中文摘要

嵌入式系统发展的日新月异，芯片制造技术的不断革新，迅速扩展了 32 位微处理器的应用领域。目前 32 位微处理器种类繁多，主要有 ARM、x86、PowerPC、68K/ColdFire 和 MIPS 等。ColdFire 是 Freescale 公司在 68K 的基础上推出的高性价比、高集成度 32 位微处理器，并且该系列还不断有新产品推出。芯片的推广应用离不开功能强大的开发工具。本课题设计并实现了一套基于ColdFire 的评估系统，它由硬件评估板和配套的软硬件开发工具组成，可以解决目前国内开发工具严重依赖进口、开发资料较少等一系列问题，为用户提供了一种功能完备、操作简单、价格低廉、能满足学习和开发双重需求的实验平台。

本课题开发了针对 ColdFire 系列微处理器的 BDM 调试头以及 Windows 平台上的 SdIDE for ColdFire 嵌入式集成开发环境。文中以 MCF5271 为例，设计并制作了 SDEVB5271 评估板，将开发工具成功地运用在该评估板之上，构建了整套的 SDEVB5271 评估系统，可提供性能评估、应用开发等功能。文中详细阐述了整个开发的流程，可以对开发其他 ColdFire 型号芯片的评估系统以及类似的嵌入式产品提供借鉴和参考。

本文首先给出了 SDEVB5271 的硬件设计、各硬件模块的原理框图以及硬件测试流程；其次分析了 ColdFire 的 BDM 工作原理并给出 BDM 调试头的硬件设计和底层驱动程序的编写；然后阐述了软件设计中使用的关键技术，包括编译、连接脚本的编写，代码的下载与执行；最后给出了嵌入式应用程序的基础知识和一个 IO 口的实验程序，同时还给出了 μCLinux 在 SDEVB5271 上的移植实例。

关键词：评估系统，ColdFire，MCF5271，交叉编译器，连接器，BDM，μCLinux

ABSTRACT

The rapid development of embedded system and the continual renovation of manufacturing process enlarges the fields of 32-bit microprocessor. There are many kinds of 32-bit processors, mainly including ARM, x86, PowerPC, 68K/ColdFire and MIPS. Derived from 68K, ColdFire is a kind of 32-bit microprocessor having the aggressive price/performance and the high integrity. The chip 's applications need powerful development tools. Thus, an evaluation system for ColdFire is designed and implemented, which consists of the hardware evaluation board and the related development tools. So that, a serial of problems such as too much dependency on import tools and lacking of development documents can be well solved. Additionally, an evaluation system with all-purpose, simple operation, low-price meets the needs of both study and development for all users.

A BDM cable for ColdFire CPUs and an IDE named SdIDE for ColdFire running on windows platform is developed. Taking MCF5271 as an example, this paper has designed an evaluation board SDEVB5271, applied the new development tools to it，constructed the whole evaluation system of SDEVB5271 which can be used to evaluate and develop applications. The paper describes the flow of the whole development in detail, which can provide reference for the development of other ColdFire chips and other embedded productions.

First, this paper shows the hardware design, the theory block diagram of each hardware module and the test flow of them. Second, the paper analyses the work theory of the ColdFire BDM and gives the hardware design of BDM cable and the way to write the drivers. Third, it realizes the key technology of the software design such as how to write makefile and linkfile and the download and execution of the code. At last, it gives the basic knowledge of an embedded application and an experiment on I/O port. At the same time, the paper even gives a real example of porting μClinux to SDEVB5271.

Keywords: Evaluation System, ColdFire, MCF5271, Cross-Compiler, Linker, BDM, μCLinux

目　　录

第一章 概述

嵌入式系统是当前最热门最有发展前途的 IT 应用领域之一。它除了在传统的工业控制、电信设施中发挥巨大的作用，还在家用电器、医疗保健设备、数码相机、多媒体播放设备、手机、PDA、网络设备等各个领域扮演着重要的角色。嵌入式处理器是嵌入式系统的核心，是控制、辅助系统运行的硬件单元。目前，嵌入式系统中使用的处理器主要有：8 位、16 位、32 位、64 位嵌入式处理器和数字信号处理器(DSP)，而 32 位嵌入式处理器已成为主流。

本章首先介绍了 Freescale 公司(原摩托罗拉半导体部)的 ColdFire 系列微处理器的基本特性，接着分析了当前 ColdFire 系列微处理器开发工具的现状，然后给出了基于 ColdFire 评估系统的设计思路及课题意义，最后为本文的工作和组织结构。

1.1 ColdFire 系列微处理器

Freescale 32 位微处理器分为 PowerPC 系列、ColdFire 系列以及 Dragon Ball 系列等。ColdFire 系列是重要的组成部分之一，它可以应用于工业控制、仪器仪表、民用产品、网络产品等领域。

ColdFire 源于 Freescale 的传统 68K 指令集架构(ISA)，最初是在 20 世纪 80 年代末开发的，在计算和嵌入式应用中得到了广泛使用。由于 68K 属复杂指令流(CISC)类 CPU，不容易做得很小，速度上不去，ColdFire CPU 采用指令长度可变的精简指令流(RISC)技术，精简掉部分 68K 的指令，又与 68K 指令集兼容，并使多数指令可以在 1 个周期内完成，使 CPU 内核可以做得很小，速度可以达到 400MIPS，成本可以降到很低，更适用于嵌入式应用[1][2]。

ColdFire CPU 的应用领域不断扩大——从工业自动化系统到喷墨打印机和 MP3 播放器——提供片上功能，满足特定嵌入式应用的需求。ColdFire 系列目前已推出 50 多种型号的芯片，而且还不断有新产品推出。为了适应市场对更多连接的需求，Freescale 推出了多种 ColdFire 连接选择，包括 10/100 以太网、USB2.0、PCI、CAN 和其它串行口；为了适应市场对工业应用进行复杂、实时控制的需求，Freescale 又在 ColdFire 架构产品上集成了增强型时间处理单元(eTPU)；为了适应市场对安全性的要

求，Freescale 在 ColdFire 设备上提供了加密加速器，作为可选模块。

ColdFire 系列产品由最初的 Version 2(包括第一款 ColdFire 微处理器 MCF5206 以及 MCF52xx)发展到 Version 3(MCF5307)、Version 4(MCF54xx)，直到最新版的超流水线结构的 Version 6，其指令执行速度也相应的由 25MIPS@33MHz(0.8μm 工艺)提高到 610MIPS@333MHz(0.13μm 工艺)。目前 ColdFire 已经推出了采用 V2、V3、V4 核心的芯片，V5、V6 为后续版本。各个版本之间的比较如下[3]：

① V2 核心

- 两个独立的、解耦(Decoupled)的两级流水线；
- 单周期局部总线，具有统一的 Cache、RAM 和 ROM。

② V3 核心

- 两个独立的、解耦的四级指令获取流水线和两级指令执行流水线；
- 两级流水线局部总线，具有统一的 Cache、RAM 和 ROM。

③ V4 核心

- 两个独立的、解耦的四级指令获取流水线和五级指令执行流水线；
- 哈佛结构的指令和数据分开的高速缓冲存储器，可获得更宽的带宽；
- 指令分支加速结构。

④ V5 核心

- 和 V4 基本相同的流水线组织；
- 双执行流水线和大容量分支 Cache。

⑤ V6 核心

- V6 为超流水线结构。

1.2 课题背景

1.2.1 常见的 32 位微处理器及其应用

目前，在嵌入式领域得到广泛应用的 32 位嵌入式微处理器主要有 Intel 开发的嵌入式 x86 处理器，采用 ARM 公司 ARM 核心的 ARM7、9、10 等系列，Freescale 和 IBM 等公司生产的采用 PowerPC 内核的 PowerPC 系列，MIPS 公司开发的 MIPS 系列以及 68K/ColdFire 系列等。

x86 系列 CPU 性价比较高，软件兼容性好，开发资源丰富，开发平台简单，主要应用在工业场合和部分民用市场(例如机顶盒等)，相对来讲耗电较大。ARM 因其低功耗以及 ARM CORE 的快速设计发展，在手持设备领域(PDA、手机等)取得了领导地位。PowerPC 和 MIPS 则以其在网络通信处理方面的高速、高可靠性等特点主要应用于数据通信领域。68K/ColdFire 凭借其优异的性价比、丰富的外围设备控制接口，在智能家电领域、低端网络设备(远程监控、Internet 网络通话等)、数据安全加密设备、以太网集线器、服务器类应用、销售点终端打印机甚至家用路由器等方面得到广泛应用[4][5]。

1.2.2 ColdFire 系列微处理器的开发工具及评估板

Freescale 推荐的 ColdFire 开发工具为 Metrowerks 公司的 CodeWarrior，它是专门面向 Freescale 所有 MCU 与 DSP 嵌入式应用开发的软件工具。这类商业的嵌入式开发软件，提供 Windows 平台上的开发方式，有相当完善的功能，但是这些软件一般价格昂贵，而且界面和帮助文档都是英文，购买和技术支持也不方便，不适合国内用户使用。

相对于商用软件的高昂价格，使用开源的、免费的 GCC 工具链进行嵌入式开发也是不错的选择。GCC(GNU C Compiler) 是 GNU 推出的在 Linux 或 UNIX 环境下运行的功能强大、性能优越的多平台交叉编译器，主要用于对 C 语言，包括 C、C++、Object C 等的交叉编译。GCC 交叉编译器几乎可以支持所有的 32 位 CPU，也支持像 Atmel 的 AVR 单片机，Freescale MC68HC11/12 这样的 8 位、16 位 MCU[6]。由于 GCC 是 Linux 下的编译器，使用 GCC 一般要求用户装有 Linux 操作系统。GCC 沿用了 Linux 的命令行方式，且由于其功能强大，命令参数选项特别多，对于初学者来说，要花费相当多的时间来学习如何使用。采用这种开发方式，显然没有那些专门面向某种 CPU 的商用软件那样方便，特别是所有的操作都是基于命令行的，没有 Windows 平台下集成开发环境的那种简单、易操作性。

要进行某种 CPU 的开发，除了有相应的软件开发工具，一般还需要有一块针对该 CPU 的硬件评估板。使用评估板，可以让用户在进行项目分析初期就可以完成测试，开发，评估，软件升级，产品功能演示等一系列功能，使用户不必制作 PCB 板就可以熟悉该芯片的指令系统，编程框架，实际性能等，以缩短用户的研发周期。

Freescale 对于 ColdFire 系列的每种型号都提供相应的评估板，其功能完善，外围接口丰富，而且还带有硬件调试工具(如 BDM 调试头)，但价格昂贵，基本在 300～900 美元之间。而国内主要的嵌入式产品提供商如深圳英蓓特、复旦金海博、上海存思等主要针对 ARM 市场，提供 ARM 的开发板产品。目前，只有作为 Freescale 半导体全球设计联盟成员之一的华恒科技推出了 ColdFire 的评估板，但其品种较少(只有针对 MCF5213/MCF5249 的)，而且开发资料不够全面，价格也相对较高。

对于国内的开发者来说，国外的软硬件价格昂贵，购买、技术支持不方便；国内的嵌入式产品提供商提供的评估板种类少，可选择性差，而且开发工具均使用 Linux 环境下的 GCC，入门门槛较高。这在一定程度上限制了 ColdFire 系列 CPU 在国内的推广与应用，同时也导致开发板价格相对较高、学习资料缺乏等问题。

1.3 设计思路

本课题着眼于 ColdFire 系列 CPU 在国内的推广与应用，将构建一套基于 ColdFire 的评估系统，这套系统具有操作简单，价格低廉，技术资料全面，可扩展性强等特点。整个评估系统由软件开发环境，硬件开发工具，硬件评估板以及一些示例程序组成。

软件开发环境是一套在 Windows 平台的 IDE(集成开发环境)，它以工程的方式组织源文件，方便用户编辑。IDE 的编译器调用 GCC，和Linux 下在命令行中使用 GCC 不同，IDE 提供了图形化的使用方式，用户只需要点击鼠标就可以完成编译选项配置、Makefile 生成、后台调用 GCC 编译等工作，并将编译的信息以友好的界面呈现给用户。同时，IDE 还提供代码写入、执行、固化等功能。

硬件开发工具主要是指编程、调试工具。代码生成后，要通过某种方式下载到目标板上才能执行。ColdFire 系列 CPU 内部没有任何的监控程序，因此要完成代码的下载，必须要有相应的硬件开发工具。ColdFire 系列一般带有 BDM(背景调试模式)调试模块，为此，本课题将制作针对 ColdFire 系列的 BDM 调试头。

有了软硬件开发工具，还要有这些工具的应用对象——硬件评估板。ColdFire 系列目前已推出 50 多种具体型号的芯片，本课题将从中选取一款 CPU 制作一套硬件评估板。硬件评估板一般要具有一些常用的对外接口，丰富的外围设备以及引出一些 CPU 引脚以便以后扩展。最后，编写一些实验程序，来测试软硬件开发工具及评估板硬件各个模块。

虽然本课题中以 ColdFire 的一款芯片为例设计了一块硬件评估板，但所有的软硬件开发工具却是通用的。BDM 调试头提供了 Freescale 的 BDM 标准接口；编译模块可以通过传递芯片型号，实现对不同芯片的编译优化；写入模块可以根据 CPU 类型及 Flash 型号进行程序的下载、固化。整个评估系统为用户进行 ColdFire 的开发提供了方便、简单的开发模式。

1.4 课题意义

嵌入式系统在我国已逐渐形成产业，其在生产研发、高校教学等方面得到了大力推广。研究基于 ColdFire 的评估系统具有如下现实意义：

① 为广大嵌入式爱好者提供一套良好的学习平台。目前国内外嵌入式开发人才稀缺。一方面，是因为这一领域的门槛较高，不仅要懂较底层软件，而且必须懂得硬件的工作原理，所以非专业 IT 人员很难切入这一领域；另一方面，是因为这一领域较新，目前发展太快，很多软硬件技术出现时间不长或正在出现(如 ARM 处理器、嵌入式操作系统、无线传感网络等)，掌握这些新技术的人很少。嵌入式人才稀少，根本原因可能是进行嵌入式开发需要相应的开发板和软硬件开发工具，大多数人无条件接触这一领域。本评估系统的推出，正好可以满足这一需求。

② 为广大嵌入式产品研发人员提供一套廉价实用的工具。目前比较流行的 32 位微处理器主要是 Freescale 的 ColdFire 和 ARM。ColdFire 系列处理器在工业控制中占有绝对地位；而 ARM 处理器是当今最流行的嵌入式处理器内核，它在家电和通信行业中应用十分广泛。目前市场上针对 ARM 的开发工具特别多，而 ColdFire 的却很少。ColdFire 不仅具有 32 位微处理器的高性能，而且还具有 8 位微控制器的简单、易用等特性，正被越来越多的人所使用。

③ 为研发类似的嵌入式产品提供不可多得的借鉴和经验。本课题详细介绍了整个系统的开发流程及技术细节。文中给出了在 Windows 平台上执行 GCC 的方法，交叉工具的构建，Makefile 的书写技巧以及连接器的使用都适合开发其他 CPU 架构的集成开发环境。硬件评估板的设计、测试方法也可在设计其他电路时借鉴。

1.5 本文工作和论文结构

1.5.1 本文工作

本文的主要工作安排如下：

(1) 制作硬件评估板

① 硬件模块划分

② 芯片选型

③ 了解和确定各硬件模块的外围电路

④ 硬件评估板原理图设计

⑤ 绘制 PCB，联系厂家制作

⑥ 元件采购

⑦ 焊接、测试、完成调试

(2) 制作硬件开发工具

① 分析 ColdFire 处理器中 BDM 调试模块的原理

② 设计并制作 BDM 调试头

③ 编写 BDM 驱动程序，实现对目标 CPU 操作的常用命令

(3) 软件设计与调试

① SdIDE 集成开发环境的总体设计

② SdIDE 集成开发环境通用部分实现

③ 准备 ColdFire 的交叉开发工具

④ 了解连接器的工作原理，编写芯片初始化代码和连接脚本文件

⑤ 完成 SdIDE 对 ColdFire 的支持，并编写用于代码写入的外部程序

⑥ SdIDE 集成开发环境的编程实例

⑦ 移植 μCLinux 到评估板(板级移植)

⑧ 总结整个开发过程

1.5.2 论文结构

本文第一章介绍了基于 ColdFire 的评估系统的课题背景、意义以及设计思路；

第二章详述了 SDEVB5271 的硬件设计和现实，介绍了评估板采用的各种芯片的主要特性及电路图，最后还给出了硬件测试流程及测试心得；

第三章首先分析了 ColdFire 微处理器中 BDM 调试模块的工作原理，然后阐述了 BDM 调试头的硬件设计以及驱动程序的编写；

第四章是软件的详细实现，包括交叉开发工具的使用方法以及程序的下载；

第五章给出了在 SDEVB5271 的应用实例，包括裸机程序的编写以及 μCLinux 的移植；

最后对全文进行了总结与展望。

第二章 评估板硬件设计

SDEVB5271 硬件评估板采用了 MCF5271 高性能微处理器，包括 64K 的内部 SRAM、16M SDRAM、2M Flash、2 个串口、1 个以太网口等，可以运行 μC/OS、μCLinux 等嵌入式操作系统。本章讲述了 SDEVB5271 设计的详细过程，包括芯片选型原则，各模块使用的芯片介绍，硬件评估板与各模块的电路连接。最后给出了硬件测试方法以及测试体会。

