《DSP 嵌入式系统开发典型案例》

作者：华清远见

第 2 章　高速数据采集系统设计

本章简介

数据采集即利用计算机获取信息，通过数据采集过程，真实世界的信息变成计算机可以识别的二进制代码，并可供计算机存储和进一步处理，或者通过网络传输。数据采集卡就是实现数据采集过程的工具，对于计算机来说，就好比是人的眼、鼻、舌一样。

外界的信息一般是模拟信号，计算机是用数字信号，数据采集系统通过模拟数字转换（即 AD）芯片来完成这一任务，这一过程称为采样。如果每秒钟采样速率少于 50 幅图像，那么我们看到的电影或电视画面，就会是不连续的，采样过程也有同样的问题，对于视频、多媒体信息来说，每秒钟的采样的次数必须要足够高，才可以真实反映这些信息。信息越丰富，采样的次数就要求越高，这实际上就是信息论中著名的奈奎斯特采样定理。因此，对于信息越来越丰富的社会来说，采样的速度要求越来越高。从收音机、黑白电视机、彩色电视机到交互式多媒体终端，数字化所需要的数据采样速度呈爆炸式增长。采样的速度快必然要求后面处理过程也要加快，采用高速的数字信号处理（DSP）专用芯片，组成高速数据采集系统甚至是超高速数据采集系统成为很多领域计算机获取信息的标准外部设备。

2.1 高速数据采集概述

高速数据采集系统具有极强的通用性，可广泛应用于军事、工业生产、科学研究和日常生活中。就像其他计算机技术一样，随着数字化生活的到来，高速数据采集系统在日常生活中的应用越来越显著。智能化建筑中各种信息，包括温度、湿度、声音、视频等各种信号都必须通过高速数据采集系统才能进入系统计算机，供智能建筑其他系统进一步处理。对于每个家庭来说，各种家用电器的智能化数字化的第一步，也是通过高速数据采集系统将外界的信息数字化。特别是对于家庭录像来说，需要同时记录视频和音频，可能还有文字等其他信息，这时需要的采样速度是非常高的，这对高速数据采集系统的性能要求极为显著。高性能的高速数据采集卡一般来说相当昂贵，这主要是由于高速电子器件成本和制作工艺，以及高密集的技术含量造成的，不过随着社会的市场需求和技术的进步，高速数据采集卡的价格不会阻碍其在日常生活中的广泛应用。

典型基于 DSP 的数据采集系统一般包括：AD 模数转换芯片、DA 数模转换芯片、SDRAM 动态数据存储元件、Flash 静态数据存储元件、HPI 主机接口、USB 接口、PCI 接口等。典型的数字信号处理过程如图 2.1 所示。

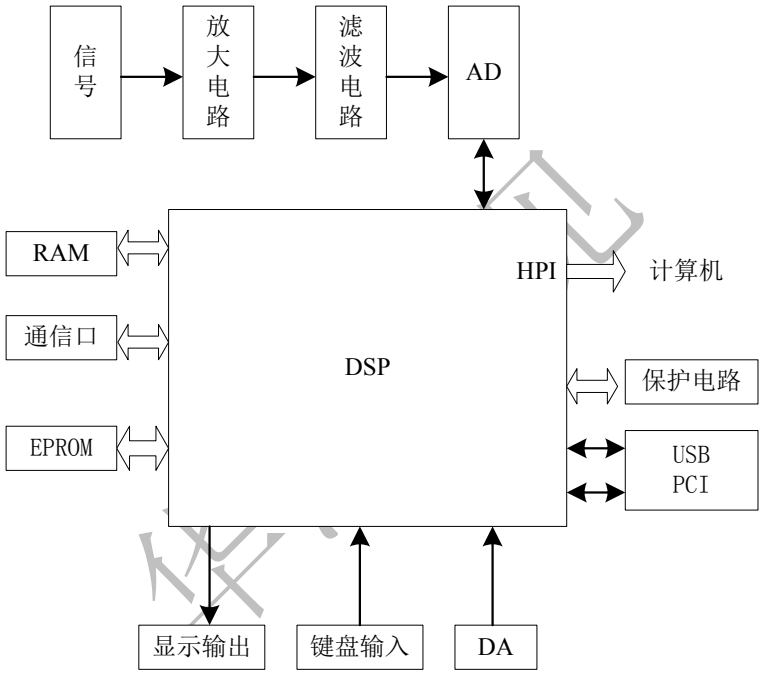

图 2.1　典型的 DSP 应用系统

图 2.1 是一个典型的 DSP 应用系统。输入信号可以是各种形式，它可以是语音信号、调制后的电话信号、编码的数字信号、压缩的图像信号，也可以是各种传感器输出的信号。如果输入信号的幅度较小或者过大，一般都需要经过放大器单元将输入信号幅度放大或者缩小后，送到 AD 进行模数转换；如果输入信号带有较大的噪声，一般需要经过一个硬件的模拟滤波单元，将信号滤波整形后，送到 AD 进行模数转换。AD 能将模拟信号变换成数字信号，但必须满足香农采样定理，也就是为了保证不丢失信息的所有信息，采样频率必须高于输入信号最高频率的 2 倍。AD 变换后得到的数字信号输入到 DSP 芯片，再由 DSP 芯片对该数字信号进行各种数字信号算法的处理。

AD 完成数据采集的功能，在 DSP 系统中处于首要的地位，DSP 对 AD 的控制是大部分 DSP 系统的重要组成部分。第 2 章和第 3 章分别介绍高速和中速两种数据采集系统的研制。

目前在模拟信号到数字信号的转换过程中，模拟信号一般分为射频、中频和低频等 3 种频率段的信号。射频信号一般经过模拟下变频，将信号的中心频率下移到中频段或者低频段，在满足带通采样定理情况下，使用 AD 进行数字转换。由于有模拟下变频技术，使得数据采集仅仅受限于信号的带宽，而不受限于信号的最高频率。通常认为如果数字逻辑电路的频率超过 50MHz，而且工作在这个频率之上的电路已经占到了整个电子系统一定的份量（比如说 1/3），就称为高速电路。相应的，对于并行采样系统，如果采样频率达到 50MHz，并行 8bit 以上；对于串行采样系统，如果采样频率达到 200MHz，一般将这种采样系统称为高速数据采集。目前高速数据采集使用较多的采样频率一般在 50M～100MHz 之间。

TI 公司的 TMS320C6000 系列 DSP 由于具有较高的运行速度和总线速度，成为高速数据采集中广泛使用的数据处理芯片，TI 公司也相应的配套一些高速的模数转换芯片。本章以 TI 公司的 TMS320C6203B 和高速 AD 转换器 ADS5422 为例，介绍高速数据采集系统的设计以及软硬件调试方法。

2.2 案例要求和应用对象

案例要求：实现对信号频率在 30MHz 以下的模拟信号的采样，并分析信号的频谱，将频谱结果通过 USB 接口传送到计算机保存和显示。

分析案例要求，本案例的关键点如下。

（1）模拟到数字信号的转换速度达到高速。

（2）USB 传输的时间必须尽量短，从而减少在处理器和计算机通信之间的时间开销。

（3）高速转换后，采样数据很大，处理器处理数据的速度必须跟上 AD 的速度。

对于关键点（1），本案例中的模拟信号最高频率为 30MHz，AD 的转换速度受限于采样定理，必须达到 60MHz 以上，本案例选择 ADS5422 高速模数转换器。

对于关键点（2），尽量减少处理器和计算机通信之间的时间开销，使用较快的 USB 2.0 协议的控制权，优化 USB 传输方案，争取实现较快的传输速度。本案例选择 CY7C68013 型号的 USB 控制权。

对于关键点（3），处理器如何处理庞大的采样数据（每秒 60M 个数据）。首先确定使用 DSP 进行核心处理器，并且不再使用其他的辅助处理器（例如 FPGA 等）。有两种方法确保跟上 AD 的转换速度，简化数据处理算法或者丢弃部分数据。第一种方法，简化数据处理算法。采用较高处理速度的 C64 系列 DSP（其最高处理速度为 1000MHz），如果处理每个数据，平均到每个数据的指令周期为 1000M/60M，不到 17 条指令。这里假设所有的指令都是单指令周期，这在实际中是不可能实现。17 条指令就连一个简单数字滤波器也很难完成。

第二种方法，丢弃部分数据。由于本案例只要求分析信号的频谱，并不需要对每个数据进行处理，只需要定时地对数据进行处理，将频谱分析结果上报计算机就可以。定时的时间可以根据具体的项目进行设置。这样，DSP 每次处理一批数据，在处理数据的时间内，停止对数据的采样，DSP 处理完该批数据后，然后启动 AD，开始下一批数据的采样。

丢弃部分数据的方法对于信号的频谱分析、载波分析、故障检测等非连续数据的场合是可行的方法，对于数字通信、文件传输等对误码率要求较高的无线传输场合不可行。

系统的基本框图由 AD、FIFO、DSP 以及 USB 接口组成，其框图如图 2.2 所示。

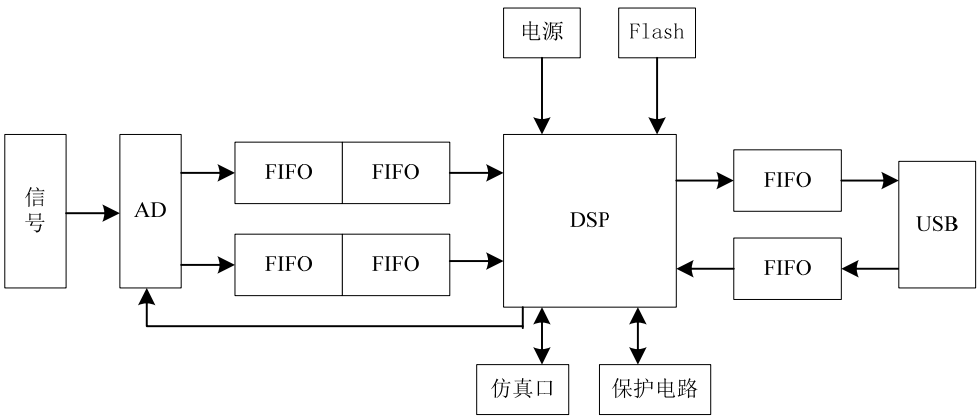

图 2.2　硬件结构框图

系统由 AD 转换器、数据存储单元（FIFO）、DSP 和 USB 接口电路，以及相应的电源转换电路、Flash 程序保存单元等组成。下面将详细介绍这些单元的设计。

2.3　器件的选择

2.3.1　AD 转换器的选择

AD 转换器的主要指标如下。

（1）分辨率（Resolution）。指数字量变化一个最小量时模拟信号的变化量，定义为满刻度与 2n 的比值。分辨率又称精度，通常以数字信号的位数来表示。

（2）转换速率（Conversion Rate）。是指完成一次从模拟转换到数字的 AD 转换所需的时间的倒数。积分型 AD 的转换时间是毫秒级属低速 AD，逐次比较型 AD 是微秒级属中速 AD，全并行/串并行型 AD 可达到纳秒级。采样时间则是另外一个概念，是指两次转换的间隔。为了保证转换的正确完成，采样速率（Sample Rate）必须小于或等于转换速率。因此习惯上将转换速率在数值上等同于采样速率也是可以接受的。常用单位是 Ksps 和 Msps，表示每秒采样千/百万次（Kilo / Million Samples Per Second）。

（3）量化误差（Quantizing Error）。由于 AD 的有限分辨率而引起的误差，即有限分辨率 AD 的阶梯状转移特性曲线与无限分辨率 AD（理想 AD）的转移特性曲线（直线）之间的最大偏差。通常是 1 个或半个最小数字量的模拟变化量，表示为 1LSB、1/2LSB。

（4）偏移误差（Offset Error）。输入信号为零时输出信号不为零的值，可外接电位器调至最小。

（5）满刻度误差（Full Scale Error）。满刻度输出时对应的输入信号与理想输入信号值之差。

（6）线性度（Linearity）。实际转换器的转移函数与理想直线的最大偏移，不包括以上 3 种误差。

AD 的其他指标还有绝对精度（Absolute Accuracy）、相对精度（Relative Accuracy）、微分非线性、单调性和无错码、总谐波失真（THD，Total Harmonic Distotortion）和积分非线性等。

对于 AD 转换器，选取的标准主要决定于采样频率和位数，以及价格、供货周期、应用情况等其他因数。生产高速 AD 的主要厂家有 AD 公司、Maxim 公司以及 TI 公司（也就是 BB 公司）。这三家公司在高速 AD 上的产品种类不是很多，根据对各种 AD 芯片的查阅，选择 TI 公司的 AD 转换芯片 ADS5422 。

ADS5422 是 14bit 的最高采样频率可达 62Msps 的高速 AD 转换芯片，采用单一 5V 电源供电，在采样频率为 10M 时其最大动态范围为 82dB，最高信噪比达到 72dB，其数字量输出可以直接和 5V 或者 3.3V 的 CMOS 芯片连接，模拟量输入的峰峰值为 4V，可以直接输入 0.5～4.5V 的模拟量，封装形式为 64 脚的扁平四方封装，目前 TI 的官方报价为 29 美元/片（一次购买千片以上的单价）。国内也有该芯片出售，国内价格在 300 元左右。

14bit 的 AD 转换适应信号的范围为 $10\lg(2^{14})$dB = 42dB，基本上可以适应各种应用场合。ADS5422 的采样频率的大小由其输入时钟决定，输入时钟的范围可以在 16ns～1μs，输入时钟为 16ns 时对应采样频率为 62MHz，AD 可以接受 3V 或者 5V 的 TTL 或者 CMOS 电平。DSP 可以提供该时钟信号，并且可以软件设置输入时钟的各种特征量，包括时钟频率、高电平宽度等，基本上可以满足 AD5422 对时钟信号的要求。这里确定 AD 的实际采样频率为 60MHz。这样，一秒钟内采样的数据量为 50M 个，由于 DSP 系统无法及时处理这些数据，在数据处理之前，必须将这些数据保存起来，使用 FIFO 保存 1M 个数据，也就是 20ms 内的采样数据，1M 个数据采集结束开始信号处理。由于高速 AD 采样导致信号不稳定，甚至出现错误。将设计多层板，加强布线的合理性，从电路板上尽可能去除干扰；其次提高算法的效率，节省计算时间。

和 ADS5422 功能接近的其他型号的 AD 还有 AD 公司的 AD9244。和 ADS5422 相比，两者数据位数都是 14bit，在信噪比上两者相近，时钟输入和操作方法相近，电源都是 5V，输出数字信号都可以和 3.3V 的芯片兼容；其主要优点是功耗是 ADS5422 的一半，500mW；其主要缺点是输入模拟电压峰峰值为 ADS5422 的一半，只有 2V。

AD 公司其他的高速 AD 芯片还有 AD6644，为其早期产品，操作方法和 ADS5422、AD9244 不一样，AD6644 功耗达到 1.3W。和 AD9244 相比，没有什么优点，AD9244 是其替代产品。

高速 AD 的另外一个厂家 Maxim 公司也有一批高速 AD 产品，但采样频率在 40MHz 以上没有 14bit 数据的 AD，其产品优势主要集中在中速 AD 上。

2.3.2 FIFO 的选择

FIFO 大致可以分为 3 种类型，异步 FIFO、触发 FIFO、同步 FIFO。FIFO 一般都提供状态标志信号，这些状态信号包括空标志、满标志以及半满标志。此外，大部分 FIFO 还提供专门的字长和字深扩展引脚。

根据 AD 和 DSP 的接口，可选择美国 IDT 公司的 FIFO 芯片 IDT72V2113，IDT72V2113 具有 512K×9bit 的数据空间、133MHz 的操作时钟、最快 7.5ns 的读写周期、单一 3.3V 电源供电、80 脚 TQPF 封装的同步 FIFO。目前 IDT72V2113 的国内价格为 300 元左右。

由于有 1M 多的数据量，而且 IDT72V2113 只有 512K×9bit 的数据单元，所以必须做字长和字深扩展。如果不做字长和字深扩展将选择更大数据空间的 FIFO，而目前更大的 FIFO 市场上很少，价格也很高。

IDT72V2113 和 DSP 的接口使用 DSP 的扩展总线接口，可以方便地实现两者之间的连接。IDT72V2113 的写时钟和 AD 的采样时钟一样由 DSP 同一个引脚提供，DSP 的该引脚可以同时驱动这两个芯片；IDT72V2113 的读时钟由 DSP 的扩展总线接口提供；IDT72V2113 的其他控制信号，包括写使能、读使能、部分复位等信号均由 DSP 提供。

2.3.3 DSP 的选择

根据数据的处理时间，选择 TMS320C6203B 型号的 DSP，其时钟频率为 250MHz 和 300MHz 两种，选择后者，考虑高频电路情况，实际时钟频率可以达到 300MHz，内部指令时钟达到 2400MHz。TMS320C6203B 内部具有 4Mbit 的数据空间和 3Mbit 的程序空间、3 个多通道缓冲串口（可以做通用 I/O 口使用）、4 个 DMA 通道、32bit 的扩展总线、两个定时器、双电源供电（I/O 口为 3.3V，芯片核电压为 1.5V）、功耗低、384 脚 BGA 封装、目前官方报价为 84 美元左右，国内价格 1000 元左右。

