摘　要

随着小卫星技术的发展，卫星的功能密度越来越大，这对星载计算机的性能提出了更高的要求，传统 CPU 越来越难以胜任现代小卫星繁重的飞行任务。而 ARM 微处理器具有性能高、体积小、功耗低、操作系统便于移植等特点，具有应用于航天领域的广泛前景。

本文给出了基于 AR91R40008 处理器的星载计算机的设计方案，详细论证了方案的可行性与技术细节，以及相应的设计所采用的一些关键技术，最后给出了系统的调试过程。分析和调试的结果表明本系统能够满足任务需求。

本系统的设计与完成，为商用微处理器在航天领域的应用探索出了一条新的途径，也为星载计算机进行高可靠冗余设计与并行处理设计奠定了良好的基础。

关键字：ARM；星载计算机；1553B 总线

Abstract

Deng Pingke(computer Application)

Directed by Ling Baojun Zhang Shancong

With the development of small satellite technology, more and more function units are incorporated in the on-board computer. The traditional microprocessors can not satisfy the need of high capacities of on-board computers. A new generation of on-board computer with the ARM microprocessor presented in the thesis provides a new way in this field.

This thesis presented the scheme of designing the on-board computer based on the AR91R40008 processor, and studied the feasibility and some details of the scheme. Some important techniques were discussed.

According to the results of the experiments, the fruits of this project can satisfy the mission of the small satellite. It provides a good foundation for the on-board computer with high credibility redundancy design and parallel processing ability of the designed computer.

Keyword: ARM; On-Board Computer(OBC); 1553B Bus

第一章 绪论

1.1 本课题的背景、目的及意义

卫星的发展经历了一个由小到大又由大到小的变化过程。目前大多数小卫星均为一体式结构的设计框架，一颗卫星本身就具有某种完整的实用功能，这正体现了现代小卫星的本质——具有高功能密度。而传统大卫星虽然功能相对完整，但是设备也相对复杂，由此产生了实用性不够强、可靠性下降、研制周期长等问题。

与传统卫星相比，小卫星的应用具有设备复杂度低、通信时间延迟小、应急能力与灵活性强、实用性与可靠性高、系统建设周期短、投资和发射与运营成本相对大卫星低等优点。而小卫星的星载计算机作为卫星管理的总枢纽，不仅要完成传统的姿态控制任务，而且也要承担数据采集与存储、数据校验、数据回放、星务管理的新的任务，地位十分重要。

小卫星由卫星平台与卫星有效载荷两部分组成，其功能组成如图 1-1 所示。小卫星平台为有效载荷的服务系统，主要完成功能为：支持有效载荷；保持有效载荷的正确指向；维持有效载荷的合适温度；提供电源、指令和遥测；使有效载荷处于正确轨道，并保持在这个轨道；提供数据存储和通信功能等。因此，小卫星平台可以分为以下的分系统：星载计算机、电源、姿轨控制、推进、测控、热控、结构与机构。

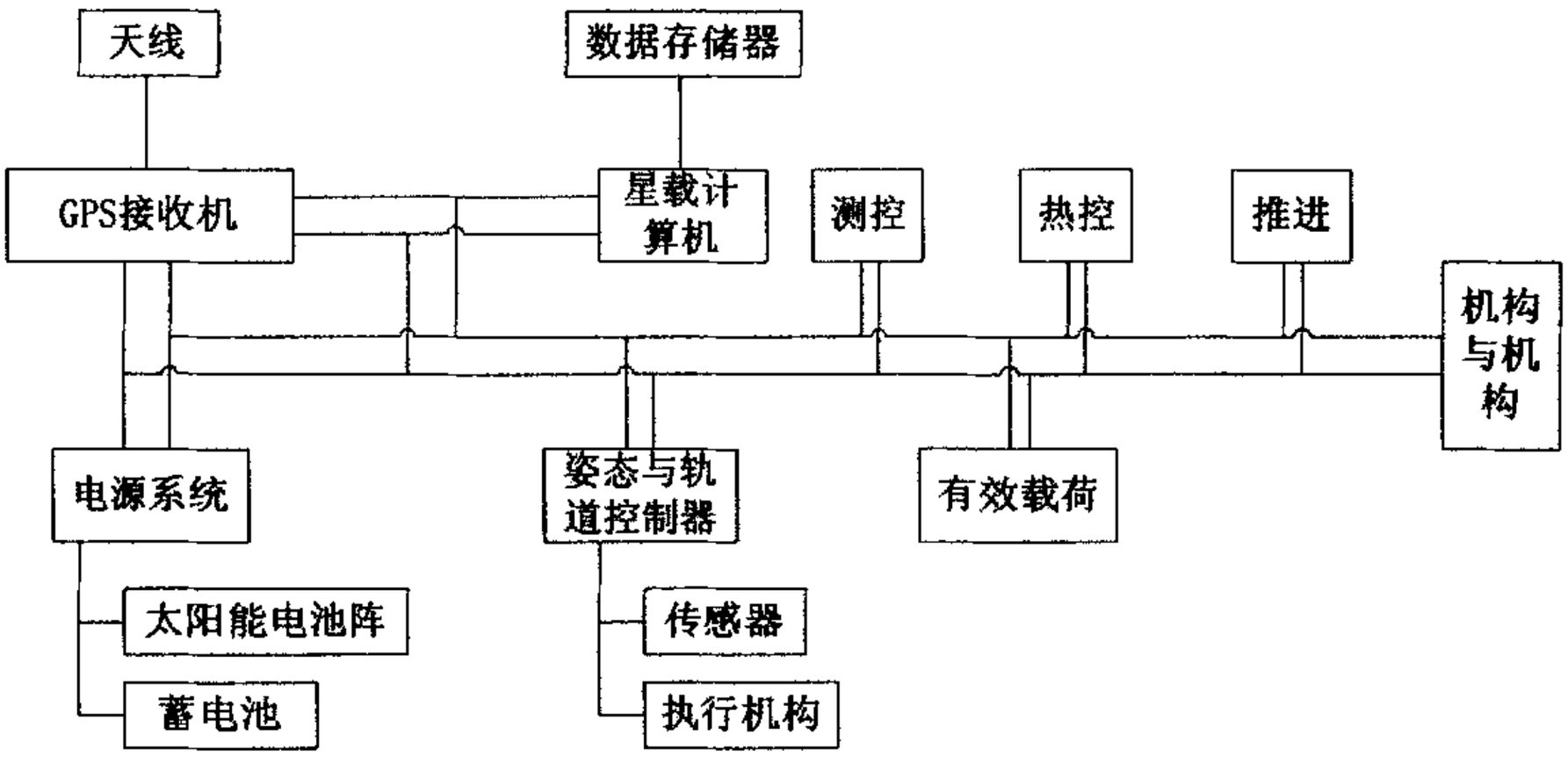

图 1-1：小卫星系统功能组成

在小卫星平台下，星载计算机负责星上数据与程序的存储、处理以及各分系统的协调管理，主要硬件有 CPU、系统时钟、内存、用户存储器等。随着现代计算机技术的发展，计算机的处理能力和通信能力都有了长足发展。而作为现代高科技代表的航天工程，对计算机的依赖性也越来越大。

从我国的航天计算机现状来看，星载计算机单机处理能力大多不超过

10MIPS，系统结构设计上采用单机固定结构，整机多冗余结构，并且大部分未采用操作系统，少量采用类似 DOS 的单任务操作系统。这样的水平已经不能满足复杂度越来越高的航天工程对星载计算机性能的更高的要求。

小卫星技术对我国航天的发展具有极其重要的意义。研究小卫星平台上的星载计算机系统也具有重大的意义和广阔的应用前景。为了跟踪国际先进航天电子系统的发展，针对我国小卫星平台的研究，我们开展了基于 ARM 的高性能星载计算机研究。此计算机研制的成功，将广泛应用在低成本卫星平台上，对我国卫星平台的发展有很大的应用价值。

1.2 国内外星载计算机发展现状

近年来，随着我国航天工程项目的增加，航天工程项目中使用的计算机的种类和数量也越来越多，计算机技术在我国航天工程中的重要性日见凸现。由于我国半导体工业的整体水平比较低，当前我国航天计算机仍然大量采用 8031、80186 等处理器。这些处理器处理能力低，不适合进行多任务处理。对小卫星这种结构简单，对 CPU 依赖性大的系统，很难胜任。

在最近的十年中，冷战的结束半导体产品在商业领域中的应用越来越广，商业领域对半导体工业的投资也越来越大，使得半导体工业的技术进步也越来越集中于商业芯片领域。随着技术的进步，相对于军用级芯片(或航天级芯片)而言，商业级芯片有着更快的发展速度。

虽然，因为工艺上的原因，商业级的芯片在可靠性、抗辐照等方面与军用元器件还有一定差距；但在速度、功耗、体积、封装、结构、价格等方面，与军用器件相比，却拥有巨大优势，而且目前这种优势还在不断地扩大。另外，由于美国对我国的航天高技术进行封锁，同时我国的半导体工业(特别在超大规模集成电路方面)相对落后，我们也不得不探索各种技术途径来提高我们的航天用电子设备的技术水平，降低生产成本，所以有必要对商用芯片在航天领域中的使用进行研究。

目前，已经有大量的商业 CPU 结构被改造后用于空间飞行，目前主要的星载处理器有 Motorola PowerPC 系列（RHPPC，RAD6000，RAD750），SUN 公司的 SPARC 系列（EH32），MIPS 系列（Mongoose，RH32），INTEL 公司的 X86 系列（80386、80486、Pentium、Pentium-II 的空间飞行器板）及 ARM 系列。由于 INTEL 公司在 PC 上的巨大成功，使得 X86 系列在航天领域应用的最广泛；PowerPC 系列在大型航天器中得到了广泛应用，如美国火星探测器（PowerPC750）、NASA 的先进飞行计算机（PowerPC604）；SPARC、ARM、和 MIPS 系列 CPU 均为 CPU 联盟，不同厂家通过 SPARC、ARM 和 MIPS 公司的授权生产各种兼容的 CPU，因此这三种 CPU 的生产

商非常多；ARM 系列在家电及多媒体，移动通信上有广泛应用，美国的 GPS 使用 ARM；SPARC 在高端服务器上应用的较多，欧空局的标准有效载荷计算机采用 SPARC7；MIPS 是 Aeroflex 公司的高性能 CPU，应用领域相对前面要少。

表 1-1 目前常用的飞行计算机处理器

任务	处理器	任务	处理器
Cassini	1750A	Sojourner	80c85
Cluster(ESA)	1750A	Galileo AACS	ATAC(bit slice)
MSTI-1，2	1750A	SPOT-4	F9450
Rosetta(ESA)	1750A	EO-1/WARP	Mongoose V
EOS Terra	1750A(2)	IecSat Glas	Mongoose V
EOS Aqua	1750A(4)&8051(2)	MAP	Mongoose V, UTMC 69R000
EOS Aura	1750A(4)&8051(2)	CGRO	NSSC-1
Clementin	1750A,32Bit RISC	Topex/Poseidon	NSSC-1
MSTI-3	1750A,R-3000	UARS	NSSC-1
Pluto Express	32Bit RISC	EUVE	NSSC-1, 1750A
Sampex	80386,80387	HST	NSSC-1/386, DF-224/486
SMEX	80386,80387	Coriolis	RAD6000
SWAS	80386,80387	Deep Space-1	RAD6000
TRACE	80386,80387	Gravity Probe B	RAD6000
WIRE	80386,80387	HESSI	RAD6000
FUSE	80386,80387,68000	MARS 98	RAD6000
Surrey Micro Sat	80386EX(2)	MARS Pathfinder	RAD6000
UoSat-12	80386EX(2)	SIRTF	RAD6000
FAST	8052	SMEX-Lite	RAD6000
Health Sat-II	80c186, 80c188	Swift	RAD6000
PoSat-1	80c186,TMS320C25, TMS320C30	Triana	RAD6000

1.3 本文的主要研究内容和成果

本课题作为小卫星平台数据处理单元（DHU）的预研，在借鉴其他研究成果的基础上，突破传统的设计观念，设计并实现了一种用于空间环境的、高可靠性的、基于实时嵌入式系统的可重构航天计算机系统。在此，

本文主要对以下几个方面进行了研究：

1、按照航天器计算机系统的各阶段设计要求，对星载计算机设计过程中的重点问题及实现方法做具体分析。

2、针对航天飞行器的飞行环境的要求，对星载计算机的关键器件进行了选择，并对大规模可编成器件在空间飞行器上的应用进行了探索。

3、按照嵌入式系统的调试方法，建立了嵌入式系统的调试环境，对嵌入式系统的在星载计算机上的移植进行了尝试。

第二章 设计过程中的重点与实现方法

本章将按照航天器计算机系统的各阶段设计要求，对星载计算机设计各阶段中的重点及实现方法做具体分析。

2.1 计算机系统定义阶段

计算机系统定义阶段主要包括系统要求的定义、体系结构的选择以及计算机系统的组成单元，其核心是体系结构的选择。

体系结构是计算机系统研制的框架，其模型是根据具体飞行任务的要求以及构造计算机系统基线的工作要求建立的。体系结构提供了系统各组成单元的有关信息，描述它在高层次上的相互关系。体系结构中，网络类型和接口的定义是我们最关心的物理特征，它既包括数据网络或总线的物理结构，也包括协议或者总线各部分之间的逻辑关系。星载计算机的体系结构主要包括两种类型：集中式体系结构和分布式体系结构，其中分布式体系结构又分为环形结构和总线型结构。我们可以根据以下要素决定系统的体系结构：

a) 体系结构能否满足飞行任务的目的

b) 体系结构的复杂程度如何

c) 体系结构是否有利于系统测试

d) 体系结构是否有利于系统维护

我们根据小卫星体积小、功能密度大的特点，选择集中式体系结构作为本系统的结构类型。由于小卫星内部节点不多，每个节点都已定义清楚，而且它们都只与一个处理器相连，采用这种结构是很有效的。它也比较可靠，一个节点的故障不会影响其他节点。但是中心处理器也是一个潜在的单点故障源，必须对它仔细设计以减少或消除这种风险。

每种体系结构都规定了一种数据总线结构及相应的通信协议，它们是整个体系结构的一部分。集中式体系结构通常有一个离散接口或模拟接口，使用串行或并行总线，通信协议开销比较小。表 2-1 列出了两种体系结构可选用的数据总线标准。

表 2-1 可选用的数据总线标准

可选择的体系结构	数据总线标准
集中式体系结构	RS232
	RS422
	RS485
	MIL-STD-1553
	离散信号：有模拟信号线

分布式体系结构	RS422 双绞线
	RS232 双绞线
	FDDI 光纤
	MIL-STD-1553 双绞线
	MIL-STD-1773（光纤标准总线）
	CAN 双绞线
	SAE-4074 同轴电缆 / 光纤

总线的选择应在满足任务需要的情况下，根据 CPU 的处理能力与总线的容错要求，尽量提高可靠性，如有可能，应考虑总线冗余。

2.2 计算机资源的估计阶段

有了功能划分方案和系统体系结构之后，就可以确定所需的处理任务，决定对数据的要求，估计软件规模和吞吐量要求。在这个阶段，决定了处理器的处理要求及计算机的存储空间大小。

我们在“软件要求规范”和“接口要求规范”这两个文件中规定了对处理任务和系统接口关系的要求。在确定星载计算机设计要求的同时，还应对小卫星必须完成的各种任务进行分类，确定所必须的处理任务。星载计算机的处理软件主要分为四个类别：控制系统软件、系统管理软件、飞行任务数据处理软件和操作系统软件。不同的软件对星载计算机提出的要求不同。

星上控制系统软件包括姿态确定与控制，它要求有一个输入激励，其响应是系统状态的变化。这个软件涉及高精度的数学计算并且有严格的实时性要求。

飞行任务数据处理软件需要处理大量的数据，这些数据边采集边进行压缩和预处理。这需要很大的存储能力，以便存储大批数据。

操作系统软件直接管理计算机的各种资源，控制对不同任务的资源分配。其基本管理功能包括任务调度、时间管理、中断处理、输入输出设备管理、其他外部设备驱动、系统测试及内存故障管理。这些功能对于计算机都是必不可少的，我们通常把操作系统看作是应用软件的额外开销。

我们根据处理软件的类型，可以估计软件的规模及吞吐量。星载计算机应用软件的初步估计如表 2-2 所示：

表 2-2 星载计算机应用软件的初步估算见下表：

功能链	功能	ROM(KB)	RAM(KB)	吞吐量(KIPS)	执行频率(HZ)	单位时间内的吞吐量(KIPS)	任务数
通信	命令处理	2	8	7.0	10.0	0.7	1
	遥测处理	2	5	3.0	10.0	0.9	1
姿态敏感器信号处理	速率陀螺	1.6	1	9.0	10.0	0.9	1
	太阳敏感器	1.0	0.2	1.0	1.0	1.0	1
	磁强计	0.4	0.2	1.0	2.0	0.5	1
	星敏感器	4	30	2.0	0.01	200	1
姿态确定与控制	GPS 数据采集	1	2	2.0	1.0	2.0	1
	运动方程积分	4	0.4	15.0	10.0	1.5	1
	误差确定	2	0.2	12.0	10.0	1.2	0
	进动控制	6.6	3	30.0	10.0	3.0	1
	磁控	2.0	0.4	1.0	2.0	0.5	1
	推力器控制	1.2	0.8	1.2	2.0	0.6	1
	反作用飞轮控制	2.0	0.6	5.0	2.0	2.5	1
	星历表推算	7.0	5	4.0	0.5	8.0	1
轨道确定与控制	轨道推算	26	8	20.0	1.0	20.0	1
星务管理	自主运行	4	2	2.0	1.0	2.0	1
	上行数据管理	2	1	2.0	10.0	0.2	1
	下行数据管理	3	2	4.0	10.0	0.4	1
	1553B 通讯管理	8	2	30.0	20.0	1.5	2
	时间和同步	1	1	0.8	1.0	0.8	1
	有效载荷监视	1	0.6	2.0	1.0	2.0	1
	系统参数存储	1.6	1.4	1.0	1.0	1.0	1
	故障监测与校正	8	2	20.0	1.0	20.0	1
其他功能	能源管理	2.4	1	5.0	1.0	5.0	1
	热控	8	2	3.0	0.1	30	1
	卡尔曼滤波	18	2	100	0.01	10 000	0
合计		117.8	81.8	283	—		25