2.1 硬件选型

2.1.1 CPU 的选取

一般来说，CPU 是一个嵌入式系统的核心。一切外围电路的设计都是围绕 CPU 的特性来考虑的。因此，选择一个合适的处理器是一件相当重要的事情。选择处理器时，除了技术方面的因素，非技术原因也占了很大成分，其主要原因见表 2-1[3]。

表 2-1 选择处理器需要考虑的几个问题

序号	影响因素	原因
1	价格	在产量和成本约束相当严格的场合，价格是非常重要的因素。在一般的嵌入式系统中，十几美元的 CPU 算是比较昂贵的
2	性能	仔细评估 CPU 的性能能否满足系统的处理需求(满负荷运行时)，最好使用评估板评估一下实际的处理能力。有的 CPU 内部有 MAC 单元，运算密集时应优先选用这样的 CPU
3	封装	目前，QFP、BGA 的封装较多。BGA 焊接要求严格，必须使用机器焊接，可测试性差
4	功耗	对于电池供电的产品，需要非常重视 CPU 的功耗问题
5	指令集	RISC 的主频高，但代码密度和运行效率稍低。CISC 有时也是不错的选择
6	EMC	应优先选用对 EMC(电磁兼容)作过优化设计的 CPU
7	总线形式	最好能与大多数外围器件实现无缝连接
8	MMU	MMU 内存管理单元对于有些操作系统是必需的
9	生命周期	选用一种处理器之前，要对它的 Roadmap 有很清楚的了解。尽量选用生命周期较长的产品
10	工具支持	包括编译、调试环境、操作系统支持等
11	市场定位	了解芯片的定位，低端还是高端，消费电子还是工业用

本课题的设计目的是设计一套基于 ColdFire 的评估系统，因此只能从 ColdFire 系列微处理器中选取一款合适的 CPU。由于通用低端外设的联接扩展和网络市场空间的飞速发展，需要高性能、低价位的解决方案。Freescale 推出的 MCF527x 系列微

处理器就是针对这种需求而设计的，其综合性能较以前的产品有了大幅度提高，而功耗及价格却更低，为用户提供了更多的选择。第二代 ColdFire V2 内核的工作性能也上了一个新台阶，可在 166 MHz 的时钟频率下提供高达 159 MIPS 的处理能力(Dhrystone 2.1)[7]。考虑到本实验室的实际情况，决定选用该系列中的 MCF5271，它有 QFP 和 BGA 两种封装，可以选择较易焊接的 QFP 的封装形式，而且其样片可以直接从 Freescale 网站申请。

2.1.2 外围器件的选取

外围器件决定着 CPU 能否完成其相应的功能扩展，外围器件的选择原则如下：

① 能否完成系统的要求。在选择时，一定要仔细阅读芯片的参考手册以及一些应用实例，充分了解器件的性能及应用场合。

② 接口电压与形式。外围器件的电压最好能够从电路板上直接获取，避免为此新增多余的电压转换芯片，最好能选用与系统总线无缝接口的芯片。

③ 尽量选用标准产品。选用引脚、封装、功能兼容的产品会降低采购的风险。

根据以上原则，本硬件评估板根据功能需求，选择了以下芯片，如表 2-2 所示。

表 2-2 各模块芯片选型

设备类型	模块名	备注
存储	Flash	AMD29LV160DB，2Mbyte
	SDRAM	HY57V641620HG，8Mbyte*2
输入/输出	以太网	DM9161，物理层芯片
	串口	MAX3232，2 路串行口
	A/D 转换	TLC2543，11 路输入 10 位精度

2.2 芯片简介

2.2.1 MCF5271 微处理器

(1) MCF5271 微处理器主要性能

MCF5271 微处理器是 Freescale 半导体公司于 2004 年推出的一款高性能、低成本的 32 位微处理器。该芯片采用 ColdeFire V2 内核，内部集成了 10/100Mbps 快速以太网控制模块 FEC；具有硬件 DES/3DES/AES 加密功能；队列式串行外围接口 QSPI、I^2C 接口、增强乘法加法运算单元 eMAC；具有 64KB 的片内 SRAM，8KB 的可配置

cache；32 位的 SDRAM 控制器[8]。其引脚图和内部功能模块框分别参见附录 A.1 和附录 A.2。

(2) 快速以太网控制器 FEC 简介

MCF5271 集成的快速以太网控制器(FEC)支持 10Mbps 和 100Mbps 的以太网和符合 IEEE802.3 协议的网络，该模块具有如下特性：

① 支持三种不同的以太网标准物理接口：10Mbps IEEE 802.3 MII 接口、100Mbps IEEE 802.3 MII 接口和 10Mbps 7 线接口(工业标准)；

② IEEE 802.3 全双工流量控制；

③ 在 50MHz 的系统时钟下即可支持全双工操作(200Mbps 吞吐量)，在 25MHz 的系统时钟下即可支持半双工操作(100Mbps 吞吐量)；

④ 如果检测到冲突则无需经过处理器总线，直接从发送 FIFO 缓冲区进行数据重传；

⑤ 自动刷新接收 FIFO 缓冲区以去除冲突碎片，同时自动进行地址识别，无需利用处理器总线。

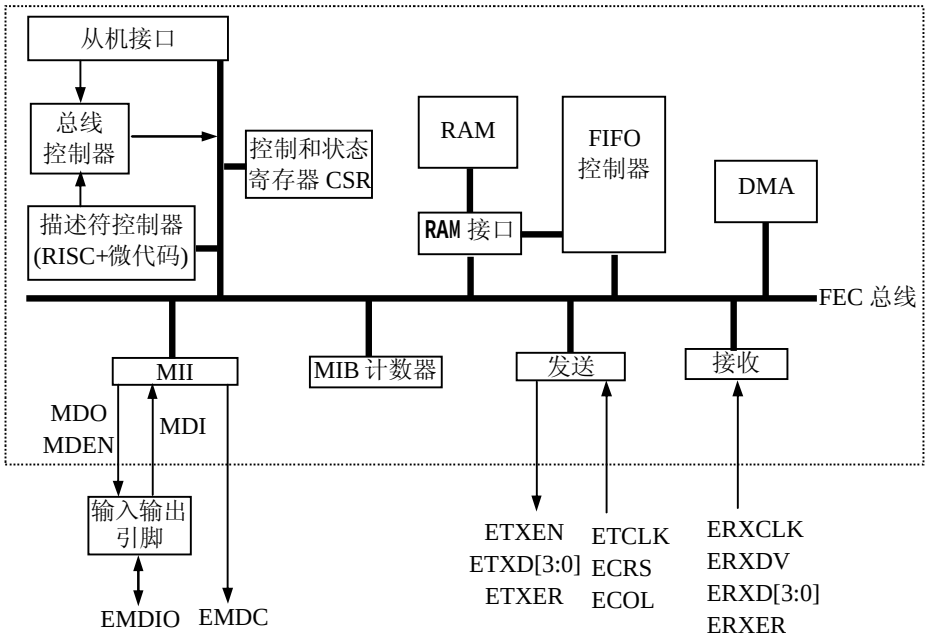


图 2-1 FEC 模块图

FEC 模块的内部结构如图 2-1 所示。图中描述符控制器是一个 RISC 结构的控制器，该控制器具有以下功能：

① 初始化 DMA 传输，对 DMA 通道进行高层控制；

② 解释缓冲块描述符；

③ 对接收到的以太帧进行地址识别；

④ 生成用于发送冲突补偿定时器的随机数。

图中的 MII 模块功能为对外部的网络物理层设备进行控制和读取状态，与标准含义的 MII 接口不同。其中 EMDC 为数据管理时钟，EMDIO 为串行输入输出引脚，所有的物理设备控制信号和状态信号都从该引脚输入输出。标准 MII 接口中的数据发送和接收引脚位于图中的发送和接收模块。

DMA 模块提供了多个数据传输通道，通过这些通道，FEC 模块中的数据发送/接收、描述符发送/接收就可以独立运行，互不干扰。需要注意的是这里的 DMA 是指快速以太网控制模块的 DMA 引擎，该 DMA 引擎只负责对快速以太网的数据进行传输，与通用含义的 DMA 不相同。

MCF5271 微处理器中的 FEC 模块使用了缓冲块描述符的概念。以太网最大的数据帧长度可以达到 1518 个字节，而且可能短时间内收到多个以太帧，如果仅仅使用 FEC 模块中的存储器来存储显然是不够的，所以 FEC 模块的以太帧数据必须暂存在 FEC 外部的存储器中，这其中可能存在的一种情况是一帧的数据存放在多个缓冲块中。为了对这种以太帧进行处理，必须有一种方法将这些缓冲块联系起来，缓冲块描述符(Buffer Descriptors)就是起这个作用的。缓冲区描述符包含起始地址(32 位对齐的指针)、数据长度和状态/控制信息，DMA 引擎根据缓冲区描述符发送数据。

(3) 队列式串行外设接口 QSPI 简介及内部逻辑功能

串行外设接口 SPI 是 Freescale 公司提出的同步串行外设接口，它允许 CPU 与各种外设接口器件以串行方式进行通信和交换信息[9]。

MCF5271 芯片提供队列式串行外围接口(Queued Serial Peripheral Interface)，称其为队列式 SPI，是因为该模块可以提供可编程传输队列，允许同时有 16 个用户排队等候数据传输，而且在传输时不需要 CPU 参与。该模块具有如下特性：

① 无需用户干预，可编程队列最多支持 16 个传输；

② 支持从 8 位到 16 位的传输长度；

③ 4 个外设片选信号，最多可以控制 15 个外围设备；

④ 在主频 75MHz 的情况下，可以设置的波特率从 147.1Kbps 到 18.75Mbps；

⑤ 可编程控制传输前后的延时时间；

⑥ 可编程 QSPI 时钟相位和极性；

⑦ 对于连续传输，提供环绕模式。

MCF5271 使用 80 个字节的静态 RAM 执行队列操作，这部分 RAM 是 QSPI 和 CPU 可以共同访问的静态 RAM 空间，它分为以下 3 部分，16 个命令控制字节(简称命令 RAM)，16 个数据发送字(简称发送 RAM)和 16 个数据接收字(简称接收 RAM)。这 80 个字节被组织成 16 个队列项，每个队列项包含 1 个指令控制字节、2 个数据发送字节和 2 个数据接收字节。

需要进行数据通信时，用户首先要进行 QSPI 初始化操作：将命令队列写入命令 RAM，将待发送的数据写入发送 RAM，然后允许 QSPI 数据传输。QSPI 执行队列中的命令并通过 QSPI 中断寄存器 QIR 的 SPIF 标志位指示命令已经执行完毕。当 QSPI 数据传输结束后，QDLYR 寄存器中的 SPE 位自动清零。此时可以读取接收 RAM 中收到的数据。

2.2.2 存储器件

(1) Flash

由于成本和制造工艺等原因，很多 32 位微处理器无法集成足够大的 Flash 存储器模块，甚至没有 Flash 模块，这就大大限制了 32 位微处理器优越的控制和计算功能的发挥。解决这个瓶颈的有效而简单的方法就是对 32 位微处理器进行 Flash 存储器扩展。

由于 32 位微处理器的应用场合较为复杂，所以一般都要架设嵌入式操作系统。Flash 存储器的扩展主要目的是对代码存储空间的扩充。通常需要满足以下要求[1]：

① 应用程序可以不需要调入 RAM 而直接在 Flash 上运行；

② 可以对 Byte ／Word 单位存储单元进行直接操作；

③ 扩展方便，所需额外硬件，软件少；

④ 数据存储可靠性高。

在选择 Flash 存储芯片时，主要考虑容量和类型两个因素。对于 32 位微处理器，常用的嵌入式操作系统核心小于 1MB，16M 位的 Flash 芯片能完全满足嵌入式操作系统及应用程序的需要。Flash 存储器主要有 NOR 型和 NANR 型，它们的性能比较如表 2-3 所示。

表 2-3 NOR 型和 NAND 型 Flash 性能比较

项目	NOR 型	NAND 型
程序运行方式	直接在 Flash 上运行	需 MTD 支持，在 RAM 中运行
容量	小	大
同容量价格	高	低
编程单位	Byte	Block(512Byte)
接口	SRAM 接口，足够的地址引脚	复杂的 I/O 口串行存取数据
使用复杂性	有地址线，可以像其他存储器一样使用，直接，简单	须先写入驱动程序，写入信息技巧性极强，须有虚拟映射

由表 2-3 中 NOR 型和 NAND 型 Flash 的性能比较可知，NOR 型 Flash 存储器主要用来存储代码，如系统代码空间的扩展。NAND 型 Flash 则适合于大量数据存储，如 MP3 文件管理等[10]。

本设计采用 16M 位的 NOR 型 Flash 存储器扩展方案。硬件板采用 AMD 公司生产的 AM29LV160DB，它的容量达 2Mbyte。AM29LV160DB 有如下特性[11]：

- 3.3V 单一供电，编程、擦除电压只需要 3V；

- 0.23 微米芯片制造技术；

- 8 个 8Kbyte 的可引导块、127 个 32Kbyte 的普通块；

- 10 万次可重复擦写；

- 125 摄氏度下数据仍可保存 20 年；

- CFI 兼容(Common flash Memory Interface)。

(2) SDRAM

与 Flash 存储器相比较，SDRAM 不具有掉电保持数据的特性，但其存取速度大大高于 Flash 存储器，且具有读/写的属性。SDRAM 在系统中主要用作程序的运行空间、数据及堆栈区。具有操作系统的系统在启动时，CPU 首先读取启动代码，在完成系统初始化后，程序代码一般应调入 SDRAM 中运行，以提高系统的运行速度。同时系统及用户堆栈、运行数据也都在 SDRAM 中。

评估板采用了 Hynix 公司的 HY57V641620HG[12]，该芯片存储空间达 8Mbyte，非常适用于需要高速大容量存储器的应用。它有如下一些特性：

- JEDC 标准 3.3V 单一电源供电；

- 内部分为 4 块，每块容量 1,048,576*16 位(16M 位)；

- 自动刷新和自刷新，刷新速度达 4096 周期/64ms；

- 总线最高时钟频率可达 160MHz。

2.2.3 输入输出类器件

(1) 以太网

MCF5271 内部具有 FEC 模块，提供标准的 MII(Media Independent Interface)接口，因此只需要外接一片物理层芯片便可实现以太网通信。评估板选用了 Davicom 公司的 DM9161 作为物理层芯片，该器件有如下特点[13]：

- 工作电压 3.3V；
- 完全兼容 IEEE 802.3u；
- 集成 10Base-T 和 100Base-TX 收发器；
- 支持 MII 或 RMII(Reduced MII)接口；
- 支持全双工和半双工；
- 支持连接状态 LED。

(2) A/D 转换

在嵌入式控制中，通过 A/D 采样可以感知外部环境的状态等信息，由于 MCF5271 内部没有集成 A/D 模块，故评估板采用一片专用的 A/D 转换芯片来实现类似的功能。评估板采用了 Texas Instrument 公司的 TLC2543 模数转换芯片，特性如下[14]：