使用 TMS320C6203B 的 32bit 扩展总线接口连接 FIFO，实现从 FIFO 读取数据；使用 TMS320C6203B 的定时器输出信号控制 AD 的采样频率；使用 TMS320C6203B 的 EMIF（外部存储器接口）连接 Flash 程序单元；同时使用 TMS320C6203B 的 EMIF 接口连接 USB 接口设备；此外，TMS320C6203B 必需的电压转换芯片、时钟信号、分频电路等外围电路这里不再一一说明。

信号处理的算法编成程序保存在外部的 Flash 芯片上，供 DSP 上电读程序到其内部 RAM 单元，全速运行程序。

其他型号的 DSP 芯片，TI 的 C5000 系列芯片将满足不了处理时间要求，C64 系列的芯片可以满足系统要求，但性能较高。对于 C62 系列的 DSP，由于需要外部总线接口连接同步 FIFO，而在该系列中，只有 3 种型号有该接口，分别为 C6202、C6203、C6204。其中 C6204 价格最为便宜，报价仅有 20 美元，但其内部 RAM 只有 16KWord，最大的缺点是其时钟频率最高只有 200MHz，显的过低，很难满足实际要求。C6202 和 C6203 基本上一样，主要的区别只是其内部 RAM 大小不一样，C6202 为 32KWord，C6203 为 128KWord，两者的时钟频率都有 250MHz 和 300MHz 两种，这样方便产品升级，此外，两者的管脚完全兼容，价格分别是 C6202 为 55 美元左右，C6203 为 85 美元左右。

2.3.4 USB 控制器的选择

目前市场上 USB 的控制器很多，例如 National Semiconductor 公司的 USBN9602，Plilips 公司的 PDIUSBD12 以及 Cypress 公司的 USB 控制器。选择 Cypress 的 USB 控制器 CY7C68013。CY7C68013 具有以下优点。

（1）基于 RAM 的“软”系统解决方案，不需要 ROM 或其他的固化存储器，只使用片内的程序/数据 RAM。可通过主机下载的方式来配置 USB 接口，将需要在 CY7C68013 上运行的固件，存放在主机上，当 USB 设备连上主机后，下载到设备上，这样就实现了在不改动硬件的情况下很方便地修改固件，使接口系统的修改和升级变得非常简单。可使外设硬件的更新和升级更加方便快捷。

（2）数据吞吐量完全符合 USB 2.0 协议要求，并向下兼容 USB 1.1X 协议，可以向用户提供足够的端口、缓冲区和快速的传输速度。可提供 USB 协议所要求的 4 种传输方式：控制传输、中断传输、批量传输和同步传输，因而能满足用户对各种类型数据传输的需求。

（3）片上的智能串行接口引擎（SIE）执行所有基本的 USB 功能，将嵌入的 MCU 解放出来以用于实现其他丰富的功能，以保证持续高速有效的数据传输，使用户摆脱了复杂的协议细节，简化了用户配置代码，加快了开发过程。

（4）内嵌增强型 8051 处理器，兼容 8051 指令系统；一个指令周期仅需 4 个时钟周期，可提供标准 8051 处理器 3 倍以上的处理能力。双数据指针，方便了数据块的转移。使用片内 RAM 作为数据/程序存储器，非复用数据/地址总线，使程序执行速度更快，并且使其同外部器件的连接更加简单。

（5）具有 4KB 的大容量 FIFO 用于数据缓冲，当作为从设备时，可采用 Synchronous/ Asynchronous FIFO 接口与主设备（如 ASIC、DSP 等）连接；当作为主设备时，可通过通用可编程接口（GPIF，Generally Programmable Interface）形成任意的控制波形来实现与其他从设备连接，能够轻易地兼容绝大多数总线标准，包括 ATA、UTOPIA、EPP 和 PCMCIA 等。

（6）具有独特的休眠模式，可以降低系统功耗，延长器件的使用寿命。

2.3.5 Flash 的选择

Flash 称为闪速存储器，是一种高速的、电擦除、电改写的非易失性的存储器。因它具有速度快、容量大、功耗低以及成本低等优点，所以在电子、计算机、通信、军事以及航空航天等领域得到了广泛应用。Flash 的选择，主要考虑以下几个方面的因素。

（1）可靠性。闪速存储器的可擦除编程次数和数据的保存时间是选购的重要标准之一。一般选择可重复擦写的次数在 10 万次以上，保存时间在 100 年以上的芯片。

（2）容量。根据系统的要求，选择合适的大小。容量和价格成正比，容量越大，价格越高。

（3）读写时间。读写时间是关键性指标。一般选择和 DSP 能达到最佳配合的芯片。读写速度和价格成正比，读写速度越快，价格越高。

（4）写周期和擦除周期功耗。

（5）和 DSP 芯片的兼容性。包括速度、电压、时钟以及硬件连接等方面。

除了上述几个因素以外，选择 Flash 芯片还需考虑到芯片的价格、封装形式、供货周期、技术支持以及普及程度等情况。

根据以上原则本案例选择美国 SST 公司的 128K×8bit 的闪速存储器 SST29LE010。SST29LE010 是 SST 公司的 29 系列 128K×8bit 多用途闪速存储器。它的主要特点和技术指标如下。

（1）高的可靠性。可承受 10 万次擦写、数据保持时间大于 100 年。

（2）快速写操作。每页 128Byte，页写时间为 5ms（典型值），片写时间为 5s（典型值）、有效字节写循环周期是 39μs（典型值）。

（3）低功耗。工作时功耗为 10mA，休眠时功耗为 10uA。

（4）兼容性好。兼容 TTL 电平和 CMOS 电平。

（5）单一 3.3V 电压供电，可以直接和 DSP 连接。

（6）达到 JEDEC（电子器件工程联合会）标准的 Flash EEPROM 管脚输出和命令集。

2.4 硬件电路设计

2.4.1 ADS5422 的电路设计

ADS5422 的内部结构如图 2.3 所示，ADS5422 的时钟信号由 CLK 或者 $\overline{CLK}$ 从外部引入，和一般 AD 转换器不同的，外部时钟信号必须经过内部时钟电路提供给内部各个逻辑单元，由于 ADS5422 的内置时钟电路，使得 ADS5422 对各种时钟都兼容，包括正弦波或者方波、TTL 电平或者 CMOS 电平、单端时钟信号或者差分时钟信号。外部模拟信号从 IN 和 $\overline{IN}$ 引脚输入，首先经过一个高带宽的线性跟随器（T / H）单元，确保 AD 具有很高的无寄生动态响应；经过跟随器后的信号到达 ADS5422 的中心单元，一个 14bit 的管道型（也称为直通型）（Pipelined）AD，将模拟信号转换为数字信号，同时 ADS5422 内部的梯度选择单元以及模式选择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，在一次数据输出之后可以将 ADS5422 的引脚 $\overline{OE}$ 设置为高，从而设置数据输出寄存器为高阻状态，以节省芯片的功耗。此外，ADS5422 提供外部数据输出电压驱动 VDRV 引脚，一般接 5V 电压，但如果将引脚连接到 3.3V 电压，数据输出将可以直接接到 3.3V 的外部器件上。

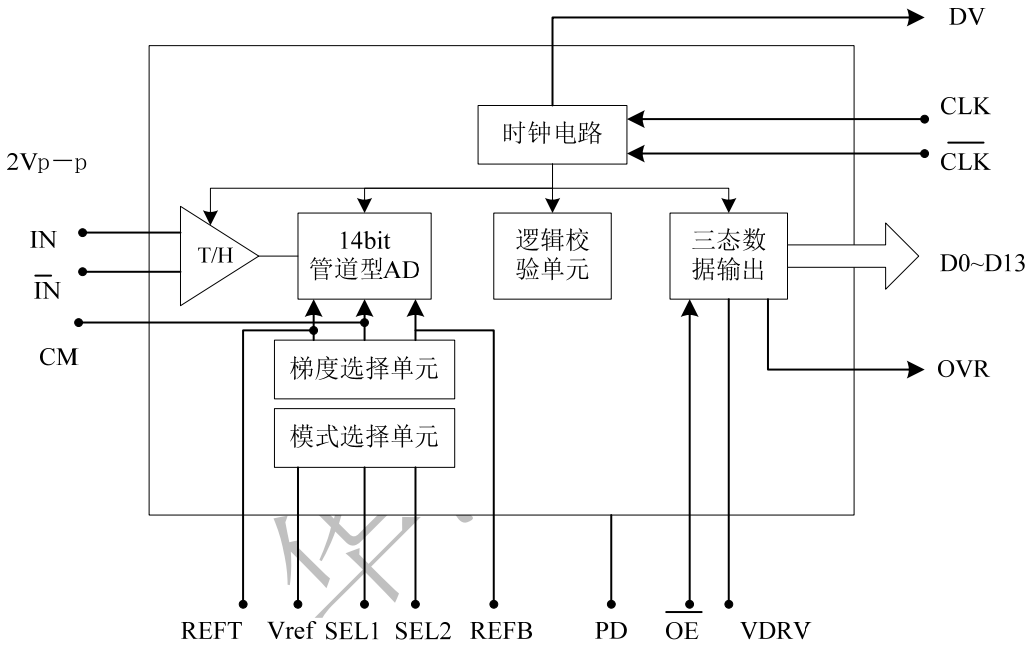

图 2.3　ADS5422 内部结构框图

ADS5422 的模拟信号输入可以采取单端输入方式或者差分输入方式两种。虽然单端输入方式连接比较简单，但抗噪性能差，所以一般采取差分输入方式。这样，可以尽量减少信号噪声以及电磁的干扰，尤其是采用差分输入方式可以将所有的偶次谐波通过正反两个输入信号基本上互相抵消。此外为了保证在高频下信号的失真最小，增加输入信号的抗噪性能，即使是单端输入信号也建议在信号输入端前加上前置放大器以及 RF 变压器，将单端信号转换成差分信号再输入到 AD。

ADS5422 的模拟信号差分输入方式需要同时使用 IN 和 $\overline{IN}$ 引脚，下面以单端信号转换成差分信号为例说明 ADS5422 的差分信号输入方法，其硬件连接方法如图 2.4 所示。

图 2.4 中首先使用放大器 OPA687 以及 RF 变压器将单端信号转换成差分信号，然后输入到 ADS5422，其中 ADS5422 的公共端 CM 和 RF 变压器的公共端连接，RF 变压器的匝数比根据信号确定，可选择线艺公司的 TTWB2010 产品或者其替代产品。为了增强信号的稳定性，在 ADS5422 的每个信号的输入前加上 RC 元件组成的低通滤波电路，图 2.4 中推荐值 R_t 为 50Ω，R_{in} 为 22Ω，C_{in} 为 10pF，这些元件也可以根据具体的信号进行调整，一般情况下电阻值在 10～100Ω 之间，电容值在 10～200pF 之间。

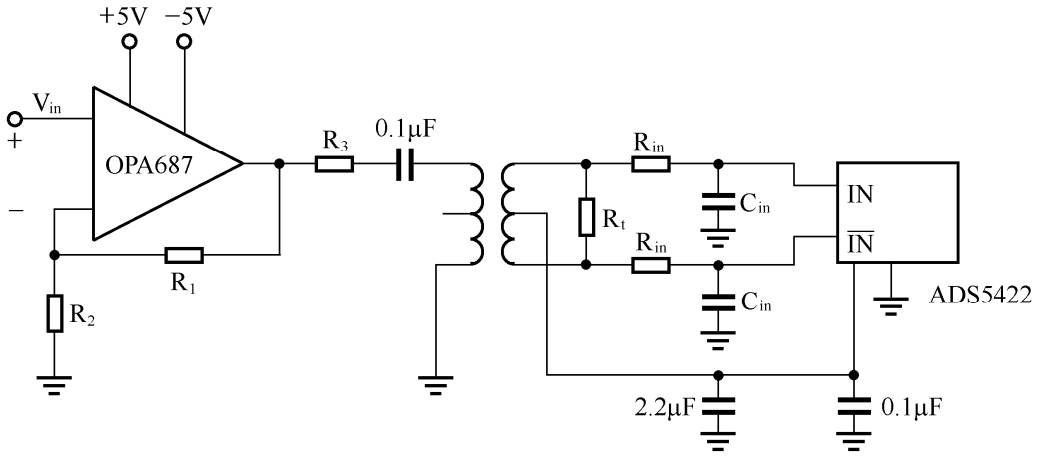

图 2.4　ADS5422 的差分信号输入连接

ADS5422 和 TMS320C6203B 的硬件连接如图 2.5 所示。使用 TMS320C6203B 的 32bit 外部扩展总线接口（XBUS）连接 ADS5422，实现 XBUS 从 ADS5422 读取数据并存储在 TMS320C6203B 的内部 RAM 中（由于 ADS5422 只有 14bit 数据，实际上只需要使用 XBUS 的低 14bit）；使用 TMS320C6203B 的定时器输出信号 TOUT0 提供精确稳定的时钟给 ADS5422，控制 AD 的采样频率，并且该时钟可根据定时器参数由软件设置，增加 AD 采样频率的灵活性。在 DSP 内部寄存器中，将多通道缓冲串口（MBSP）的引脚配置成通用的 I/O 引脚，使用 DR0、DR1 以及 DX0 引脚读入或者写入 ADS5422 的控制信号 OVR、DV 以及 $\overline{OE}$ 。ADS5422 的详细电路如图 2.6 所示。

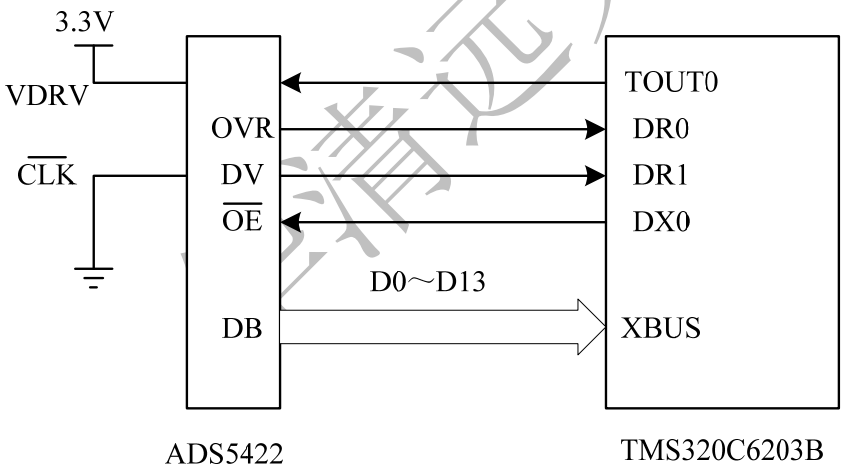

图 2.5　ADS5422 和 TMS320C6203B 的连接

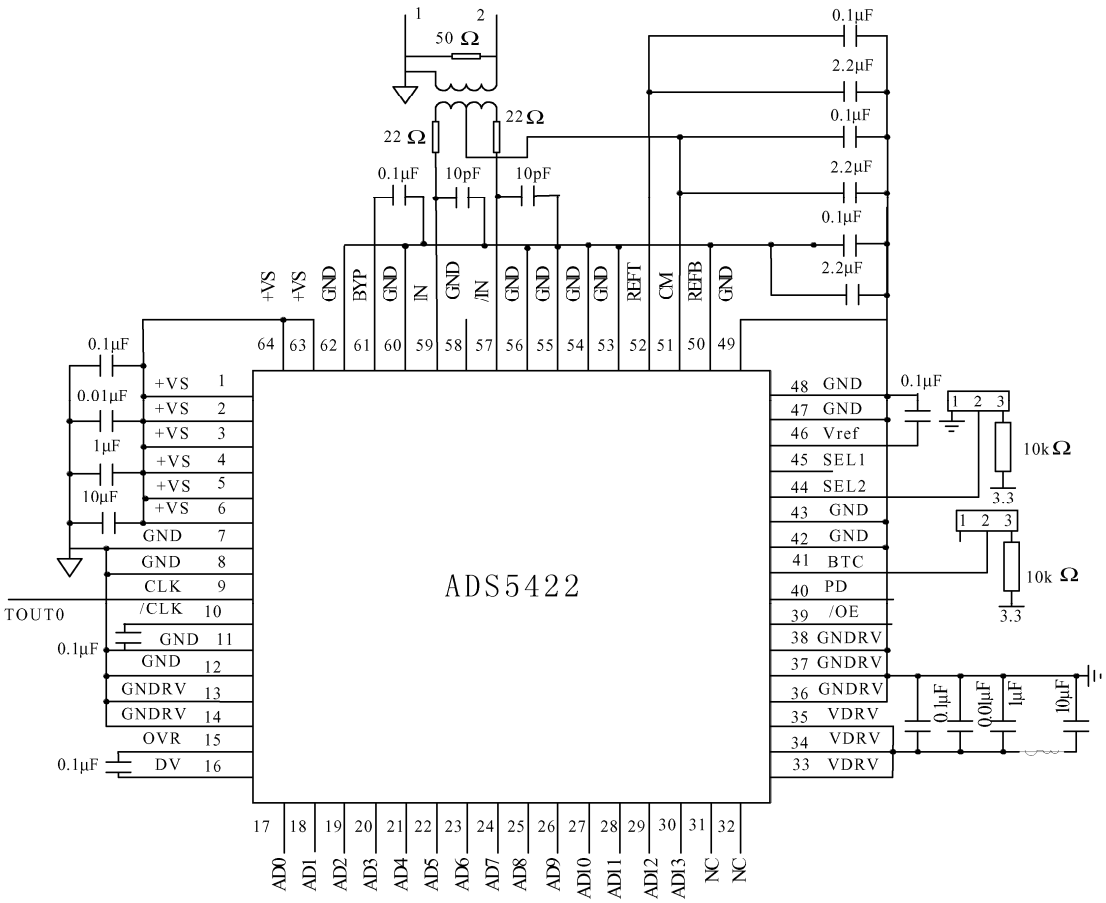

图 2.6　ADS5422 的详细连接电路

2.4.2　IDT72V2113 的电路设计

IDT72V2113 的内部结构如图 2.7 所示。主要由一个内部 RAM 阵列以及读写控制单元、读写指针单元、输入输出寄存器、标志信号以及复位单元组成。其内部 RAM 阵列采用先进先出设计技术，外部数据首先存到输入数据寄存器，再传送到 RAM 阵列，依次保存，数据的先后顺序通过修改写指针确认。RAM 阵列始终检测输出数据寄存器的状态，一旦为空，RAM 阵列的数据送到输出数据寄存器，外部设备可以直接从输出数据寄存器读出数据，数据的读顺序通过读指针来控制。通过设置输出使能 $\overline{OE}$ 引脚为高状态来禁止数据的输出，以减低芯片的功耗。为了方便数据的读写，IDT72V2113 还增加了一些对数据读写的控制信号，包括读写使能、读写时钟以及字宽控制等。