在估计操作系软件规模时，我们可以以表 2-2 中的数据为基础，由于实际系统中各部件的任务各不相同，所以应当灵活应用这些数据。星载操

作系统的规模和吞吐量估计如下表 2-3 所示

表 2-3 星载操作系统的规模和吞吐量估计

功能	ROM(KB)	RAM(KB)	吞吐量(KIPS)
应用软件	117.8	81.8	283
接口驱动程序	4	2	32.0
操作系统	19	1.5	0.5
浮点及数学运算	15	3	21
合计	155.8	88.3	336.5

根据表 2-3，我们得出计算机中 RAM 的容量最小为 88.3KB，外接 FLASH 存储器的容量最小为 155.8KB。当考虑软件在编写过程中的增长余量及在轨运行时的备份量时，认为软件从系统需求评审到发射，软件规模和吞吐量会增加一倍，在轨备份量取计算机容量的 30%—50%。当备份量取 50%时，我们可以算出 RAM 需求量为 264.9KB，FLASH 容量为 467.4KB，吞吐量为 673KIPS。操作系统采用抢占式的时间片轮换调度算法，其本身要求达到的任务级响应为 0.1ms。我们根据最紧张的一个任务卡尔曼滤波算法，我们需要的处理能力最少为 10MIPS。对 ARM7 处理器，处理能力为 0.9MIPS/MHz，系统时钟 66MHz 时，CPU 占用率约为 1.0%，因此每秒钟由于任务调度导致的最大累计误差不超过 10ms。；对 ARM9 处理器，处理能力为 1.3MIPS/MHz，系统时钟为 120MHz 时，CPU 占用率约为 0.43%，因此每秒钟由于任务调度导致的最大累计误差不超过 4.3ms。这两种 CPU 均能达到最大执行频率 50HZ 的要求。

2.3 研制阶段

在研制过程中，存在着许多问题，它们会影响方案设计。其中比较重要的主要有元器件的选择、系统的可靠性设计和软件开发环境的选择等。

空间飞行系统电路通常要求符合 MIL-STD-883b，S 级，辐射加固（总剂量），单粒子事件翻转（SEU）冗余部分。MIL-STD-883 为测试与屏蔽标准。MIL-M-38510 涉及抗辐射方面的内容，S 级专门针对空间飞行器应用。元器件在地面充分地进行筛选和试验，同时采取一定的降额。各类元器件的降额使用参数参照国军标《GJB/Z35-93 元器件降额准则》执行。单机要通过环模试验、老练试验及电磁兼容性等项试验，确保设备在空间环境下工作的安全性及可靠性。外围的存储器访问考虑增加 EDAC 控制，我们可以采用专门的 EDAC 芯片或者用 FPGA，来实现这一功能。为了便于研究

和工程实现，元器件应具有从商业级到军品的各个级别的系列产品。选用的元器件应在能满足性能要求的前提下，尽可能降低成本。

星载计算机是小卫星平台的最重要的部分之一，它的可靠工作直接关系着卫星的成功与否。因此，采取适当的冗余和保护措施来保障其正常工作是尤为重要的。在本设计中，我们主要采用系统的可重构设计来提高系统的可靠性。

可重构设计是指在数据处理系统中，软、硬件模块能根据变化的数据流或算法进行重新配置或重新设置。

系统的可重构设计可以分两个方面：

a 软件可重构

设置操作系统，使其能够根据任务的变化加载不同的任务处理程序，同时，允许在轨上行程序，进行任务加载，实现软件系统的在轨重构。

b 硬件可重构

采用 FPGA 器件，用 CPU 实现对 FPGA 的加载。当系统发生局部故障时，通过对 FPGA 的重新加载，对故障区域进行隔离，从而实现硬件系统的在轨重构。

在选择编程语言时，首先要考虑的因素是程序在运行实时性上有没有严格的要求。在选定了某种语言后，就需要考虑它具有什么样的开发环境和编译软件，是否具备相应的开发人员。高级语言一般与宿主机（指软件开发时所用的计算机）上的开发工具及软件工程环境相适应。选择了宿主机和目标机以后，必须有一个交叉编译软件可供使用。交叉编译软件在宿主机上运行，产生可在目标机上运行的代码。由于主机和目标机往往运行着不同的操作系统，而且处理器的体系结构也彼此不同，这就提高了嵌入式开发的复杂性。

2.4 计算机系统总装的测试阶段

系统总装和系统测试就是要把各个分系统组合起来成为一个能满足最高层系统要求的结构配置。从这个意义上来说，总装与测试就是系统工程。通过测试，使我们对系统性能增强了信心，确定系统要求已被满足，相信任何超出系统要求而发生的事情不会对系统构成危害，系统仍能发挥其正常功能。但是，直到在轨运行阶段，我们才能确定是否正确的对计算机系统进行了验证。即使在投入在轨运行以后，也还要进行在轨综合测试，以保证计算机分系统继续正常工作。

由于本课题作为 DHU 项目的一部分，其系统测试随 DHU 整机测试一起进行。根据以上分析，我们提出了本课题的初步方案设计框图，如图 2-2

所示：

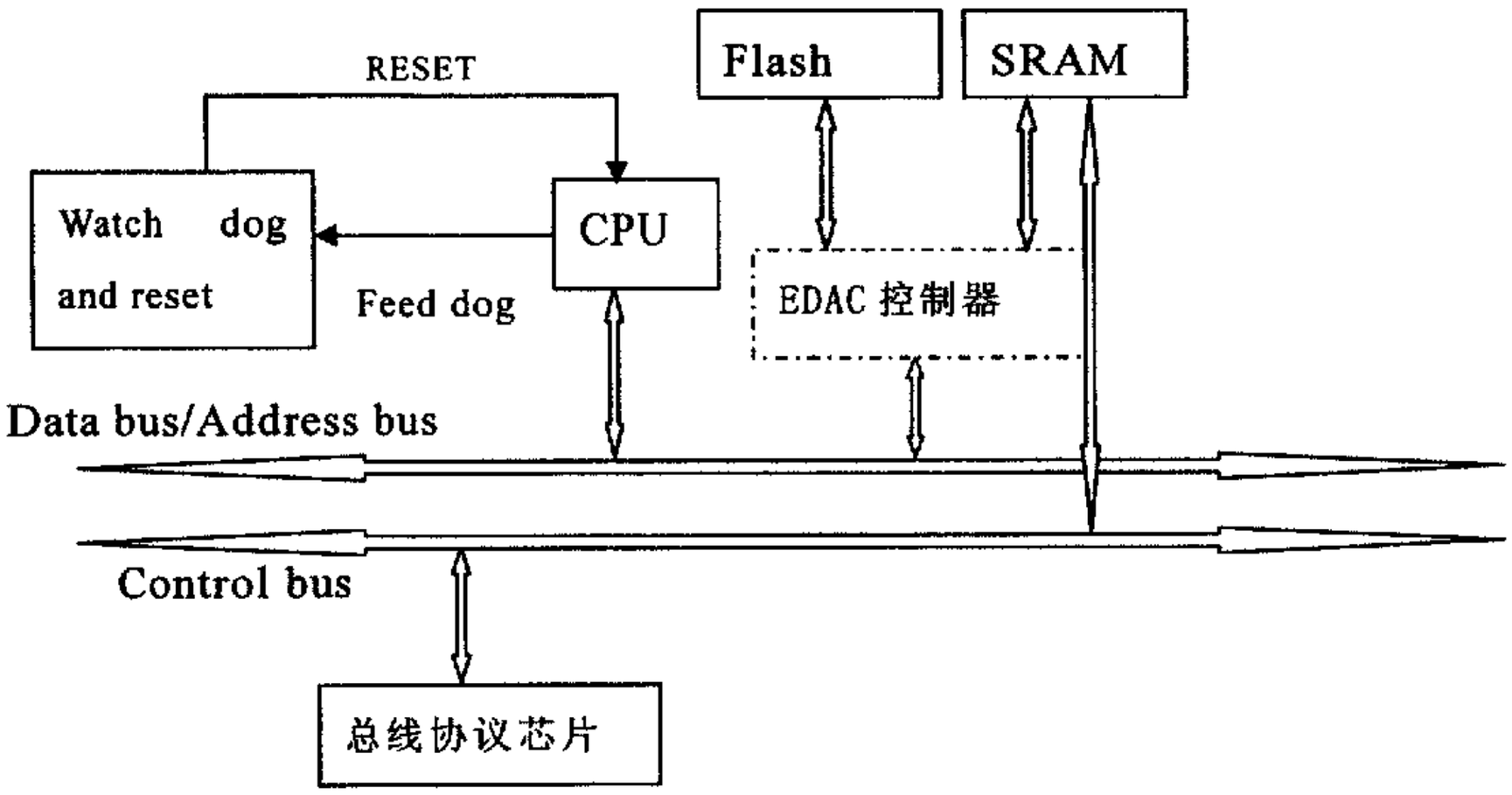

图 2-2 初步方案设计框图

本硬件系统由 CPU、总线接口电路、看门狗电路、程序存储器、数据存储器等组成。其中 CPU 是系统的核心，其他电路均作为 CPU 外围电路与 CPU 相连。程序存储器分别为操作系统和应用程序提供存储空间。这样，系统可以根据不同任务加载不同的应用程序。SRAM 不仅为系统提供了足够的程序运行空间，同时也提供足够的系统数据和用户数据的应用空间。同时，系统采用专门的大容量 Flash 提供星务数据的存储。

需要说明的是由于时间的限制，本文没有实现 EDAC 控制器的设计。

第三章 系统硬件的总体设计

3.1 硬件总体构架

系统硬件主要由中央处理器（CPU）单元、微处理器监控单元、存储器单元、总线接口单元、电源单元、时钟单元及译码电路组成。

CPU 单元是整个系统的核心单元，它配置有看门狗电路，以防止长时间运行软件出现程序跑飞或死机。存储器单元包括 SRAM 和 Flash，SRAM 用来存储系统运行的临时数据，FLASH 用来存储系统的软件程序和星务工程参数。时钟单元从 GPS 接收机获得精确的定时脉冲，为系统提供精确时间。电源单元为整个系统提供不同电压的电源。操作系统在上电复位后从 Flash 加载到 CPU 内部的 SRAM 运行。总线接口单元完成卫星平台全部设备之间以及挂接在总线上载荷设备间的数据通信管理。时间管理单元完成飞行时间基准计时和校时，也可为定时器复位电路提供基准时钟。硬件总体结构如图 3-1 所示：

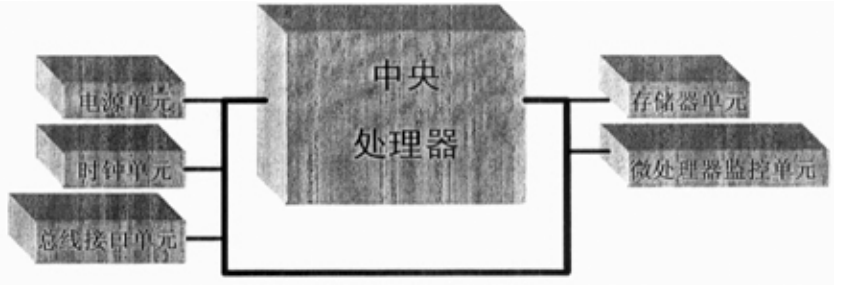

图 3-1 硬件总体结构图

以下我们将对各单元电路的功能及器件的选型做详细的说明。

3.2 CPU 单元的功能及器件选择

对于计算机系统，中央处理器是系统的核心，它完成整个系统的控制、信号处理及数据处理功能。处理器的选择至关重要，直接关系到系统开发的难度和进度，对于整个项目的成功非常重要。我们通常根据以下因素来选择处理器：

a) 性能：处理器必须具备足够的性能来处理任务，满足实时多任务处理的要求；
b) 功耗：卫星能提供的功率有限，要求星载计算机采用功耗低的处理器；
c) 体积与封装：卫星上空间狭小，芯片尺寸要求尽可能小；
d) 实现：根据具体的应用和需要的外围接口，确定处理器；

e) 工具支持：是否有支持软件创建、调试、系统集成、代码调整和优化的工具；
f) 操作系统支持：针对所选处理器是否有优化的操作系统支持，能够缩短应用开发周期；
g) 仿真支持：是否有成熟的处理器仿真器支持；
h) 应用支持：是否提供用于外围设备的驱动程序、板级支持包以及其它“启动套件”；
i) 成本：嵌入式应用对成本非常敏感，性价比是否高；
j) 算法复杂性：特定处理器能高效地运行特定算法，选择的处理器应该尽可能与应用相匹配；
k) 安全性与可靠性高；

综合以上因素，本系统拟采用 ARM 处理器。由于该型处理器在国内航天器上较少使用，因此基于 ARM 系列处理器的选型问题将作为一个关键问题进行考虑。

3.2.1 ARM 处理器简介

第一个 ARM 处理器原型于 1985 年在英国剑桥诞生，随后 ARM 公司(Advanced RISC Machines Limited)于 1990 年成立，作为微处理器行业的一家知名企业，设计了大量高性能、廉价、耗能低的 RISC 处理器、相关技术及软件。技术具有性能高、成本低和能耗省的特点。

ARM 将其技术授权给世界上许多著名的半导体、软件和 OEM 厂商，每个厂商得到的都是一套独一无二的 ARM 相关技术及服务。

目前，总共有 30 家半导体公司与 ARM 公司签订了硬件技术使用许可协议，其中包括 Intel、IBM、LG 半导体、NEC、SONY、菲利浦和国民半导体这样的大公司。至于软件系统的合伙人，则包括微软、SUN 和 MRI 等一系列知名公司。

ARM 处理器具有三大特点：耗电少功能强、16 位/32 位双指令集和众多合作伙伴。ARM 是设计公司，本身不生产芯片，采用转让许可证制度，由合作伙伴生产芯片。ARM 通过该模式建立起新型的微处理器设计、生产和销售商业模式。正是这种商业模式，使得采用 ARM 技术的微处理器遍及各类电子产品。目前，ARM 对于高性能、低功耗、低成本的嵌入式应用领域，作为 32 位嵌入式 RISC 微处理器业界的领先供应商，ARM 占有超过 75%的市场。

ARM 提供一系列内核、体系扩展、微处理器和系统芯片方案。由于所有产品均采用一个通用的软件体系，所以相同的软件可在所有产品中运行

（理论上如此）。典型的产品如下。

（1）CPU 内核

ARM7：小型、快速、低能耗、集成式 RISC 内核，用于移动通信。

ARM7TDMI(Thumb)：这是公司授权用户最多的一项产品，将 ARM7 指令集同 Thumb 扩展组合在一起，以减少内存容量和系统成本。同时，它还利用嵌入式 ICE 调试技术来简化系统设计。

ARM9TDMI：采用 5 阶段管道化 ARM9 内核，同时配备 Thumb 扩展、调试和 Harvard 总线。在生产工艺相同的情况下，性能可达 ARM7TDMI 的两倍之多。

（2）体系扩展

Thumb：以 16 位系统的成本，提供 32 位 RISC 性能，特别注意的是它所需的内存容量非常小。

（3）嵌入式 ICE 调试

由于集成了类似于 ICE 的 CPU 内核调试技术，所以原型设计和系统芯片的调试得到了极大的简化。

（4）微处理器

ARM7 系列（包括 ARM710、ARM710T、ARM720T 和 ARM740T）：低价、低能耗、常规系统微型处理器，配有高速缓存（Cache）、内存管理、写缓冲和 JTAG。

ARM9 系列（包括 ARM940T、ARM920T 系列）：低价、低能耗、高性能系统微处理器，配有 Cache、内存管理和写缓冲，具有丰富的外围电路，用于多媒体开发。

StrongARM 系列：性能很高、同时满足常规应用需要的微处理器。由 ARM 公司与 DEC 联合研制，后来授权给 Intel。SA1110 和 SA1111 作为平台 CPU 与接口扩展器件，通过与其他接口芯片配合来完成扩展接口的设计任务。

3.2.2 ARM 处理器在航天领域的应用

ARM 处理器从 ARM7 概念产品面世至今只有 10 年左右的时间。在此期间，由于 ARM 处理器具备高性能低功耗的特点，国内外的航天研究机构都因为纷纷开展了这方面的应用研究。在国外，美国空军学院的 FalconSat-2 号钠星也采用了 StrongArm SA1100 处理器。Surrey 大学的小卫星平台 SNAP(Surrey Nanosatellite Applications Platform)也将 ARM 处理器作为 OBC(在轨计算机)的主处理器。除此之外，Surrey 大学的一些星上关键部件（例如星跟踪器 Altair-HB）也使用了 ARM 处理器。在国内，目前有清华小卫星采用了 StrongArm 处理器，并已成功发射运行。另外，国

内负责星上计算机研制的专业单位航天部 771 研究所也开展了 ARM 处理器的研究，并进行了一些试验验证工作。

目前星上主流的处理器还有 X86、1750、ERC32 等体系结构。为了便于比较，下面选取各自具有代表性的产品 486DX2-66、MA31750、TSC695 与 ARM7 产品 AT91R40008 进行比较。

表 3-1 几种嵌入式 CPU 的比较

项目	AT91R40008	486DX2-66	MA31750	TSC695
厂商	ATMEL	INTEL	DYNEX	ATMEL
产品等级	工业品	工业品	抗辐照品	军品
最大功耗(W)	0.055	5	0.3	1.5
典型工作频率(MHZ)	66	33	16	20
处理能力(MIPS)	63	54	2.2	14
产品价格	很低	较低	较高	较高
商业化支持环境	全	全	较全	不全
通用性	较好	好	差	较差
升级产品	ARM9/ARM10/ ARM11	暂无	不祥	TSC695E

3.2.3 ARM CPU 选型

下面从应用的角度，对选择 ARM 芯片时所应考虑的主要因素做一详细的说明。

1、ARM 内核

如果希望使用 WinCE 或 Linux 等操作系统，就需要选择 ARM720T 以上带有 MMU(memory management unit)功能的 ARM 芯片，ARM720T、StrongARM、ARM920T、ARM922T、ARM940T 都带有 MMU 功能。而 ARM7TDMI 没有 MMU，不支持 Windows CE 和大部分的 Linux,目前只有 uCLinux 等几种 Linux 不需要 MMU 的支持。