- 2.7-5.5V 工作电压；
- 片内时钟源；
- SPI 接口；
- 11 路 A/D 转换，12 位精度。

(3) 串行

评估板采用了 Maxim 公司生产的 MAX3232 作为串行电路的驱动芯片，该芯片支持两路串行收发。

2.3 硬件评估板设计

SDEVB5271 采用 2 层 PCB 板的设计，CPU 频率 100MHz，总线频率 50MHz，接口和外围设备丰富，实物如图 2-2 所示。

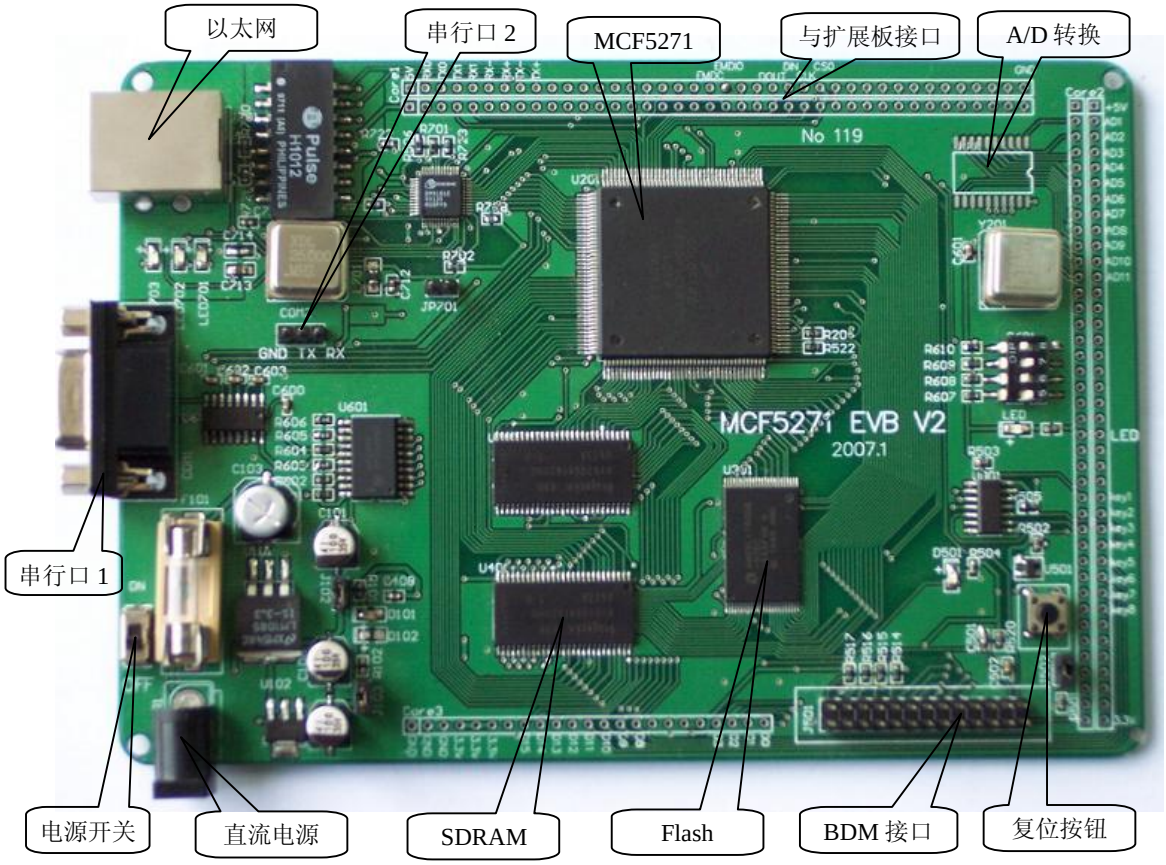


图 2-2 评估板硬件板

本节将给出硬件评估板的各外围设备的原理框图，并对各外围设备的接口进行说明，在附录 B 中给出了硬件评估板的所有原理图。

2.3.1 电源

SDEVB5271 硬件评估板采用单一的 5V 直流电源直接供电。使用了两个电压转换芯片(1085,1117)可以分别提供 3.3V 和 1.5V 的直流电源。5V 电源可供 AD 转换芯片(TLC2543)使用，同时和扩展板的 5V 相连，可以给扩展板供电或者从扩展板取电；3.3V 为存储芯片及网络芯片提供工作电源；1.5V 是 ColdFire 内核使用的工作电压。

电源电路的好坏，直接决定着整个系统能否稳定的工作。通过电源接适当的滤波电容，可以提高整个电路的抗干扰性。为了防止不慎的短路对评估板上的芯片造成损坏，在 5V 电源的入口处加了一个保险丝，可以在电路电流过大时，自动切断电源，从而保护整个电路板。为了将电源电路和其他的模块隔离，特将 3.3V、1.5V 与其他模块相接的地方加一个跳线，这样当电源电路不稳定时，可以断开电源电路与其他电

路的连接，从而快速的判断出是哪部分电路的问题。

2.3.2 CPU 支撑电路

CPU 支撑电路主要包括电源滤波电路、晶振电路、复位电路以及 CPU 配置电路。电源滤波电路确保了输入芯片的电压是稳定、可靠的。晶振电路产生 CPU 工作所需的外部时钟信号，可以使用有源晶振或无源晶振。本设计中采用的是 25MHz 的有源晶振，电路相对简单。复位电路主要是保证 CPU 能够可靠地复位，当复位芯片检测到电压小于复位电压时，就会立即产生一复位电平。在选择复位芯片时，一定要根据 CPU 所要求的复位时间以及复位电压，选择一款合适的复位芯片[15]。本设计中使用的 IMP 公司生产的 IMP811。该芯片产生的复位脉冲时间为 140ms，完全可以满足 MCF5271 的最长复位时间 10us。

MCF5271 可以工作在多种模式下，这些配置可以通过微处理器的外围引脚的设定来控制，如图 2-3 所示，RCON、JTAG_EN、CLKMOD[0:1]为 MCF5271 的外围引脚，故可以通过拨码开关设定，RCON 为复位配置选择脚，JTAG_EN 用来设置是否允许进入调试模式，CLKMOD[0:1]用来设置时钟频率。另外一些配置需要通过对数

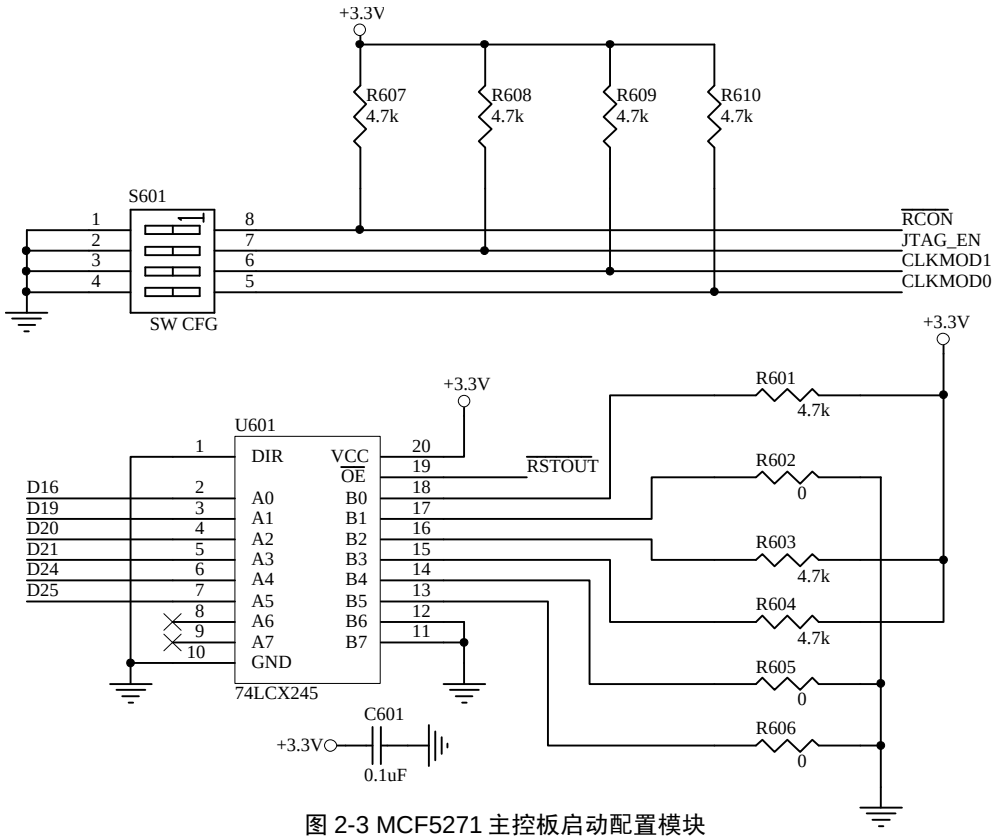


图 2-3 MCF5271 主控板启动配置模块

据总线中的 D16、D19~D21、D24、D25 进行信号接电阻上拉至 3.3V 或下拉至地来确定。为了防止这些配置对数据总线进行数据传输时的干扰，故使用一片 74LCX245 进行缓冲，同时兼有隔离的作用。具体各个引脚的配置功能在此不详细讲述，仅仅列出主控板的配置。

D[19:20] = 01　　系统从 16 位设备端口启动

D21 = 1　　引脚驱动能力为全额输出

D[24:25] = 00　　地址线为 A[23:21]

2.3.3 Flash

AM29LV160DB 的存储容量为 16M 位，一般采用 48 脚 TSOP 封装，16 位数据宽度，在字节模式下数据宽度为 8 位，在字模式下数据宽度为 16 位。AM29LV160DB 的逻辑框图如图 2-4 所示。各信号功能如表 2-4 所列。

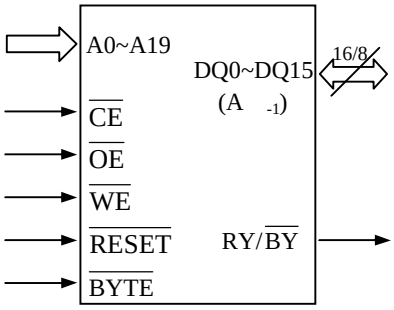


图 2-4 AM29LV160DB的逻辑框图

表 2-4 AM29LV160DB引脚功能

引脚	功能
A0~A19	20 根地址线。在字节模式下，DQ15/A_1 为 21 位字节的地址最低位
DQ0~DQ14 DQ15/A_{-1}	数据线。在字模式下，DQ0~DQ15 为 16 位数据线。在字节模式下，DQ0~DQ7 为 8 位数据线，DQ8~DQ14 为高阻态，DQ15/A_{-1} 为地址最低位
$\overline{CE}$	片选信号，低电平有效。在对 AM29LV160DB进行读写操作时，该引脚必须为低电平
$\overline{OE}$	输出使能，低电平有效。在读操作时有效，写操作时无效
$\overline{WE}$	写使能，低电平有效。在对 AM29LV160DB进行编程和擦除操作时，控制相应的写命令
$\overline{RESET}$	硬件复位，低电平有效。当复位时，AM29LV160DB立即终止正在进行的操作
$\overline{BYTE}$	8 位/16 位模式选择。低电平为字节模式，DQ0~DQ7 为数据线，DQ8~DQ14 为高阻态，DQ15/A_{-1} 为地址最低位。高电平为字模式，DQ0~DQ15 为数据线
RY/$\overline{BY}$	就绪/忙状态指示。用于指示写或擦除操作是否完成。当 AM29LV160DB正在进行编程或擦除操作时，该引脚为低电平；操作完成时为高电平，此时可读取内部数据

2MB 的 Flash 采用 16 位数据宽度，地址空间为 0xFFE0 0000~0xFFFF FFFF。Flash 与 CPU 的连接如图 2-5 所示。

AM29LV160DB 的 $\overline{BYTE}$ 接上拉电阻，使 AM29LV160D 工作在字模式。

AM29LV160DB 的 $\overline{OE}$ 与 MCF5271 芯片的 OE 相连。

AM29LV160DB 的 $\overline{WE}$ 与 MCF5271 芯片的 R/W 相连。

AM29LV160DB 的 $\overline{\text{RESET}}$ 与 MCF5271 芯片的 RESET 相连。

AM29LV160DB 工作在字模式，其地址总线 A19~A0 与 MCF5271 芯片的地址总线 D31~D16 相连。

AM29LV160DB 的 $\overline{\text{CE}}$ 与 MCF5271 芯片的 CS0 相连，这样可以设置为从 Flash 启动。

AM29LV160DB 的 RY/$\overline{\text{BY}}$ 悬空，它的编程与擦除操作的工作状态可以通过查询片内的相关寄存器判断。

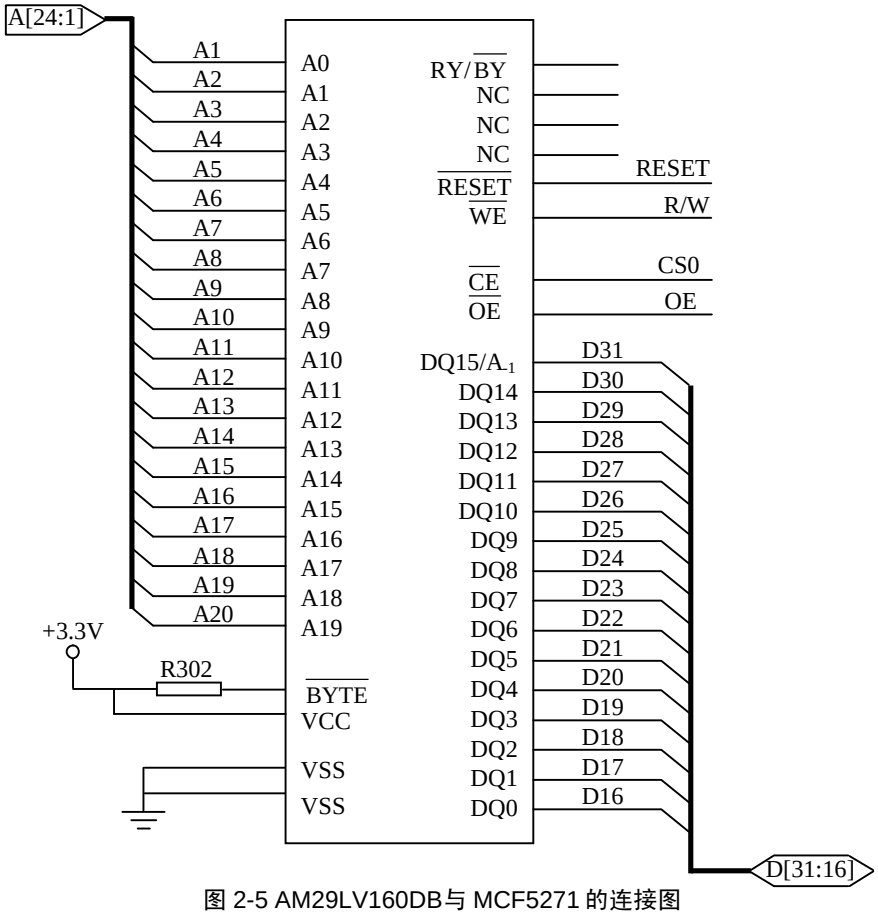


图 2-5 AM29LV160DB与 MCF5271 的连接图

2.3.4 SDRAM

MCF5271 内含同步 DRAM(SDRAM)控制器，可以通过外扩 SDRAM 的方式，增加系统的内存。SDRAM 常用 16 位或 32 位的数据总线。MCF5271 的 SDRAM 控制器的主要特点如下：

- 与 JEDEC 兼容的各种 SDRAM 器件的接口；

- 可将 16 位或 32 位的 MCF5271 数据总线连接到 SDRAM 存储阵列；

- 支持 16~256M 位的器件；

- 专用的存储体地址引脚使得不同大小的 SDRAM 可在同一块 PCB 板上使用；

- 页大小是 256~1024 的列地址；

- 假定在 66MHz 时页命中，突发读的访问时间为 6-1-1-1，突发写访问时间为 3-1-1-1；

- CAS 的锁存为 1 和 2；

- 最多同时激活 4 个存储体；

- SDRAM 掉电和自刷新；

- 在系统时钟降至 5MHz 时，刷新定时器预分频因子保持 15.6us 的刷新周期；

- SDRAM 自动初始化。

SDRAM 具有存储空间大、价格低的优点，但必须进行定时刷新。ColdFire 芯片的片内带有的 SDRAM 控制器，可完成对 SDRAM 的定时刷新。

ColdFire 常用的 SDRAM 为 32 位数据宽度，工作电压为 3.3V，主要有 SAMSUNG 的 K4S641632F、HYUNDAI 的 HY57V641620HG 等。

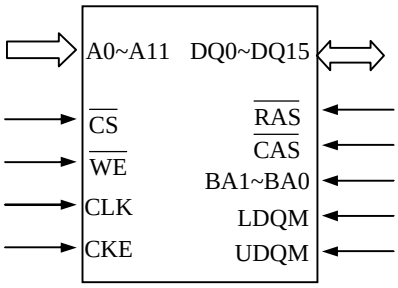


图 2-6 HY57V641620HG 逻辑框图

本文采用的 HY57V641620HG 存储容量为 4*1M*16 位(64M 位)，常用封装为 54 引脚 TSOP，JEDEC 标准 3.3V 电源供电，兼容 LVTTL 接口，支持自动自刷新，16 位数据宽度。HY57V641620HG 的逻辑框图如图 2-6 所示。各信号功能如表 2-5 所列。

表 2-5 HY57V641620HG 引脚功能

引脚	功能
CLK	芯片时钟输入
$\overline{CS}$	禁止或使能所有输入信号(CLK、CKE、LDQM、UDQM 除外)
CKE	时钟使能
A11~A0	行/列地址复用。行地址：RA11~RA0；列地址：CA7~CA0
BA1,BA0	块地址选择。用于片内 4 个块的选择。在行地址锁存时，所选择的块被激活；在列地址锁存时，对所选择的块进行读/写
$\overline{RAS}$	行地址锁存。当 RAS 为低时，CLK 上升沿锁存行地址，允许行锁存和预充电
$\overline{CAS}$	列地址锁存。当 CAS 为低时，CLK 上升沿锁存列地址，允许列存取
$\overline{WE}$	写使能。允许写操作和行预充电，从 $\overline{CAS}$ 、$\overline{WE}$ 激活时锁存数据
LDQM,UDQM	数据输入/输出屏蔽。当 LDQM，UDQM 激活时，块数据输入。在写模式下，屏蔽数据输出
DQ15~DQ0	16 位数据线。