大容量数据存储是高速数据采集系统迫切需要解决的问题，例如，一个 20MHz 采样速率、8bit 的 ADC，在一秒钟的时间内所采集到的数据量是 20MB。虽然 IDT72V2113 的单片容量是 512K×9bit，可以很好地满足一般的数据采集系统的需要，但是，对于高速、无间隔的数据采集系统来说，一片的容量是不够的。IDT72V2113 便于扩展的特性可以很容易地解决这个问题。其容量扩展可以分为字长扩展和深度扩展，且不需要外部控制电路，电路设计及软件开发十分方便。

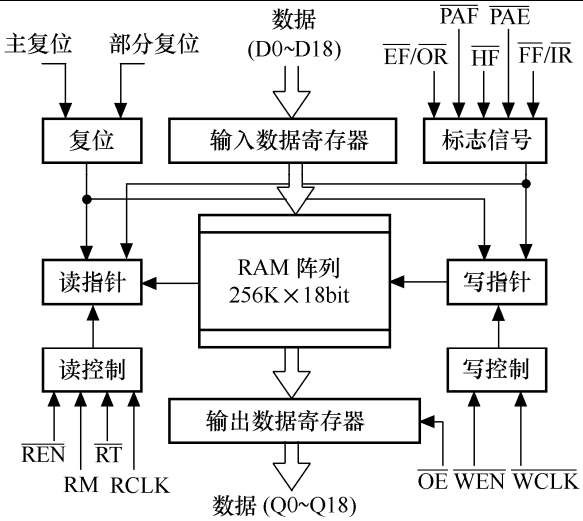

图 2.7　IDT72V2113 内部结构框图

IDT72V2113 的字长扩展比较简单，只要把各个芯片的控制信号连在一起就可以实现。需要注意的是 $\overline{EF}/\overline{OR}$ 和 $\overline{FF}/\overline{IR}$ 两个引脚，在标准模式下这两个管脚的功能为 $\overline{EF}$ 和 $\overline{FF}$，把各个芯片这两个管脚分别相与；在 FWFT 模式下，这两个管脚功能为 $\overline{OR}$ 和 $\overline{IR}$，把各个芯片的这两个管脚分别相或，这样就可以确保同步读写每一个 IDT72V213。IDT72V2113 的字长扩展的连接如图 2.8 所示。

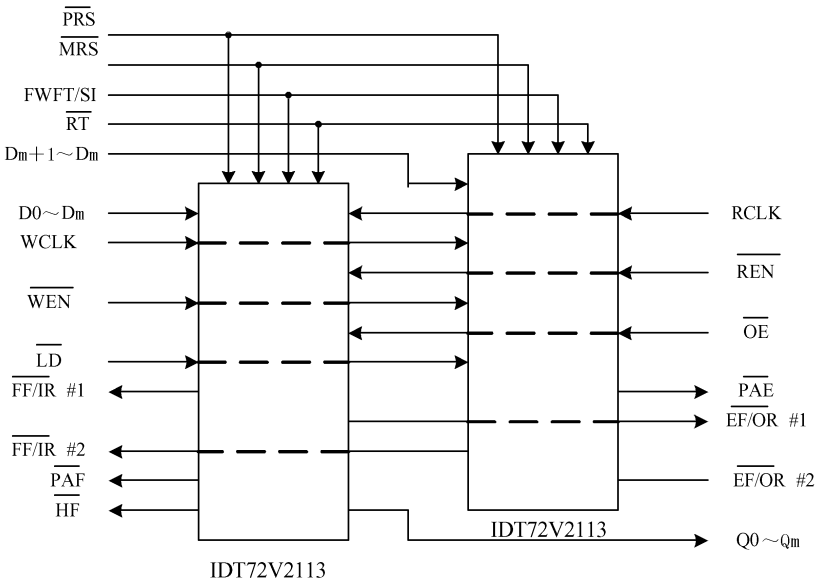

图 2.8　IDT72V2113 的字长扩展

IDT72V2113 的深度扩展方式仅仅适用于 FWFT 工作模式。两片 IDT72V2113 的字深扩展的硬件连接如图 2.9 所示，其中，传输时钟可以选择写时钟和读时钟中频率高的那个时钟信号。工作原理：当有数据写入第一片 FIFO 中后，其输出允许信号（$\overline{OR}$）低有效，从而使第二片 FIFO 的写使能信号有效；同时，只要第二片 FIFO 中仍有空间，它的输入允许信号（$\overline{IR}$）有效（低有效），从而使第一片 FIFO 的读使能信号（$\overline{REN}$）有效，这样，在传输时钟的驱动下，数据由第一片 FIFO 向第二片 FIFO 传送，直到第二片 FIFO 写满为止，以后的数据将储存在第一片 FIFO 中。通过深度扩展，两片 IDT72V2113 可形成容量为 1M×9 bit 的数据缓冲。

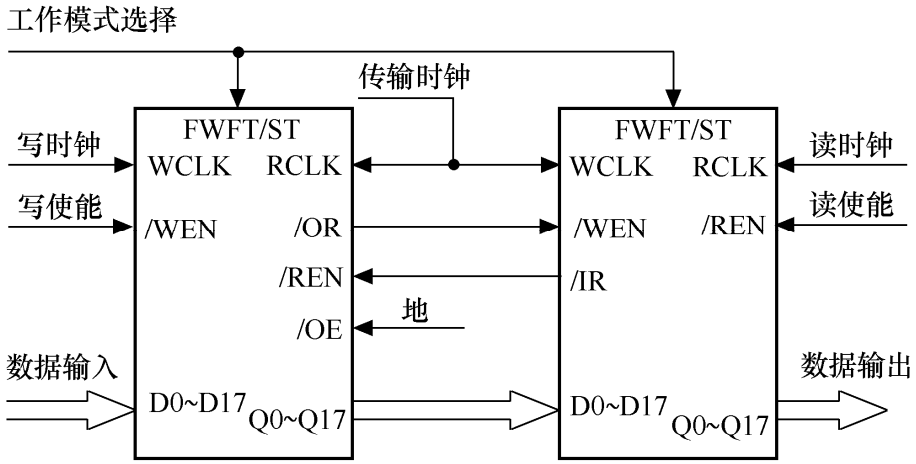

图 2.9　IDT72V2113 的字深扩展

IDT72V2113 不仅可以通过字长扩展和深度扩展来实现容量扩展，而且可以将两者结合起来，进行更大容量的扩展，如用 4 片 IDT72V2113 扩展成容量为 1M×18 bit 的数据缓冲，连接方法如图 2.10 所示。数据同步输入到 FIFO-1 和 FIFO-2，FIFO-1 和 FIFO-2 做字长扩展；FIFO-1 的数据输出到 FIFO-3，FIFO-1 和 FIFO-3 做字深扩展；同样的，FIFO-3 和 FIFO-4 做字长扩展，FIFO-2 和 FIFO-4 做字深扩展。这种连接方式下，FIFO-1 和 FIFO-2 的读时钟以及 FIFO-3 和 FIFO-4 的写时钟必须同步，为了方便，一般将这 4 个时钟信号以及 FIFO-1 和 FIFO-2 的写时钟一共 6 个时钟信号连接在一起，由一个时钟信号提供，需要注意的是提供到 6 个信号的时钟必须有较强的驱动能力，否则不能提高有效的时钟周期。如果 FIFO-1 和 FIFO-2 有数据写入，这些数据将自动写到 FIFO-3 和 FIFO-4 中。

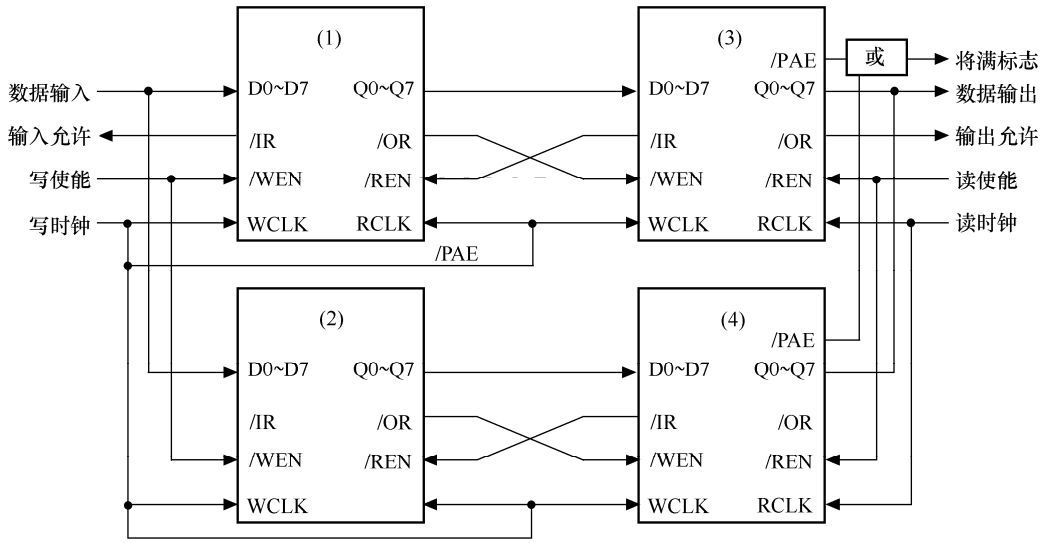

图 2.10　IDT72V2113 的容量扩展

IDT72V2113 与 TMS320C6203B 的连接是通过 TMS320C6203B 外部扩展总线（XBUS）的 $\overline{\text{XCE3}}$ 空间，数据通过 DMA 方式从 IDT72V2113 传送到 TMS320C6203B 的片内 RAM 中，具体连接如图 2.11 所示。

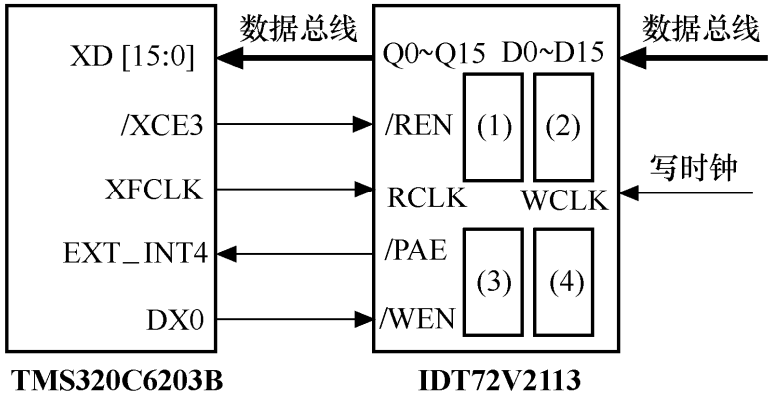

图 2.11　IDT72V2113 与 TMS320C6203B 的连接

图 2.11 中，4 片 IDT72V2113 经过字长和深度扩展形成 2MB 的数据输入缓冲，输入数据总线（D0～D15）、输出数据总线（Q0～Q15）、读使能（ $\overline{\text{REN}}$ ）、读时钟（RCLK）、写使能（ $\overline{\text{WEN}}$ ）、写时钟（WCLK）和将空标志信号（ $\overline{\text{PAE}}$ ）是由 4 片 IDT72V2113 的相应信号组合形成的； $\overline{\text{XCE3}}$ 为外部扩展总线的空间选择信号，XFCLK 为外部扩展总线的输出时钟，EXT_INT4 是 TMS320C6203B 的外部中断信号 4，DX0 用作通用输出口，控制 IDT72V2113 的写使能信号。

上电后，TMS320C6203B 进行初始化，外部扩展总线的 $\overline{\text{XCE3}}$ 空间设置为同步 FIFO 读模式，DMA 通道 0 配置为每次传输含 8 帧，每帧 128Byte，同步事件设置为外部中断 4，触发极性为高电平；手动启动 DMA 通道 0，设置 DX0 为低电平；随着数据不断写入 IDT72V2113，当 IDT72V2113 中的数据量大于 128 Byte 时，IDT72V2113 的将空标志信号（ $\overline{\text{PAE}}$ ）由低电平变为高电平，使得 TMS320C6203B 的外部中断信号有效，从而触发 DMA 传输；TMS320C6203B 的 DMA 通道 0 开始通过外部扩展总线读取 8×128 Byte 的数据，存储于内部 RAM 中，然后向 TMS320C6203B 发送中断，通知 TMS320C6203B 处理数据；TMS320C6203B 处理完数据后，重新启动 DMA 通道 0，进行下一次 DMA 传输；如此循环，直到处理完所有数据。总结以上，IDT72V2113 的详细电路如图 2.12 和图 2.13 所示。图 2.12 和图 2.13 为并行和 ADS5422 连接的两片 FIFO。