2、系统时钟控制器

系统时钟决定了 ARM 芯片的处理速度。ARM7 的处理速度为 0.9MIPS/MHz，常见的 ARM7 芯片系统主时钟为 20MHz-133MHz；ARM9 的处理速度为 1.1MIPS/MHz，常见的 ARM9 的系统主时钟为 100MHz-233MHz，ARM10 最高可以达到 700MHz。不同芯片对时钟信号的处理不同，有的芯片只有一个主时钟频率，如 Cirrus Logic 的 EP7312 等；有的芯片内部时钟控制器可以分别为 CPU 内核和 USB、UART、DSP 等功能部件提供不同频率的时钟，如 PHILIPS 公司的 SAA7550 等芯片。

3、内部存储器容量

在不需要大容量存储器时，可以考虑选用有内置存储器的 ARM 芯片。

4、GPIO 数量

在某些芯片供应商提供的说明书中，往往申明的是最大可能的 GPIO 数量，但是有许多引脚是和地址线、数据线、串口线等引脚复用的。这样在系统设计时需要计算实际可以使用的 GPIO 数量。

5、中断控制器

ARM 内核只提供快速中断(FIQ)和标准中断(IRQ)两个中断向量。但各个半导体厂家在设计芯片时加入了自己不同的中断控制器，以便支持诸如串行口、外部中断、时钟中断等硬件中断。外部中断控制是选择芯片必须考虑的重要因素，合理的外部中断设计可以很大程度的减少任务调度的工作量。例如 PHILIPS 公司的 SAA7750，所有 GPIO 都可以设置成 FIQ 或 IRQ，并且可以选择上升沿、下降沿、高电平、低电平四种中断方式。而 Cirrus Logic 公司的 EP7312 芯片，只有 4 个外部中断源，并且每个中断源都只能是低电平或者高电平中断，这样在用于接收红外线信号的场合时，就必须用查询方式，会浪费大量的 CPU 时间。

6、nWAIT 信号

外部总线速度控制信号。不是每个 ARM 芯片都提供这个信号引脚，利用这个信号可以使 CPU 直接与其他处理器（如 80186）或接口芯片（如 DDC 64843）连接，不需要外加高成本的专用控制芯片。另外，当需要扩展外部 DSP 协处理器时，此信号也是必需的。

7、扩展总线

大部分 ARM 芯片具有外部 SDRAM 和 SRAM 扩展接口，不同的 ARM 芯片可以扩展的芯片数量即片选线数量不同，外部数据总线有 8 位、16 位或 32 位。

8、UART 接口

几乎所有的 ARM 芯片都具有 1~2 个 UART 接口，可以用于和 PC 机通讯或用 Angel 进行调试。一般的 ARM 芯片通讯波特率为 115200bps。

9、时钟计数器和看门狗

一般 ARM 芯片都具有 2～4 个 16 位或 32 位时钟计数器和一个看门狗计数器。

10、电源管理功能

ARM 芯片的耗电量与工作频率成正比，一般 ARM 芯片都有低功耗模式、睡眠模式和关闭模式。

11、DMA 控制器

有些 ARM 芯片内部集成有 DMA(Direct Memory Access)可以和硬盘等外部设备高速交换数据，同时减少数据交换时对 CPU 资源的占用。

根据以上各条及综合本系统设计的需要，这里选择了 ATMEL 公司的 ARM7-TDMI 内核 CPU AT91R40008 作为本系统的主控 CPU。其主要原因如下：

首先，AT91R40008 的外部总线与我们目前使用的 80C186 在硬件接口上是兼容的，其外部数据总线为 16 位，地址总线为 24 位，并有 8 根独立的片选，寻址空间从 186 的 1MB 扩展到高达 4Gb，这样为软件的扩展留有充足的空间；16 位数据总线可以配置成位 8 位或 16 访问模式，完全独立的读写控制信号可以直接和存储器或外设相连。

第二，186 在 25MHz 时的处理能力约为 1MIPS，而 AT9140008 的处理能力在 33MHz 时约为 31MIPS，66MHz 时约为 63MIPS。采用这款 CPU 将极大地提高系统的处理能力，为操作系统的良好实时性奠定了基础。

第三，现有的仿真器 ARM-STAR 不仅支持 ARM7 内核 CPU，同时也支持 ARM9、ARM10 内核 CPU，可以很方便的为我们的产品升级换代。

3.2.4 AT91R40008 介绍

本系统选择基于 ARM7 内核的 ARM7TDMI™处理器 AT91R40008。该处理器主要有以下功能。

a) ARM7TDMI™含 Thumb 指令的 ARM7 内核

T：16 位压缩指令集 Thumb；

D：在片调试（debug）支持，允许处理器响应调试请求暂停；

M：增强型乘法器（multiplier），可以产生全 64 位结果；

I：嵌入式 ICE 硬件提供片上断点和调试点支持。

b) 内部 32 位数据总线，外部 16 位数据总线，支持 8 位、16 位读写。

c) 256KB 片内 SRAM。

d) 可编程扩展总线接口（EBI）4 个独立片选信号，最多可以扩展到 8 个。

e) 4 个外部中断，包括一个 FIQ 中断。

f) 32 位可编程 PIO。

g) 3 通道定时器/计数器。

h) 2 个 USART。

i) 可编程内部看门狗。

j) 高级电源管理。

k) 0Hz to 70MHz 工作频率。

l) -40° C to +85° C 工作温度范围。

m) 嵌入式 ICE。

AT91R40008 硬件结构图如下图 3-2 所示：

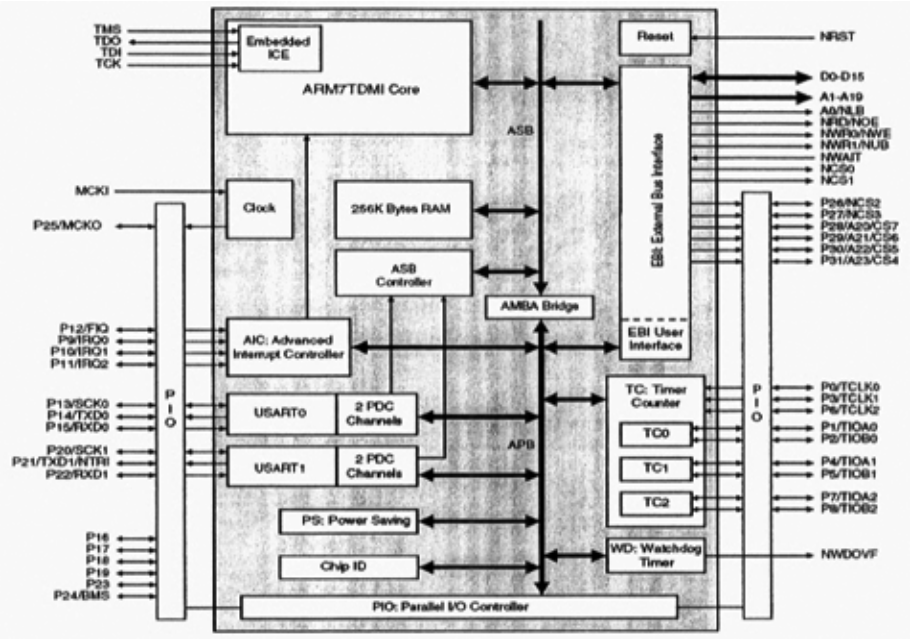

图 3-2 AT91R40008 硬件结构图

3.3 存储器单元的选择

本系统的存储器包括程序存储器和数据存储器。数据存储器可以是 SRAM 或 Flash，程序存储器可以是 PROM、E^2PROM、Flash 等。程序存储器中保存操作系统和应用程序，其重要性要求和 CPU 等同。根据可靠性要求，需要考虑冗余。用于存储应用程序的存储器，其内容因任务、设备而变化，采用 E^2PROM 等可重复编程的器件。另外，采用在线可编程器件的好处在于便于在轨软件维护（On-Board Software Maintenance OBSM），这点对于任务灵活多变的 DHU 单元非常重要。而目前我国航天器中绝大多数软件是固化的，只有少量可编程的软件，这种设计已跟不上目前航天任务的需求。

数据存储器中存放的不同数据发生错误对于系统的影响是不一样的。对于堆栈区、中断向量区、变量区发生错误就意味着软件运行发生错误，因而需要有可靠的对应措施，如动态纠错编码；对于存放工程参数或星务数据的存储区，其发生错误对软件的运行影响不大，无需动态纠错。对于 E^2PROM 和 SRAM，根据任务的不同或存储时间的长短应采用不同的可靠性措施。对于 SRAM 和闪存，可能的预防和纠正错误的措施是采用纠错编码，如 RS 编码。对于程序存储器 E^2PROM 和闪存等在容量上都应有一定的冗余，以便在局部发生故障时利用系统重构实现故障隔离和故障恢复。

根据以上考虑，我们分别对数据存储器和程序存储器器件的选型分别做以介绍。

3.3.1 BOOT Flash 的选择

BOOT Flash 是系统中的静态存储器件，用于存储操作系统和应用程序。系统中选用 ATMEL 公司 AT29LV 系列 Flash。该类型 Flash 是小扇区型 Flash，不仅具有单周期改写功能（擦除和写入），而且具有较高的读写速度。该系列 Flash 体积小，功耗低，在编程时不需要提供额外的电压，具有在系统改写功能，其接口与 AT91R40008 提供的静态存储接口兼容，可以直接连接。

目前 AT29LV 系列提供 1Mbit、2Mbit、4Mbit 三种容量的芯片，每种都提供 PLCC 和 TSOP 封装。PLCC 封装的尺寸小于 TSOP 封装。系统要求具有 256KB 的静态数据存储容量，采用后两种芯片都可。

根据系统的可靠性设计要求，我们采用两片 AT20LV020 芯片作为 BOOT Flash。该芯片的容量为 256KB，每扇区 256 字节，共 1024 个扇区，TSOP 封装。该芯片还具有以下特点：

a) 单一 3.3V 供电
b) 快速读取时间：100ns
c) 低功耗
d) 15mA 工作电流
e) 40 μA CMOS 静态电流
f) 首尾两个 8K 字节带保护的安全区
g) 快速扇区编程，周期 20ms
h) 内部控制器与定时器
i) 典型耐久度：大于 10000 次擦写
j) CMOS 和 TTL 兼容输入输出
k) 工作温度：-40°C—+85°C

该芯片内部结构如图 3-3 所示：

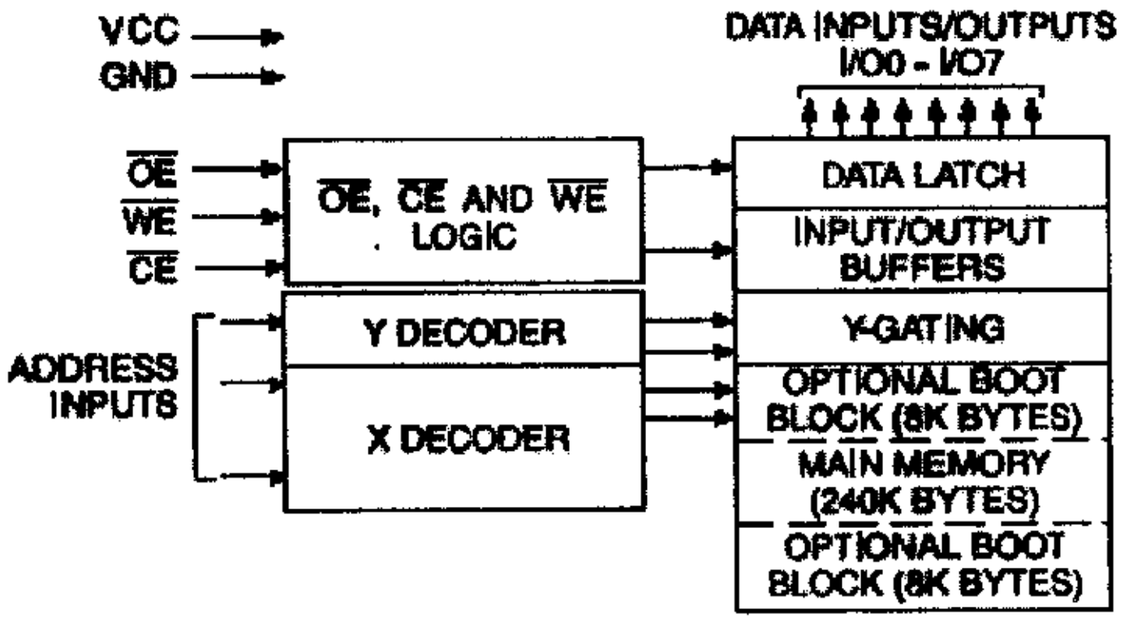

图 3-3 AT29LV020 结构图

在使用中应注意，由于 CPU 工作频率较高，因而要采用读出速度较快的型号与 CPU 读写时序配合。在实际使用中采用-10TI 的型号，以达到与 CPU 的访问匹配。

3.3.2 星务数据存储器的选型

星务数据存储器要求最少 256MB 的存储空间，同时采用非易失性器件，保证在系统断电或重新复位后能保留以前的星务数据。非易失性存储器主要包括 PROM、EPROM、E^2PROM 及 Flash 等。由于要求在系统更新，因此只能采用 E^2PROM 或 Flash 等可反复写入的器件实现。目前市场上没有这么大容量的 E^2PROM，只能采用 Flash 器件做星务数据存储器。由于单芯片的容量越高，系统中使用的芯片数量越少，在印制板上所需要的面积就越小。因此，为减少系统的整机尺寸，在系统中应尽量使用容量大的芯片。

芯片市场上，大容量的 Flash 芯片由于用量较少，订货周期较长。考虑到项目的实际要求和 Flash 芯片市场的实际状况，最后选用市场上货源比较充足的规格：K9K2G08U0M。该芯片的容量为 2Gb，TSOP 封装，在系统中使用一片。该芯片具有以下特点：

a) （2K+64）bit×8bit 的读/写缓存

b) 最大 50ns 的块访问时间

c) 块编程周期 300us

d) 块擦除周期 2ms

e) 控制/地址/数据总线复用

f) 具有硬件数据保护功能，数据保存时间可长达 10 年

g) 工作温度：-40℃—+85℃

该器件外形如图 3-4 所示：

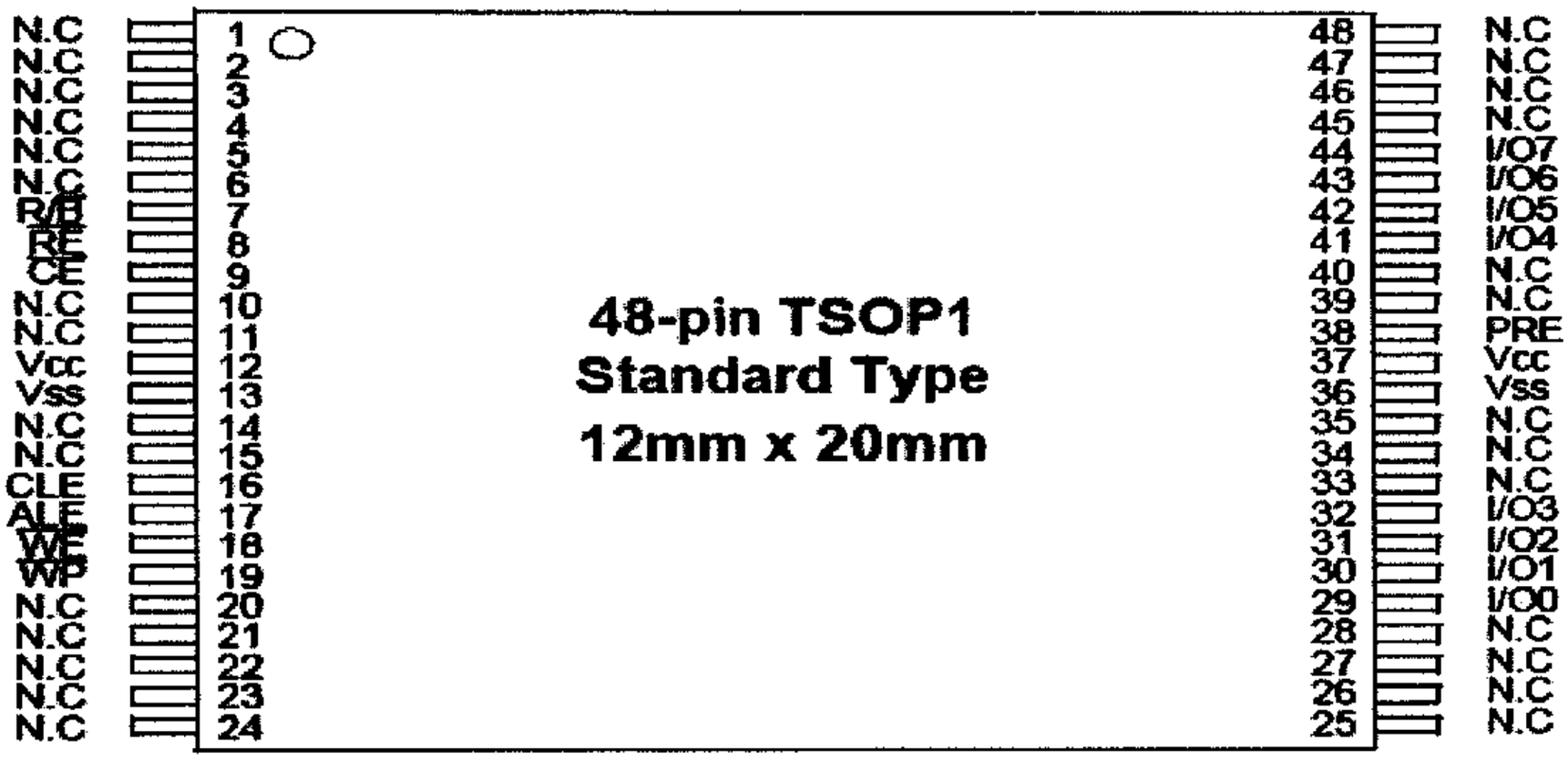

图 3-4　　K9K4G08U0M TSOP 封装外形图

3.3.3. SRAM 的选择

SRAM（静态随机存取存储器）是一种高速易失性存储器件。它是大多数计算机系统中一个关键部分。本系统要求板载外部 SRAM 容量不小于 1MB。目前生产大容量 SRAM 的厂商主要有 ISSI、Cypress、Aeroflex、RENESAS、ST 等。由于 ISSI 公司的高速异步 CMOS 工艺 SRAM 具有集成度高、功耗低、读写速度快、面积小及市场供应充足等优点，所以我们选择了 ISSI 的 61LV51216 作为板载 SRAM，该芯片的容量为 1MB，TSOP 封装，在系统中使用一片。该芯片具有以下特点：