表 2-6 MCF5271 与 SDRAM 接口(32 位数据宽度、8 根列线)

MCF5271 引脚	A15	A14	A13	A12	A11	A10	A9	A17	A18	A19	A20	A21	A22	A23
行线	15	14	13	12	11	10	9	17	18	19	20	21	22	23
列线	2	3	4	5	6	7	8	16						
SDRAM 引脚	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13

本设计使用的 HY57V641620HG 含有 8 根列线，MCF5271 采用 32 位数据宽度。此时，MCF5271 的 SDRAM 控制器对行、列线的分配如表 2-6 所示[8]。根据 MCF5271 的引脚定义，设计的 SDRAM 控制电路如图 2-7 所示。

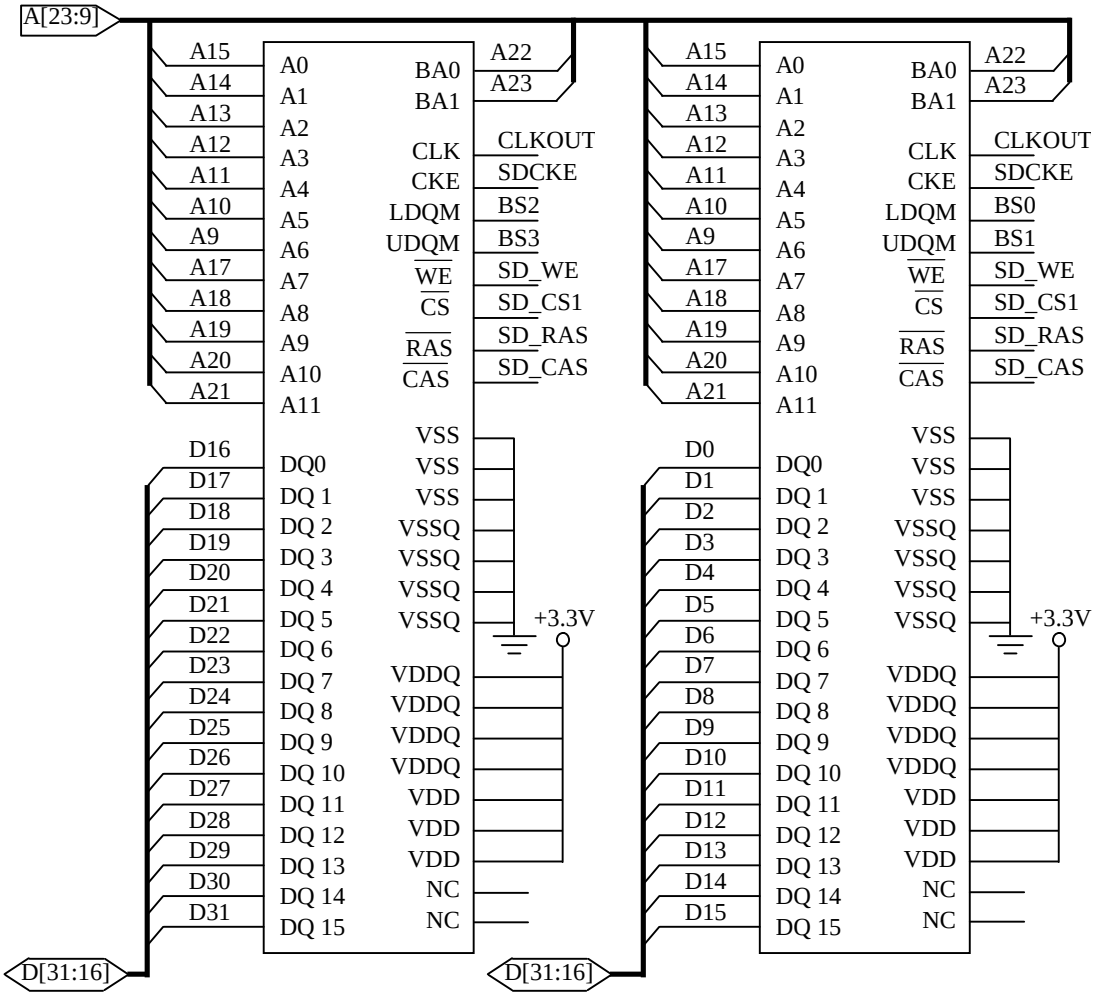


图 2-7 HY57V641620HG 与 MCF5271 的连接图

由图 2-7 可以看出，采用了两片 16 位 SDRAM 并作 32 位数据宽度的连接方法，MCF5271 的 32 条数据线中，高16 位 D[31:16]和低 16 位 D[15:0]分别与两片 SDRAM 相连。SDRAM 的 A[13:0]分别和 CPU 的 A[23:17]、A[9:15]相连作为行地址线；SDRAM

的 A[7:0]分别与 CPU 的 A[16]、A[8:2]相连，作为列地址线；SDRAM 的 BA[1:0]分别和 CPU 的 A[23:22]相连，作为块选择；负责存储低 16 位的 SDRAM 的 LDQM、HDQM 与 CPU 的 BS0、BS1 相连，作为高低字节选择线，负责存储高 16 位的 SDRAM 的 LDQM、HDQM 与 CPU 的 BS2、BS3 相连，作为高低字节选择线。CAS(列选择)、RAS(行选择)、WE(写使能)、CKE(时钟源使能)、CLK(时钟源)等控制信号线也分别和 CPU 的 nCAS、nRAS、SDWE、SDCKE、SDCK 引脚相连。由于 160 脚 QFP 封装的 MCF5271 没有专用的 SDRAM 片选信号，两片 SDRAM 的 CS 片选可以与 QSPI_CS1 复用。16MB SDRAM 的地址空间为 0x0000 0000~0x01FF FFFF。

2.3.5 以太网模块

MCF5271 内部包含一个 FEC 模块，用于OSI 中的物理层和数据链路层之间的硬件实现，同时该芯片也提供了一个标准的 MII 接口，只需要外扩一个物理层器件便可实现以太网通信。本评估板外扩了一个物理层器件 DM9161，它可通过 MII 接口和 MCF5271 相连，经过一个隔离器件(H1012)，提供了 RJ45 接口，如图 2-8 所示。

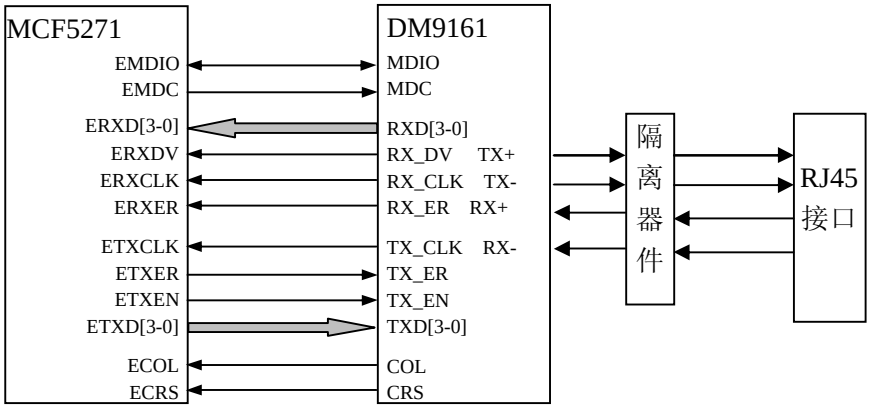


图 2-8 以太网物理层 DM9161 连接图

MCF5271 芯片 MII 接口的信号线可以分为三组：用作数据管理的信号线 EMDIO 和 EMDC；反映介质状态的信号线 ECOL 和 ECRS；其余为接收/发送信号线。

EMDIO 为双向信号，在 DM9161 和 MCF5271 之间传输控制信息和状态信息。控制信息由 MCF5271 发给 DM9161，状态信息由 DM9161 反馈给 MCF5271。EMDC 为输出信号，作为 EMDIO 数据传输的参考时钟。

ECOL 为输入信号，该信号有效时，表示检测到传输介质上有冲突。ECRS 为输入信号，该信号有效时，表示检测到传输介质忙；否则表示传输通道处于空闲状态，

可以进行传输。

ERXD[3-0]用来接收 DM9161 发送的 4 位数据；ERXDV 用来表示从 DM9161 接收到的数据是否有效，高电平为有效；ERXCLK 为 DM9161 提供给 MCF5271 的数据接收时钟信号，其工作频率为数据接收速度的 25%；ERXER 为输入信号，该信号有效时表示检测到当前在 DM9161 上传输的帧出错。

ETXD[3-0]用来向 DM9161 发送 4 位数据；ETXEN 为输出信号，该信号有效时表示 MII 总线上有数据，可以启动发送操作。ETXCLK 为输出信号，作为数据发送的参考时钟信号。

2.3.6 串口

MCF5271 片内可支持 3 路串行模块，评估板引出了其中的第 1 路和第 2 路，基本可满足一般应用的需求。使用 MAXIM 公司的 MAX3232 串行收发器，该器件可支持两路串行收发。MCF5271 和 MAX3232 的连接如图 2-9 所示。

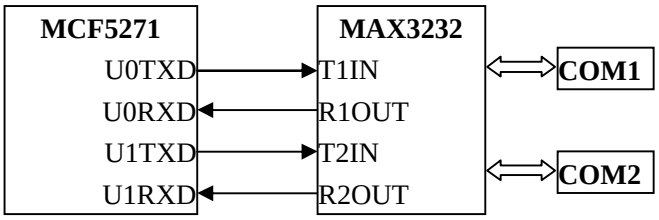


图 2-9 串行电路连接图

MAX3232 的主要信号描述如表 2-7 所示。

表 2-7 MAX3232 信号描述

引脚	名称	描述
7	T2OUT	第 2 路发送输出
8	R2IN	第 2 路接收输入
9	R2OUT	第 2 路接收输出
10	T2IN	第 2 路发送输入
11	T1IN	第 1 路发送输入
12	R1OUT	第 1 路接收输出
13	R1IN	第 1 路接收输入
14	T1OUT	第 1 路发送输出

2.3.7 A/D 转换

TLC2543 与 MCF5271 通过 QSPI 接口相连，占用 SPI 片选的 CS0 位，如图 2-10 所示。

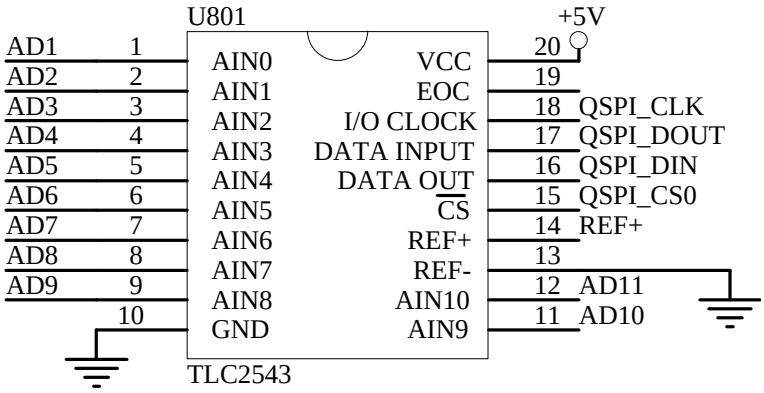


图 2-10 AD 扩展电路连接图

TLC2543 可采集 11 路 A/D 模拟输入信号，可连接各种温度、光敏传感器等，测量各种物理量输入。TLC2543 的主要信号描述如表 2-8 所示。

表 2-8 TLC2543 信号描述

引脚	名称	描述
15	CS	片选信号
18	I/O CLK	输入输出时钟
17	DIN	数据输入
16	DOUT	数据输出
1-9，11-12	AIN[11:0]	模拟信号输入
14	REF+	参考电压正
13	REF-	参考电压负

2.3.8 与扩展板接口

评估板设计了 3 排 SIP 接口，通过这些扩展接口可以和作者所在的研究室开发的扩展板对接。扩展板主要集成了 8/16 位单片机实验中常用的各种模块的接口电路，这样核心板只需设计 CPU 的基本工作电路以及与扩展板的接口，就可以在扩展板上完成该芯片所有模块的基本实验。本评估板通过与扩展板的连接可以完成的实验有：2 个串口、以太网、SPI 通信、I^2C 通信、AD 采样、键盘、液晶显示等。

2.4 硬件评估板测试流程及体会

2.4.1 测试方法

在制作硬件电路板的时候，由于电路原理设计、制板、焊接以及元件自身的问题等都会引起硬件电路板的不通，因此，在测试时，必须遵守一定的方法。硬件测试的方法有如下几种：

①利用万用表来测量，主要是测量引脚的电压是否正常，以及两个引脚是否短接或虚焊。

②利用示波器来测量，主要是测量晶振是否能够起振，捕获沿跳变以及观测引脚的时序转换。

③利用 BDM 调试头测试 CPU 内部模块及外部的总线扩展。

④编写基本的模块测试程序。

2.4.2 测试流程

根据硬件评估板包含的硬件模块，测试流程如下：

(1) 电源测试

焊接好两片电压转换芯片 LM1085-3.3、1117-1.5 和电源接头以及滤波电容后，可以测量 LM1085 及 1117 的输入电压是否为+5V，输出电压是否分别为+3.3V 和+1.5V，并且输出稳定、无波动。

(2) 主控芯片测试

主控芯片 MCF5271 的基本工作电路包括晶振电路、芯片配置电路、复位电路以及芯片电源滤波电路。根据各模块的功能，首先焊接晶振电路。由于使用的是有源晶振，因此不需要 CPU 的干预就可以直接用示波器测出产生的晶振是否为 25MHz。然后焊接芯片配置电路，通过万用表测量特定引脚的电压信号是否正确。最后就可以焊接主控芯片了。由于 MCF5271 的内部 ROM 中没有固化的程序，因此必须借助 BDM 调试头来测试 CPU 是否能够正常工作。将 BDM 调试头与评估板上的 26 针 BDM 接口连接，运行 PC 端的测试软件，即通过 BDM 调试头使芯片进入 BDM 模式，如果能够成功，则可以说明 CPU 工作正常。为了进一步测试，可以编写一个小灯闪的基本程序，通过 BDM 调试头写入到内部的 SRAM 中，然后运行之。

(3) SDRAM 测试

两片 HY57V641620HG 焊接好后，可以将上面测试 SRAM 的小灯闪程序编译到 SDRAM 的地址空间，并通过 BDM 调试头写入到 SDRAM 中，然后运行这个程序(具体过程可参见第四章)。如果小灯不停的闪烁，则说明 SDRAM 工作正常。不过，由于 SDRAM 较难焊接，并且引脚信号基本都是地址线和数据线，因此实际的调试过程中很少出现一次就会成功的情况。当运行结果不正确时，原因一般为地址线或数据线

虚焊或相邻两根线短接导致的。为了快速的找到问题的原因，可以编写一个对SDRAM某个地址写入和读出的函数，如果写入的数据和读出的数据不一致，则可以通过比较具体的数据值，找到对应的数据位。

(4) Flash 测试

焊接好 AM29LV160DB后，可以通过 BDM 调试头对 Flash 进行读取、擦除以及写入的操作。如果操作不成功，在说明 MCF5271 与 Flash 的连接有问题，可以按照调试 SDRAM 的方法找到问题所在。

(5) 以太网测试

DM9161 是一片高性能物理层芯片,它可以自动检测出对方是 10M 还是 100M 网卡，并建立连接。因此，将以太网模块焊接好后，可以直接将评估板上的 RJ45 接头通过一根交叉网线与 PC 机的网卡相连。如果可以建立连接，则说明 DM9161 工作正常。在测试 CPU 与 DM9161 的通信时,可以先通过 MCF5271 的 MII 接口读写 DM9161 的寄存器开始。以上都测试通过之后，就可以编写收发一个数据包的程序了。

(6) 其他模块的测试

QSPI、I^2C、A/D 转换、串口以及与扩展板的接口的测试主要是要编写响应的测试子程序，其硬件的连接相对简单，测试相对容易。

2.4.3 测试体会

(1) 深刻理解电路原理图

电路原理图是进行硬件设计的基础，在画原理图时，往往会参考芯片制造厂商提供的一些应用笔记，而具体的原理可能没有搞清楚，就开始进行 PCB 的布板。在调试硬件时，只有当调试不通时，才会开始认真研究电路的原理图，这样不仅浪费制板费，还浪费时间。作者在设计以太网模块时，就是参考了 AT91RM9200 的网络模块设计，而没有注重原理的理解，结果导致第一版的网络部分需要跳线。因为 9200 和 DM9161 的接口方式是 RMII，而 MCF5271 只提供标准的 MII 接口。

(2) 分模块测试

焊接的时候，要焊一个模块，测一个模块，不要一次焊多个模块。一个模块测试通过了，才可以继续焊下面的。因为一旦有错误，就可以很快的定位到是哪个模块的问题。

(3) 注重电源电路

电源电路的好坏直接决定着整个系统能否稳定工作。开始焊的第一块板子的3.3V的电源有的地方是3.3V，但有的地方却是3.6V，居然大于3.3V。后来用力按板子，有时就好了，所以判断有地方虚焊。后来经查发现，原来是转3.3V的电源芯片的GND引脚虚焊。第二块板子的电源也有问题：芯片的多个3.3V的电源引脚测量出来的电压不相同，在3.0V~3.3V之间。后来换了个晶振就好了，估计是晶振质量不好，对电源产生了较大干扰。为了防止元器件工作时产生的高频噪声对整个电源电路的影响，可以在器件的电源和地之间加上0.1uF左右的电容。