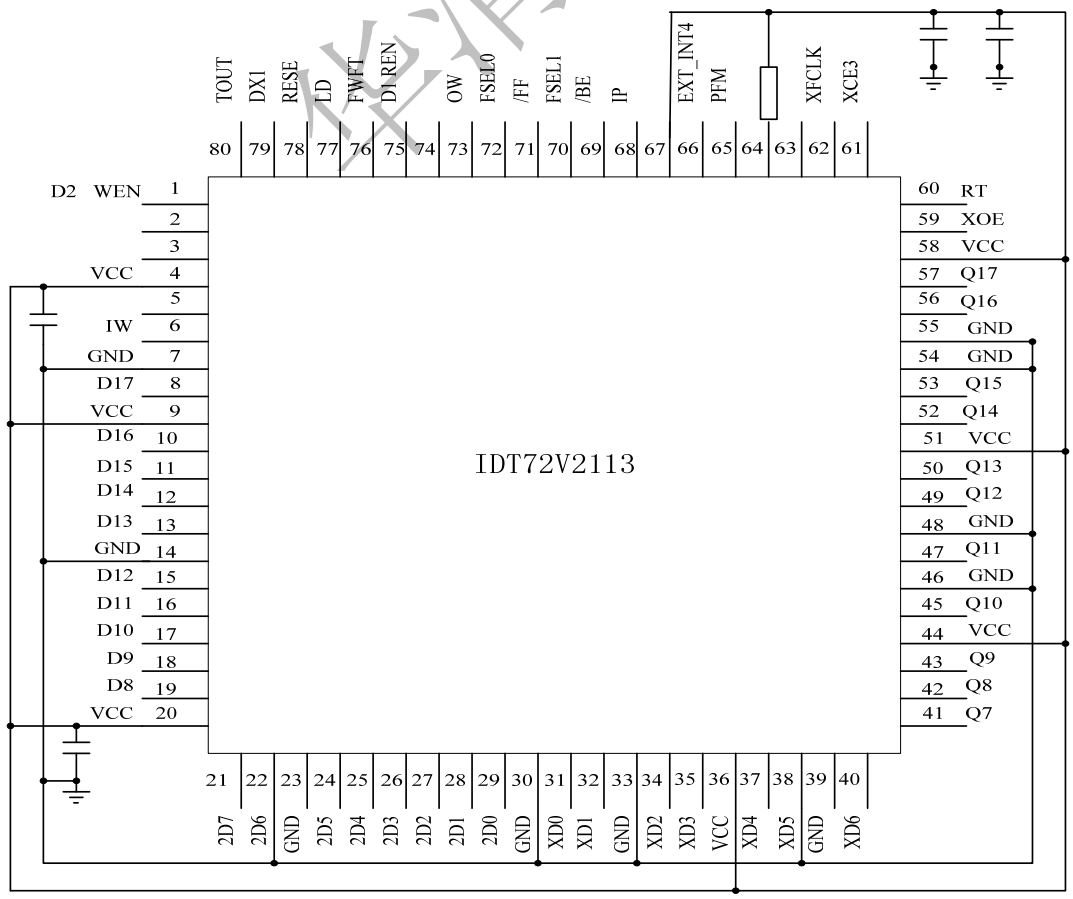

图 2.12　IDT72V2113 的详细电路 1

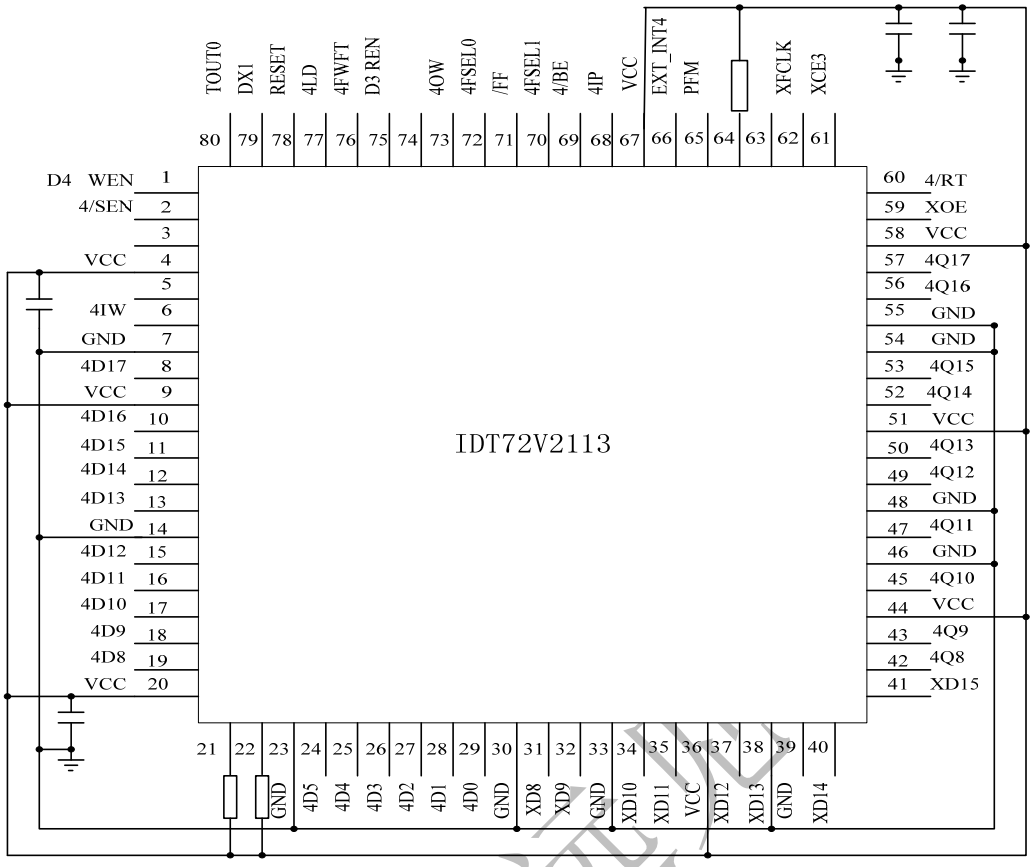

图 2.13　IDT72V2113 的详细电路 2

2.4.3　CY7C68013 的电路设计

USB 控制器 CY7C68013 的功能框图如图 2.14 所示。其功能框图主要包括 5 个部分。

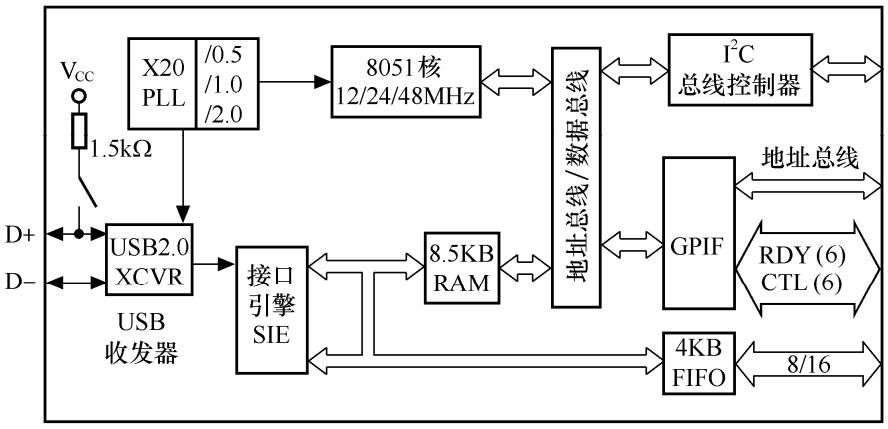

图 2.14　CY7C68013 的功能框图

（1）收发器。USB 和 PC 机的接口只有 4 根电源线：+5V、GND、D+和 D−。数据以 480Mbit/s 的全速信号或者以 240Mbit/s 的半速信号在 D+和 D−信号线上差分传输。收发器固化在芯片上，不需要外部电路，一般只用来选择全速或者低速的上拉电阻。

（2）智能接口引擎（SIE）。SIE 通过包排序、信号产生、信号检测、CRC 产生、CRC 校验、NRZI 数据编码、位填充、包标识产生和解码等功能来处理 USB 通信协议，并保证传送到 USB 电缆上的数据字节以 LSB 开头。

（3）通用微控制器以及片上 RAM。USB 控制器采用通用的 8051 微控制器，即编程语言为 51 系列单片机的通用语言，可使实际操作更加灵活方便。USB 接口控制器的 8051 代码编写的底层程序可以固化在 ROM 内，但为了确保 USB 接口控制器能与外设达到最佳连接，通常使用的方法是，将 8051 代码编写的底层程序通过 USB 口从主机下载到内部 RAM，以便于修改、调试和更新底层程序。之所以能下载 8051 代码编写的底层程序，是因为 USB 接口控制器一上电就能在硬件上自动完成枚举全过程，而不需要任何软件支持。完成枚举后便可作为一个 USB 设备（此时为缺省 USB 设备）与计算机通信，即可进行底层程序下载。下载完后，8051 微控制器的内核脱离复位状态开始执行代码。此外，下载完后还可以通过底层程序对 USB 设备重新配置，这个重新配置过程叫做再枚举。

（4）I^2C 总线控制器。主要为完成将 USB 控制器作为主设备时必须的配置。

（5）片内 FIFO。其容量为 4KB，可以快速实现与不同速度的外设通信。

CY7C68013 是一个非常方便的 USB 2.0 实现方案，它提供与 DSP 或者 MCU 连接的接口，连接方法有两种：Slave FIFOs 和 Master 可编程接口 GPIF。以下介绍 Slave FIFOs 异步读写方式。Slave FIFOs 方式是从机方式，DSP 可以像读写普通 FIFO 一样对 CY7C68013 内部的多层缓冲 FIFO 进行读写。具体的电路连接如图 2.15 所示。

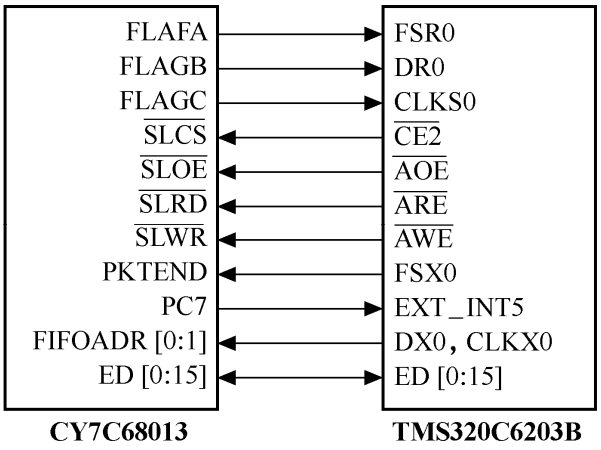

图 2.15　USB 与 DSP 的连接

FLAGA、FLAGB 和 FLAGC 是 CY7C68013 内部 FIFO 的状态标志，TMS320C6203B 通过通用 I/O 口来获得 FIFO 的空、半满（由用户设定半满的阈值）和满等状态信息。TMS320C6203B 对 CY7C68013 内部 FIFO 的选择以及数据包的提交也是通过通用 I/O 口来实现。TMS320C6203B 通过 EMIF 接口的 CE2 空间对 CY7C68013 进行读写操作。工作过程为：DSP 通过 USB 向 PC 发送数据时，首先查看空、半满和满这 3 个状态信号，然后向 USB 写入适当大小的数据，以保证数据不会溢出；PC 机通过 USB 向 DSP 发送命令字时，USB 通过中断方式通知 DSP 读取命令字。CY7C68013 的详细电路如图 2.16 所示。

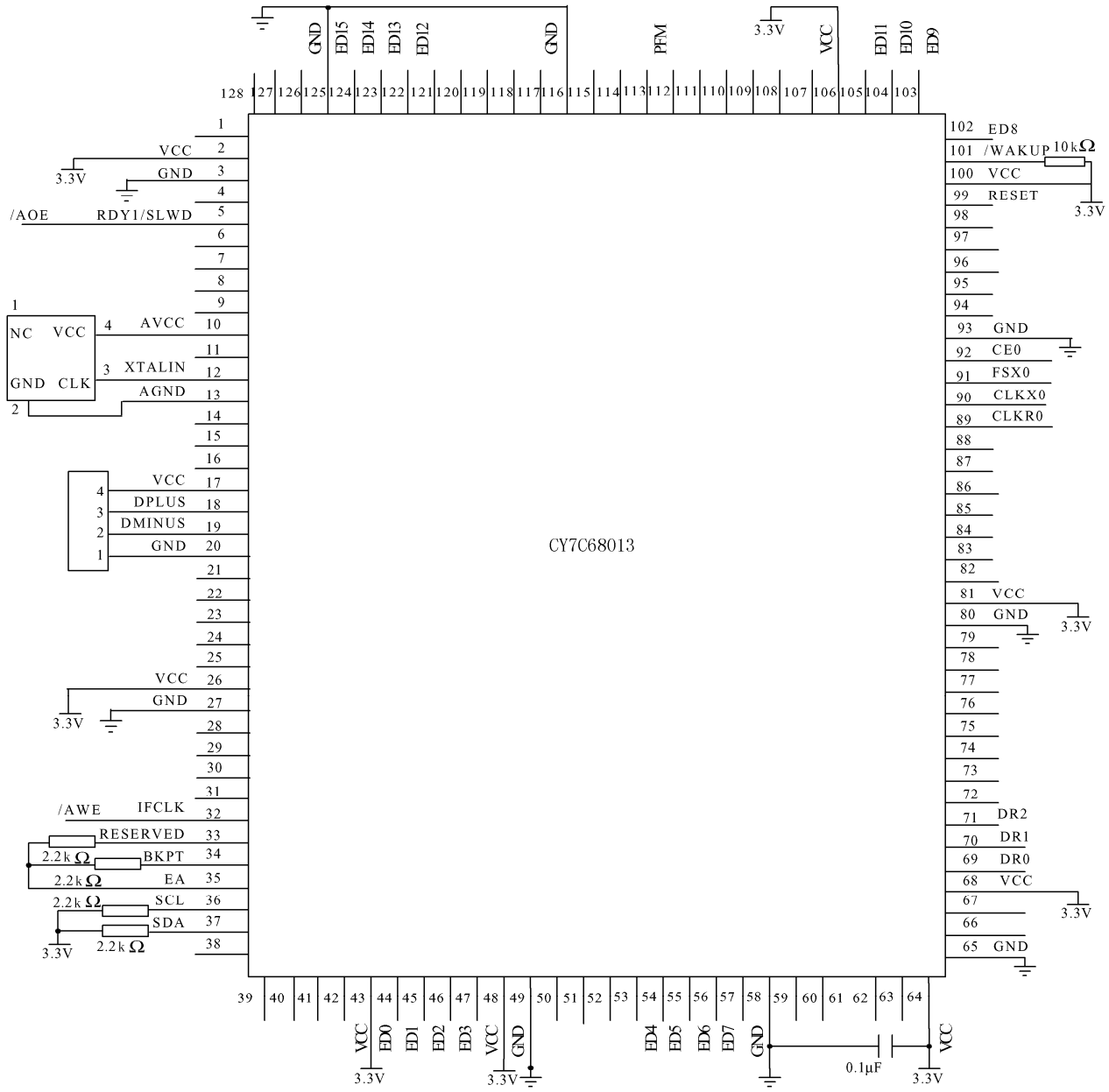

图 2.16　CY7C68013 的详细电路

2.4.4　SST29LE010 的电路设计

闪速存储器 SST29LE010 连接到 TMS320C6203B 作为其外部程序存储器，供 DSP 上电时启动载入程序（BOOTLOAD）使用，其作用是将 Flash 中保存的程序载入 DSP 中运行。TMS320C6203B 与 Flash 的连接如图 2.17 所示。其地址和数据总线连接到 TMS320C6203B 的 EMIF 接口总线上，Flash 的片选信号连接到 DSP 的 CE1 引脚，配置成 DSP 的 CE1 空间，CE1 引脚在上电复位后为低电平。此外，Flash 的读写信号分别连接到 EMIF 接口的读写信号引脚上。

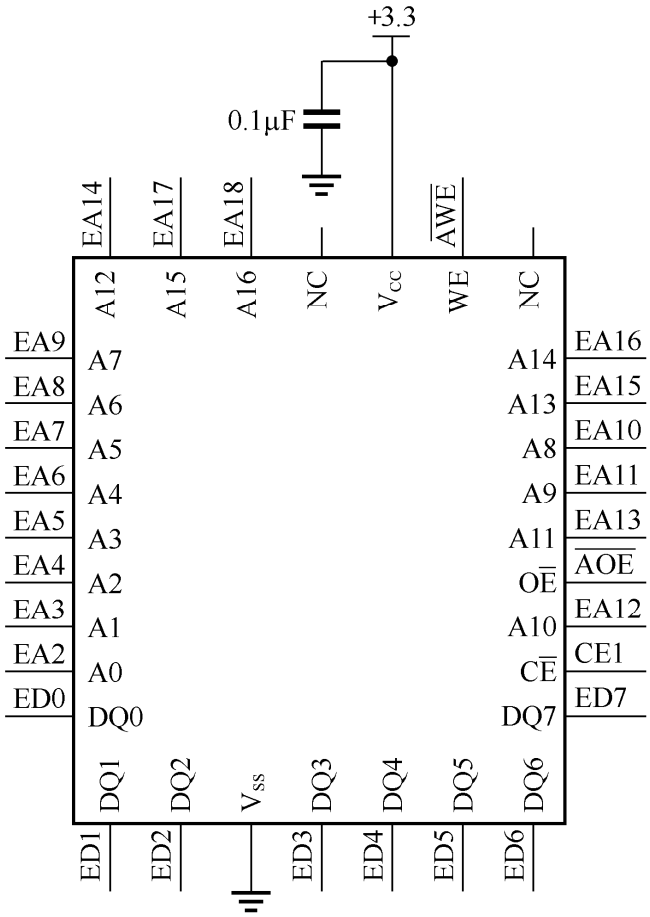

图 2.17　TMS320C6203B 和 Flash 的连接

2.4.5　电源和复位电路设计

在 TI 公司的 DSP 系列中，TMS320C6000 系列的 DSP 一般都采用 3.3V 和 1.5V 电压供电，其中 I/O 采用 3.3V 电压，芯片内核采用 1.5V 电压。实际常用的只有 5V 电压，所以必须采用电压转换芯片，将 5V 电压转换成 3.3V 和 1.5V，供 DSP 使用。TI 公司提供很多专门的电压转换芯片，供用户选择使用。有关电压转换的芯片很多，本案例选择 TI 公司的 TPS70348 系列单电源芯片，该芯片可以同时输出 3.3V 和 1.5V 两种电压，供 DSP 使用。