a) 单一 3.3V 供电

b) 允许单独访问高 8 位和低 8 位数据

c) 访问时间：8、10、12ns

d) 三态输出

e) 工作温度：-40°C—+85°C

芯片内部结构图如图 3-5 所示：

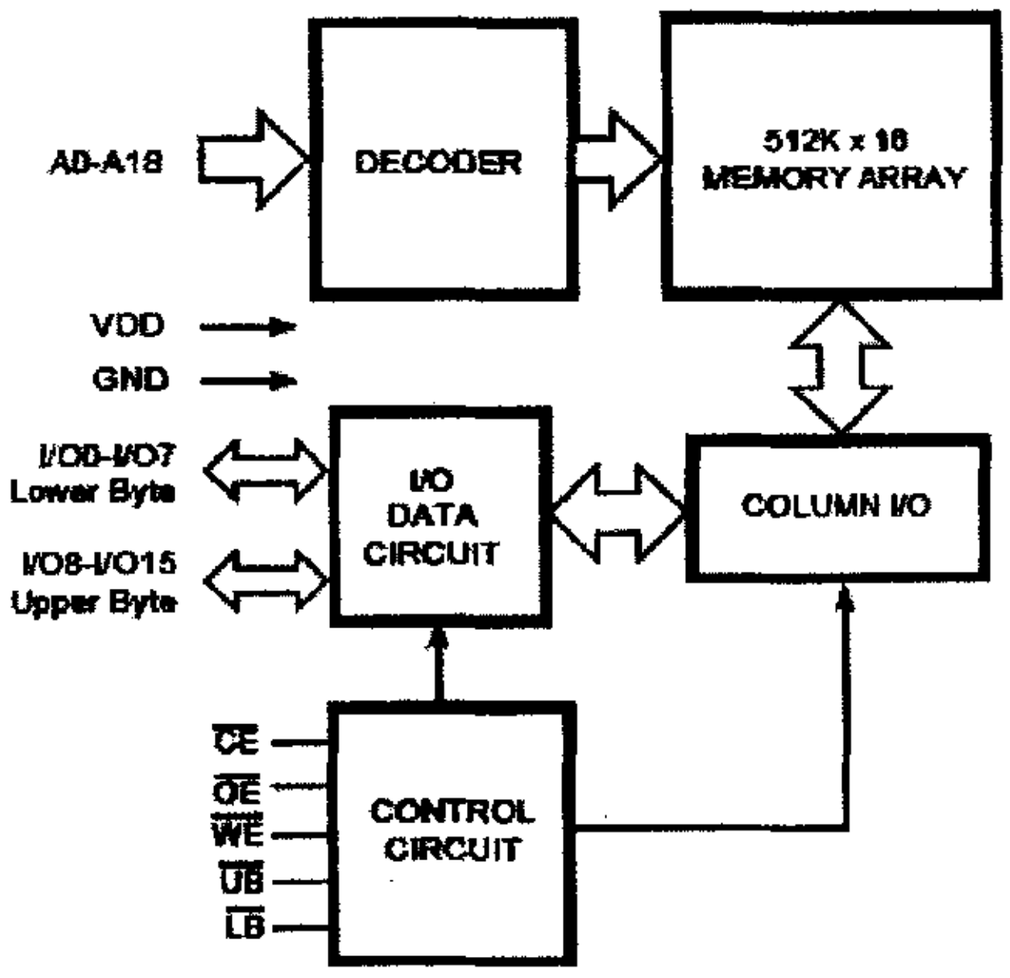

图 3-5 61LV51216 内部结构图

3.4 总线接口单元选择

随着小卫星功能的不断增加，系统越来越复杂，各个分系统之间的通信也越来越多，对总线的速率与容错性能的要求也越来越高，因此，对星载计算机系统总线的选择也是一个重要问题。目前国内外航天领域常用的系统总线主要有 RS-485/422、1553B、I^2C 及 CAN 等串行总线。

我国 SJ-5 卫星采用了 RS-485 总线。该总线为主从命令/响应结构，由一个控制节点和若干个下级节点组成，下级节点只能在主控节点的查询下工作，难以满足实时性要求，数据传输率随着总线长度的增加而下降。RS-485 标准只对电气接口做了规定，而没有通信协议等高层内容，并且，RS-485 标准没有错误检测机制，一旦有主节点故障，整个系统将瘫痪，可靠性无法保障。

1553B 是一种高可靠的串行通信总线，采用中央集权方式进行控制。总线组成包括一个总线控制器（BC），负责总线调度、管理，是总线通讯的发起者和组织者；若干个与总线勾连在一起的远置终端 RT(最多不超过 31 个)，另外还有一种总线监视器（BM），用于监视总线的运行。该总线采用指令应答方式实现系统通讯，采用冗余通道和奇校验以及相应的错误处理来提高系统通讯的可靠性。

控制区域网 CAN 总线是 80 年代初出现的一种工业现场总线，最初是为了解决汽车中大量的控制系统与测试仪器之间的数据交换而开发的一种串行数据通信协议。CAN 总线成本低，结构灵活，具有国际标准 ISO-11898。自 90 年代以来，随着小卫星技术的逐渐成熟及应用的不断扩大，英国萨瑞

大学和德国空间研究机构率先在小卫星中使用 CAN 总线。CAN 总线采用多主结构，当多个 CAN 节点同时发送报文的情况下，总线访问冲突通过“位仲裁”法则解决。

作为星载计算机总线，首先考虑的是可靠性。因此，我们选择可靠性最高的 1553B 总线作为专用接口总线，选用 DDC 公司的 DDC64843 芯片作为 1553B 协议芯片。

3.4.1 1553B 总线简介

MIL-STD-1553B 总线是美国空军电子子系统联网的标准总线，是一种中央集权式的串行总线。总线组成包括一个总线控制器——负责总线调度、管理，是总线通讯的发起者和组织者；若干(最多不超过 31 个)远置终端，另外还可以有一种设备即总线监视器，用于监视总线的运行。该总线采用指令应答方式实现系统通讯，采用冗余通道和奇校验以及相应的错误处理来提高系统通讯的可靠性。

1553B 是总线接口规律和信号特性的标准，它在物理层上对硬件部件所产生的电信号特性作了严格的规定，在数据链路层和网络层对错误监测的方法和指令响应的格式也作了严格的定义。由于 1553B 总线具有极高的可靠性，因而在航空、航天、军事等领域的电子联网系统中得到广泛应用。

1553B 总线采用异步数据传输方式，码速率 1Mbps，即每秒 10^6 位，数据编码采用曼彻斯特 II 型码，差分传输，一般下采用屏蔽双绞线作为传输介质。

一个典型的 1553B 总线硬件系统的拓扑结构见图 3-6。总线本身是一个二冗余的结构，包括总线 A 和总线 B，二者互为冗余备份，所有的总线设备(也称为总线接口单元，即 Bus Interface Unit BIU)BC、RT、MT 都以并联方式共享总线的主线部分，主线与子线之间采用变压器耦合，子线与 1553B 设备之间也采用变压器耦合。总线上只能有一个总线控制器(Bus Controller BC)，不多于 31 远置终端(Romte Terminal RT，某些文献也称其为远程终端或者远置单元)，总线监视器（BM）是可选的，用于监视总线通讯，一般不参与通讯，也可以 RT 兼容方式参与通讯。

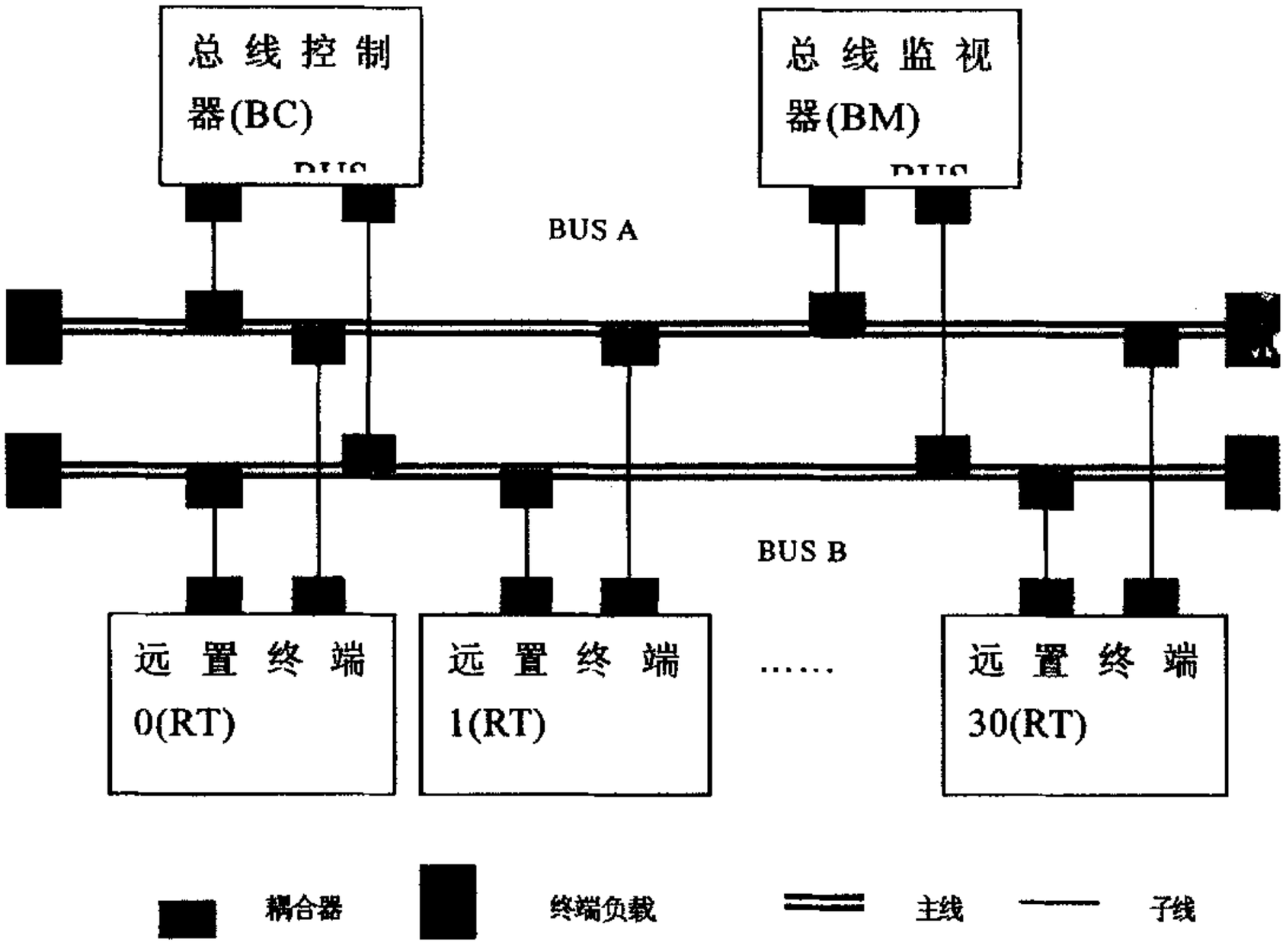

图 3-6 1553B 拓扑结构

3.5 复位电路的选择

复位电路的基本功能是：系统上电时提供复位信号直至系统电源稳定后撤销复位信号。为可靠起见，电源稳定后还要经一定的延时才撤销复位信号，以防电源开关或电源插头分一合过程中引起的抖动而影响复位。复位电路的可靠性是影响系统稳定与可靠的主要因素之一。

在本系统中，由于 ARM 的高速、低功耗、低工作电压导致其噪声容限很低，因此系统对电源的纹波瞬态响应性能、时钟源的稳定度、电源监控可靠性等诸多方面也提出了更高的要求。ARM 监控技术是复杂并且非常重要的。

分立元件实现的监控电路受温度、湿度、压力等外界的影响较大，而且对不同元件影响不一致。分立器件较大的面积，过多、过长的引脚容易引入射频干扰，同时功耗大也使很多应用难以接受。而集成器件能很好的解决此类问题。目前也有不少微处理器中集成监控电路，由于制造成本和工艺技术原因，此类监控电路大多数是用低电压 CMOS 工艺实现的，比起用高电压高线性度的双极工艺制造的专用监控电路，性能还有一段差距。因此，使用专业的监控电路是十分必要的。