(4) 电路的电流

在焊某个模块前，特别是焊芯片时，很容易短接，而且又不好随便拿掉，所以在焊之前可以测量一下当前的电流值，然后等焊好芯片之后，再测量此刻的电流值是否在一个合适的范围内。如果电流出现异常，可以及时发现，同时也便于定位出错电路。

(5) 同类型信号线的分布

在各种微处理器的输入输出信号中，总有相当一部分是相同类型的，例如数据线、地址线。对这些相同类型的信号线应该成组、平行分布，同时注意它们之间的长短差异不要太大，采用这种布线方式，不但可以减少干扰，增加系统的稳定性，还可以使布线变得简单，印刷电路板的外观更美观。

第三章 ColdFire 系列 BDM 调试头的设计与实现

嵌入式系统上通常没有我们所熟知的显示器(最多仅有一个很小的 LCD)，也没有很完备的类似于 PC 机上的调试开发环境，因此调试嵌入式软件具有一定的难度。目前在调试嵌入式软件方面主要有两大发展趋势：一种是基于硬件的调试技术；另一种是基于软件的调试技术，它们各有所长。本章介绍的 BDM 调试头就是一种硬件调试工具，配合底层的驱动程序及高端的软件，可以实现对目标芯片的调试。

3.1 芯片调试技术

大多数嵌入式微处理器生产商已意识到嵌入式软硬件调试的困难，因此它们会在芯片中集成一个支持在线调试的模块。使用特殊的接口将之连接到 PC 机的串口、并口或 USB 口上，通过运行在 PC 机上的调试软件可以实现读写指定位置的 Flash 或者 SDRAM 数据、复位、设置断点、单步运行等调试操作；在指令运行停止时刻，可以方便地观察各个寄存器、存储器、外围接口的值。由于这种方法使用的是基于硬件的调试技术，因此在实时性调试方面发挥着重要的作用。通常，编写调试嵌入式 RTOS 都是使用该技术。

32 位 CPU 几乎无一例外地内置了片上调试电路(OCD)，宿主机只要通过很简单的命令就可以控制 CPU 的运行、读写存储单元以及 CPU 内部的寄存器等。只要是 CPU 指令可以完成的工作，都可以通过宿主机控制 OCD 来完成。下面具体介绍一下三种常见的 OCD 接口模块[16][17]。

(1) JTAG。JTAG(Joint Test Action Group，联合测试行动小组)是 1985 年指定的检测 PCB 和 IC 芯片的一个标准，1990 年被修改成为 IEEE 的一个标准即 IEEE1149.1，通过这个标准，可对具有 JTAG 接口芯片的硬件电路进行边界扫描和故障检测。边界扫描测试有两大优点：一个是方便芯片的故障定位，迅速准确地测试芯片管脚的连接是否可靠，提高测试检验效率；另一个是，具有 JTAG 接口的芯片，内置一些预先定义好的功能模式，通过边界扫描通道来使芯片处于某个特定的功能模式，以提高系统控制的灵活性和方便系统设计。JTAG 接口是国际上常用的一种标准接口，具有连接电路电路简单，调试方便，使用广泛等诸多优点。现在使用最多的 32 位 ARM 系列

微处理器大多带有这种接口。

(2) BDM。Freescale 的背景调试模式(Background Debug Monitor，BDM)也是一种常见的调试模块，广泛用于 Freescale 所生产的各类微处理器之上。如 8 位机中的 HCS08 系列，16 位机中的 S12 系列，以及 32 位机中的 68K/ColdFire，PowerPC 系列等。

(3) OnCE。OnCE 的全称是 On-Chip Emulation。它是另一种常见的调试模块，在 Motorola 568xx 系列 DSP 和 M*Core 上都包含有该模块。值得注意的是：M*Core 里除了包含有 OnCE 模块之外，也包含有 JTAG 模块。

3.2 ColdFire 微处理器的 BDM 调试模式

ColdFire 系列微处理器支持 JTAG 和 BDM 两种调试模式。虽然 JTAG 调试具有不占用系统资源并能够调试没有外部总线的芯片，但是由于 JTAG 是通过串行依次传递数据的，速度比较慢，因而只能进行软件断点级别的调试；而且其自身还不能完成实时跟踪和多种事件触发等复杂调试功能。因此，ColdFire 系列微处理器一般采用 BDM 调试模式。

3.2.1 ColdFire 调试模块概述

ColdFire 系列微处理器中的调试模块针对不同场合的应用分别提供了 3 种调试支持：实时跟踪调试、BDM 调试和实时调试。

① 实时跟踪调试：能够跟踪应用程序的动态执行路径。ColdFire 以 8 位并行输出总线实现实时跟踪，为外部仿真系统提供处理器运行状态和数据。

② 背景调试模式(BDM)：提供复杂的 ColdFire 处理器的底层调试。通过 BDM 处理器被暂停，各种命令发送到处理器，以便访问存储器和寄存器。外部仿真器使用 3 个引脚，实现串行的全双工通信。

③ 实时调试：BDM 需要处理器暂停才能调试，对于大多数的实时嵌入式应用系统，这是做不到的。调试中断使实时系统执行一个特定的服务子程序，它能快速存储关键寄存器的内容和变量，并使系统返回到正常操作。由于硬件支持处理器的当前操作和 BDM 内部指令，开发系统能访问存储的数据。

所有 ColdFire 调试信号都是单向的，而且与处理器内核时钟信号的上升沿相关。这几种调试方式都共用 26 脚的 BDM 调试引脚信号，这些信号的定义如表 3-1 所示。

表 3-1 ColdFire 调试模块信号

信号	功能描述
开发串行时钟(DSCLK)	内部同步输入。在串行通信端口的数据传送计数。最大的工作频率是 PSTCLK 的 1/5。在 DSCLK 的同步上升沿，DSI 上的数据输入被采样，DSO 改变状态
开发串行输入(DSI)	内部同步输入，提供串行通信端口对调试模块的数据输入
开发串行输出(DSO)	为调试模块响应提供串行输出通信
断点(BKPT)	中断请求输入信号。在当前指令执行完后，处理器进入停机状态。停机状态在 PST[3:0]上对应的值是 0x0F
处理器状态时钟(PSTCLK)	处理器状态时钟信号。它指示外部开发系统采样 PST 和 DDATA信号的时刻
调试数据(DDATA[3:0])	默认情况下，这些输出信号表明了寄存器断点状态，也可以表示获取的地址和操作数。对数据的获取是由 CSR 控制的。另外，执行 WDDATA指令将获取 DDATA上的操作数。这些信号在处理器的每个时钟周期会被刷新
处理器状态(PST[3:0])	报告处理器状态的输出信号。它指示了处理器的当前状态，但与当前的总线传输无关。PST 的值也是在每个处理器时钟周期被刷新

调试系统主要由 3 部分组成，如图 3-1 所示。要调试的嵌入式系统硬件经 BDM 调试头与计算机的并口相连，PC 方的调试软件通过 BDM 调试头驱动程序提供的编程接口，可以实现对目标系统的调试。

图 3-1 BDM 调试系统组成框图

BDM 调试头作用是完成从并口与 BDM 调试端口之间操作时序、操作逻辑及电压的转换；BDM 驱动程序则是完成宿主机与待开发的嵌入式系统的通信过程处理，如合成一定格式的数据包、解释接收到的应答数据或者微处理器的状态数据等；BDM 调试软件则是功能软件，完成对 Flash、SDRAM 或 CPU 内部寄存器的读写及程序执行流程控制等功能。

3.2.2 不同 BDM 接口的比较

在 Freescale 的众多微处理器中，有多种不同类型的 BDM 接口。HCS08、S12 系列采用单线连接的 BDM 接口。它对时序的要求特别严格，通过特定的通信频率，实

现数据的双向、异步的单线通信。它可以通过简单的电路转换直接与 RS-232 接口连接。BDM 模式下的指令并不是 HCS08、S12 的微指令，而是通过改变 PC 寄存器的值去执行 ROM 中的特定代码。CPU32 系列内部有专门用来调试的微指令，它能够暂停正常的机器码，而去执行来自串行调试接口的 BDM 命令。这些命令可以用来访问所有的 CPU32 的寄存器以及片内或片外的存储空间。ColdFire 的 BDM 模式类似于 CPU32，并且它将程序的单步执行和一些逻辑功能直接集成在 ColdFire CPU 的内部，所以其 BDM 调试接口相对简单；同时，ColdFire 的 BDM 指令集中新增了实时监控的功能。PowerPC 与 CPU32 的 BDM 接口则基本相同[18]。

3.3 BDM 调试头的硬件设计

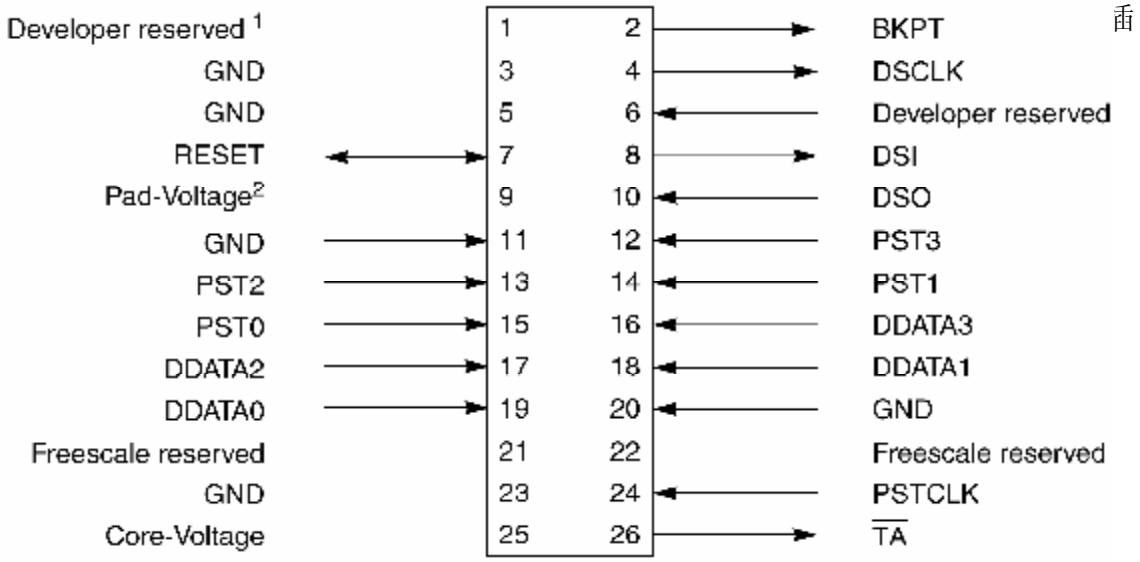


图 3-2 BDM 接口引脚连接规范

3.3.1 BDM 的串行通信时序分析

调试模块用两个输入(DSCLK 和 DSI)和一个输出(DSO)，实现同步协议，其中 DSO 是相对于处理器时钟上升沿的延迟。开发系统采用串行通信的主方式，并且一定要产生 DSCLK。BDM 接口的串行操作时序分析如图 3-3 所示。从图 3-3 中可以看

出，所有的事件都是基于处理器时钟(PSTCLK)的上升沿的。串行通信的运行频率是 DC 到 PSTCLK/5 频率，使用全双工模式。这里的 DSCLK 的作用类似于周期性的使能信号，当 DSCLK 为高时，在 PSTCLK 时钟的上升沿时允许所有的状态转换，即 DSI 被采样，DSO 被驱动。每次串行数据的传输可以分为 4 个阶段：C1，C2，C3 和 C4。在 DSCLK 的高电平期间，数据从 DSI 输入，经过 2 个 PSTCLK 周期的同步(C1 和 C2)而被采样，然后在 DSCLK 的低电平期间 PSTCLK 的第 1 个上升沿(C3)来临时 BDM 状态机改变状态。随后在第 2 个 PSTCLK 上升沿(C4)，DSI 准备传送新的数据，DSO 上则输出原来数据输入相对应的应答数据。BDM 状态机在 DSI 上数据采样检测到的时候改变状态，当所有的数据传输完毕，BDM 状态机的状态也就不会有什么改变[1][8][19]。

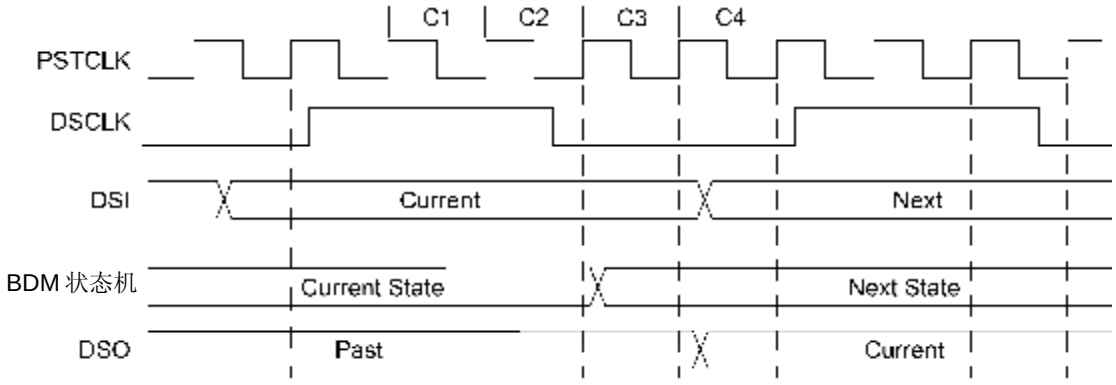


图 3-3 BDM 串行接口时序图

C1～C4 描述如下:
C1 —— DSI 的第一个同步周期(DSCLK 为高)
C2 —— DSI 的第二个同步周期(DSCLK 为高)
C3 —— BDM 改变状态取决于 DSI 和整个的输入数据传送是否结束
C4 —— DSO 改变为下一个值

3.3.2 XC9536XL 芯片介绍

由上节的分析可以看出，BDM 下的通信需要满足一定的时序及逻辑条件，采用传统的电路设计时需要很多元器件，电路过于复杂，从而造成系统成本升高，可靠性差。本设计中采用一片 CPLD(复杂可编程逻辑器件)来达到简化设计的目的。CPLD 是一种整合性较高的逻辑元件。它具有编程灵活、集成度高、设计开发周期短、适用范围宽、开发工具先进、设计制造成本低、对设计者的硬件经验要求低、标准产品无需测试、保密性强、价格低廉等特点，可实现较大规模的电路设计[20][21]。采用了 CPLD

的产品具有性能提升，可靠度增加，PCB 面积减少及成本下降等优点。

CPLD 元件基本上是由许多个逻辑方块(Logic Blocks)所组合而成的。而各个逻辑方块均相似于一个简单的 PLD 元件(如 22V10)，逻辑方块间的相互关系则由可编程的连线架构。常见的 CPLD 元件有 Altera 公司的 Max5000 及 Max7000 系列；Cypress 的 Max340 及 Flash370 系列；Xilinx 的 CoolRunner 及 XC9500 系列等。

本设计中选用了 Xilinx 公司的 XC9536XL[22]，其主要特点有：

- 3.3V 电源供电
- 4ns 的引脚间逻辑延迟，最高系统频率可达 200MHz
- 支持 In-System Programming(ISP)和 IEEE1149.1(JTAG)边界扫描
- 0.35μmCMOS FashFLASH 技术
- 增强的数据安全特性
- 多种封装形式，包括 VQFP、TQFP、PLCC

3.3.3 BDM 调试头硬件实现

根据 BDM 接口的信号定义，可以看出这些信号都是单向的。其中要注意的是：PSTCLK 是调试模块从处理器中获取的，所有的输入和输出信号都是基于 PSTCLK 的上升沿的，而 DSCLK 则是外部开发系统产生的，控制着串行通信的时序。CPLD 与并口接口的信号有：

输入信号 —— DSI，DSCLK，BREAK，RESET，TEA

输出信号 —— DSO，ACK

CPLD 与 BDM 接口的信号有：

输入信号 —— PSTCLK，DSO， PST0 ～PST3

输出信号 —— BKPT，DSCLK，RESET，DSI，TEA

因此，共需配置 11 个输入端和 7 个输出端，故选用 Xilinx 公司的 XC9536XL 器件即可满足要求。XC9536XL 与 PC 机标准并口以及与 BDM 接口的引脚连接如图 3-4 所示。