TPS70348 的硬件电路如图 2.18 所示。TPS70348 除了提供电源外，还提供上电复位和手动复位信号到 DSP。

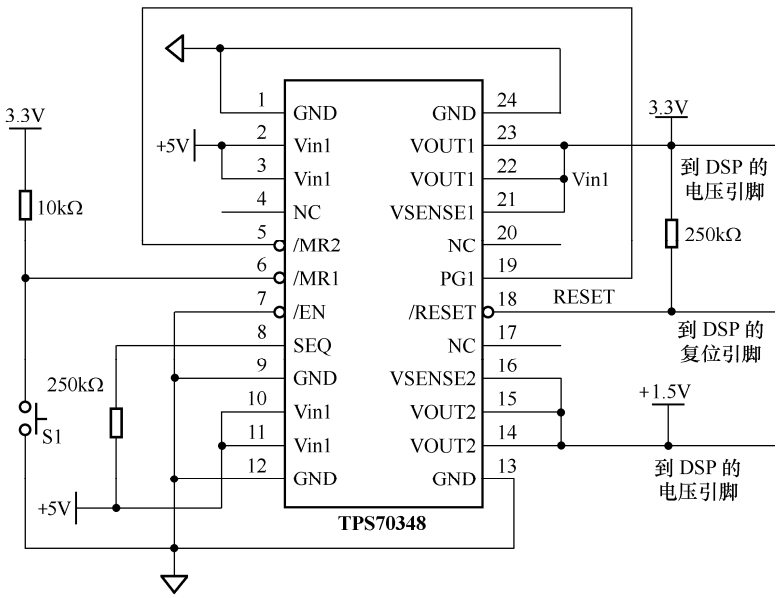

图 2.18　TPS70348 的电路设计

2.4.6　时钟电路设计

TMS320C6000 系列 DSP 的时钟引脚为 X1 和 X2/CLKIN。采用有源晶振，则直接将晶振的输出连接到 X2 引脚，接法如图 2.19 所示。

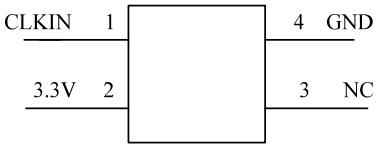

图 2.19　有源晶振的连接

DSP 有一组引脚 CLKMD0～CKLMD2，可以用来调整 DSP 工作频率的高低，并由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上乘以设定的倍数，倍数与 CLKMD0～CLKMD2 的关系如表 2.1 所示。每种型号 DSP 的倍数关系不同，即使同一种型号的 DSP，不同封装的 DSP 倍数关系也不同，表 2.1 则是以 GLS 封装的 TMS320C6203B 为例。

表 2.1　　CLKMD0～CLKMD2 与分频关系

CLKMD2	CLKMD1	CLKMD0	时 钟 模 式
0	0	0	PLL×1
0	0	1	PLL×4
0	1	0	PLL×8
0	1	1	PLL×10
1	0	0	PLL×6
1	0	1	PLL×9
1	1	0	PLL×7
1	1	1	PLL×11

一般 DSP 芯片的 PLL 电路都有 PLLV、PLLF 以及 PLLG 引脚。这些引脚是为了确保输入时钟的稳定性而特别设计的。一般 PLLF 和 PLLG 引脚连接到电容电阻网络上，如图 2.20 所示。图中 RC 的推荐值为 R1=60.4Ω、C1=27nF、C2=560pF，或者 R1=45.3Ω、C1=47nF、C2=10pF。

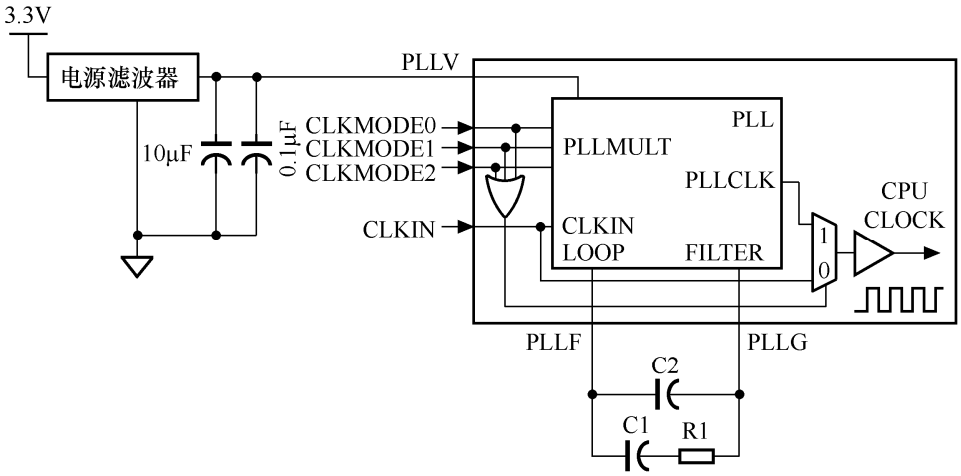

图 2.20　PLL 电路的连接

PLL 电路的 PLLV 引脚连接到电磁兼容滤波器的输出端。电磁兼容滤波器实际上是一个三端口的穿心电容，该电容可以选择 TDK 公司的 ACT4518 系列或者松下公司的 EXCCET103U 的电容，价格在 1 元以下。但这些器件不易购买，尤其是对小批量的用户。为此，也可以不使用电磁兼容滤波器，直接将 PLLV 引脚连接一批容值大小不同的电容上，然后接地，也可以达到稳定时钟信号的目的。

2.4.7 JTAG 仿真口电路设计

连接测试组（JTAG，Joint Test Action Group）接口用于连接最小系统板和仿真器，实现仿真器对 DSP 的访问，JTAG 接口的连接需要和仿真器上的接口一致。不论什么型号的仿真器，其 JTAG 接口都满足 IEEE 1149.1 的标准。满足 IEEE 1149.1 标准的 14 脚 JTAG 接口如图 2.21 所示。

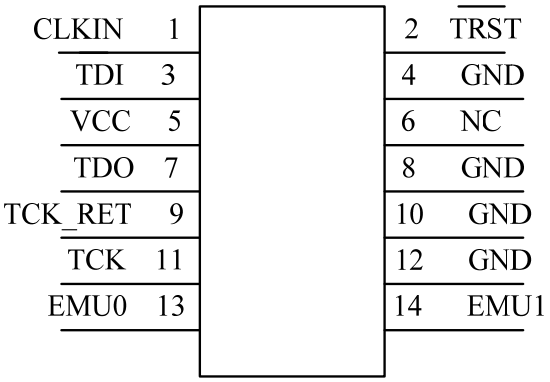

图 2.21　14 脚仿真口引脚

一般情况下，最小系统板需要引出双排的 14 脚插针和图 2.21 所示的一致，图中引脚间隔为 0.1 英寸，引脚宽度为 0.025 英寸，引脚长度为 0.235 英寸。在大多数情况下，如果开发板和仿真器之间的连接电缆不超过 6 英寸，可以采用图 2.22 接法。需要注意的是其中 DSP 的 EMU0 和 EMU1 引脚都需要上拉电阻，推荐阻值为 4.7kΩ或者 10kΩ。如果 DSP 和仿真器之间的连接电缆超过 6 英寸，必须采用图 2.23 接法，在数据传输引脚加上驱动。

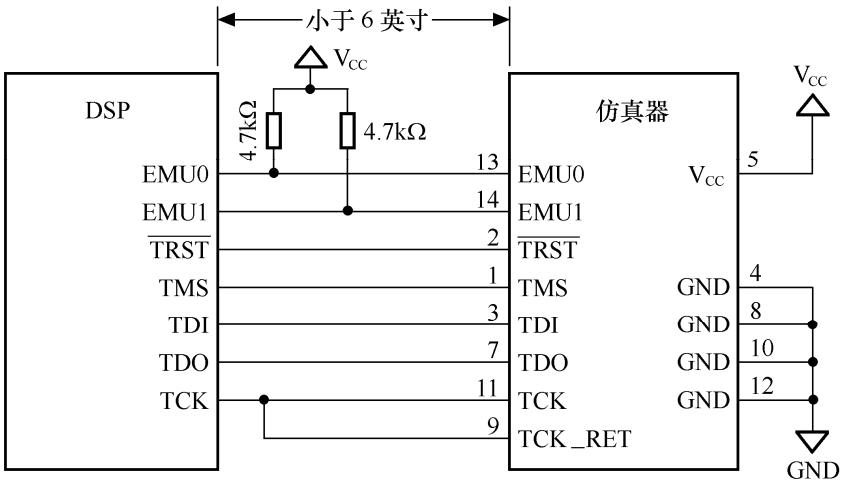

图 2.22　小于 6 英寸的 JTAG 连接方法

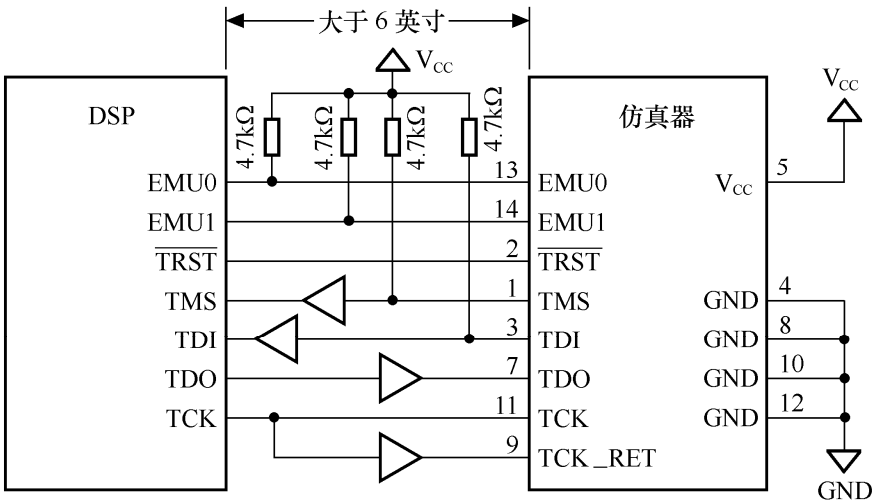

图 2.23　大于 6 英寸的 JTAG 连接方法

2.4.8 其他电路设计

其他电路设计如下。

（1）上拉电阻或者下拉引脚。DSP 芯片的有些引脚必须接 4.7kΩ 或者 10 kΩ 的上拉电阻，不同型号的芯片这些引脚有所不同，一般情况下这些引脚包括：未使用的中断信号、READY（数据准备好输入引脚）、$\overline{\text{HOLD}}$（保持输入引脚）、EMU0（仿真中断引脚 0）、EMU1（仿真中断引脚 1）、所有的 XBUS 总线的数据引脚、一些保留未使用的 RSV 引脚等。

（2）信号灯。系统板上可加入信号灯，用于指示最小系统的电源情况。当电源指示灯出现异常情况时可及时断电，以保护电路不被损坏。信号指示灯一般有：+5V 的电源指示灯（电路板供电正常）、电压转换输出 3.3V 指示灯（I/O 供电正常）、电压转换输出 1.8V 指示灯内（核供电正常）以及其他信号指示灯。

（3）测试孔。BGA 封装的 DSP 芯片在焊接之后，无法直接通过焊盘检测到每个引脚的状态，为此必须将一些可能需要测试的引脚通过连线引出。同时也可以将设计时不能确定的引脚也引出，这样确保在以后的改动中可以直接从这些测试孔跳线。

2.5 软件系统调试方法

2.5.1 ADS5422 的调试

ADS5422 的启动工作十分简单，只需要提供给 ADS5422 时钟信号，ADS5422 则根据这个时钟信号开始采样。但由于 ADS5422 是高速 AD，一般其输出的数据信号和模拟信号有 10 个时钟周期的延迟。ADS5422 在输出数字信号的同时，提供一个数据有效输出信号，该信号可以通知处理器来读取 ADS5422 的数字信号，但该信号的有效脉冲非常短，需要非常高速的处理器才能识别该信号。一般情况下，不建议使用该信号来通知处理器读取数据。ADS5422 的采样还存在一个特别需要注意的地方，就是该芯片启动的时间相当长，这也是高速（几十兆）AD 的缺点。如果将一个 60MHz 的时钟输出到 ADS5422，ADS5422 则马上开始采样，但此时的采样频率不是 60MHz，而是低于 60MHz，然后逐渐缓慢的上升到 60MHz，直至稳定。这也是 ADS5422 的时钟建立时间，一般需要 5μs 的时间。如果时钟为 60MHz，则前 500 个采样数据的采样频率为未知，不能使用。ADS5422 的采样情况如图 2.24 所示。

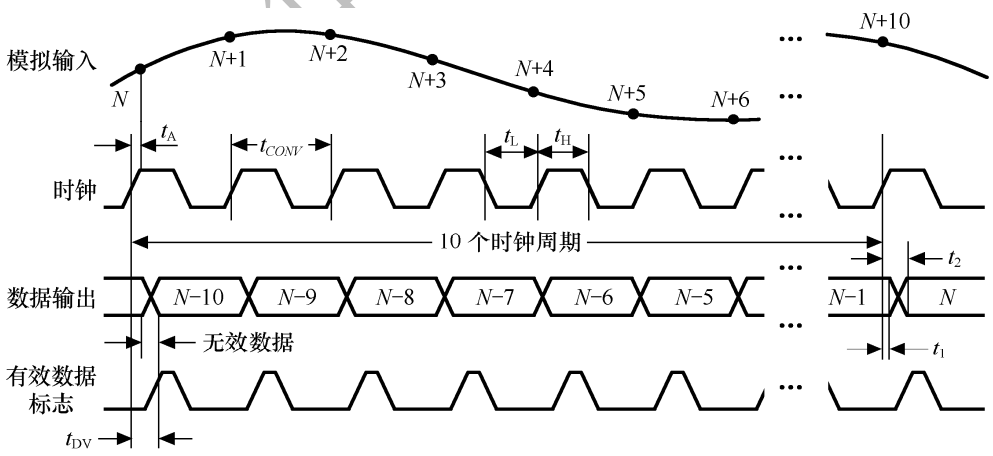

图 2.24 ADS5422 的采样

ADS5422 的数据输出可以输出使能引脚（$\overline{\text{OE}}$）屏蔽，这是因为高速 AD 启动采样的时间较长，一般不通过关闭采样时钟来禁止数据输出。而是一旦启动了 AD 采样，AD 将持续采样，如果需要禁止数据输出，则将 $\overline{\text{OE}}$ 引脚置高，这样，处理器将不再接收数据到来信号，不会去读取 AD 的数据。

此外，ADS5422 还提供一个数据溢出引脚（OVR），如果模拟信号输入超过 AD 的范围，AD 的数据输出将一直为正的或者负的最大值，在这种情况下，AD 的 OVR 引脚将持续输出高电平通知处理器数据溢出，如果模拟信号一段时间后恢复正常值，该引脚也将自动输出低电平。DSP 控制数据采集的软件程序流程如图 2.25 所示。

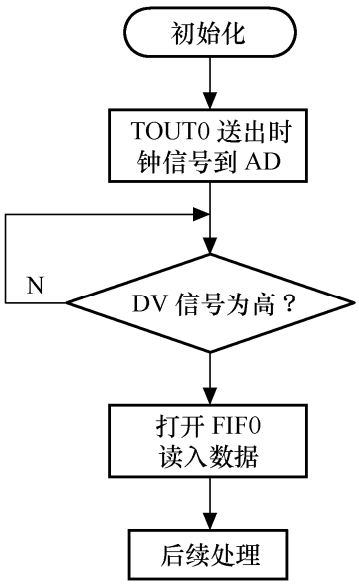

图 2.25　数据采集程序流程图

设置定时器参数提供 TOUT0 信号到 ADS5422，ADS5422 收到时钟信号后开始采样。由于高速 AD 特有的延迟特性，也就是在收到采样信号后，ADS5422 至少需要在 10 个信号周期后才可以提供采样数据，所以设置好 AD 的时钟后，让 ADS5422 一直工作于采样状态，通过控制 ADS5422 的$\overline{OE}$引脚控制数据的输出，当 DSP 检测到 DV 信号为高后，打开 DSP 的 DMA 控制器读入数据，读入一批数据后设置$\overline{OE}$引脚为高禁止数据输出，DSP 开始算法处理，并将处理后的结果输出或者保存，然后设置$\overline{OE}$引脚为低，ADS5422 数据输出，开始下一次数据处理。按照硬件连接方法，AD 采样部分详细的程序编写流程如图 2.26 所示。

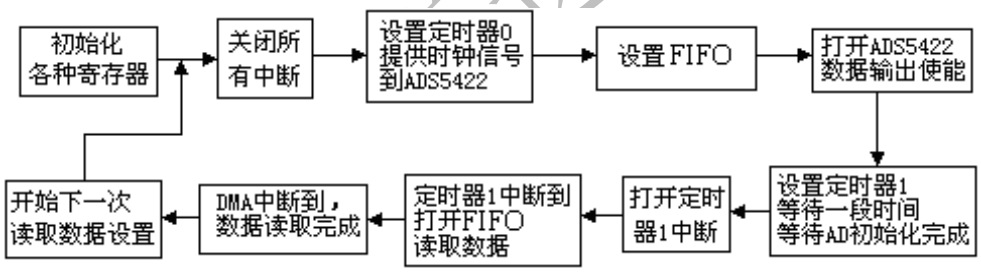

图 2.26　详细的程序编写流程

2.5.2　IDT72V2113 的调试

IDT72V2113 的输出和输入数据的宽度可以配置成 9bit 或者 18bit 的方式，在实际中大部分的外设都是 8bit 或者 16bit 的系统，一般 IDT72V2113 的一个（9bit 方式）或者两个（18bit 方式）数据引脚不需要用做数据引脚，该引脚可以用来控制外设的控制信号。IDT72V2113 的输出和输入数据的宽度由 IW 和 OW 引脚在主复位时的状态决定，如表 2.2 所示。