本系统选择专用复位芯片 MAX706T。其主要特点是：

a) 复位延迟时间＞200ms

b) 手动复位输入

c) 掉电复位保护

d) 100uA 静态工作电流

e) 集成看门狗定时器

MAX706T 的功能框图如图 3-7 所示：

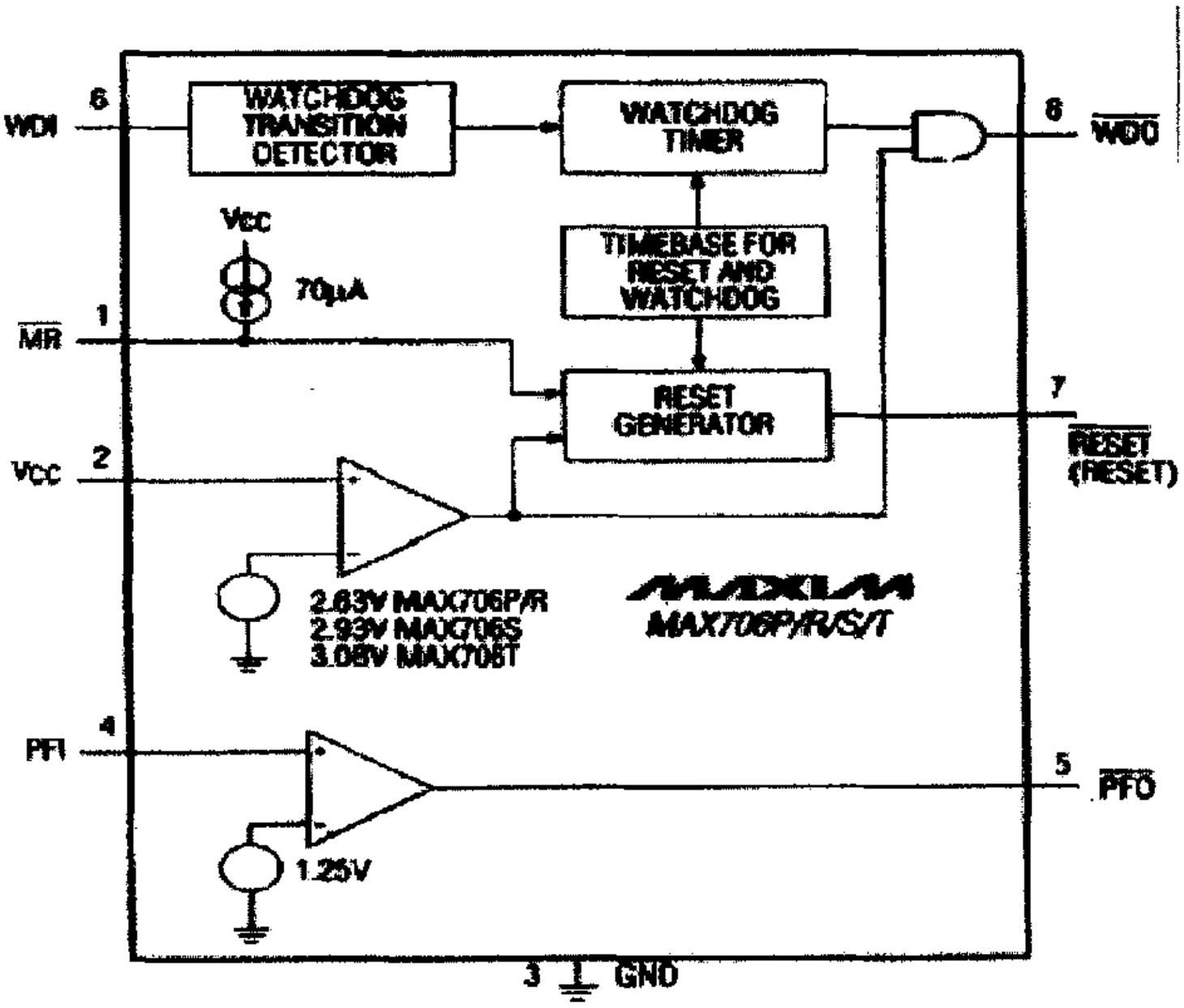

图 3-7　MAX706T 的功能框图

3.6 详细结构图及说明

根据上述内容，我们得到了本系统的详细结构框图，如图 3-8 所示：

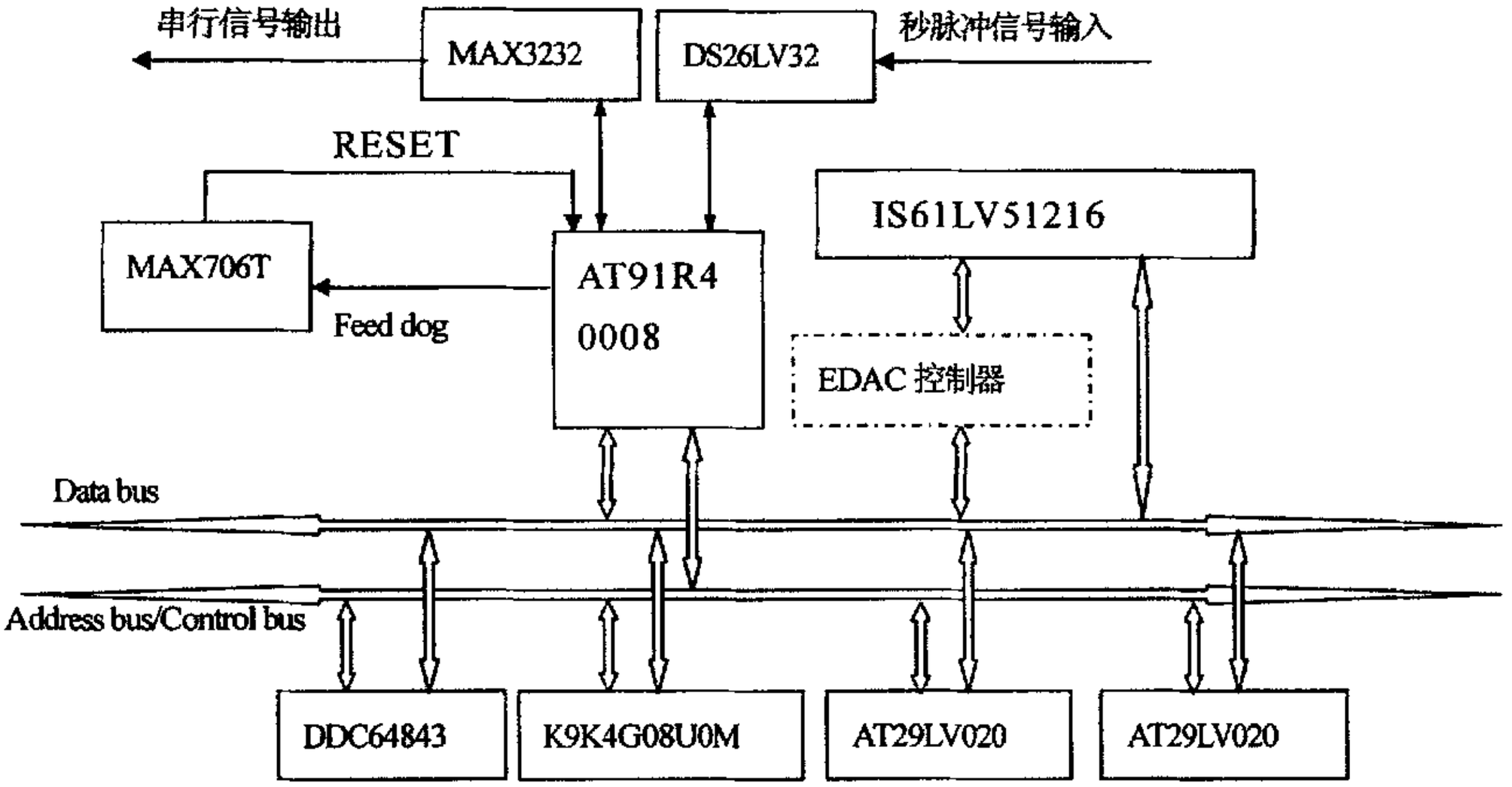

图 3-8 系统详细结构框图

我们根据图 3-8 实现了系统的原理图设计，PCB 设计，并进行了系统调试。

第四章 详细设计

在上一章，我们根据系统结构图和所选择的器件，实现了系统的原理设计，这一章对系统的各个部分做详细说明。

4.1 电源模块设计

本系统的输入电源电压为 5V，使用的电压主要有两种：3.3V 和 1.8V。其中 1.8V 为 CPU 内核电压，其他电路均采用 3.3V 电压，因此需要系统同时提供两种电源。电源模块的设计如下图 4-1 所示：

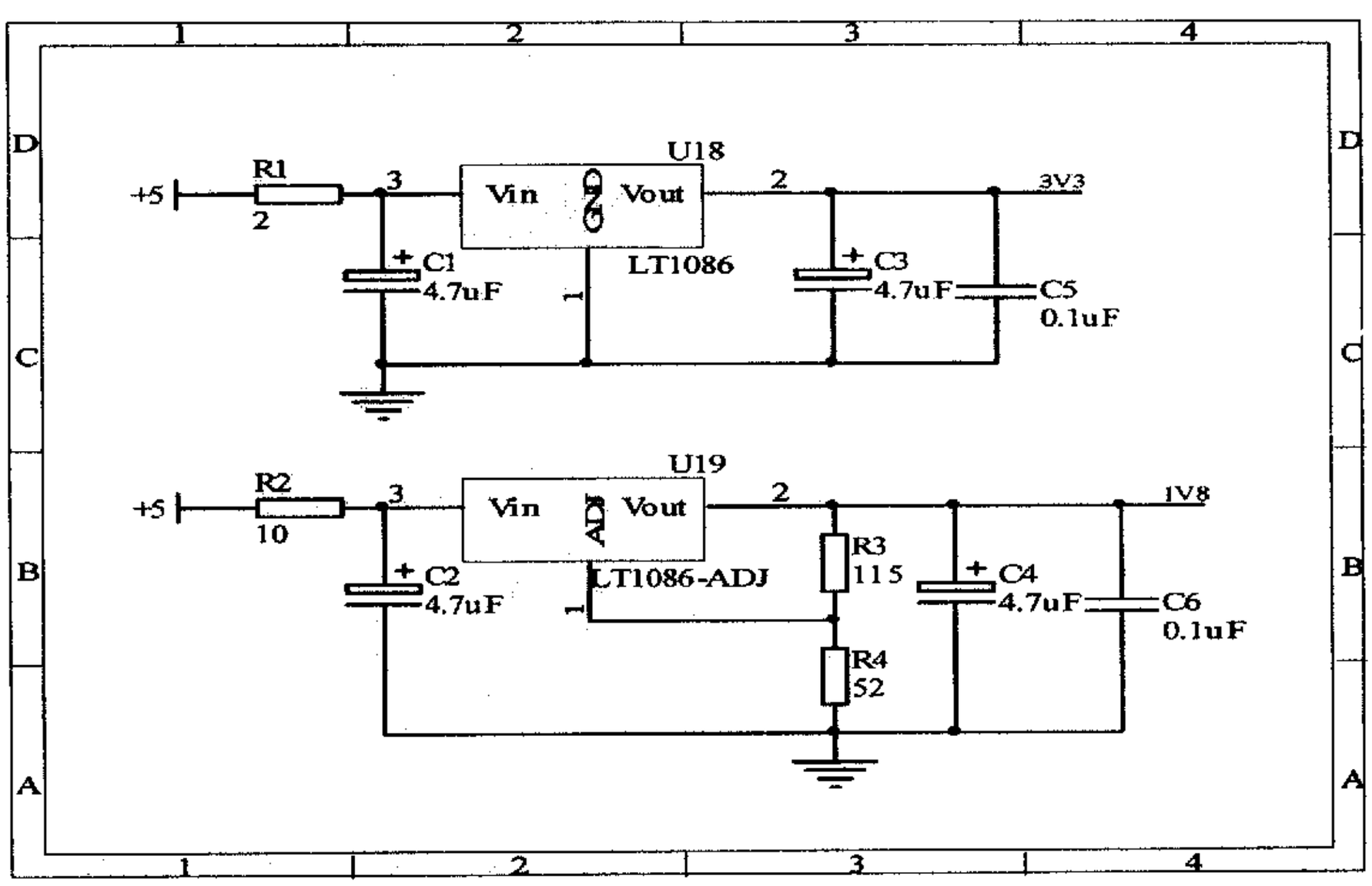

图 4-1 电源模块原理图

LT1086 系列 LDO 的最小落差电压为 1V，典型落差电压为 1.3V。在 5V 输入时，系统的最大工作电流<130mA，因此在 U18、U19 的输入端电阻 R1、R2，起限流保护作用。U18 为固定输出 3.3V 电压，因此直接在 U18 两端接电源滤波电容即可使用。U19 为可调输出 LDO，因此需要加分压电阻 R3、R4。R3、R4 的选择根据如下公式 1 所示：

$$Vout = Vref(1+\frac{R2}{R1}) + Iadj * R2 \qquad (1)$$

其中 V_{ref}=1.25V，I_{adj}=50μA

图中 C3、C4 为钽电解电容、用于去除电源的低频噪声。C5、C6 为高频独石电容，用于去除高频噪声。

4.2 CPU 模块设计

CPU 模块主要包括 CPU、时钟输入、CPU 外围电路及 JTAG 接口三个

部分。如下图 4-2 所示：

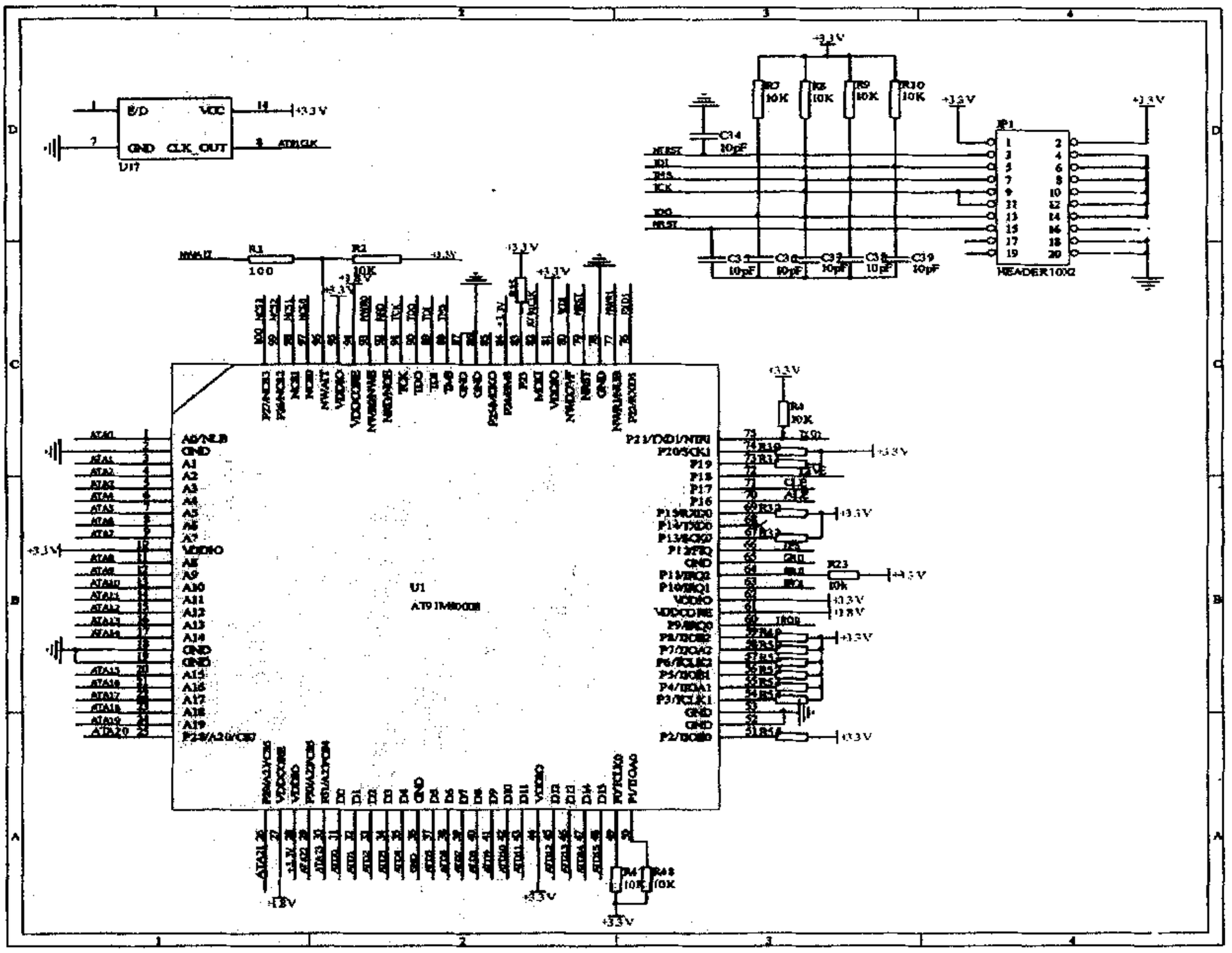

图 4-2 CPU 模块原理图

CPU 采用 ARM7TDMI 内核 AT91R40008,时钟采用 66MHz 有源晶振。

JTAG 接口信号 TDI、TDO、TCK、NRST 分别于 CPU 的对应引脚相连，同时分别接 10KΩ 上拉电阻与 10PF 滤波电容，以改善仿真器输出的信号特性。NTRST 信号为 JTAG 仿真器复位信号，与本地 CPU 无关。在接近 CPU 的 TCK 引脚处，将 TCK 和 RTCK 信号相连，用于为仿真器提供反馈时钟。

CPU 的 75 脚 NTRI 要求用 10KΩ 电阻上拉至 VCC，以保证使用仿真器时，目标板上的本地处理器处于工作状态。

CPU 的 96 脚 NWAIT 信号用 10KΩ 电阻上拉至 VCC，以防止 CPU 在工作时受到外部信号的干扰而产生错误的插入等待。

当CPU与低速外设相连时,CPU与外设之间的握手等待信号从NWAIT端接入，为 CPU 的读写周期提供额外的插入等待信号。

4.3 微处理器监控模块

WD(Watchdog Timer)是嵌入式系统中普遍采用的抗干扰和可靠性保证措施。

AT91R40008 本身支持二级 WD。第一级 WD 利用了 AT91R40008 内部的 WD 单元，通过软件可以将溢出周期编程为 1ms 到 2sec（在 33MHz 系统时钟下）。第二级 WD 采用独立的 CPU 监控器实现，系统中提供了 WD 调用服务．由用户程序调用完成 WD 的设置、启动和更新。WD 启动以后，第一级在设定时间内没有被更新即产生溢出，触发硬中断。同时二级 WD 启动，中断响应过程中回调用户处理程序。并复位一级 WD。如果中断没有得到响应，第二级 WD 将在设定时间后溢出并触发系统复位。

外部微处理器监控模块主要由 MAX706T 及外围电路组成，用来提供系统复位信号及看门狗信号，同时对电源进行监控。电路如下图 4-3 所示

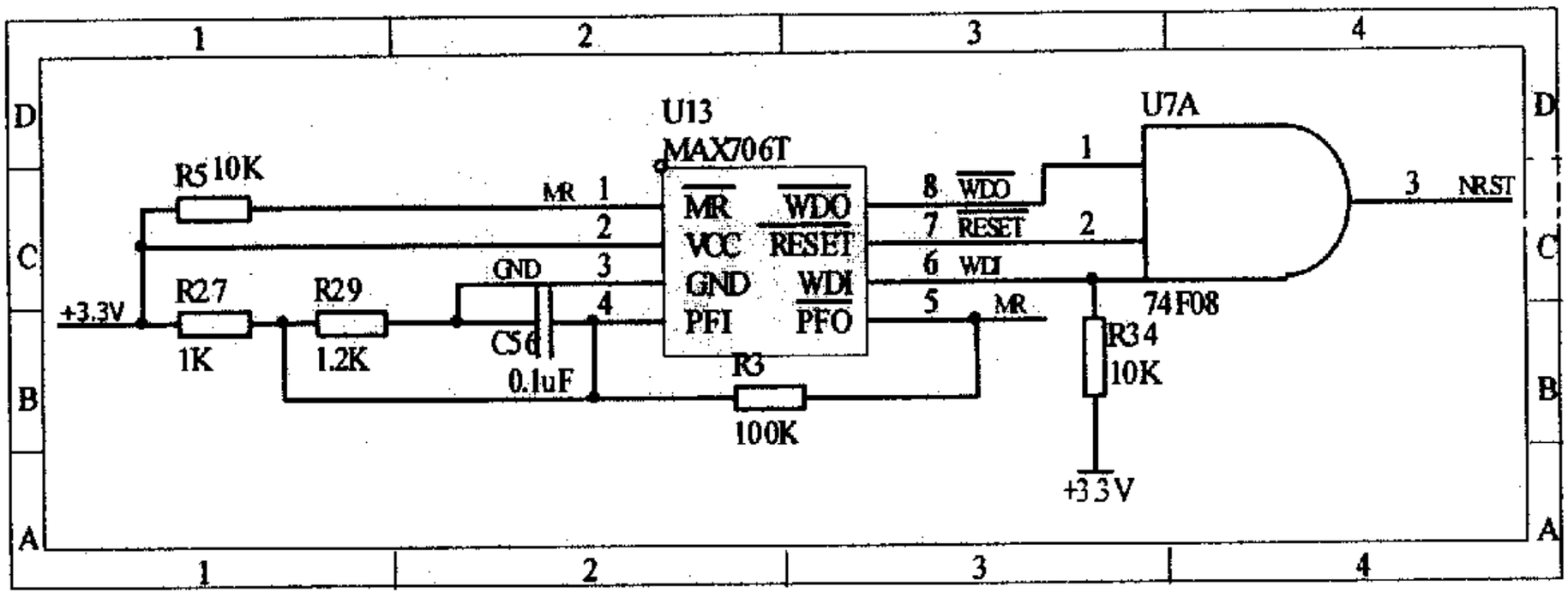

图 4-3　复位模块原理图

MAX706T 为 3.3V 低功耗微处理器监控电路，当上电复位时，当电源电压 VCC 和手动复位端 $\overline{MR}$ 低于 MAX706T 的复位门限电压 3.08V 时，$\overline{RESET}$ 始终保持低电平。它在电源电压达到门限电压后产生一个 200ms 的低电平脉冲，以确保系统在电源稳定之后复位，以减少因电源干扰而引起异常复位的概率。MAX706T 的看门狗响应时间为 1.6s，在这段时间内，WDI 信号必须变化一次，以保证其内部的看门狗比较器复位。否则将在 $\overline{WDO}$ 端产生一个低电平脉冲信号，使系统复位。MAX706T 还具有掉电保护功能。当 PFI 端的输入电压低于 1.25V 时，$\overline{PFO}$ 输出一个低电平同时吸收电流，否则，$\overline{PFO}$ 保持高电平。因此，我们根据 CPU 正常工作的最低电压 2.7V 设计掉电复位电平，以保证当电压低于 2.7V 时，系统能够复位。

4.4 存储器模块

本系统采用 Flash+SRAM 的存储器结构，使用两片 AT29LV020 作为双冗余的程序存储器，用于操作系统和应用程序的存储。

AT29LV020 是 ATMEL 公司生产的带坏块检测和数据保护的 CMOS 型 Flash，第一片 Flash 用于存储底层操作系统和应用程序，第二片 Flash 用于应用程序备份和为在轨程序动态加载的预留空间[5]。这样分配，实现了

应用程序的动态加载和在轨更新。AT91R40008 片内 256KB SRAM 用于存储运行时的程序。

当系统启动时，BOOT 程序自检后加载操作系统到 SRAM 中，然后将控制权交给操作系统。操作系统自检后，根据系统任务状态字，加载相应的应用程序到内存中。此后，程序将在 SRAM 中运行，不需要再去访问 Flash 中的程序。1MB 外部 SRAM 提供了足够的空间用于程序的运行和用户数据的缓存；256MB 外部 Flash 用于存储星务数据。同时，为了适应空间环境的运行，存储器部分拟采用动态纠错技术（EDAC）来解决单粒子翻转问题。电路如图 4-4 所示：