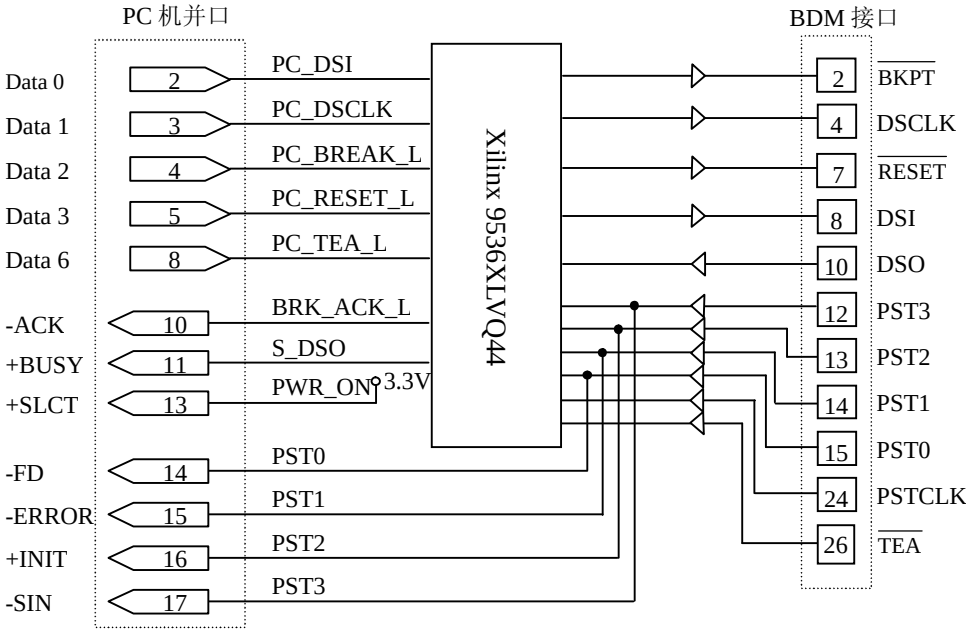


图 3-4 BDM 调试头硬件原理图

PSTCLK 信号通过一个 IBUFG 器件连接到 XC9536XL。IBUFG 即输入全局缓冲，是与专用全局时钟输入管脚相连接的首级全局缓冲[23]。所有的输入信号通过一个 FD 触发器，PSTCLK 作为统一的触发脉冲，这样就实现了所有的信号受控于 PSTCLK 的上升沿，即只有在 PSTCLK 上升沿时，FD 触发器的输出等于输入，否则输出保持上一个状态不变。当 PST[3:0]=0xFF 时，表明处理器暂停，因此可以将 PST3，PST2，PST1，PST0 通过一个与门 AND4 输出到 BRK_ACK_L，作为处理器的应答信号[24]。

为了实现对 XC9536XL 的编程，同样需要相应的软硬件工具。本设计中使用的硬件工具是一个 JTAG编程器，软件为 Xilinx 公司高性能设计开发工具 ISE (Integrated Software Environment) 6.1，它可以实现从输入、综合、CPLD 适配、仿真到编程下载的整个流程。设计输入有多种表达方式，最常用的是原理图方式和 Verilog HDL 硬件描述语言两种。在这里采用的是较简单的原理图方式。根据上面的分析，定义引脚的输入、输出，实现所要求的逻辑及时序操作。具体的原理图请参见附录 C。

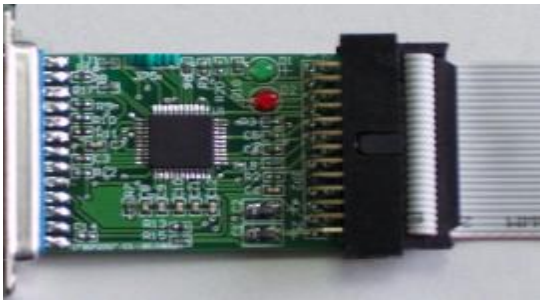


图 3-5 BDM 调试头实物图

硬件板为两层 PCB 板，实物图如图 3-5 所示。

3.4 BDM 调试头驱动程序的设计

BDM 调试头已经实现了并口与 BDM 口的时序与逻辑操作转换，按照一定的数据格式传送和接收数据则是 BDM 调试头驱动程序要完成的工作，驱动程序对数据的处理分为 2 层：底层完成 BDM 数据包的收发，上层实现 BDM 的各种调试命令。这些命令接口将会在第四章的程序写入部分中被调用。

3.4.1 BDM 的数据通信格式

BDM 串行通信使用双工模式，数据可在主控设备和从控设备之间同时发送和接收，每次传送的数据块由一个 17 位的数据包组成，该数据包由一个状态/控制位和一个 16 位数据字组成。

接收与发送包格式：

基本的接收、发送包由 16 个数据位和 1 个状态位组成。数据格式如下所示：

表 3-2、3-3 分别给出了 BDM 接收和发送包中各个字段的详细说明。

表 3-2 接收 BDM 包字段说明

位	名称	说明
16	S	状态位。表示 CPU 产生下面的消息的状态。除非在存储器访问周期，未就绪的响应可以被忽略；否则，调试模块在 32 个处理器时钟周期之后能接收新的串行传输。 S　DATA　消息 0　xxxx　有效数据传输 0　0xffff　状态 OK 1　0x0000　响应未就绪，重来 1　0x0001　错误——终止总线周期，数据无效 1　0xffff　非法命令
15～0	DATA	数据。内含由调试模块向开发系统发送的消息，响应消息总是一个单字，数字字段编码同上

表 3-3 发送 BDM 包字段说明

位	名称	说明
16	C	控制位，保留。由开发系统发命令和初始化传输数据，应清 0
15～0	DATA	数据。内含由开发系统向调试模块发送的数据

在了解了数据发送、接收的具体格式后，就可以通过对 PC 机并口的编程来实现最基本的收发 BDM 数据包的程序了。

3.4.2 BDM的串行通信实现

BDM 调试头通过并口与计算机相连，要实现 BDM 下的串行通信需要对并口编程。并口是从简单的打印机接口发展而来，早期的并口是单向口，它只是简单的把数据从计算机传送到打印机。随着双向并口的出现，并口的用途也越来越广泛。相对于串口它有传输快、可靠性高，编程简单的特点，在近距离通信方面正在逐步取代串口。并口主要分为三种：SPP(标准并行接口)，EPP(增强型并行接口)和 ECP(扩展型并行接口)。一个并行接口一般占有 3 个端口地址。其中第一个地址是基地址，即数据寄存器，第二个是状态寄存器，第三个是命令寄存器。各个寄存器位定义及对应的引脚如下所示[25][26]。

		7	6	5	4	3	2	1	0
数据寄存器	位定义	D7	D6	D5	D4	D3	D2	D1	D0
	引脚	2	3	4	5	6	7	8	9
状态寄存器	位定义	BUSY	ACK	PE	SLCT	ERR	—	—	—
	引脚	11	10	12	13	3	—	—	—
控制寄存器	位定义	—	—	—	—	SIN	INIT	ALF STRB	
	引脚	—	—	—	—	17	16	14	1

数据寄存器是可读写的，通过操作数据寄存器可以实现并口信号的输入和输出；状态寄存器为只读，用来检测输入并口的信号；控制寄存器为只写，可以输出一些控制信号。根据 BDM 调试头的硬件原理图中引脚分配，定义以下有关并口操作的宏：

```
#define CF_PE_DR_DATA_IN        0x01    //              数据输入 DSO
#define CF_PE_DR_CLOCK_HIGH     0x02    //              数据同步时钟 DSCLK
#define CF_PE_DR_BKPT           0x04    //BKPT                  引脚
#define CF_PE_DR_RESET          0x08    //              复位信号
#define CF_PE_DR_TEA            0x40    //              传输允许

/*通过并口向目标芯片输入 0/1 操作的宏*/
#define CF_PE_DR_MASK   (CF_PE_DR_DATA_IN\
            |CF_PE_DR_CLOCK_HIGH \
                | CF_PE_DR_BKPT | CF_PE_DR_RESET\
    | CF_PE_DR_TEA)

#define CF_PE_MAKE_POS_DR(flags)  ((flags) & (CF_PE_DR_DATA_IN\
      | CF_PE_DR_CLOCK_HIGH))
#define CF_PE_MAKE_NEG_DR(flags)  ((~(flags)) & (CF_PE_DR_BKPT \
      | CF_PE_DR_RESET | CF_PE_DR_TEA))
#define CF_PE_MAKE_DR(flags)     ((CF_PE_MAKE_POS_DR(flags) | \
     CF_PE_MAKE_NEG_DR(flags)) \
                & CF_PE_DR_MASK)
```

```
/*并口获取处理器状态*/
#define CF_PE_SR_PST1       0x08   //              处理器状态位 1
#define CF_PE_SR_DATA_OUT   0x80   //              数据输出 DSI

#define CF_PE_CR_NOT_PST0   0x01   //              处理器状态位 0
#define CF_PE_CR_NOT_PST2   0x02   //              处理器状态位 2
#define CF_PE_CR_NOT_PST3   0x04   //              处理器状态位 3
```

一个标准的 BDM 包由 17 位数据组成，通过控制 DSI 和 DSCLK 的电平变化，将数据一位一位的传输到目标芯片。首先传输 MSB(最高有效位)，在传送一位后，读取 DSO 上的返回的数据位。此功能的实现代码如下所示。

```
/*---------------------------------------------------------------*
 * 功  能:通过并口，实现  BDM  的串行通信                        *
 * 参  数: *self--    记录  BDM  通信所需的信息                  *
 *      wval --          要发送的数据                          *
 *      holdback--       用来调整要发送的数据位数              *
 * 返  回:  无                                                  *
 *---------------------------------------------------------------*/
static void
cf_pe_serial_clocker (struct BDM *self, unsigned short wval, int holdback)
{
 unsigned long shiftRegister;
 unsigned char dataBit;
 unsigned int  counter;

 shiftRegister = wval; //   记录要发送的数据
 counter       = 17 - holdback; //       计算要发送的数据位数
 /*  开始循环发送*/
 while (counter--)
 {
  //   首先发送高位 MSB
  dataBit = ((shiftRegister & 0x10000) ? CF_PE_DR_DATA_IN : 0);
  shiftRegister <<= 1;
  outb (CF_PE_MAKE_DR (dataBit), self->dataPort);//   数据上线,DSCLK 为低电平
  if (self->delayTimer)  //     延时
   bdm_delay (self->delayTimer);
  outb (CF_PE_MAKE_DR (dataBit | CF_PE_DR_CLOCK_HIGH),
        self->dataPort);  //          数据保持,DSCLK 设为高电平
  if (self->delayTimer)
   bdm_delay (self->delayTimer);
  outb (CF_PE_MAKE_DR (dataBit), self->dataPort);//   数据保持,DSCLK 为低电平
  if (self->delayTimer)
   bdm_delay (self->delayTimer);
  //   从 DSO 中读取数据,并从 shiftRegister 的低位开始存放
  if ((inb (self->statusPort) & CF_PE_SR_DATA_OUT) == 0)
```

```
        shiftRegister |= 1;
    }

    outb (CF_PE_MAKE_DR (0), self->dataPort);// 输出数据平时保持低电平

    self->readValue = shiftRegister & 0x1FFFF;
}
```

3.4.3 BDM 的调试命令

BDM 调试模块为外部开发系统提供了 12 个基本类型的命令(用助记符表示)：RAREG/RDREG(读地址/数据寄存器)，WAREG/WDREG(写地址/数据寄存器)，READ(从存储器读数据)，WRITE(向存储器写数据)，DUMP(与 READ 命令结合使用转储数据块)，FILL(与 WRITE 命令结合使用填充数据块)，GO(继续执行程序)，NOP(不做任何操作，可以被用作一个空指令)，RCREG(读系统控制寄存器)，WCREG(向系统控制寄存器写入数据)，RDMREG(读调试模块寄存器)，WDMREG(写调试模块寄存器)。各种命令和它对应的应答数据的格式可以参考 ColdFire 的用户编程手册。上层驱动程序最终的任务就是实现这 12 种基本命令和应答数据的收发，并为调试软件提供相应的函数调用接口。

ColdFire 系列的 BDM 命令包括一个 16 位的操作字，随后是一个或多个可选的扩展字，如图 3-6 所示，表 3-4 给出了各个字段的说明。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
操作码						0	R/W	操作数大小		0	0	A/D	寄存器		
扩展字															

图 3-6 BDM 命令格式

表 3-4 BDM 命令字段说明

位	名称	描述
15～10	操作码	每个调试命令对应一操作码
9	0	保留
8	R/W	操作数传递的方向 0：开发系统写数据到 CPU 或存储器 1：CPU 传送数据到开发系统
7～6	操作数大小	指定操作数字节数。一个命令执行单字节的存储器读取将忽略高 8 位的数据，数据将返回在低 8 位中 操作数大小　　位数 00　字节　8　位 01　字　16　位 10　长字　32　位

		11　　保留　　—
5～4	00	保留
3	A/D	地址/数据。决定该寄存器字段描述的是数据还是地址
2～0	寄存器	指定要操作的寄存器个数

一些指令需要扩展字作为地址及立即数。因为只有绝对长地址寻址，所以需要两个扩展字做地址。长字存取被强制按长字对齐，而字存取被强制按字对齐。立即数可以是一或二个字长。字节和字数据都需要一个扩展字，而长字数据需要两个扩展字。操作数和地址的传输字的最高有效位在前[27]。

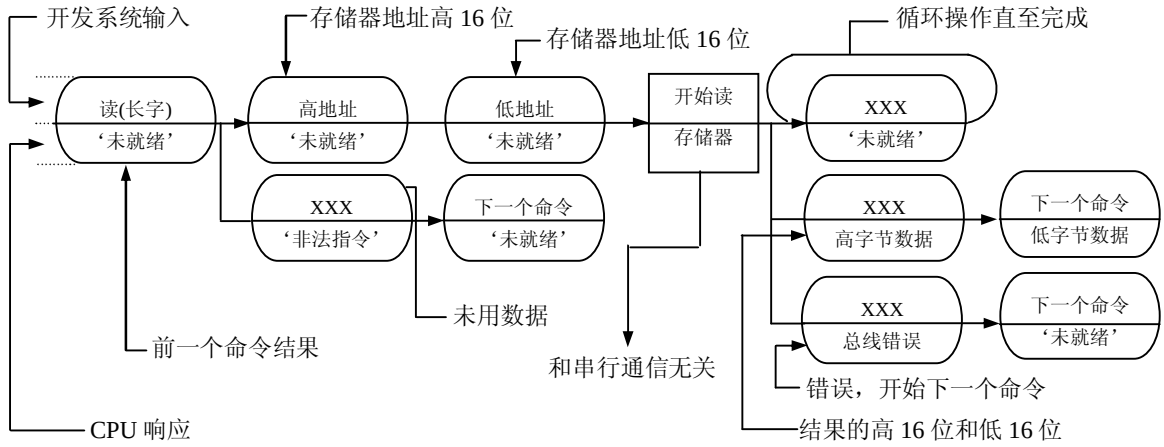


图 3-7 BDM 命令序列图

图 3-7 中的命令序列图给出了命令的串行总线传输。每个椭圆形的上半部或下半部代表一个 17 位的 BDM 包传输。其中上半部表示开发系统送往调试模块的数据，下半部表示调试模块对之前开发系统命令的响应。指令和结果重叠交叉使延迟最小。

序列具体如下：

在周期 1 中，开发系统的命令被执行(图中以 READ 命令为例)。如果不要求结果，调试模块或者响应前一命令的低位结果，或者响应前一命令的全部状态。

在周期 2 中，开发系统提供高 16 位地址。调试模块返回未就绪的响应，除非收到的命令编码未完成，这用非法指令编码显示。如果发生该情况，开发系统将再次传送这个命令。

在周期 3 中，开发系统提供低 16 位地址。调试模块总是返回“未就绪”的响应。

在周期 3 完成时，调试模块开始存储器读操作。在存储器存取期间，串行传输返回“未就绪”的响应。

在存储器存取完成之后，在2 个串行传输周期内结果被返回。对于完成字节读操作的任何命令，响应数据的高 8 位是不确定的，访问的数据在低 8 位返回。在最后的

传输期间，下一个命令的操作码被送到调试模块。如果存储器或寄存器存取被总线错误终止，返回错误状态(S=1,DATA=0x0001)，而不是结果数据。

根据每个命令的执行序列，就可以实现 BDM 的全部调试命令。这部分程序的编写可以参考 www.sourceforge.org 网站上的开源项目 BDM tools 。本文实现了主要的 BDM 调试命令以及一些对并口的基本操作。其中函数列表如表 3-5 所示。

表 3-5 BDM 驱动部分主要的函数列表

函数	说明
void init_port(unsigned short lpt_port)	初始化并口，参数为要初始化的并口号
void set_io_delay_cnt_value(unsigned short new_io_delay_cnt)	设置并口通信的时钟频率
bool go_to_background (void)	通过 bkpt 引脚，使 CPU 进入 BDM 模式
unsigned char read_data_byte(unsigned int address) unsigned short read_data_word(unsigned int address) unsigned int read_data_long(unsigned int address)	读内存操作。支持字节、字、长字等数据长度
void write_data_byte(unsigned int address, unsigned short data) void write_data_word(unsigned int address, unsigned short data) void write_data_long(unsigned int address, unsigned int data)	写内存操作。支持字节、字、长字等数据长度
unsigned int get_control(unsigned int address) void put_control(unsigned int address, unsigned int data)	读写控制寄存器
unsigned int get_data_reg(unsigned char register_number) void put_data_reg(unsigned char register_number, unsigned int data)	读写数据寄存器
unsigned int get_add_reg(unsigned char register_number) void put_add_reg(unsigned char register_number, unsigned int data)	读写地址寄存器
unsigned int get_pc_value(void) void put_pc_value(unsigned int newpc)	读写 PC 寄存器的值
void resume(void)	让 CPU 从当前 PC 所指处开始执行