表 2.2　　IDT72V2113 的输入输出数据宽度配置

IW	OW	数据输入宽度	数据输出宽度
低电平（L）	低电平（L）	18bit	18bit
低电平（L）	高电平（H）	18bit	9bit
高电平（H）	低电平（L）	9bit	18bit
高电平（H）	高电平（H）	9bit	9bit

IDT72V2113 的将满或者将空标志是提示 FIFO 中的数据状态，如果数据将满，外设则需要尽快将数据读走，否则数据将出现满的情况，导致数据的丢失；同样道理，如果数据将空，外设则不要去读取其中的数据，否则数据被读空，无效数据被读走，导致错误数据产生。FIFO 何时提示外设其中的数据将满或者将空由 FSEL0、FSEL1 以及 $\overline{LD}$ 引脚决定。这些引脚配置一个空间大小，如果数据超过这个大小，将提示将满信息，如果数据小于这个大小，将提示将空信息，FSEL0、FSEL1 以及 $\overline{LD}$ 引脚状态和配置空间大小如表 2.3 所示。

表 2.3　　IDT72V2113 的空间配置

$\overline{LD}$	FSEL0	FSEL1	状态	空间大小
低电平	低电平	高电平	001	16383
低电平	高电平	低电平	010	8191
低电平	高电平	高电平	011	4095
高电平	低电平	高电平	101	2047
高电平	低电平	低电平	100	1023
高电平	高电平	低电平	110	511
高电平	高电平	高电平	111	255
低电平	低电平	低电平	000	127

2.5.3　CY7C68013 的调试

开发 USB 接口相当大的工作量是关于 USB 软件的开发，USB 软件包括 3 方面的工作：固件（Firmware）设计、驱动程序设计和主机端应用程序的设计。USB 设备启用之前，必须编写 3 个程序：一个是负责 USB 接口调用程序，用于 PC 机识别 USB 设备；一个是安装 USB 时的信息文件，用于介绍 USB 设备；一个是 USB 接口控制器使用的程序，用于传输数据。下面对这 3 个程序分别做一些简单的说明。

（1）负责 USB 接口调用程序。Cypress 公司提供了制作该程序的模板。首先，将 KEIL51 编译生成的 16 进制文件转换为可写入 USB RAM 中的二进制文件，然后将二进制文件拷入模板中。再调用设备驱动套件（DDK，Device Driver Kit），在 DDK 环境下对模板进行编译，生成驱动程序的系统文件。USB 通过通用驱动（GPD，General Purpose Driver）实现上层软件同 USB 的通信。

GPD 提供一种用户模式的界面，完成 USB 设备的请求和数据传输。Cypress 公司的开发工具包中提供了其开发面板的源程序，而其开发面板的设计就是基于 GPD 的，这能使开发者在例子程序的指引下，快速地编写出用于通信的应用软件。不过，GPD 的设计思想是服务于一般用户，其接口函数具有通用性。通过 GPD 提供的接口函数的原型，可以实现各种 USB 操作。包括实现负责 USB 设备的请求，即打开 USB 设备；负责 USB 的 I/O 口控制；通过改变 I/O 控制代码（IOCTL，I/O Control Code）实现各种操作。

当 USB 设备插入计算机时，计算机和 USB 设备之间能产生一个枚举过程。计算机能检测到有设备插入，自动发出查询请求；USB 设备回应这个请求，送出设备的 Verdor ID 和 Product ID；计算机根据这两个 ID 装载相应的设备驱动程序，完成枚举过程。

（2）安装 USB 时的信息文件。安装文件的任务就是将驱动程序文件绑定到特定的 VID/PID。主要是说明哪一个文件是负责 USB 接口调用程序，哪一个文件是 USB 接口控制器需要下载的文件。Cypress 公司提供了一个标准的 USB 安装信息文件，但用户还要将说明、版本号、日期、生产商等信息加到安装信息文件中才行。

（3）USB 接口控制器使用的程序。USB 接口控制器使用的程序是在 PC 机找到 USB 设备后将此程序下载到 USB 接口控制器内，以实现 USB 接口控制器对 FIFO 和 USB 接口的监控以及数据读写。这个文件因为涉及到下层硬件的连接，不可能提供通用的模板，需要用户自己编写。一般是通过对 USB 接口控制器的端口进行操作，来实现计算机和底层硬件设备之间的数据通信。

固件是运行在 CY7C68013 上的程序，可采用汇编语言或C 语言设计，其主要功能是控制 CY7C68013 接收并处理 USB 驱动程序的请求（如请求设备描述符、请求或设置设备状态，请求或设置设备接口等 USB2.0 标准请求）、控制 CY7C68013 接收应用程序的控制指令、通过 CY7C68013 存放数据并实时上传至 PC 等等。

本案例中的固件设计思路如下，其框架程序流程如图 2.27 所示。

（1）使 CY7C68013 工作于异步从 FIFO（AFIFO，Asynchronous Slave FIFO）模式。相应的寄存器操作为 IFCONFIG = 0xCB。CY7C68013 具有多种工作方式，除了可以作为能够产生任意控制波形的主控芯片外，还可以作为一些处理器的从设备。作为从设备，可选择异步或者同步工作方式。由于 DSP 的关系，本方案选择 CY7C68013 为异步从设备。

（2）将 4KB 的 FIFO 对应到两个端点（EndPoint），即 EndPoint2 和 EndPoint6。相应的寄存器操作为：EP2CFG = 0xA0，EP6CFG = 0xE2。EndPoint2 与 EndPoint6 分别对应 2kB 的内部 FIFO（下面分别称作 FIFO2，FIFO6），存放 USB 需要上传与接收的数据。其中 EndPoint2 为 OUT 型，负责从主机接收数据；EndPoint6 为 IN 型，负责向主机发送数据。另外 EndPoint2 与 EndPoint6 均采用批量（BULK）传输方式，这种方式相对于其他 USB2.0 定义的传输方式具有数据可靠、传输速率高等特点，是最常用的传输方式。

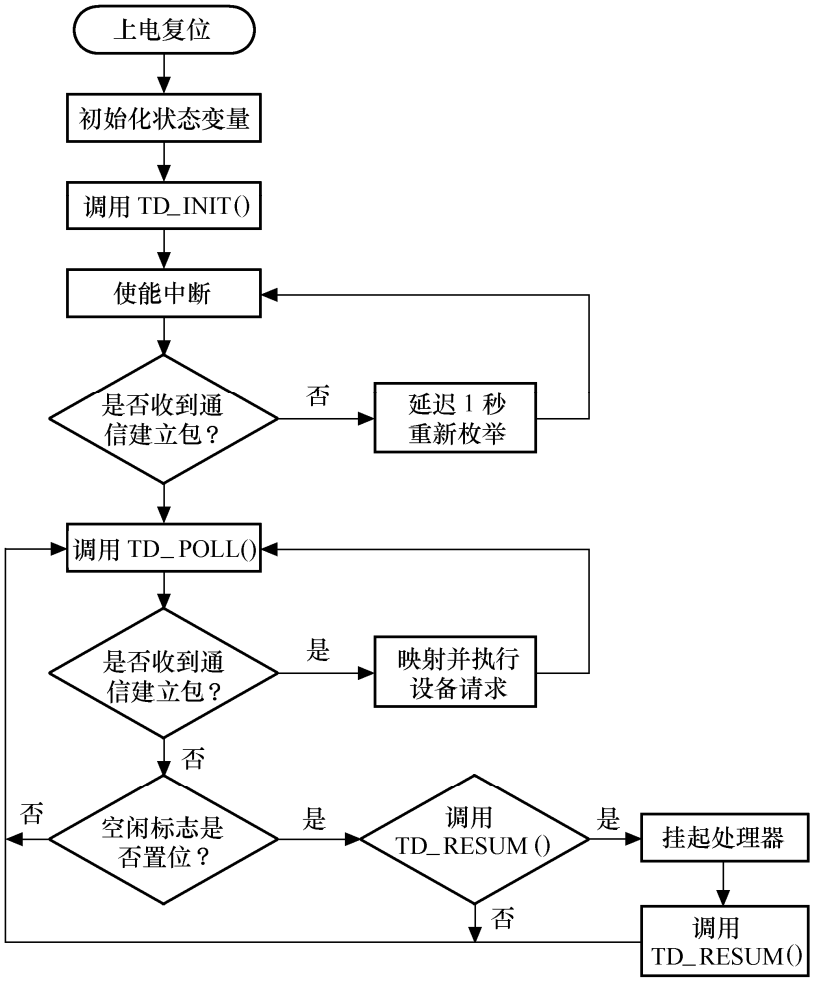

图 2.27　框架程序流程

（3）对 FIFO 进行配置。相应的寄存器操作为 EP2FIFOCFG = 0x11，EP6FIFOCFG = 0x0D。本案例将 FIFO2，FIFO6 设置成自动方式。这里所谓“自动”，是指在数据的传输过程中，不需要 CY7C68013 的 8051 内核参与。如有特殊需要可以设成手动方式，这样 8051 就可以对数据进行修改，如图 2.28 所示。

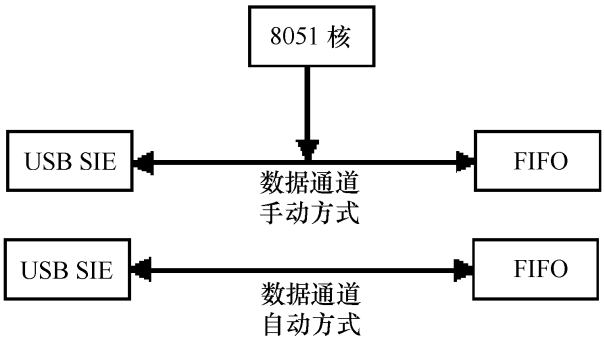

图 2.28　FIFO 的配置方式

（4）其他操作。为了完善整个 USB 传输功能，提高固件的健壮性，还必须配以其他设计，这包括 FIFO 的自行清空复位，个性化命令等功能。

USB 系统驱动程序采用分层结构模型，分别为较高级的 USB 设备驱动程序和较低级的 USB 函数层。其中 USB 函数层由两部分组成：较高级的通用串行总线驱动程序模块（USBD）和较低级的主控制器驱动程序模块（HCD）。在上述 USB 分层模块中，USB 函数层（USBD 及 HCD）由 Windows 提供，负责管理 USB 设备驱动程序和 USB 控制器之间的通信；加载及卸载 USB 驱动程序；与 USB 设备通用端点（EndPoint）建立通信并执行设备配置、数据与 USB 协议框架和打包格式的双向转换任务。目前 Windows 提供有多种 USB 设备驱动程序，但并不针对实时数据采集设备，因此 USB 设备驱动程序需由开发者自己编写。

开发 USB 设备驱动程序，可采用 Numega 公司的开发包 Driver Works 和 Microsoft 公司的 2000DDK，并以 Visual C++ 6.0 作为辅助开发环境。Driver Works 提供的驱动向导，可根据用户的需要，自动生成代码框架，减少了开发的难度，缩短了开发的周期。但是，Cypress 公司为了方便用户开发 USB 接口，在 CY7C68013 的开发包中提供了一个通用驱动程序，该程序可不加修改经 DDK 编译后直接使用。

主机应用程序主要实现从高速数据采集处理板读取处理后的数据、存储、显示处理结果以及向数据采集处理板发送控制命令。从逻辑上讲，USB 协议将 USB 信息通道按配置（Configuration）、接口（Interface）和端点（EndPoint）3 个层次进行了划分。通常一个 USB 设备可以由内部的软件代码设置一个或几个配置信息，但在任意时候都只有一个配置是激活的，每个配置下都包含一个或几个接口信息，每个接口信息实际上对应着 PC 机（主机）和 USB 设备之间一类完整意义上的通信任务。所以，要完成一个接口对应的通信任务，往往需要几个端口来协同工作。这里端口的概念是对称的，即是 USB 设备和 PC 机上都有一个相同逻辑意义上的端口，而信息就在这两个端口之间的逻辑管道上交互，如图 2.29 所示。

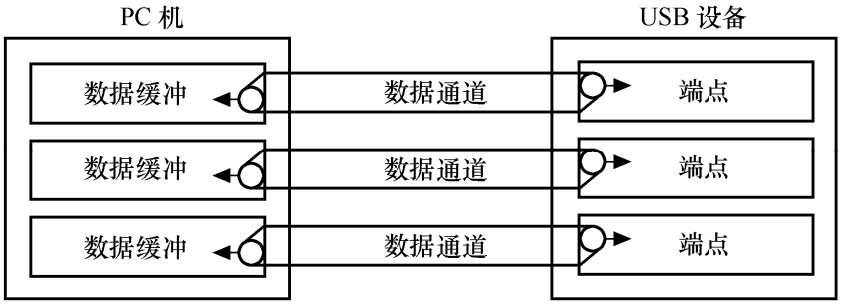

图 2.29　PC 机与 USB 的通信

使用每个管道（Pipe）进行数据通信之前，必须确定每个端口的特性，端口支持的传输类型为 4 种：控制传输（Control Transfer）、中断传输（Interrupt）、块传输（Bulk Transfer）和同时传输（Isochronous Transfer）。另外，还要确定端点（EndPoint）支持的传输方向（IN 和 OUT），IN 表示信息从 USB 设备传输到 PC 机，OUT 表示信息从 PC 机传输到 USB 设备，具体的设置已经在固件设计中给出了。PC 机使用的操作系统为 Windows2000，所使用的应用程序开发工具是 Visual C++ 6.0。

2.5.4　SST29LE010 的调试

闪速存储器的读操作与普通的存储器读操作基本上一致，具体的读周期时序如图 2.30 所示，先后有 7 个读周期。

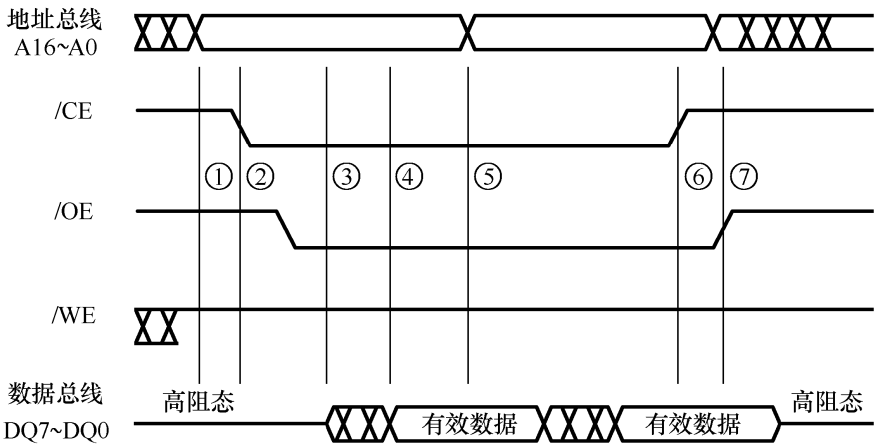

图 2.30　Flash 读周期时序

第 1 个周期，DSP 提供地址信号到 Flash，选通 Flash 的数据单元，这个周期与其他信号都无关。

第 2 个周期，将 Flash 的片选信号 $\overline{CE}$ 置为低电平，选通 Flash 芯片，此时 Flash 将地址信号锁存。

第 3 个周期，将 Flash 的读信号 $\overline{OE}$ 置为低电平，同时将写信号 $\overline{WE}$ 置为高电平。

以上 3 个周期，都是以 DSP 为主，所有的信号由其提供，Flash 被动接受信号。

第 4 个周期，Flash 数据总线输出数据。

第 5 个周期，DSP 从数据总线读取数据，并改变地址信号，Flash 输出下一个数据。

第 6 个周期，数据读周期结束，首先将 Flash 的片选信号置为高电平。

第 7 个周期，将 Flash 的读信号置为高电平，Flash 不输出任何数据。

需要注意的是，DSP 在每读取一个数据后，必须在 $\overline{CE}$ 或 $\overline{OE}$ 引脚上给出一个上升沿标志，通知 Flash 已经将数据读取，之后 Flash 会自动将下一个存储单元的数据送到数据线上。

Flash 的写操作相对复杂一些，它需要一串命令字序列，写入 Flash 的命令寄存器来完成相应的命令，并逐页进行编程，同时清除已有的程序。

SST 公司建议总是激活 SDP（Software Data Protection），在接收到三字节的写命令字时，Flash 得以确认下一步将进行写操作，有效防止产生误写；然后，载入所要写的页地址，在 $\overline{WE}$ 或 $\overline{CE}$ 中的上升沿由 T_{BLCO}（字节装载循环周期）定时器初始化写周期，写入相应数据，直到全部 128 个字节数据写完；之后 Flash 必须等待一定的时间 T_{WC} 才可以进行下一页的写入。写操作的具体时序图如图 2.31 所示。图中，T_{AH} 的时间不能低于 70ns，T_{WP} 的时间不能低于 120ns，T_{DS} 的时间不能低于 50ns，T_{BLC} 的时间必须在 0.05～100μs 之间，T_{BLCO} 的时间不能低于 200μs，T_{WC} 的时间不能高于 10ms。