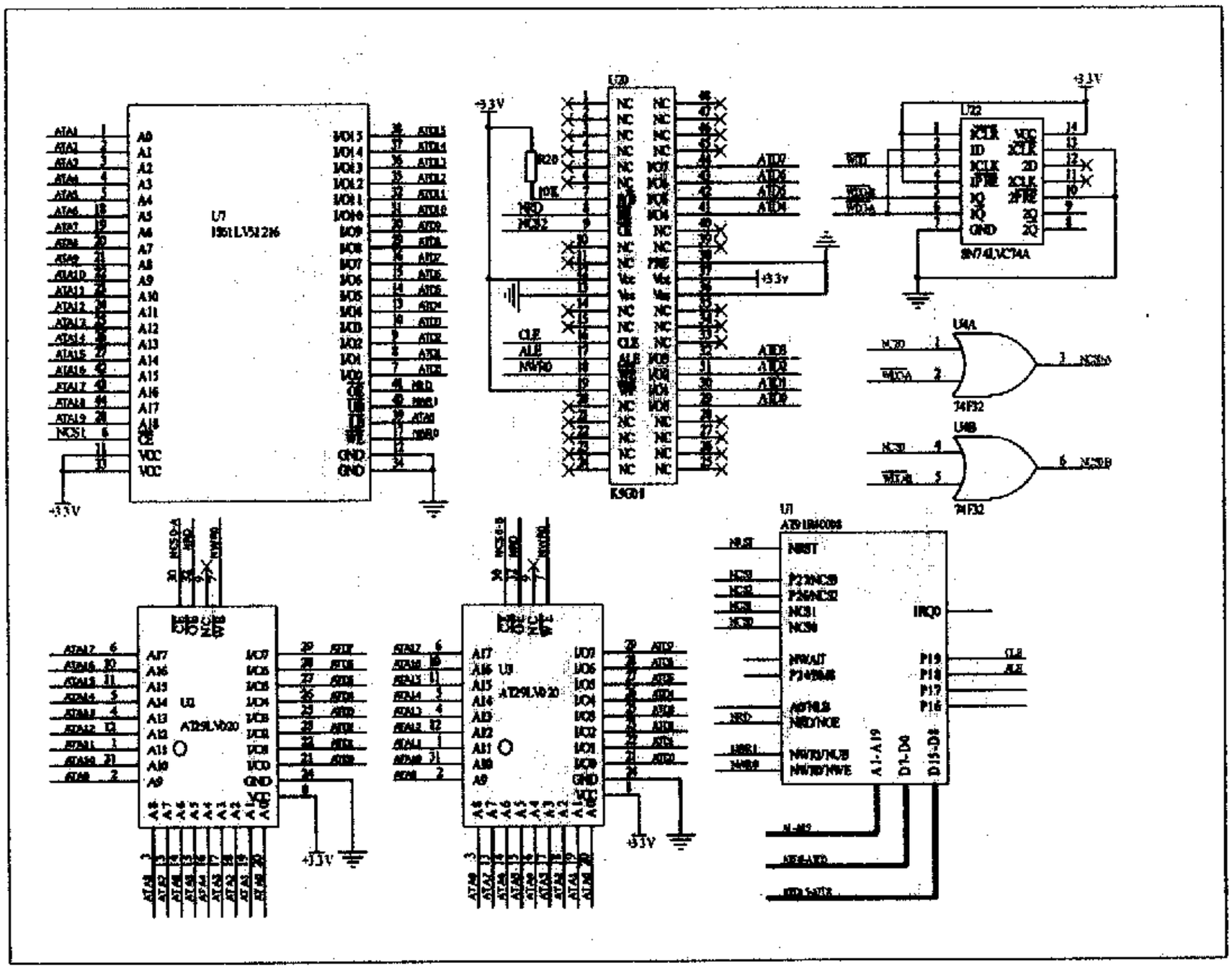

图 4-4 存储器模块原理图

AT29LV020 有 8 根数据线，因此我们在配置 CPU 的启动方式时将 BMS 置“1”，采用外部 8 位方式启动。同时用 NRD/NWR0 作为 Flash 的读/写信号。IS61LV51216 是 512K×16bit SRAM，采用 byte 选择访问方式。ARM 对 16bit 的储存器有两种访问方式，一种是 byte 写访问方式，它支持 2byte 写和一个读信号；另一种是 byte 选择访问方式，它支持对 16 位数据的高 8 位和低 8 位的分别访问或同时 16 位访问方式，其读写信号是完全独立的。这两种访问方式通过填充 EBI_CSR 寄存器中的 BAT 位进行选择。

在采用 byte 选择方式访问 16 位存储器时应注意：

a) A0/NLB 信号用作 NLB 方式，用于使能低 8 位读写操作；

b) NWR1/NUB 信号用作 NUB 方式，用于使能高 8 位的读写操作；

c) NWR0/NEW 信号用作 NEW 方式，用于 8 位或 16 位写操作；

d) NRD/NOE 信号用作 NOE 方式，用于 8 位或 16 位读操作；

采用 byte 选择访问方式访问 16 位存储器的连接图如 4-5 所示：

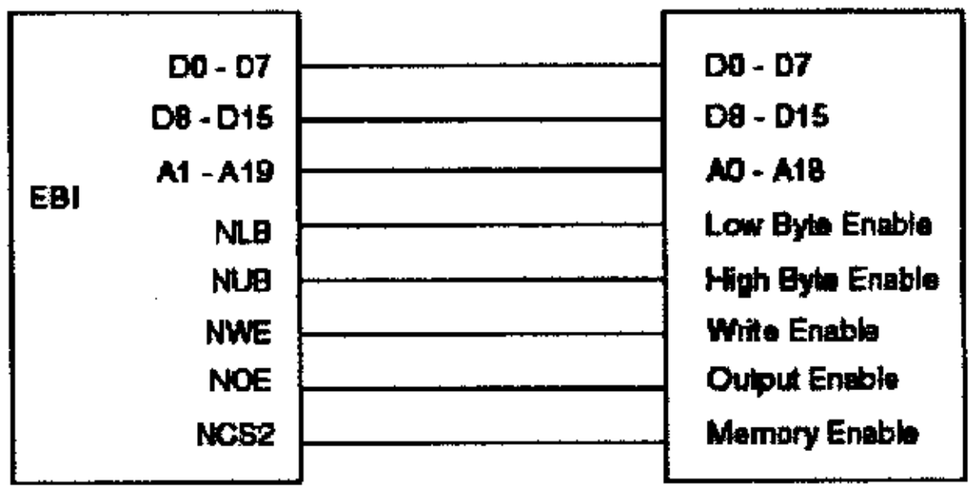

图 4-5　采用 byte 选择访问方式访问 16 位存储器

4.5 1553 模块设计

本系统选用 DDC 公司的 DDC64843 芯片作为 1553B 总线接口协议芯片。

根据初始化参数的不同，DDC64843 芯片可分别工作在 BC、RT、MT 三种不同的工作模式，自动完成总线通信，并在通讯的消息结束或出现异常时发出中断服务请求。CPU 根据中断服务请求对通讯进行控制和管理。

DDC64843 采用变压器耦合方式连接，以保证总线的隔离。DDC64843 与 ARM 有多种连接方式，在本系统中采用 16 BIT BUFFERED 模式。这种方式是 DDC64843 最普通的连接方式，它与 ARM 之间通过 14 个控制信号、16 位数据总线、17 位地址总线直接相连，这样为处理器提供一个直接的、共享的 RAM 接口。电路如图 4-6 所示：

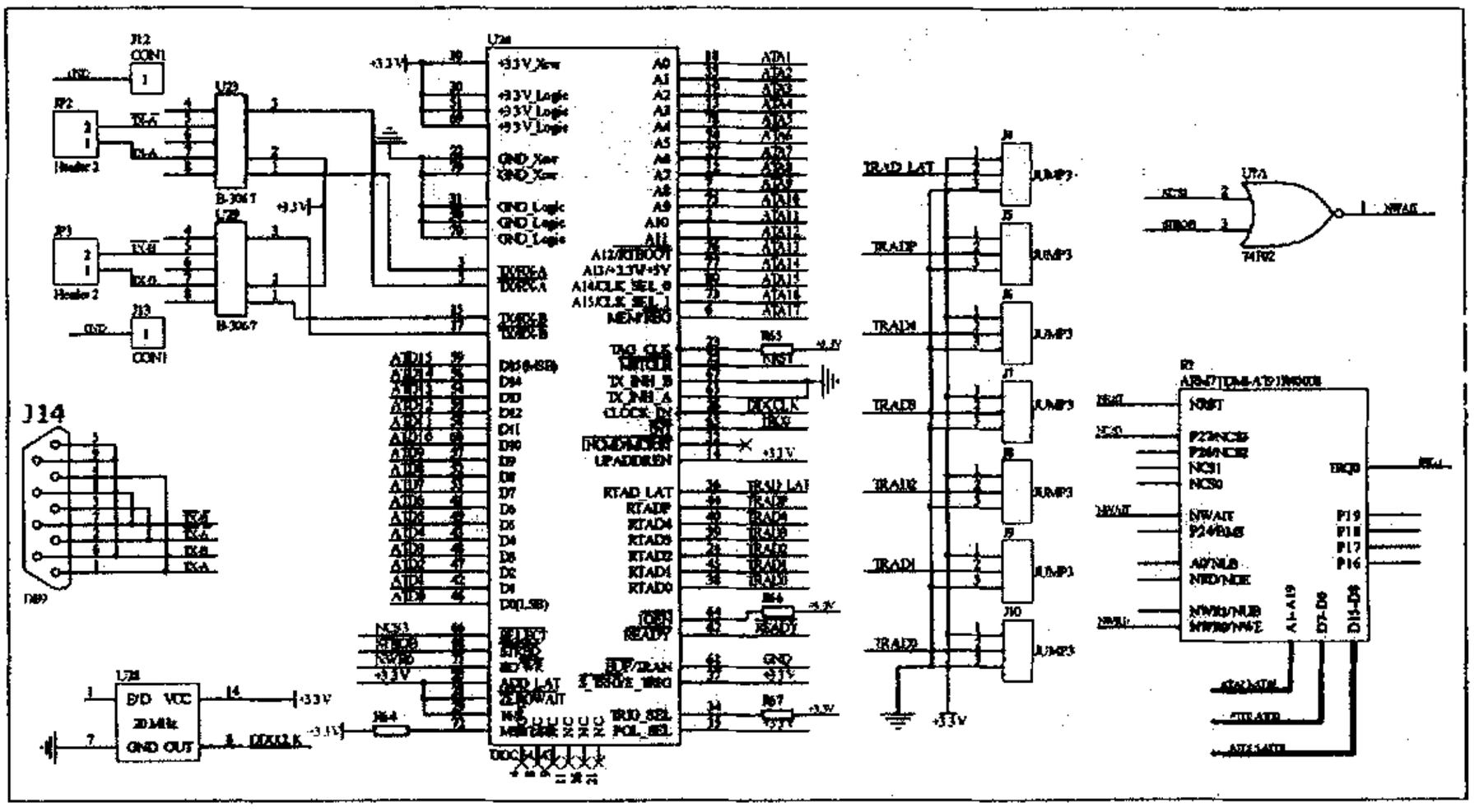

图 4-6　64843 的 16 BIT BUFFER 连接模式

DDC64843 可以通过不同的配置方式在 10、12、16 或 20MHz 时钟频率下工作。本系统中通过指定#6 配置寄存器中的值来选择不同的工作频率。由于 CPU 工作在 66MHz，DDC64843 相对 CPU 来说是个低速设备，因此必须在 DDC64843 与 CPU 之间增加握手信号。这里，将 DDC64843 的片选信号和 STROB 信号“或”之后送入 CPU 的 NWAIT 端，这样可以为 CPU 读写 64843 时插入额外的等待周期。

当 64843 工作在 RT 状态时，需要通过配置不同的 RT 地址来保证其在系统中有唯一的地址。与 64843 对应的变压器为 LVB-3067，每一个 LVB-4203 对应一个 1553 通道，耦合系数为 1：2.7。

第五章 系统调试及集成

本章通过介绍对 AT91R40008 的调试过程，详细说明了 AT91R40008 的启动过程和内存映射方法，并且介绍了通过 JTAG 接口对 FLASH AT29LV020 编程的过程。

5.1 ARM 系统调试方法简介

开发嵌入式系统时，选择合适的开发工具可以加快开发进度，节省开发成本。因此一套含有编辑软件、编译软件、汇编软件、链接软件、调试软件、工程管理及函数库的集成开发环境（IDE）是必不可少的。使用集成开发环境开发基于 ARM 的应用软件，包括编辑、编译、汇编、链接等工作全部在 PC 机上即可完成，调试工作则需要配合其他的模块或产品方可完成，目前常见的调试方法有以下几种：

1、指令集模拟器

部分集成开发环境提供了指令集模拟器，可方便用户在 PC 机上完成一部分简单的调试工作，但是由于指令集模拟器与真实的硬件环境相差很大，因此即使用户使用指令集模拟器调试通过的程序也有可能无法在真实的硬件环境下运行，用户最终必须在硬件平台上完成整个应用的开发。

2、驻留监控软件

驻留监控软件（Resident Monitors）是一段运行在目标板上的程序，集成开发环境中的调试软件通过以太网口、并行端口、串行端口等通讯端口与驻留监控软件进行交互，由调试软件发布命令通知驻留监控软件控制程序的执行、读写存储器、读写寄存器、设置断点等。驻留监控软件是一种比较低廉有效的调试方式，不需要任何其他的硬件调试和仿真设备。ARM 公司的 Angel 就是该类软件，大部分嵌入式实时操作系统也是采用该类软件进行调试，不同的是在嵌入式实时操作系统中，驻留监控软件是作为操作系统的一个任务存在的。

驻留监控软件的不便之处在于它对硬件设备的要求比较高，一般在硬件稳定之后才能进行应用软件的开发，同时它占用目标板上的一部分资源，而且不能对程序的全速运行进行完全仿真，所以对系统实时性要求严格的情况不是很适合。

3、JTAG 仿真器

JTAG 仿真器也称为 JTAG 调试器，是通过 ARM 芯片的 JTAG 边界扫描端口进行调试的设备。JTAG 仿真器比较便宜，连接比较方便，通过现有的 JTAG 边界扫描端口直接与 ARM CPU 内核通信，属于完全非插入式(即不使用片上资源)调试，它无需目标存储器，不占用目标系统的任何端

口，而这些是驻留监控软件所必需的。另外，由于 JTAG 调试的目标程序是在目标板上执行，仿真更接近于目标硬件，因此，许多接口问题，如高频操作限制、电线长度的限制等被最小化了。使用集成开发环境配合 JTAG 仿真器进行开发是目前采用最多的一种调试方式。

4、在线仿真器

在线仿真器使用仿真头完全取代目标板上的 CPU，可以完全仿真 ARM 芯片的行为，提供更加深入的调试功能。但这类仿真器为了能够全速仿真时钟速度高于 100MHz 的处理器，通常必须采用极其复杂的设计和工艺，因而其价格比较昂贵。在线仿真器通常用在 ARM 的硬件开发中，在软件的开发中较少使用，其昂贵的价格也是在线仿真器难以普及的因素。

5.2 系统调试的软、硬件环境介绍

5.2.1 硬件仿真器介绍

本系统硬件调试工具使用 ARM STAR JTAG 仿真器。ARM STAR 是用于 ARM 处理器内核软件调试的专用工具，符合 IEEE 1149.1 规范，与 ARM Multi-ICE 完全兼容。它可以很好的与 ADS1.2 集成开发环境相结合，支持全线 ARM 处理器内核，可以满足更多用户对 ARM 处理器内核软件的开发、调试需求。

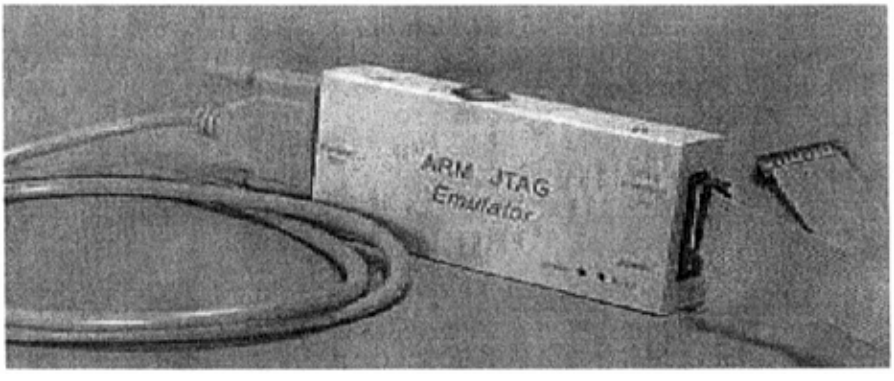

图 5-1 ARM STAR 仿真器

ARM STAR 主要有以下特征：

a) 支持所有内建 Embedded-ICE 逻辑单元的 ARM 处理器
b) 支持多个内核系统
c) 连接简便，兼容不同电压的目标系统
d) 除 JTAG 扫描链外，不占用目标板上的其它任何资源
e) 支持实时硬件断点
f) 用户可通过 JTAG 修改寄存器、存储器
g) 支持程序下载及实时调试

h) 通信速度快，最高可达 10Mbps
i) JTAG 速度可配置，以满足不同调试对象的需求
j) 支持所有符合 RDI 1.50 或 RDI 1.51 规范的调试工具软件
k) 提供丰富的例程和使用说明
l) 支持多种目标平台，如 ATMEL、Samsung、Intel、Philips、Sharp、Cirrus Logic 等
m) 体积轻巧、性能稳定
n) 支持的操作系统有：
o) Windows 95/98/NT/2000/ME/XP
p) X86 RedHat Linux 6.2/7.1/7.2