3.5 本章小结

芯片调试技术的发展给开发人员调试硬件平台带来了极大的方便。调试模块是基于处理器的底层调试，直接“面对”硬件，所以在嵌入式系统的硬件平台设计中，可以使用它来验证硬件平台的正确性。本章首先介绍了芯片调试技术的相关知识，通过对 ColdFire 的 BDM 调试模式下各引脚信号的分析，给出了 BDM 调试头的硬件设计。BDM 调试头可以完成 PC 机并口与目标系统板之间信号的时序与逻辑转换，BDM 驱动程序为高层软件提供了简单、易用的编程接口。

第四章 软件设计

本章重点阐述 SDEVB5271 评估系统软件的具体实现，其中包括 SdIDE 集成开发环境通用部分介绍、SdIDE for ColdFire 交叉开发工具使用以及利用 BDM 调试头进行代码写入的方法。

4.1 SdIDE 通用模块介绍

4.1.1 SdIDE 简介

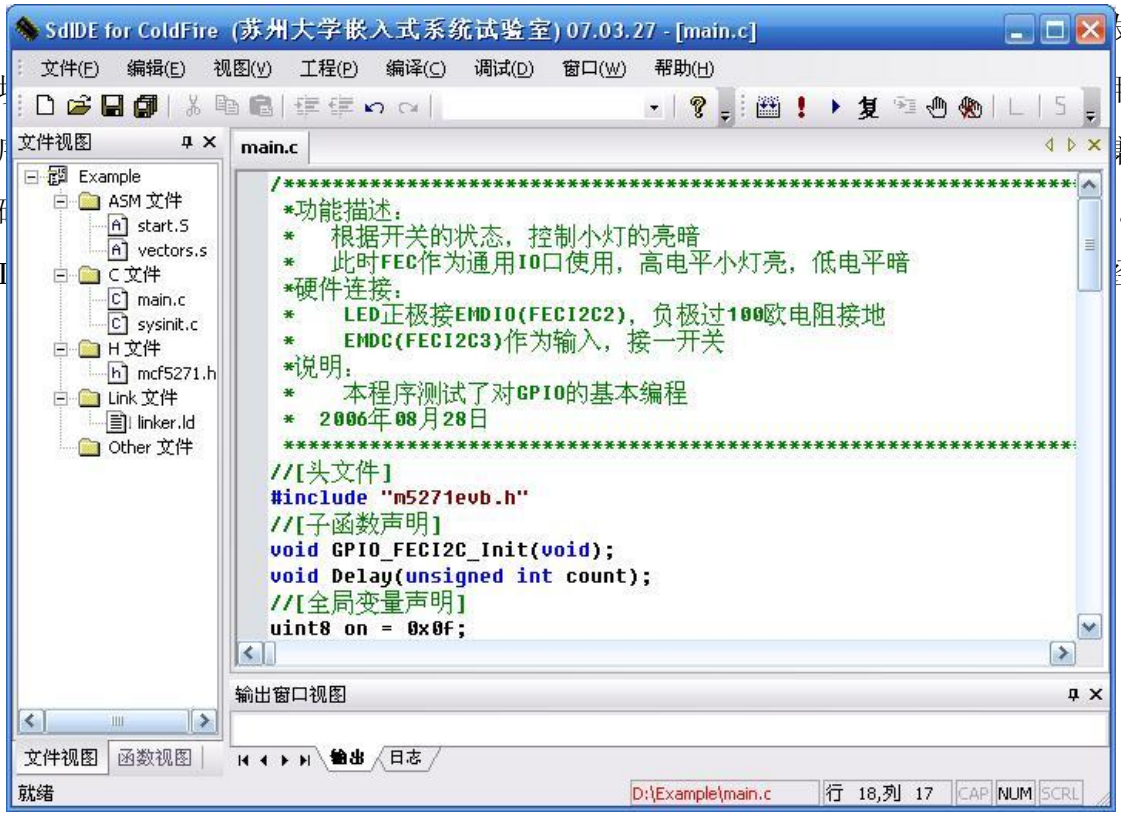


图 4-1 SdIDE for ColdFire

陆续开发出 SdIDE for HC08、SdIDE for ARM 等系列[28]。但由于当时并没有考虑到软件平台的芯片无关性，所以每开发一种针对新 CPU 的系统，都需要重新编写整个软件，软件不可复用。此次，本实验室在开发 ColdFire 和 S12 系列平台的集成开发环境时，对整个软件平台进行了重新设计。新的 IDE 在保留原来所有功能的基础上，充分考虑了软件的芯片无关性，将与芯片无关的部分独立出来，将与芯片有关的部分留出统一的接口。这样，再增加对新的 CPU 支持时，不需要对软件做任何修改，只需修改一些配置文件以及编写独立的接口部分软件，就可以完成对该系列芯片的支持。

4.1.2 SdIDE 通用模块

在设计新的 IDE 时，通过分析各种不同硬件平台之间的关系发现，芯片间的主要区别在于使用不同的交叉编译器、不同的代码写入程序。交叉编译器的不同可以通过通过配置 Makfile 文件中的参数解决，而对代码的写入部分，由于不同系统差别很大，可以通过调用独立的 Flash 操作外部程序的方法来解决；同时，软件的很多功能都可以通过修改相应的配置文件来选择，增加了其灵活性及多样性。SdIDE 在原来功能不变的基础上，根据用户在实际使用时发现的不足以及新的设计要求，主要对如下方面进行了改进。

(1) 增强了代码编辑功能。

① 新增对单个文件的导航功能。当打开一个文件时，可以列出该文件中所有的变量定义及函数声明，用户只要双击就会直接跳转到相应的位置；

② 新增了对整个工程的全局搜索支持。在对代码编辑或浏览代码时，经常要在所有文件中查找某个字符串，全局的搜索功能极大地方便了这一需求；

③ 支持对汇编指令码的高亮显示。

(2) 优化了编译功能。主要是对 Makefile 的书写进行了优化，使之自动生成文件的依赖性规则。

(3) 将代码写入部分与其他部分分离。即单独地编写一个下载代码的程序，提供代码的多种下载方式；有的芯片内存比较大，这样在调试目标代码的时候就可以先让程序在内存中运行，调试通过后，在写入到 Flash 中，避免了对 Flash 的重复擦写。

(4) 对菜单重新进行了分类，更符合用户的使用习惯。

SdIDE 对源文件以工程的形式组织，其整个的模块图如图 4-2 所示。图中的代码

写入模块用虚线标出，表示代码写入部分并不属于 SdIDE 通用部分的一部分，而只是在此提供了调用写入模块程序的接口。

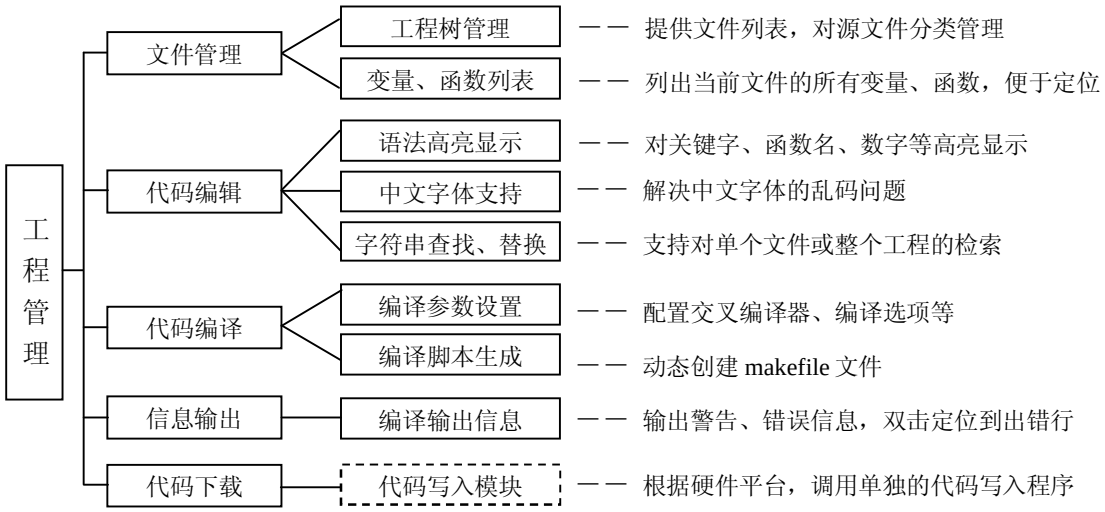


图 4-2 SdIDE 模块图

4.2 交叉编译器

不同系列的芯片在编译时使用的交叉编译器也不相同。如果要构建 ColdFire 的集成开发环境，首先必须要有 ColdFire 的交叉编译器。在软件的通用模块设计中，通过 Makefile 中的参数可以指定要使用的编译器的名称。本节主要介绍交叉编译器的基本知识以及创建交叉编译器的方法。

4.2.1 关于交叉编译器

所谓交叉编译器(Cross-Compiler)就是指可以在一个平台上生成另一个平台上的可执行代码的编译器。在嵌入式系统开发过程中，由于嵌入式系统本身的资源有限，一般都是采用一台宿主机运行交叉编译器，生成可以在目标芯片执行的可执行代码。目前，GCC(GNU C Compiler)是最常用的交叉编译器，它已成为开发众多 16 位和 32 位嵌入式处理器的首选工具。它支持多种不同类型的处理器，主要包括 ARM、MIPS、PowerPC、ColdFire/68K、M*Core 以及 x86 等[29]。使用可自由访问的源代码可以在 Windows 或 Linux 平台上构建一个完整的基于 GCC 的嵌入式工具链。这个工具链包括[30]：

- GNU GCC C/C++编译器

- 汇编器
- 连接器
- 嵌入式系统的标准 C 库
- GDB 代码级调试器(可选)

4.2.2 构建嵌入式开发的 GCC 工具链

ColdFire 的交叉工具链虽然可以从网上下载，但 GCC 的版本往往太老，尤其是针对 Windows 的版本更新更慢，这样在编译某些较新版本的程序(如 u-boot1.2.0)时就会编译不过。这时就需要用新版本的 GCC 重新构建交叉工具链。制作交叉编译器的时候，GCC 有所谓的 target，host 和 build 的概念。交叉编译器所生成的代码运行在 target 上面，交叉编译器本身运行在 host 上，而制作交叉编译器的机器是 build。一般来说，host 与 build 为同一台机器。因此可以在 x86-pc-windows 系统上制作一套 ColdFire 的交叉编译器，该编译器生成在 ColdFire 处理器上运行的代码[31]。

GCC 是 Unix/Linux 平台的编译工具，为了在 Windows 平台上构建 GCC 工具链，需要安装 Cygwin。Cygwin 是一个基于 DLL 的 Unix 仿真层(位于 Win32 之上)。它提供了 Unix 风格的环境，包括 Bash 外壳和 GNU 工具，这样就可以建立交叉编译器工具[32]。

值得注意的是使用 Cygwin 的最终的交叉编译器是动态连接 Cygwin DLL 的常规 Windows 可执行文件，不需要从 Cygwin 的 Bash 外壳运行最终的交叉编译器工具。

(1) 安装 Cygwin

Cygwin 目前由 Redhat 负责维护，可以到其网站上下载。由于我们只是使用 Cygwin 编译交叉编译器，所以只需要安装 cygwin 的最小组件。

- GCC —— 用来编译交叉编译器，主要是用到了 GCC-CORE
- bintuils —— GCC 用到的工具集，当选择 GCC 的时候，这一项会自动被选中
- make —— 工程编译管理器
- perl —— 一种脚本语言
- flex —— 用来模式识别(Used for pattern recognition)
- patchutils —— 用来对程序打补丁

(2) 下载需要的软件包

构建交叉编译器用到的软件包主要有：GCC，binutils 和 libc。GCC 的主要功能是 C 编译器，binutils 最重要的成员是汇编编译器和连接器，还包括一些二进制代码工具。libc 通常是 C 或 C++标准库。注意这三部分是彼此独立的，也就是说，GCC 并不是非要 binutils 中的工具，也可以使用其它汇编编译器和连接器，也可以使用其它 C 程序库。libc 可以有多种选择，在此选用 newlib。首先把下载好的软件包解开：

```
mkdir c:/build
cd c:/build
tar -jxf binutils-2.17.tar.bz2
tar -zxf gcc-4.1.1.tar.gz
tar -zxf newlib-1.14.0.tar.gz
mkdir build-bin
mkdir build-gcc
mkdir build-newlib
```

目录 binutils-2.17，gcc-4.1.1 和 newlib-1.14.0 会生成。建议把这三个软件包的源文件目录和编译目录分开，因此最好另建三个目录：build-bin，build-gcc，build-newlib。

(3) 开始编译

GCC 交叉编译器要分为四个阶段构建，这是因为有个死循环问题：构建交叉编译器首先需要构建库，但是库又不能在没有交叉编译器的情况下构建。这个问题解决方法就是建立一个最小的唯 C(C-only)编译器，它足够编译 newlib 库即可，然后重新建立一个完全的 C/C++。

制作交叉编译器的步骤是：

- 编译 binutils
- 编译一个最简单的 gcc
- 编译 libc
- 再次编译 gcc，生成功能完整的 gcc

① 编译 binutils

```
cd /cygdrive/c/build/build-bin
/cygdrive/c/build/binutils-2.17/configure \
      --target=m68k-elf \
      --prefix=/cygdrive/c/coldfire --nfp
make all install
```

--target=TARGET'

指定软件面向的系统平台。这主要在程序语言工具如编译器和汇编器上下文中起

作用。如果没有指定，默认将使用'--host'选项的值。

--prefix=PREFIX

'--prefix'是最常用的选项。制作出的 Makefile 会查看随此选项传递的参数，当一个包在安装时可以重新安置它的结构独立部分。上述命令将目标安装在 C:\coldfire 中。

--nfp：目标机器没有浮点数处理器。

② 编译最简 gcc

编译一个最简的 gcc，它只能编译 C 程序：

```
cd /cygdrive/c/build/build-gcc
/cygdrive/c/build/gcc-4.1.1/configure \
    --target=m68k-elf \
    --prefix=/cygdrive/c/coldfire \
    --with-newlib --without-headers \
    --enable-languages=c --disable-threads --nfp
make all install
```

③ 编译 newlib

再用刚才编译好的 gcc 和 binutils 编译 newlib：

```
cd /cygdrive/c/build/build-newlib
CFLAGS=-O2 CXXFLAGS=-O2
/cygdrive/c/build/newlib-1.14.0/configure \
    --target=m68k-elf --prefix=/cygdrive/c/coldfire --nfp
make all install \
    CC_FOR_TARGET=/cygdrive/c/coldfire/bin/m68k-elf-gcc \
    AS_FOR_TARGET=/cygdrive/c/coldfire/bin/m68k-elf-as \
    LD_FOR_TARGET=/cygdrive/c/coldfire/bin/m68k-elf-ld \
    AR_FOR_TARGET=/cygdrive/c/coldfire/bin/m68k-elf-ar \
    RANLIB_FOR_TARGET=/cygdrive/coldfire/bin/m68k-elf-ranlib
```

④ 再次编译 gcc

最后重新编译 gcc，这次它充分利用了刚编好的 newlib，可以支持 C 之外的其它语言了。

```
cd /cygdrive/c/build/build-gcc
/cygdrive/c/build/gcc-4.4.1/configure \
    --target=m68k-elf --prefix=/cygdrive/c/coldfire \
    --with-newlib \
    --with-headers=/cygdrive/c/build/newlib-1.14.0/newlib/libc/include \
    --enable-languages=c++ --disable-threads --nfp
make all install
```

4.2.3 Makefile

make 命令执行时，需要一个 Makefile 文件，以告诉 make 命令怎么样去编译和连接程序。Makefile 带来的好处就是——“自动化编译”，一旦写好，只需要一个 make 命令，整个工程完全自动编译，编译的时候编译器会根据文件的依赖性规则决定是否需要重新编译某些源程序，从而可以极大的提高软件开发的效率[33]。

(1) Makefile 的规则

一个简单的 Makefile 规则可以使用如下代码表示：

```
target: dependency file1 dependency file2 […]
    command1
    command2
    […]
```

上述规则中 target 是要创建的目标文件或者 Linux 系统支持格式的可执行文件。dependency file 是创建 target 需要的依赖文件列表。command 是创建 target 时使用的命令组，每个命令行的首字符必须是 Tab。创建目标文件时，make 会先检查依赖文件列表中的文件是否比 target 新，如果是，则根据下面的规则，重新编译目标文件；否则什么都不做。