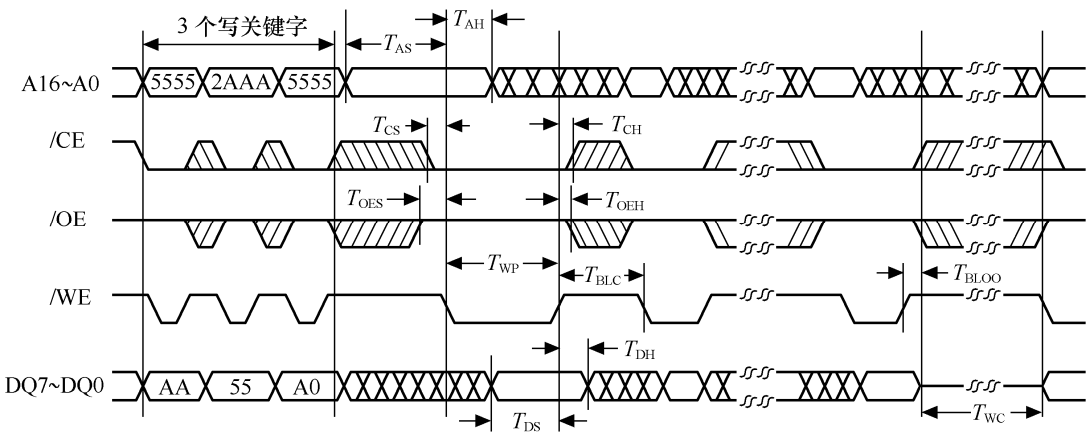

图 2.31　Flash 写周期时序

闪速存储器 SST29LE010 可进行软件擦除，它允许用户一次清除整个存储器内容，有利于快速擦除整个芯片。一般情况下进行写操作时，芯片能自动擦除已写入的数据。

此外，将数据写入闪速存储器 SST29LE010 可以在不激活 SDP 的情况下进行，但这样写入数据有时会导致写入失败，为保证写入的数据完全正确，一般需重复进行两次写入。

为优化系统写循环时间，闪速存储器 SST29LE010 提供了两种检测写循环完成的方式，即检测 Data Polling（DQ7）和 Toggle（DQ6）状态。一般情况下，非易失性写与系统是异步的，所以上述两位的读写可能会同时发生。如果发生这种情况，系统将会得到一个错误的结果，产生假象，例如合法的数据有可能与 DQ7 和 DQ6 冲突。为防止这种假象，当系统得到错误结果时，软件程序将重复读两次。如果两次读都是有效的，表明写周期完成，否则继续。

Flash 可以不擦除直接编程，但这样可能导致第一次编程出现错误数据写入，需要进行第二次编程。在编程之前擦除一次，就不会产生以上问题，但增加程序的复杂性。Flash 的擦除就是使其每个数据位都恢复为 1 状态，即全 FF 状态。对 Flash 的擦除操作需要 6 个总线周期，总线时序如图 2.32 所示。

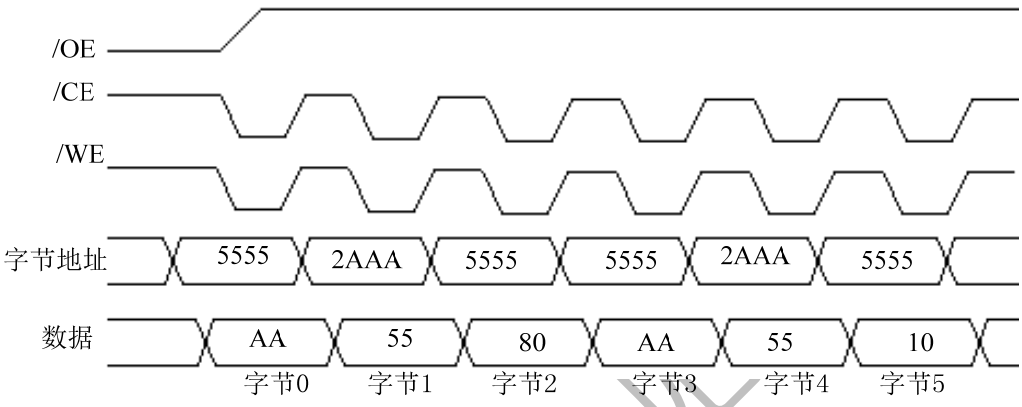

图 2.32　SST29LE010 的擦除时序

从图 2.32 可以看出，各总线周期的操作如下。

- 第 1 总线周期——向 Flash 的 2AAAH 地址的存储单元写入数据 55H。
- 第 2 总线周期——向 Flash 的 2AAAH 地址的存储单元写入数据 55H。
- 第 3 总线周期——向 Flash 的 5555H 地址的存储单元写入数据 80H。
- 第 4 总线周期——向 Flash 的 5555H 地址的存储单元写入数据 AAH。
- 第 5 总线周期——向 Flash 的 2AAAH 地址的存储单元写入数据 55H。
- 第 6 总线周期——向 Flash 的 5555H 地址的存储单元写入数据 10H。

完成上述操作后，Flash 存储器被完全擦除，内部数据恢复为初始状态，全为 FFH。

在对 Flash 进行擦除时，应对 DSP 及 EMIF 外存储器接口进行初始化，因为 SST29LE010 为 8bit 的 Flash，所以 CE1 空间定义为 8bit 读写模式。对 Flash 存储器进行字节编程之前，需要对它进行 3 个周期的编程指令操作，总线时序如下。

- 第 1 总线周期——向 Flash 的 5555H 地址的存储单元写入数据 AAH。
- 第 2 总线周期——向 Flash 的 2AAAH 地址的存储单元写入数据 55H。
- 第 3 总线周期——向 Flash 的 5555H 地址的存储单元写入数据 A0H。

在编程之后，应对 Flash 编程的正确性进行自动检查，把编程前数据的校验和编程后 Flash 中读出数据的校验和进行比较：如果相同，则编程成功；如果不相同，则编程失败。

2.5.5　电源时钟的调试

电源时钟的调试包括以下几个方面。

（1）系统上电后，检测 3.3V 和 1.5V 电压是否正常。如果正常，电源部分可以正常工作。

（2）系统上电后，直接测量 CLKOUT0 和 CLKOUT1 引脚，查看是否有时钟信号输出，以及时钟信号的频率是否和设置的一样。若 CLKOUT 信号正确，表明时钟和上电复位电路可以正常工作。

（3）连接好仿真器，查看是否能打开仿真软件 CCS。如果可以打开 CCS，表明 JTAG 仿真口和芯片启动设置可以正常工作。

（4）通过 DSP 下载程序到 DSP 中运行，查看运行结果。如果可以下载程序并运行，表明该 DSP 的外围电路正常工作，可以开始调试 AD、Flash、USB 以及 FIFO 电路。

2.6 程序代码

系统上电后，ADS5422 一直工作，采样产生的数据是否存储到 IDT72V2113 中，由 TMS320C6203B 的 DX0 引脚状态来决定。TMS320C6203B 进行初始化，外部扩展总线的 XCE3 设置为同步 FIFO 读操作模式。DMA 通道 0 配置为每次传输 1 帧，每帧 1024 个半字（HW，Half Word），同步事件设置为外部中断 4，触发极性为高电平，初始化定时器 0，定时间隔为 22 ms。

当外部同步信号到来时，启动定时器 0，手动启动 DMA 通道 0，同时设置 DX0 为低电平。ADS5422 采样产生的数据开始写入 IDT72V2113，当定时器 0 中断到来时，设置 DX0 为高电平，关闭 IDT72V2113 的写使能，采样数据不再存储到 IDT72V2113 内。随着数据不断写入 IDT72V2113，当其内部的数据量大于 1023 HW 时，IDT72V2113 的将空标志信号（PAE）由低电平变为高电平，使得 TMS320C6203B 的外部中断信号有效，从而触发 DMA 传输。TMS320C6203B 的 DMA 通道 0 通过外部扩展总线（XB）读取 1024 HW 的数据，存储于内部 RAM 中，传输结束后向 TMS320C6203B 发送中断，通知 TMS320C6203B 处理数据。TMS320C6203B 处理完数据后，通过 USB2.0 接口发送处理结果，然后重新启动 DMA 通道 0，进行下一次 DMA 传输。如此循环，直到处理完所有数据。当下一个外部同步信号到来时，进行下一轮数据采样处理过程。

2.6.1 主程序代码

主程序设置 DSP、AD 和 FIFO。

```
        .ref    start
        .def    DMAINT0     ;  DMA 中断 0
        .def    TIMERINT1   ;  定时器 1 中断

SPCR0     .equ   018c0008h
PCR0      .equ   018c0024h
SPCR1     .equ   01900008h
PCR1      .equ   01900024h
SPCR2     .equ   01A40008h
PCR2      .equ   01A40024h     ;  定义缓冲串口寄存器

XBGC      .equ   01880000h
XCECTL0   .equ   01880008h
XCECTL1   .equ   01880004h
XCECTL2   .equ   01880010h
XCECTL3   .equ   01880014h     ;  定义扩展总线寄存器

GBLCNTA     .equ   01840028h
GBLCNTB     .equ   0184002Ch
GBLIDXA     .equ   01840030h
GBLIDXB     .equ   01840034h
GBLADDRA    .equ   01840038h
```

```
GBLADDRB    .equ    0184003Ch
GBLADDRC    .equ    01840068h
GBLADDRD    .equ    0184006Ch    ;  定义 DMA 全局寄存器

PRICLT0     .equ    01840000h
SECCTL0     .equ    01840008h
SRC0        .equ    01840010h
DST0        .equ    01840018h
XFRCNT0     .equ    01840020h    ;  定义 DMA 通道 0 寄存器

CTL0        .equ    01940000h
PRD0        .equ    01940004h
CNT0        .equ    01940008h
CTL1        .equ    01980000h
PRD1        .equ    01980004h
CNT1        .equ    01980008h    ;  定义定时器寄存器

            .data
            .global    a1

            .text
start:
interrupt_init:                                  ;  初始化中断
        SUB         A0,A0, A0
        MVC         CSR,B0
        MVC         A0,ISTP
        AND         0x0FFFFFFE,B0,B0
        MVKL        08102H,A0                    ;  使能定时器 1、DMA0 和 NMI 中断
        MVKLH       0000h,A0
        MVC         B0,CSR                       ;  禁止所有的可屏蔽中断
        MVC         A0,IER
        MVKL        064H,A15                 ;  设置寄存器 a15，用于累计 DMA0 中断的次数
        MVKLH       000H,A15

start_timer0:
        MVKL        0, B0                        ;  初始化定时器 0
        MVKLH       0194h, B0
        MVKL        1, A1
        MVKLH       0h, A1
        MVKL        0301h ,A0
        MVKLH       0000h ,A0
        STW         A0, *B0++[1]
        NOP         2
        STW         A1, *B0++[1]
        NOP         2
        STW         A1, *B0--[2]
        NOP         2
```

```
        MVKL    03C1h, A0
        MVKLH   0, A0
        STW     A0, *B0
        NOP     3

mcbsps0_init:                       ;  初始化缓冲串口 0
        MVKL    SPCR0,B0            ;  将缓冲串口引脚设置成通用 I/O 引脚
        MVKH    SPCR0,B0
        LDW     *B0,A0
        NOP     3
        MVKL    0FFFEh,A1
        MVKLH   0FFFEh,A1
        AND     A1,A0,A0
        STW     A0,*B0
        NOP     3
        MVKL    PCR0, B0
        MVKH    PCR0,B0
        LDW     *B0,A0
        NOP     3
        MVKL    00003020h,A1
        MVKH    00003020h,A1
        OR      A1,A0,A0
        STW     A0,*B0
        NOP     3

mcbsps1_init:                       ;  初始化缓冲串口 1
        MVKL    SPCR1,B0
        MVKH    SPCR1,B0
        LDW     *B0,A0
        NOP     3
        MVKL    0FFFEh,A1
        MVKLH   0FFFEh,A1
        AND     A1,A0,A0
        STW     A0,*B0
        MVKL    PCR1, B0
        MVKH    PCR1,B0
        LDW     *B0,A0
        NOP     3
        MVKL    00006020h,A1            ;  设置 DX1 引脚状态为高
        MVKH    00006020h,A1
        OR      A1,A0,A0
        STW     A0,*B0
        NOP     2
        NOP
        NOP

        MVKL    5A00H, A1
```

```
        MVKLH   065h, A1
WAIT_FOR_FIFO0:                         ;  等待 FIFO0 初始化结束
        NOP
        SUB     A1,1,A1
        NOP
   [A1] B       WAIT_FOR_FIFO0
        NOP     5

        MVKL    PCR1,B0
        MVKH    PCR1,B0
        LDW     *B0,A0
        NOP     3
        MVKL    03000h,A1               ;  设置 DX1 引脚状态为低
        MVKH    00h,A1
        STW     A1,*B0
        NOP     5
        STW     A1,*B0
        NOP     5

        MVKL    5A00H, A1               ;  设置 FIFO 部分复位
        MVKLH   065h, A1
WAIT_FOR_FIFO1:                         ;  等待 FIFO1 初始化结束
        NOP     3
        SUB     A1,1,A1
        NOP     3
   [A1] B       WAIT_FOR_FIFO1
        NOP     6

        MVKL    PCR1, B0                ;  清空 FIFO 内容
        MVKH    PCR1,B0
        LDW     *B0,A0
        NOP     3
        MVKL    00006020h,A1            ;  设置 DX1 引脚状态为高
        MVKH    00006020h,A1
        OR      A1,A0,A0
        STW     A0,*B0
        NOP     2
        NOP
        NOP

        MVKL    5A00H, A1
        MVKLH   065h, A1
WAIT_FOR_FIFO2:                         ;  等待 FIFO2 初始化结束
        NOP     3
        SUB     A1,1,A1
        NOP     3
   [A1] B       WAIT_FOR_FIFO2
```

```
        NOP     6

xbus_init:                              ;  初始化扩展总线
        MVKL    XBGC, B0
        MVKH    XBGC, B0
        LDW     *B0,A0
        MVKL    04000h,A1
        MVKH    00000h,A1
        OR      A1,A0,A0
        STW     A0,*B0
        NOP     3

        MVKL    XBGC, B0
        MVKH    XBGC, B0
        LDW     *B0,A0
        MVKL    04000h,A1
        MVKH    00000h,A1
        OR      A1,A0,A0
        STW     A0,*B0
        NOP     3

        MVKL    XCECTL3,B0              ;  初始化 XCE3 空间控制寄存器
        MVKH    XCECTL3,B0              ;  选择接口类型为 32bitFIFO
        LDW     *B0,A0                  ;  设置读写时序的参数
        MVKL    0201h,A1
        MVKLH   0201h,A1
        STW     A1,*B0
        NOP     3

dma_0_init:                             ;  初始化 DMA 通道 0
        MVKL    PRICLT0,B0              ;  工作方式为帧同步，32bit
        MVKH    PRICLT0,B0              ;  源地址不变，目的地址自增
        MVKL    0040h,A1                ;  同步事件为外部中断 4
        MVKLH   0E01h,A1                ;  DMA 通道 0 停止
        STW     A1,*B0
        NOP     3                       ;  初始化 DMA 通道 0 第二控制寄存器
                                        ;  电平触发方式，高有效
        MVKL    SECCTL0,B0
        MVKH    SECCTL0,B0
        LDW     *B0,A0
        NOP     3
        MVKL    0A080h,A1
        MVKLH   0008h,A1
        STW     A1,*B0
        NOP     3

        MVKL    SRC0,B0                 ;  初始化源地址寄存器
```

```
        MVKH    SRC0,B0
        MVKL    0000h,A0
        MVKLH   7000h,A0
        STW     A0,*B0
        NOP     3                       ;  设定 SRC0 指向 XCE3 空间

        MVKL    DST0,B0                 ;  初始化目的地址寄存器
        MVKH    DST0,B0
        MVKL    1000h,A0
        MVKLH   8000h,A0
        STW     A0,*B0
        NOP     3                       ;  设定 DST0 指向内部地址 80001000h

        MVKL    XFRCNT0 ,B0             ;  设定每次中断时 DMA 读取的数据个数
        MVKH    XFRCNT0 ,B0             ;  初始化传输计数寄存器
        MVKL    02800h,A1               ;  设置成 1 帧，每帧 1024 个数据
        MVKLH   0000h,A1
        STW     A1,*B0
        NOP     3
        MVKL    GBLCNTA ,B0
        MVKH    GBLCNTA ,B0
        MVKL    02800h,A1
        MVKLH   0000h,A1
        NOP     3
        STW     A1,*B0
        NOP     3

        MVKL    0FFFCh,A1
        MVKLH   0h,A1
        AND     A0,A1,A0
        NOP     3

DX0_CLEAR:                              ;  打开 FIFO0 和 FIFO1
        MVKL    PCR0,B0
        MVKH    PCR0,B0
        LDW     *B0,A0
        NOP     3
        MVKL    0FFFFFFDFh,A1
        MVKH    0FFFFFFDFh,A1
        AND     A1,A0,A0
        STW     A0,*B0
        NOP     5

        STW     A0,*B0
        NOP     5

        MVC     CSR, B0                 ;  开中断
```

```
        OR      01H,B0,B0
        MVC     B0,CSR
        NOP     3

timer1init:
        MVKL    064H,A15            ;  设定 a15 为 DMA0 中断次数为 100 次
        MVKLH   000H,A15
        MVKL    0,  B0              ;  设置定时器 1 寄存器
        MVKLH   0198H,B0
        MVKL    0300h,A0
        MVKLH   00h,  A0
        STW     A0,     *B0
        NOP     4

        MVKL    04H,  B0
        MVKLH   0198H,  B0
        ;MVKL   08480h,A0
        ;MVKLH  01Eh,  A0
        MVKL    06EE0h,A0
        MVKLH   01Fh,  A0
        STW     A0,     *B0
        NOP     4

        SUB     A0,A0,A0
        NOP     4
        MVKL    08H,  B0
        MVKLH   0198H, B0
        STW     A0, *B0
        NOP     4

        MVKL    0,  B0
        MVKLH   0198H,B0
        MVKL    03E0h,A0
        MVKLH   00h, A0
        STW     A0,*B0
        NOP     4

waittimer1:                         ;  等待 DMA0 中断
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        B       waittimer1
        NOP     5
```

```
        NOP
        NOP

DMAINT0:                                ;  DMA0 中断服务程序
        NOP     3
        MVKL    000H,A8
        MVKLH   000H,A8
        MVKL    1000H,A3
        MVKLH   08000H,A3
        MVKL    02800H,B1               ;  累加数据的格式

LOOP:
        MVKL    000H,A8
        MVKLH   000H,A8
        NOP     4
        STW     A8,*A3

        MVKL    003FFFH,B5
        MVKLH   0,B5
        NOP     3
        LDW     *A3,A4
        NOP     4
        AND     B5,A4,A5
        SHL     A4,18,B4
        SHR     B4,18,A4
        STW     A4,*A3++
        NOP     3
  [B1]  SUB     B1,1,B1
        NOP
        NOP
  [B1]  B       LOOP
        NOP     5
        NOP
        NOP
                                        ;  累计 DMA 中断的次数
        SUB     A15,1,A15
        SHL     A15,0,B1
  [!B1] B       ENDDMA
        NOP     5
        NOP
        NOP

        MVKL    DST0,B0
        MVKH    DST0,B0
        MVKL    1000h,A0
        MVKLH   8000h,A0
        STW     A0,*B0
```

```
        NOP     3

        MVKL    SECCTL0,B0
        MVKH    SECCTL0,B0
        LDW     *B0,A0
        NOP     3
        MVKL    0A080h,A1
        MVKLH   0008h,A1
        STW     A1,*B0
        NOP     3

        MVKL    XFRCNT0 ,B0         ;  设定每次 DMA 中断读取数据的个数
        MVKH    XFRCNT0 ,B0
        MVKL    02800h,A1
        MVKLH   0000h,A1
        STW     A1,*B0
        NOP     3

        MVKL    PRICLT0,B0          ;  开启 DMA 通道 0
        MVKH    PRICLT0,B0
        LDW     *B0,A0
        MVKL    0001h,A1
        MVKLH   0000h,A1
        NOP     3
        OR      A0,A1,A0
        NOP     3

        B       IRP
        STW     A0,*B0
        NOP     3
        NOP
        NOP

ENDDMA:
        MVKL    dma_0_init,A1
        MVKH    dma_0_init,A1
        MVC     A1,IRP
        NOP     3
        B       IRP
        NOP     5
        NOP
        NOP

TIMERINT1:                          ;  定时器 1 中断，表明 FIFO 已经满
        MVKL    PCR0, B0            ;  禁止 FIFO 存储数据
        MVKH    PCR0,B0
        LDW     *B0,A0
```

```
        NOP     3
        MVKL    00003020h,A1
        MVKH    00003020h,A1
        OR      A1,A0,A0
        STW     A0,*B0
        NOP     3
        STW     A0,*B0
        NOP     3

        MVKL    0,  B0              ;  禁止定时器 1
        MVKLH   0198H,B0
        MVKL    0300h,A0
        MVKLH   0,A0
        STW     A0,*B0
        NOP     3
        NOP

        MVKL    open_dma0,A1
        MVKH    open_dma0,A1
       ;MVKL    mcbsps0_init,A1
       ;MVKH    mcbsps0_init,A1
        MVC     A1,IRP
        NOP     3
        B       IRP
        NOP     5
        NOP
        NOP

open_dma0:
        MVKL    PRICLT0,B0
        MVKH    PRICLT0,B0
        MVKL    0040h,A1
        MVKLH   0E01h,A1
        STW     A1,*B0
        NOP     3

        MVKL    SECCTL0,B0
        MVKH    SECCTL0,B0
        LDW     *B0,A0
        NOP     3
        MVKL    0A080h,A1
        MVKLH   0008h,A1
        STW     A1,*B0
        NOP     3

        MVKL    SRC0,B0
        MVKH    SRC0,B0
```

```
        MVKL    0000h,A0
        MVKLH   7000h,A0
        STW     A0,*B0
        NOP     3

        MVKL    DST0,B0
        MVKH    DST0,B0
        MVKL    1000h,A0
        MVKLH   8000h,A0
        STW     A0,*B0
        NOP     3               ;  设定 DST0 指向内部数据空间地址 80001000h

        MVKL    XFRCNT0 ,B0
        MVKH    XFRCNT0 ,B0
        MVKL    02800h,A1
        MVKLH   0000h,A1
        STW     A1,*B0
        NOP     3
        MVKL    GBLCNTA ,B0
        MVKH    GBLCNTA ,B0
        MVKL    02800h,A1
        MVKLH   0000h,A1
        NOP     3
        STW     A1,*B0
        NOP     3

        MVKL    PRICLT0,B0
        MVKH    PRICLT0,B0
        LDW     *B0,A0
        MVKL    0001h,A1
        MVKLH   0000h,A1
        NOP     3
        OR      A0,A1,A0
        STW     A0,*B0
        NOP     5
waiting:
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        B       waiting
```

```
        NOP             5

        .end
```