ARM STAR 仿真器与系统硬件连接如图 5-2 所示：

ARM_STAR 与目标板连接示意图

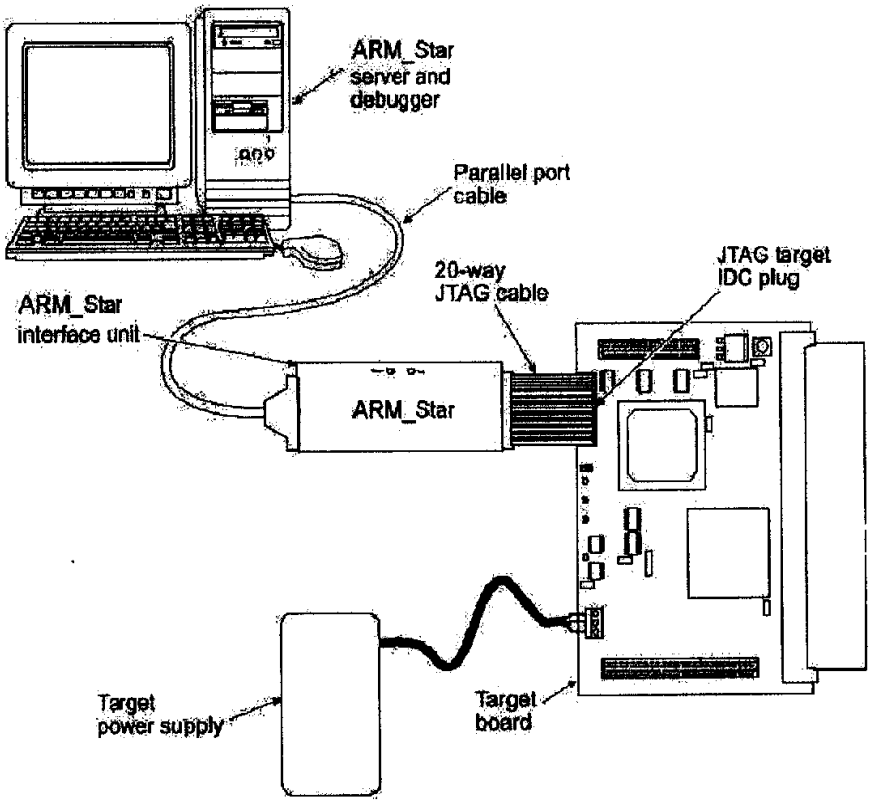

图 5-2 ARM STAR 与目标板连接示意图

在实验中应注意，因为 ARM STAR 仿真器的电源来自于目标板（约 150 毫安），所以目标板的电源设计应该考虑一定的裕量。

5.2.2 软件开发工具介绍

ARM 系统调试软件使用 ARM Developer Suite™ ADS，它是 ARM 公司开发的集成实时开发软件工具包，编译器生成的代码密度和执行速度优异，可快速低价地创建 ARM 结构应用。ADS 包括三种调试器：ARM eXtended Debugger（AXD）、向下兼容的 ARM Debugger for Windows/ARM Debugger for UNIX 和 ARM 符号调试器。其中，AXD 不仅拥有低版本 ARM 调试器的所有功能，还新添了图形用户界面，更方便的视窗管理、数据显示、格式化和编辑，以及全套的命令行界面。

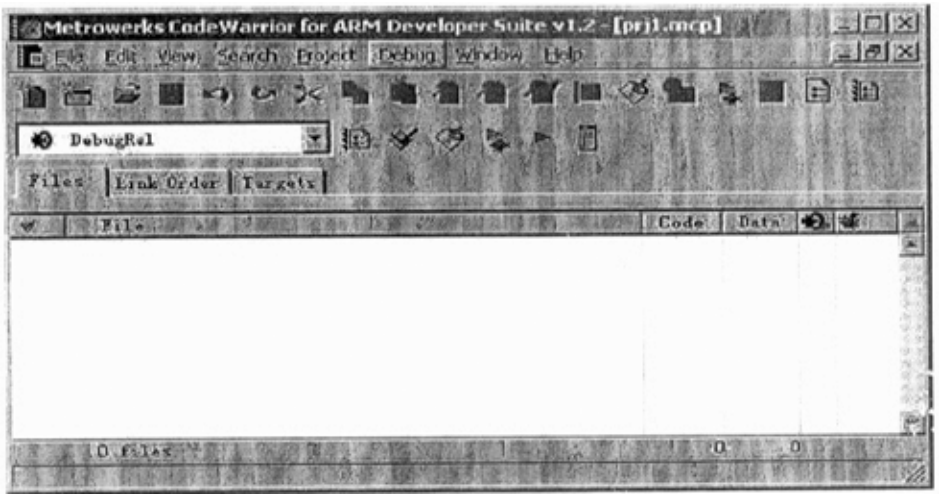

图 5-3 ADS 图形界面

ARM 的 Real-Time Trace™ 和 RealMonitor 均为重要的实时调试工具，能够提供特殊软件调试功能，可运行于带深度嵌入处理器内核的高集成系统芯片 SoC。ARM 的 Real-Time Trace 产品包括跟踪调试工具、MultiTrace 、嵌入式跟踪宏单元和 Multi-ICE。ARM 的 RealMonitor 包括 RMTarget™、RMHost™，是 ARM Developer Suite (ADS)的补充。

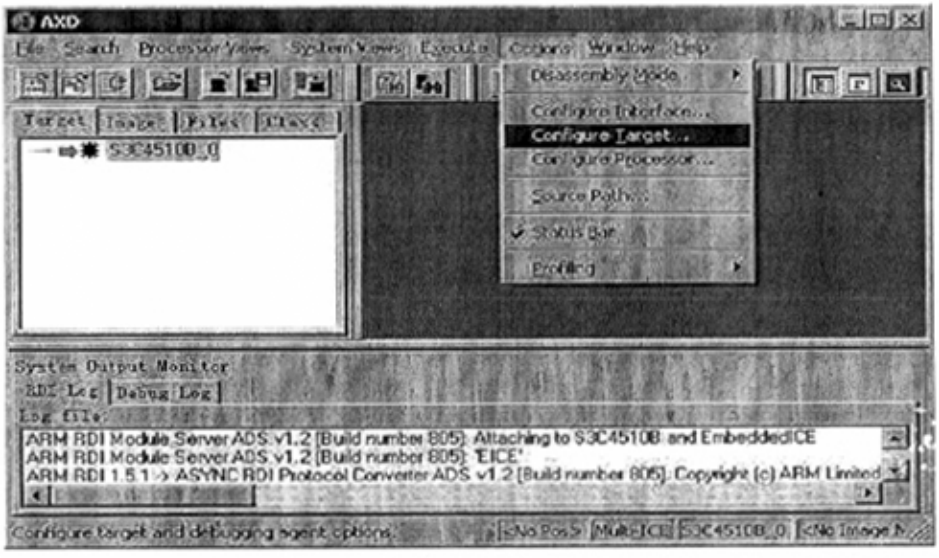

图 5-4　AXD 图形界面

5.3 SRAM 及 FLASH 调试

对 SRAM 及 FLASH 的调试目的是实现 CPU 的引导加载及对 FLASH 的编程。引导加载程序提供从指定位置更新和选择代码，同时在另一位置执行代码的途径，即使用同一处理器实现两种功能。微处理器包含引导加载功能，在构建和调试新应用程序时有更大的灵活性。

引导加载程序可以根据系统和应用程序的要求进行扩充，从而提供更为复杂的选项，例如在最终跳转到主应用程序前，进行的系统诊断、内存和系统初始化、程序验证和加载。

在嵌入式环境中，引导加载过程包含引导加载程序和 Flash 编程实用程序，Flash 编程实用程序有时集成在处理器中。引导加载程序是一段代码，它通过确定处理器使用的代码、代码所在位置，管理整个引导过程。Flash 编程实用程序编写会成为引导程序的代码。不管系统硬件复杂性如何，在嵌入式环境中设计和实现引导加载程序都是一个比较复杂的过程。

在实际应用中，代码必须在正确的内存位置执行，同时要自动避开最近擦除的代码段空间。此外，更新的代码必须加载到恰当的内存位置，并正确地写入，才能确保稳定的操作。利用引导加载程序实现这些功能，嵌入式处理器可以快速运行任意数量的配置，从而方便各个层次的开发工作。典型的处理器运行的配置环境如图 5-5 所示：

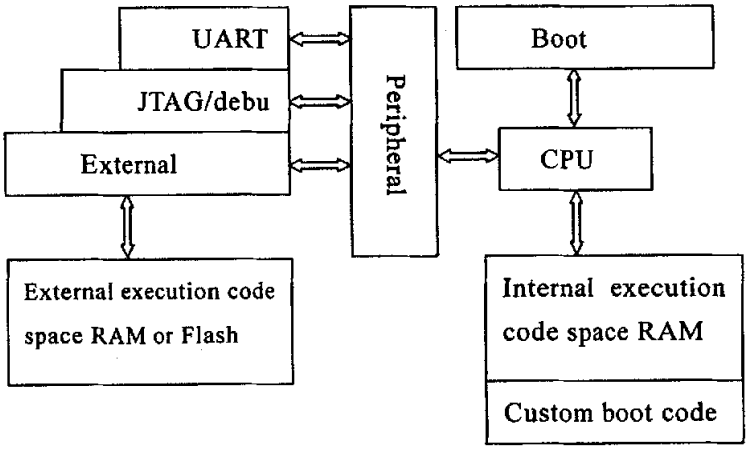

图 5-5 典型 CPU 配置框图

该系统有四个主要器件：CPU 内核（用于执行代码）；一段内存（RAM 或 FLASH，CPU 在其中执行引导加载程序）；一段代码空间（RAM 或 Flash）（用于重新编程）以及一个外设、UART 或外部存储器接口（用于加载更新代码）。

引导加载后的运行及系统存储器地址的分配是通过 Remap 实现的。当 ARM 处理器复位之后，进入引导模式，初始化用户接口寄存器，使其变为缺省直。除了 PC 为 0 以外，ARM 核内部的其它寄存器无定义。处理器从 0x0 取址。因此，必须保证系统上电时，0x0 处有指令可以执行。所以，上电的时候，0x0 地址处必定是 ROM 或者 Flash。存储器映射将 BOOT FLASH 放在 0x0，片内 RAM 重定位到 0x00300000。片内 ROM 或片外的 8/16 位 ROM 或 FLASH 映射到 0x00000000（片外 ROM 由 NCS0 选择）。执行重映射命令，即重映射寄存器的 RCB 位写 1，可以使系统退出引导模式，存储器映射返回到常规的映射关系。此时 0 页面处映射为片内 RAM，以存放动态的（可修改的）异常向量表。外部的存储器映射到用户在片选寄存器 0 中定义的地址。值得注意的是：如果这个寄存器没有写入相应的值，将会产生内部的冲突。如果要访问外部芯片（即使用 NCS1 到 NCS7），重映射命令是必须的。重映射前后，系统地址变化如下图 5-6 所示：

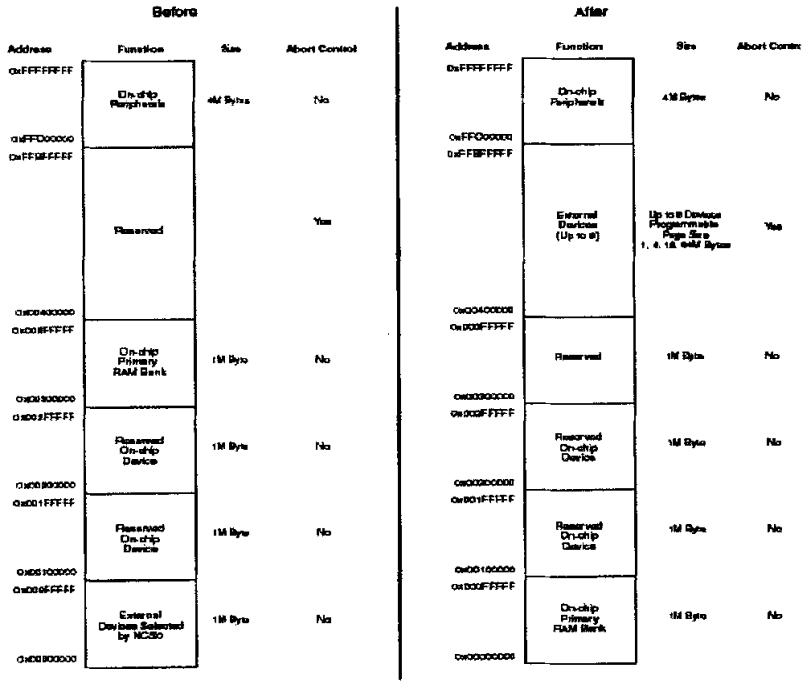

图 5-6 重映射命令执行前后的 AT91R40008 内存变化

有两个不同的方法执行 Remap 命令。第一个是标准的。将 BOOT FLASH 中的程序拷贝到片内 RAM。然后 Remap 能安全地执行。第二个是利用 ARM 内核的流水线技术，它允许在不拷贝任何代码的情况下执行 Remap。这个例子包含在 AT91 库\Library\Init\in_reset.s 中。

```
- 拷贝存储控制器的映像
ldr   r10 , PtInitTableEBI
mov   r10 , r10 , LSL  #12
mov   r10 , r10 , LSR  #12
- 装载要跳转的地址
ldr   r12 , PtInitRemap
- 拷贝片选寄存器映像到存储控制器和命令 remap
ldmia   r10 !,{ r0-r9 , r11 }
stmia   r11 !,{ r0-r9 }
- 在它的新地址跳到 ROM
mov pc, r12
PtInitTableEBI
DCD InitTableEBI
PtInitRemap
```

DCD InitRemap

PtInitVector

DCD __main

- 程序启动

InitRemap

在这种情况中“MOV pc , r12”指令在前面的“STMIA”执行以前，映射到地址 0x0 的外部引导 Flash 中取指。然后“MOV pc , lr”执行并且这跳到外部的引导区的连接地址。

在系统调试中，我们采用 JTAG 接口进行程序加载与运行。ARM STAR 支持 JTAG 下载，不需要系统在启动时对外设端口进行配置，因此简化了调试过程。

应用 ARM STAR 仿真器调试过程如下：系统上电后首先运行仿真器驱动程序，选择 ARM7TDMI 内核 CPU，保证目标板 CPU 与仿真器的通讯正常；在 ADS 的调试器 AXD 中选择 ARM STAR 的驱动 Multi-ICE.dll 文件，绑定调试器与仿真器；最后在 AXD 环境中打开 File 中的 Load Image 装载源程序后，在程序窗口中就可以看到源代码，并进行调试，程序运行正常，可以在板上观察到相应结果。

在这里需要说明的是，由于各个系统的配置 FLASH 的型号不同，需要自己编写对应型号 FLASH 的编程程序，我们在实验中完成了 AT29LV020 的下载程序，源文件见附录 1。

第六章 总结与改进

随着航天事业的发展，现代卫星出现了小型化、低成本的趋势，卫星平台设计也从大型化逐渐转向了小型化。现代卫星的功能越来越强，对完成星务管理和有效载荷控制的星载计算机也提出了更高的要求，导致其软硬件系统日趋复杂。传统的星载计算机软硬件设计方法、开发手段已逐渐不适应这种要求，这就迫切需要我们采用新的开发手段研制新型的高性能星载计算机。

根据这种要求，我们开发了基于 ARM 处理器 AT9R40008 的高性能星载计算机。本课题作为对小卫星平台数据处理单元（DHU）的重要组成部分，对卫星任务的完成有着重大的影响。本文给出了基于 AR91R40008 处理器的星载计算机的设计方案，详细论证了方案的可行性与技术细节，以及相应的设计所采用的一些关键技术，最后给出了系统的调试过程。

根据以上过程，我们初步实现了基于 AT91R40008 的星载计算机的设计，证明了系统原理的可行性，初步实现了操作系统的移植。分析和调试结果表明，该系统满足任务需求，为我国星载计算机及嵌入式操作系统的研制开发开辟了新途径，并为星载计算机进行高可靠冗余设计与并行处理技术奠定了良好的基础。

基于 ARM 的高性能星载计算机系统的硬件体系结构以及基于任务需求开发的软件系统，符合新一代航天型号任务“快、好、省”的设计原则，同时该系统具有较强的容错能力及软件上载功能。

值得指出的是，本文开发的星载计算机系统尚在如下几个方面有待改进：

第一、由于在设计过程中软件复杂性的提高，使得软件的大小超过了预期的设计，不得不在原来两片 BOOT Flash 的备份设计的基础上增加了两片相互备份的程序存储器。从而增加了器件数量，增大了设计的复杂性。在下一阶段的设计中，可以将 AT29LV020 换成容量较大的 FLASH 型号，如 AT29LV040A，以减少芯片的数量；

第二、为了添加 EDAC 电路及减少分立元件的个数，我们可以采用现场可编程逻辑门阵列（FPGA）器件来实现多个分立门电路的

功能。同时，由于采用可编程器件，我们可以对其进行在系统编程，从而实现系统重构，增加了系统的可靠性和灵活性；

第三、由于目前芯片技术的飞速发展，ARM 系列 CPU 已发展出 ARM9、ARM10、ARM11 的一系列性能更高的 CPU，我们可以在采用模块化设计的基础上，简单的改变 CPU 及其外围电路设计而保留其他单元模块，从而形成不同处理能力的系列产品。

参考文献

1. ARM Limited，ARM Architecture Reference Manual，[EB/OL] .http://www.atmel.com 2000
2. ARM Limited，ARM7TDMI(rev 4) Technical Reference Manual，[EB/OL] .http://www.atmel.com，2001
3. Atmel Corporation.AT91EB40A Evalution Board User Guide，[EB/OL] .http://www.atmel.com，2002
4. Atmel Corporation.AT91R40008 Summary，[EB/OL] .http://www.atmel.com，2002
5. Atmel Corporation.AT91R40008 Electrical Characteristics，[EB/OL] .http://www.atmel.com，2002
6. Atmel Corporation .AT29LV020 datasheet [EB/OL].http://www.atmel.com, 0565C-FLASH-05/02 Xm.
7. IEEE Standard Test Access Port and Boundary-Scan Architecture [EB/OL] http://www.IEEE.com, 2001
8. 何希顺、张跃、何荣森 嵌入式系统中的 JTAG 接口编程技术 [J] 电子技术应用 2001.12：9-12
9. 陈晔、辛洪兵 一种用于小卫星数据管理与控制的新体系结构 [J] 光机电信息 1997.11：5-9
10. 罗先武 星上数据管理中的实时多任务操作系统设计 [J] 中国空间科学技术 1997.6：15-20
11. 朱虹、王海燕 一种星载软件在轨编程功能的设计和实现技术 [J] 上海航天 2004.1：26-31
12. 卢东昕、洪炳熔、张宇 基于CCSDS的小卫星系统软件重用技术研究 [J] 高技术通信 2000.12：33-35
13. 陈明明、李忠、马永武 基于 Intel386EX 的数字式保护硬件设计 [J] 继电器 2003.05：46-48
14. 许盛柯，华更新，郭树玲. 单板双机嵌入式 486 容错子系统的设计［J］控制工程 2002.4：31-35.
15. 仉俊蝉、洪炳镑、乔永强 基于软件的星载计算机系统故障注入方法的研究 [J] 计算机工程与应用 2003.33：28-29
16. 赵勐、陈朔鹰、马忠梅 Armboot 在 EV40 评估板上的移植 [J] 单片机与嵌入式系统应用 2004.1：43-45
17. 王京林、苏洁、詹横空基于 ARM7 嵌入式内核的嵌入式应用程序调适技术 [J] 电子工程师 2000.12：9-11
18. 嵌入式系统开发面临的问题与集成开发环境的应用，[EB/OL] http://www.eetchina.com/ART_8800365272_480101_79781d57.htm，2005
19. Mike Claassen、Mark Morton 嵌入式引导加载技术 [EB/OL] http://www.memec.com/，2005

20. 广州周立功单片机发展有限公司 单片机复位电路的可靠性设计，[EB/OL]，http://www.zlgmcu.com，2003
21. 马忠梅、徐英慧等.AT91 系列 ARM 核微控制器结构与开发.[M] 北京：北京航空航天大学出版社，2003
22. 杜春雷.ARM 体系结构与编程.[M] 北京：清华大学出版社，2003
23. 徐福祥、林宝华、侯深渊等编著 卫星工程概论［M］北京：宇航出版社 2003
24. 褚桂柏 航天技术概论［M］北京：中国宇航出版社 2002
25. 于金培，杨根庆，梁旭文. 现代小卫星技术与应用［M］.上海：上海科学普及出版社，2004.3.
26. 李芳华. 星载软件可靠性设计方法［J］.上海航天. 2003.3：24-27.
27. 周立功. ARM 微控制器基础与实践［M］. 北京：北京航空航天大学出版社，2003.11.
28. 廖明宏，耿云海，吴翔龙，程光明. 小卫星资轨控与星务管理一体化设计［J］.中国空间科学技术. 2001,21(2).：31-36.
29. James R. Wertz　Wliey J. Larson 航天任务的分析与设计. [M] 北京：航空工业出版社，1992
30. 李东生、张勇等.Protel 99SE 电路设计技术入门与应用.［M］北京：电子工业出版社，2002
31. 夏宇闻 复杂数字电路与系统的 Verilog HDL 设计技术［M］北京：北京航空航天大学出版社，2003.11.
32. 霍华德·约翰逊 高速数字设计 [M] 北京：电子工业出版社，2004.05

致　谢

首先感谢我的导师林宝军老师，在三年的研究生学习生涯中，你严谨的治学态度，理性的生活方式以及在学习和生活中给予我的教诲和帮助将令我终身受益。

感谢我的指导老师张善从博士，在整个星载计算机的研制过程中，从始至终的给与我指导和支持。感谢徐志瀚、张勇、董继红等老师，是你们耐心细致地为我答疑解惑、提供各种帮助。感谢总体室的各位老师和学友们的帮助。没有各位老师的指导和帮助，我是根本无法完成工作的。是你们用无私的双手、热情的鼓励陪伴我度过了这三年的学习和工作生活，在此表示衷心的感谢！

感谢赵清同学在论文修改和校对方面所作的大量工作，没有她的帮助，我很难在短时间内完成论文的写作。

感谢张作和、李艳秋老师，感谢你们在三年的研究生学习和生活中对我们无微不至的关怀和照顾。

附　录1

AT29LV020 的编程流程图如下所示：

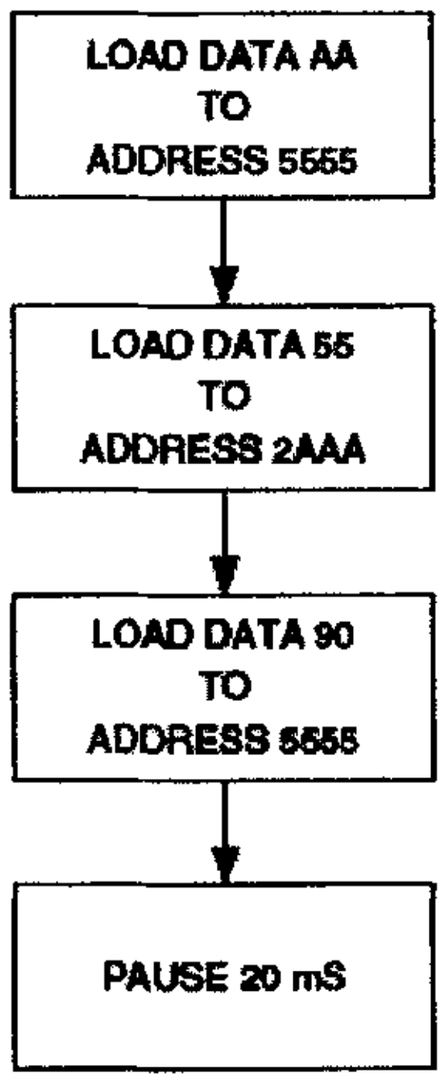

图 1　读产品 ID 号

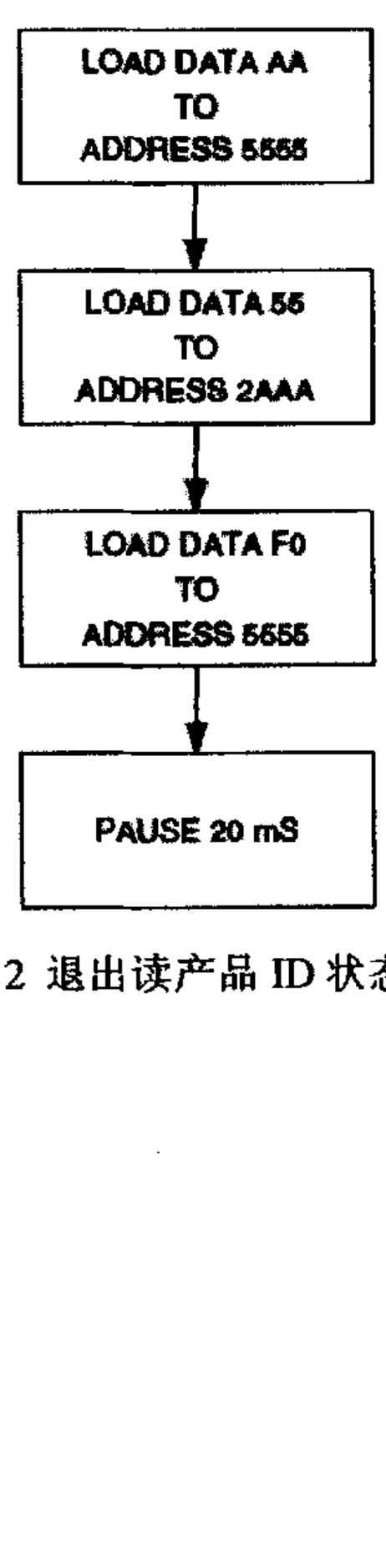

图 2 退出读产品 ID 状态

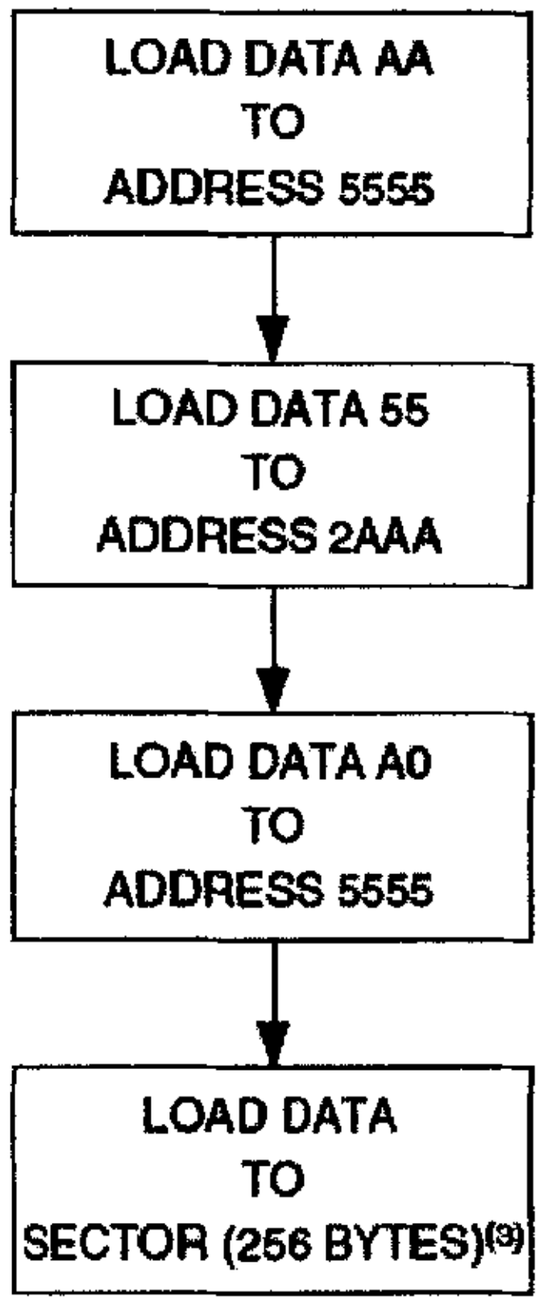

图 3 写 FLASH 流程

对 AT29LV020 编程的主要程序如下：

```
/* Include Standard c Libraries to allow stand alone compiling and operation */
#include <stdio.h>
#include <stdlib.h>
#include "flash_at49.h"
/* Target Identification */

void delay ( void )
{
      unsigned int       i ,j;
//* loop delay
      for ( i = 0 ;(i < DELAY_ERASE29 );i++ )
            {
            for ( j = 0 ;(j < 1 );j++ )
            ;
          }
}

//*-----------------------------------------------------------------------
//* Function Name       : write_flash          //向 flash 中写数据
//* Object                    : write a byte in flash
```

```
//*
//* Input Parameters       : flash_word *base_addr : Flash base address
//*                                flash_word *load_addr : Flash data address
//*                                flash_word data          : Data value
//*
//* Output Parameters     : TRUE if data has been written correctly, else FALSE
//*
//* Functions called       : wait_flash_ready
//*-----------------------------------------------------------------------
int write_flash ( flash_word *base_addr, flash_word *load_addr, flash_word data )
{
     *load_addr = data ;                                    //在 150us 内要写完一个 sector,一次写一个
byte
     return TRUE;
}

//*-----------------------------------------------------------------------
//* Function Name          : download_file_to_flash
//* Object                 : Read data from file and write it into flash memory     //从文件写入
memery
//* Input Parameters       : <addr_base> base flash address
//*                              <addr_load> address to load
//*
//* Output Parameters      : TRUE or FALSE
//*-----------------------------------------------------------------------
int download_file_to_flash ( FILE *image, const FlashAt49BVDef *flash, flash_word *addr_base,
flash_word *addr_load )
{
     flash_word    *addr_prg_sector = addr_base ;        //addr_prg_sector=0x1000000
     int           block = 0 ;          //指出一片 FLASH 中有多少 sectors
     int           nb_sector = 0 ;          //指出在一片 FLASH 中有多少中不同大小的块
     int           sector_id = 0 ;          //指出现在写的是第几个 sector
     int           sector_found = FALSE ;
     int           dataCount = 0;
     int           kbyteCount = 0;
     int           happyProgramming = TRUE;

     unsigned char    *RAMDataPtr ;
     flash_word       *flashDataPtr;
     //int sectorNum;
     unsigned int sectorSize;
     unsigned int numWordsRead;
     unsigned int wordCount;
     // Find the starting flash sector
```

```
sector_found = FALSE ;
while ( sector_found == FALSE )
{
 // compute the end address + 1 of the current sector
 //addr_prg_sector += (flash->flash_org[block].sector_size/2) ; 以 word 为单位写 flash
     addr_prg_sector += (flash->flash_org[block].sector_size) ;
//0x1000000+256*8/2 （/2）以 WORD 为单位编程   不除 2，以 BYTE 为单位编程
     // If program address lower than this, we start programming in this sector
     if (addr_prg_sector > addr_load )
     {
        // Display First sector to program
        printf ( "First Sector to program:\t%d \n", sector_id ) ;
        // If the first address to program is within a sector
        if ( addr_load > addr_prg_sector )
          {
            // Display Warning : first address within a sector
            printf ( "The first address to program is within a sector.\n" ) ;
            printf ( "First Data of the sector will be lost!\n" ) ;
          }
        sector_found = TRUE ;                          //sector_found=TRUE
        break; // leave the while loop
     }
     else
     {
        // Increment the Sector Identifier
        sector_id ++ ;
        // Increment the sector number
        nb_sector++ ;
        // If last sector in block tested
        if ( nb_sector >= flash->flash_org[block].sector_number ) {
             // Re-initialize sector number in block
             nb_sector = 0 ;
             // Increment block number
             block ++ ;
             // If last block tested
             if ( block >= flash->flash_block_nb ) {
                  // Display Error and Return False
                  printf ( "Error : Programming address is not within the Flash device's
address space\n" ) ;
                  happyProgramming = FALSE;
                  break; // leave the while loop
             }
        } //if ( nb_sector...
     } //if (( addr_prg_sector
```

```
} //while ( sector_found
/* Loop over each sector within each block, erase it, then program it */
while (happyProgramming == TRUE)
{
 printf ( "Programming current Sector:          %d\n", sector_id ) ;
 // read a whole sector from the file into RAM                    //
sectorSize = flash->flash_org[block].sector_size;
printf ( "The sectorSize is:   %d\n",sectorSize   ) ; //验证块的大小得到是正确的结果了。
numWordsRead = fread ( RAMSectorData, 1, sectorSize, image );//看读出来多少个字节
RAMDataPtr = RAMSectorData;
flashDataPtr = addr_load ;
*(addr_base + FLASH_SEQ_ADD1) = FLASH_CODE1 ;
*(addr_base + FLASH_SEQ_ADD2) = FLASH_CODE2 ;
*(addr_base + FLASH_SEQ_ADD1) = WRITE_CODE ;          //发写命令
for (wordCount = 0; wordCount < numWordsRead; wordCount ++)
{
    if ( write_flash( addr_base, flashDataPtr, *RAMDataPtr ) != TRUE )
    {
        happyProgramming = FALSE;
        printf("write flash occore wrong   \n");
        break;
    }
    flashDataPtr++;
        RAMDataPtr++;
    }

    ToggleBit( flashDataPtr );
    delay();
    /* Verify the flash sector we just wrote with the RAM data */
    RAMDataPtr = RAMSectorData;
    flashDataPtr = addr_load ;
    printf( " flashDataPtr is 0x%lx\n", (unsigned long) flashDataPtr);   //检查 addr_load 是
否被改变了。
    for (wordCount = 0; wordCount < numWordsRead; wordCount ++)
    {
        // print a dot for the user
        dataCount += 1;
        if (dataCount%1024 == 0)
        {
            kbyteCount++;
            printf("%d kbytes\n", kbyteCount);          //test kbytCount
            printf(".");
        }
/*校验数据*****************************************************************/
```

```
if (*RAMDataPtr++ != *flashDataPtr++)       //改动
{
   printf ("RAMDataPtr: 0x%lx\n",(unsigned long)RAMDataPtr);
     printf ("*RAMDataPtr: 0x%x\n",*RAMDataPtr);
     printf ("flashDataPtr: 0x%lx\n",(unsigned long)flashDataPtr);
     printf ("*flashDataPtr: 0x%x\n",*flashDataPtr);
     printf ("verify the flash sector error\n");
      happyProgramming = FALSE;
      break;
        } // leave the while loop

}//leave   for loop
addr_load = flashDataPtr;
printf("\n");
if (numWordsRead < sectorSize)       //以 byte 为单位判断文件是否读完
{
      // There is no more data in the file
      break; // leave the while loop
}
// Increment the Sector Identifier
sector_id ++ ;
// Increment the sector number
nb_sector++ ;
// If last sector in block tested
if ( nb_sector >= flash->flash_org[block].sector_number )
{
      // Re-initialize sector number in block
      nb_sector = 0 ;
      // Increment block number
      block ++ ;
      // If last block tested
      if ( block >= flash->flash_block_nb )
      {
            // Display Error and Return False
            printf ( "Error : Binary file too large for flash device\n" ) ;
            happyProgramming = FALSE;
            break; // leave the while loop
                  }
           } //if ( nb_sector...
      } // while (happyProgramm...
      return ( happyProgramming ) ;
}

//*---------------------------------------------------------------------
```

```
//* Function Name          : main
//* Object                 : Get parameter
//* Input Parameters       : none
//* Output Parameters      : none
//*-----------------------------------------------------------------
int main ( int argc, char *argv[] )
{
    flash_word          *base_addr ;
    flash_word          *load_addr ;
FILE                    *image ;
const FlashAt49BVDef    *flash ;
char str[30];
char name[256];
    //* Display Flash Downloader Header
    printf ( "\n**** %s Flash Programming Utility ****\n", TARGET_29ID ) ;
            for (;;)
            {
            printf("\nLoad address : \n\r");
        /* Get load address */
            gets(str);
            sscanf(str,"0x%x",&load_addr);
        printf("\n**** File to download ****\n\r");
            /* Get File */
gets(name);
//* If Error while opening the external Flash Image file
if (( image = fopen ( name, "rb" )) == NULL )
{
//* Display Error then exit
    printf ( "Error - Cannot open file image \"%s\"\n", name ) ;
            return ( FALSE ) ;
}
//* Else
else
{
    //* Display input file name
    printf ( "Input file is   :    %s \n", name ) ;
            }
            //* Display Load Address
            printf ( "Load address is:    %x \n", (int)load_addr ) ;
            // If FLASH Device is not recognized
            if (( flash = flash_identify( load_addr )) == (const FlashAt49BVDef
*)NULL )
            {
                // Display Error and exit
```

```
printf ( "Error - The Flash device is not recognized\n" ) ;
return ( FALSE ) ;
```