2.6.2 主程序中断向量代码

```
    .sect "vectors"
    .ref    start
    .ref    DMAINT0
    .ref    TIMERINT1

RESET_RST:
      B     IRP
      NOP
      NOP
      NOP
     NOP
     NOP
     NOP
     NOP
NMI_RST:
      B     IRP
      NOP
     NOP
     NOP
     NOP
     NOP
     NOP
     NOP
    NOP
RESV1:
      B     IRP
      NOP
      NOP
      NOP
     NOP
     NOP
     NOP
     NOP
RESV2:
      B     IRP
      NOP
      NOP
      NOP
     NOP
     NOP
     NOP
     NOP
```

```
INT4:
        B     IRP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
INT5:
        B     IRP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
INT6:
        B     IRP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
INT7:
        B     IRP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
INT8:
        B     DMAINT0
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
INT9:
        B     IRP
```

```
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
INT10:
        B     IRP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
INT11:
        B     IRP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
INT12:
        B     IRP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
INT13:
        B     IRP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
TINT0:
        B     IRP
        NOP
        NOP
```

```
        NOP
        NOP
        NOP
        NOP
        NOP
TINT1:
        B     TIMERINT1
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
```

2.6.3 主程序配置文件代码

```
-e start

MEMORY
{     SPRAM :    origin = 0x00008000, len = 0x01000
      SDRAM :    origin = 0x80000000, len = 0x10000
      INTER :    origin = 0x00000000, len = 0x01000
}

SECTIONS
{
      vectors    > INTER
      .text       > SPRAM
      .bss         > SDRAM
      .far         > SDRAM
      .stack      > SDRAM
      .system    > SDRAM
      .data       > SDRAM
}
```

2.6.4 写 Flash 程序代码

```
#include <stdio.h>
#include <c6x.h>
#include "6711.h"

int Len = 0x0a000;                                    // 预编写的程序长度
int main()
{
```

```
char *flash_ptr =(char *) 0x090000000;
char src_ptr[128] = {0};
FILE *hex_fp = NULL;
int    flag = 0,j,data,l,i,length,temp,temp1;
char hex_filename[80]="post.hex";

CSR=0x100;
IER=1;
ICR=0x0ffff;

*(unsigned volatile int *)EMIF_CE1 = 0x0ffffff03;

if ( (hex_fp=fopen( hex_filename , "r" ))== NULL )
{
    printf( "\nERROR: File %s does not exist!\n", "post.HEX" );
    exit(0);
}

if ( (fgetc(hex_fp)) == 0x02)
{
}
else
{
    exit(0);
}

printf("Programming the flash.\n");

*(char *) flash_addr1 = 0x0aa;
*(char *) flash_addr2 = 0x55;
*(char *) flash_addr1 = 0x80;
*(char *) flash_addr1 = 0x0aa;
*(char *) flash_addr2 = 0x55;
*(char *) flash_addr1 = 0x10;

temp = *flash_ptr;
temp &= 0x40;
temp1 = *flash_ptr;
temp1 &= 0x40;
while(temp != temp1)
{
    temp = *flash_ptr;
    temp &= 0x40;
    temp1 = *flash_ptr;
    temp1 &= 0x40;
}
length = Len/128;
```

```
for(l=0;l<length;l++)
{
        for (i =0;i<128;i++)
        {
                j = fscanf(hex_fp,"%x", &data);
                src_ptr[i] = data;
                if (j == EOF || j == 0)
                {
                        flag = 1;
                }
              }
        *(char *) flash_addr1 = 0x0aa;
        *(char *) flash_addr2 = 0x55;
        *(char *) flash_addr1 = 0x0a0;
       for (i=0;i<128;i++)
     *flash_ptr++ = src_ptr[i];
        temp = *--flash_ptr;
            temp &= 0x80;
            while(!temp)
            {
                temp = *flash_ptr;
                temp &= 0x80;
             }
        flash_ptr++;
}

printf("\nProgramming the flash is completed.");
}
```

2.6.5 写 Flash 配置文件代码

```
MEMORY
            {
                IPRAM           : origin = 0x0,             len = 0x0FF00
            }

SECTIONS
      {
          .vectors      > IPRAM
          .text         > IPRAM
          .bss      > IPRAM
          .cinit        > IPRAM
          .const        > IPRAM
          .far          > IPRAM
          .stack        > IPRAM
          .cio          > IPRAM
```

```
        .sysmem > IPRAM
    }
```

2.6.6 写 Flash 中断向量代码

```
    .ref    _c_int00
     .sect "vectors"
RESET_RST:
    mvkl  .S2 _c_int00, B0
    mvkh      .S2 _c_int00, B0
    B         .S2 B0
              NOP
              NOP
              NOP
              NOP
          NOP
NMI_RST:  NOP
              NOP
              NOP
              NOP
              NOP
              NOP
              NOP
              NOP
RESV1:        NOP
              NOP
              NOP
              NOP
              NOP
              NOP
              NOP
              NOP
              NOP
RESV2:        NOP
              NOP
              NOP
              NOP
              NOP
              NOP
              NOP
              NOP
              NOP
INT4:         NOP
              NOP
              NOP
              NOP
              NOP
              NOP
              NOP
              NOP
```

```
                NOP
INT5: ……   （此后所有中断后都是 8 条 NOP 指令）
INT6: ……
INT7: ……
INT8: ……
INT9: ……
INT10: ……
```

2.6.7 写 Flash 工具代码

该代码实现将 OUT 文件生成 HEX 文件的配置文件。该文件将 post.out 文件生成 post.hex，在使用文件之前用户的 OUT 文件名和该配置文件中的文件名保持一致。

```
post.out
-a
-image
-zero
-memwidth  8
ROMS
{
  Flash: org = 0, len = 0x3000,romwidth = 8, files = {post.hex}
}
```

2.6.8 DSP 读 USB 接口代码

使用 DSP 的 QDMA 方式读取 USB 接口数据，QDMA 是快速 DMA 方式，是 DSP 对外部数据读写的最快方式。使用该程序可以测试 USB 接口的数据速率。

```
                .def start
ce0ctl          .equ   0x01800004
qdma_s_opt      .equ   0x02000020
qdma_src        .equ   0x02000004
qdma_cnt        .equ   0x02000008
qdma_dst        .equ   0x0200000c
qdma_idx        .equ   0x02000010
                .text

start:
                mvc csr,b0
                and -2,b0,b0
                mvc b0,csr                ;  禁止可屏蔽中断

                mvkl   ce0ctl,a0
                mvkh    ce0ctl,a0
                mvkl   10510222h,a1
                mvkh    10510222h,a1
                stw     a1,*a0
                nop     4
```

```
                mvkl    qdma_src,a7
                mvkh    qdma_src,a7
                mvkl    80004000h,b7
                mvkh    80004000h,b7
                stw     b7,*a7
                nop     4

                mvkl    qdma_dst,a8
                mvkh    qdma_dst,a8
                mvkl    00008000h,b8
                mvkh    00008000h,b8
                stw     b8,*a8
                nop     4

                mvkl    qdma_cnt,a9
                mvkh    qdma_cnt,a9
                mvkl    00010080h,b9
                mvkh    00010080h,b9
                stw     b9,*a9
                nop     4

                mvkl    qdma_idx,a10
                mvkh    qdma_idx,a10
                mvkl    00000000h,b10
                mvkh    00000000h,b10
                stw     b10,*a10
                nop     4

                mvkl    qdma_s_opt,a11
                mvkh    qdma_s_opt,a11
                mvkl    41200001h,b11
                 mvkh    41200001h,b11

loop:
                nop
                nop
                b       loop
                nop     5
                .end
```

2.7 案例总结

本案例详细介绍了高速数据采集的系统设计。该系统以 C6203 为核心处理器，实现高频率信号的频谱分析，将频谱结果实时传输到计算机。由于 DSP 处理速度有限，无法对每个采样数据进行处理，而是采取处理一批数据，丢弃一批数据的方法。如果需要对每一个采样数据进行处理，例如一些无线通信的相关运算，最好采用 FPGA 并行处理数据，这样可以成倍加快算法运行速度，而将一些对运算要求较高的算法在 DSP 内部完成。所以，现在一些信号处理算法均采取“大规模 FPGA＋高性能 DSP 的硬件平台”框架。

高速信号处理的硬件版图布局和走线也非常重要。

（1）布线时不能只考虑线能否布通，如果 PCB 布线布局不合理，可能会大大影响性能和通信距离，这是高频电路设计的特点决定的。因此将 PCB 分为高频电路和控制电路两部分布置。PCB 使用多层板。中间层有一个单独的接地面和单独的电源面，元件面的接地面保证元件充分接地。

（2）合适的零件布局。高频电路的元件面以高频元件为中心，各元件紧靠其周围，尽可能减少分布参数的影响。需要说明的是电感的布局是非常重要的，一个经过优化的电感布局将可以给 PLL 环路滤波器提供一个合适的电压。匹配网络的元器件最好靠近高频元件，以减小杂散电感和杂散电容。

（3）将 PCB 分区为独立的模拟部分和数字部分。在电路板的所有层中，数字信号只能在电路板的数字部分布线，模拟信号只能在电路板的模拟部分布线，并且模拟电源和数字电源要分割。高频元件的直流供电必须在离电源脚尽可能近的地方用高性能的高频电容去耦。如果一个小电容再并上一个较大的电容效果会更好。高频部分的电源和数字电路部分的电源分离，高频元件的地端直接连接到接地面。

（4）高频电路的电源使用高性能的高频电容去耦，去耦电容尽可能靠近高频元件的电源端，一般还在较大容量的表面贴装电容旁并联一个小数值的电容。高频元件的电源必须经过很好的滤波，并且与数字电路供电分离。

（5）布线时，电源线和地线要尽量粗。除减小压降外，更重要的是降低耦合噪声。从单片机引入的晶体走线不能离数据线或者控制线太近。注意电源的滤波和电源线的走线。不能将数字信号或控制信号引入到 PLL 回路滤波器元件上。布线时尽量减少回路环的面积，以降低感应噪声。

（6）采用正确的布线规则。在 PCB 板上应该避免长的电源走线，所有元件地线，电源连接线，电源去耦电容必须离高频元件尽可能近，如果 PCB 设计的顶层有铺铜，地脚必须连接到铺铜面，如果 PCB 设计的底层有铺铜，与地脚的焊盘有一个过孔相连会获得更好的性能。所有开关数字信号和控制信号都不能经过 PLL 环路滤波器元件和电感附近。

（7）充分考虑电源对高频元件的影响。电源做得好，整个电路的抗干扰就解决了一大半。高频电路对电源噪声很敏感，要给高频电源加滤波电路，以减小电源噪声对高频电路的干扰。例如，可以利用磁珠和电容组成Π型滤波电路，当然条件要求不高时也可用电感代替磁珠。

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