(2) 文件的依赖性关系

一个目标文件往往依赖于多个文件，这些文件可以是.c 文件也可以是.h 文件。如

```
main.o: main.c init.c head.h
    m68k-elf-gcc –c main.c –o main.o
```

上面的 main.o 依赖于 main.c，init.c 和 head.h 这 3 个文件。当这 3 个文件中的任何一个被修改后，make 就会根据下面的命令，重新创建目标文件 main.o，这样就保证了目标文件可以及时的得到更新。

在编写 IDE 的过程中，要编译生成某个.o 文件时，它的依赖关系除了相应的.c 文件，还要找出这个.c 文件包含的所有.h 文件。只有这样，才能保证用户在对源文件修改之后，重新编译，能够得到更新后的目标代码。如何产生文件之间的依赖关系？老版本的 SdIDE 并没有解决这个问题，而是采取在每次编译前，将所有目标文件删除的方式，这样，每次编译时总是重新编译所有文件，造成编译效率不高。后来发现在使用 GCC 时，如果使用-M 选项，它会为每个 C 文件输出一个规则，并且这个规则符合 make 语法。这个规则将输出其依赖的所有头文件，包括被角括号(<>)和双引

号(“”)所包含的文件。如果可以肯定系统头文件不会被更改，可以用-MM 来代替-M 传递给 GCC，角括号包含的系统头文件将不会被包含。

```
#定义编译器
CC= m68k-elf-gcc
#定义连接器
LD= m68k-elf-ld
#定义目标文件的依赖关系
all.elf: dep main.o io.o
   $(LD)    –o all.elf main.o io.o
#定义隐含规则，所有的.c 编译成.o 均使用此规则
%.o:%.c
$(CC) -g -c $< -o $@
#以下生成源文件的依赖规则
dep:
      sed '/\#\#\# Dependencies/q' < Makefile > tmp_make
      (for i in *.c;do echo -n ;$(CC) -MM $$i;done) >> tmp_make
      cp tmp_make Makefile
      rm tmp_make
### Dependencies:
```

执行 make dep 时，将生成各文件之间的依赖关系。方法如下：使用字符串编辑程序 sed 对 Makefile 文件(这里即是自己)进行处理，输出为删除 Makefile 文件中'### Dependencies'行后面的所有行，并生成 tmp_make 临时文件。然后对当前目录下的所有 C 文件(当前目录中包含 main.c 和 io.c 两个文件)执行 GCC 预处理操作，-MM 标志告诉预处理程序输出描述每个目标文件相关性的规则，并且这些规则符合 make 语法。对于每一个源文件，预处理程序输出一个 make 规则，其结果形式是相应源程序文件的目标文件名加上其依赖关系——该源文件中包含的所有头文件列表。把预处理结果都添加到临时文件 tmp_make 中，然后将该临时文件复制成新的 Makefile 文件，并删除临时的 tmp_make 文件[34]。执行后，Makefile 改变为：

```
#定义编译器
CC= m68k-elf-gcc
#定义连接器
LD= m68k-elf-ld
#定义目标文件的依赖关系
all.elf: dep main.o io.o
   $(LD)    –o all.elf main.o io.o
#定义隐含规则，所有的.c 编译成.o 均使用此规则
%.o:%.c
$(CC) -g -c $< -o $@
#以下生成源文件的依赖规则
```

```
dep:
    sed '/\#\#\# Dependencies/q' < Makefile > tmp_make
    (for i in *.c;do echo -n ;$(CC) -MM $$i;done) >> tmp_make
    cp tmp_make Makefile
    rm tmp_make
### Dependencies:
io.o: io.c mcf5271.h
main.o: main.c mcf5271.h uart.h io.h
```

可以看到文件的最后增加了关于 io.o 和 main.o 的依赖性规则。这样当源文件被修改后，make 会自动调用已定义的隐含规则重新生成相应的.o 文件。

4.3 连接器

连接器是把编译器生产的多个目标代码文件联合在一起，生成可执行文件的工具[35][36]。在编写基于操作系统运行的文件时，连接器生成的可执行文件可以被操作系统进行地址重定位，操作系统会根据自身的资源分配该程序的运行空间，然后将程序装载到 RAM 中并运行之。这时，连接器工作时并不需要用户的干预。而在编写裸机程序时，系统的所有资源都是由程序员分配的，而且连接器生成的可执行文件直接被执行，所以在连接的时候，要事先告诉连接器可以使用的地址空间范围，然后连接器才能够生成符合条件的可执行文件。

以下小节中，将讲述有关连接器的基本知识，并结合针对 ColdFire 的连接脚本，阐述 SdIDE 如何生成可以在 SDEVB5271 上运行的可执行文件。

4.3.1 目标文件格式

在嵌入式系统中，常用的目标文件格式主要有 3 种：

COFF(Common Object File Format)：一种通用的对象文件格式；

ELF(Excutive Linked File)：一种为 Linux 系统所采用的通用文件格式，支持动态连接；

FLAT：ELF 格式有很大的文件头，FLAT 文件对文件头和一些段信息做了简化。

目前，ELF 文件是广泛使用的一种目标文件格式，GCC 交叉工具集采用的就是这种格式的目标文件。根据 ELF 文件的功能不同，ELF 可以分为三种主要类型：

(1) 适于连接的可重定位文件(Relocatable File)，可与其它目标文件一起创建可执

行文件和共享目标文件。这种文件一般是由编译器直接编译生成，而没有经过连接器连接，也可称为输入文件。执行 m68k-elf-gcc –c main.c –o main.o 后，生成的 main.o 就是这种类型的文件。

(2) 适于执行的可执行文件(Executable File)，用于提供程序的进程映像，可加载到内存执行。这种文件是由连接器连接后生成的文件，也可称为输出文件。执行 m68k-elf-ld hello.elf main.o 后，生成的 hello.elf 就是这种类型的文件。

(3) 共享目标文件(Shared Object File)，连接器可将它与其它可重定位文件和共享目标文件连接成其它的目标文件，动态连接器又可将它与可执行文件和其它共享目标文件结合起来创建一个进程映像。

不管什么类型的目标文件，一些基本的要素是必须的，即文件中应包含代码和数据。因为文件可能引用外部文件定义的符号(变量和函数)，因此重定位信息和符号信息也是需要的。一些辅助信息是可选的，如调试信息、硬件信息等。基本上任意一种目标文件格式都是按区间保存上述信息，称为段(Segment)或节(Section)。段中包含了 4 类内容：代码、已经初始化的数据、未经过初始化的存储区域、内容初始化成 0 的存储区域。每个段有相应的属性，可以为只读的、可读写的以及初始化成 0 的。

对象文件中的每一个段都有名字和大小。大多数的段还有一个相连的数据块，称之为“Section Contents”。一个被标记为可加载(Loadable)的段，意味着在输出文件运行时，Contents 可以被加载到内存中。既不是可加载的又不是可分配的(Allocatable)段，通常包含了某些调试信息。

每个可加载或可分配的输出段(Output Section)都有 2 个地址。第一个是虚拟存储地址 VMA(Virtual Memory Address)，这是在输出文件执行该段时使用的地址。第二个是加载存储地址 LMA(Load Memory Address)，这是该段被加载时的地址。基于 RAM 的系统中，这两个地址是相同的。而在基于 ROM 的系统中，当一个数据段加载在 ROM 中，后来在程序开始执行时又拷贝到 RAM 中，这时 ROM 地址是 LMA，而 RAM 地址是 VMA。

4.3.2 连接脚本

每次连接都由连接脚本控制着，脚本由连接器命令语言组成。如果没有提供一个指定的脚本文件的话，连接器会使用一个缺省的脚本。脚本的主要目的是描述如何把

输入文件中的段映射到输出文件中，并控制输出文件的存储布局。

连接脚本使用的最重要的一个命令就是“SECTIONS”，使用该命令来描述输出文件的内存布局。

在 SdIDE 集成开发环境中，只需连接器完成一些基本功能，所以连接脚本并不复杂，现在主要描述连接脚本需要完成的主要任务：

① 将中断向量表置于 Flash 的开始处。ColdFire 处理器上电复位后，首先会从 Flash 的起始位置读取 4 字节数据放入 SP 寄存器，接着再读取 4 字节的数据放入 PC 寄存器中，然后执行 PC 寄存器中的指令。

② 为初始化代码提供一些存储分配信息，如数据段的起始、结束地址等。在系统复位后，RAM 中不包含任何程序和数据，这时所有的程序和数据都保存在 Flash 中。CPU 在执行应用程序之前，要先把数据段中的数据拷贝到 RAM，并初始化部分 RAM。初始化代码中使用的地址变量值在连接的时候由连接器分配。

③ 为一些用户可配置寄存器提供参数。ColdFire 处理器中，如 IPSBAR(片内外设基址寄存器)可以由用户决定片内外设的起始地址，这个地址当然也可以在程序中直接给定，但为了增强代码的可移植性及灵活性，可以在连接脚本中给出，这样不需要修改源代码，也就是说源代码不需要重新编译，只要重新连接一次，就可以生成不同功能的代码。

SECTIONS 告诉编译器怎么将输入段映射到输出段，并指定输出段的存储位置[37]。命令格式为：

```
SECTIONS
{
    sections-command
    sections-command
        … …
}
```

每个 sections-command 可以是“ENTRY”命令、符号赋值、输出段描述和覆盖描述。这里只用到了输出段描述，完整的输出段描述格式如下：

```
section[address][(type)]:[AT(lma)]
{
  output-section-command
  output-section-command
    … …
}[>region][AT>lma region] [:phdr:phdr…][=fillexp]
```

在连接脚本中，使用 MEMORY 命令来告诉连接器哪些区域可以使用，哪些区域不能使用，配合输出段描述命令一起使用。下面给出适合 ColdFire 的连接文件，连接器将生成适合在 ROM 中执行的代码。其中的 data 数据段即占据 RAM 空间，也占据 ROM 空间。

```
/*定义输出目标文件 CPU 的体系结构*/
OUTPUT_ARCH(m68k)
/*存储器地址空间*/
MEMORY
{
   flash: ORIGIN = 0xFFE00000, LENGTH = 0x00200000  /*2M*/
   sram: ORIGIN = 0x20000000, LENGTH = 0x00010000  /*64k*/
   sdram:ORIGIN = 0x00000000, LENGTH = 0x01000000  /*16M*/
}
/*输入文件中的段的存储器映射*/
SECTIONS
{
   /*   代码段*/
   .text :
   {
      *(.romvec)
      _eromvec = ALIGN(0x4);
      *(.text)
      *(.rodata)
   } > flash       /*          如果是映射到 SDRAM 空间，只要把此处改为”sdram”*/
   /*   数据段*/
   _data_ROM = .;   /*      数据段在 Flash 中起始地址(“.”为程序计数器)*/
   .data : AT(_data_ROM)
   {
      copy_start = .; /*      数据段在 RAM 的起始地址*/
      *(.data)
      *(.shdata)
      copy_end = .; /*      数据段在 RAM 的结束地址*/
   } > sdram
   /*   未初始化数据段*/
   .bss :
   {
      bss_start = . ;  /*        未初始化数据段在 RAM 的起始地址*/
      *(.bss)
      *(.shbss)
      *(COMMON)
      bss_end = .;   /*        未初始化数据段在 RAM 的结束地址*/
   } > sdram

   . = . + 0x5000;
   PROVIDE(__SP_INIT = .);   /*     初始化堆栈*/
```

```
}
/*定义初始化时用到的系统参数*/
__IPSBAR = 0x40000000;  /*  设置片内外设基址*/
__SRAM  = 0x20000000;  /*SRAM    的起始地址*/
__SRAM_SIZE = 0x00010000; /*SRAM的大小*/
__FLASH = 0xFFE00000;  /*Flash   的起始地址*/
__SDRAM = 0x00000000;  /*SDRAM   的起始地址*/
```

4.4 代码的写入及运行

程序编译完成之后，就要将生成的代码下载到目标芯片执行。不同的芯片由于其内部寄存器不同以及 Flash 类型不同，代码下载的方式也不相同；甚至采用同种芯片的电路板因为存储扩展的不同也会有很大差异。考虑到 SdIDE 的平台无关性及可扩展性，将这部分单独地编写一个子程序，来实现代码的写入及运行。这样，不同的芯片可以根据自身的特性编写功能不同的下载程序，如支持下载到 Flash 或 RAM 等不同的存储介质中执行。同时，代码下载的程序也可以单独发布供用户使用。

SdIDE for ColdFire 目前实现了对 SDEVB5271 代码下载支持，可以使用 S-Records[38]或 bin 的文件格式。其他芯片的代码下载方式与此基本相同，可以很容易添加。本节将以 MCF5271 为例介绍下载代码的常用方法及程序实现。

4.4.1 烧写 Flash

ColdFire 系列芯片内部基本都不含有 Flash，为了存储程序和数据，必须通过外扩 Flash 的形式。SDEVB5271 采用的 Flash 芯片是 2M 的 AM29LV160DB。要实现对 Flash 的操作，主要是对 AM29LV160DB 内部的命令寄存器按一定的时序写入命令。通过这些命令或命令序列可以实现数据读取、整体擦除和数据写入等功能。表 4-1 列出了 AM29LV160DB 中与擦除、写入相关的命令序列。

表 4-1 AM29LV160DB部分命令序列表

命令	周期	第一命令		第二命令		第三命令		第四命令		第五命令		第六命令	
		地址	数据	地址	数据	地址	数据	地址	数据	地址	数据	地址	数据
读取	1	RA	RD										
写入	4	555	AA	2AA	55	555	A0	PA	PD				
整体擦除	6	555	AA	2AA	55	555	80	555	AA	2AA	55	555	10
扇区擦除	6	555	AA	2AA	55	555	80	555	AA	2AA	55	5A	30

注：RA，RD，PA，PD 分别表示将要读取的地址或数据，和将要编程的地址和数据。

在 BDM 模式下，可以像写普通内存一样直接对 AM29LV160DB 的命令寄存器进行操作。在第三章中已经介绍了如何编写 BDM 驱动，通过 BDM 提供的接口函数可以很容易实现对 Flash 的操作。

```
void write_word(unsigned long addr, unsigned short data)
{
    pBDM_CMD->write_data_byte(0xffe00aaa,0xaa); //    执行第一命令
    pBDM_CMD->write_data_byte(0xffe00555,0x55); //    执行第二命令
    pBDM_CMD->write_data_byte(0xffe00aaa,0xa0); //    执行第三命令
    pBDM_CMD->write_data_word(addr,data);   //        执行写入
}
```

因为 Flash 的地址空间映射在 0xFFFE 0000 ～0xFFFF FFFF 范围内，所以 Flash 的命令寄存器必须加上 Flash 的基址 0xFFFE 0000。

不同的评估板可能采用的 Flash 芯片也不相同，但对 Flash 的操作都是向命令寄存器写入一些命令序列。为了适应不同的 ColdFire 的评估板，Flash 写入软件可以让用户根据自己板子的硬件配置各种参数，如目标 CPU 的类型，Flash 的型号以及要写入的 Flash 的基址等，Flash 写入程序的界面如图 4-3 所示。

图 4-3 Flash 写入程序

4.4.2 程序在 SDRAM 中运行

一般来说，32 位微处理器的 RAM 空间比较大，MCF5271 片内有 64K 的 SRAM，而且扩展板又扩展了 2 片 SDRAM，使 RAM 空间达到 16M。用户在调试程序时，可以不将程序下载到 Flash 中执行，而在 RAM 空间直接运行。这样不仅减少了对 Flash 的重复擦写，而且写 RAM 的时间比写 Flash 的时间快的多，方便了用户调试程序。

芯片复位后，SDRAM 尚未初始化，所以此时 SDRAM 还不可以使用。为了将代码写入到 SDRAM 中，必须首先通过 BDM 头发送命令对 SDRAM 有关的寄存器进行配置，下面列出主要的程序代码。

```
//使中断向量寄存器 VBR 指向 SDRAM 的起始处
//VBR 的地址为 0x0801
put_control(cf_sysreg_map[BDM_REG_VBR], 0x00000000);

//设置 RAM 的基址寄存器为 0x20000000
//RAMBAR 的地址为 0x0C05
put_control(cf_sysreg_map[BDM_REG_RAMBAR], 0x20000001);

//写 PAR_SDRAM，使能 SDRAM 信号
//PAR_SDRAM的地址 IPBAR＋0x100046 0xFF
write_data_byte(IPBAR+ PAR_SDRAM,0xff);

//写 PAR_AD，使 SDRAM 工作在 32 位数据宽度
//PAR_AD的地址为 IPBAR＋0x100040
write_data_byte(IPBAR+ PAR_AD,0xe1);

//关看门狗
//WCR 的地址为 IPBAR＋0x100000
write_data_word(IPBAR+ WCR,0x0000);

//写 SDRAM 配置寄存器，最高的 CAS 地址为 A7，SDBA1、SDBA0 分别对应地址线 A24、A23
write_data_word(0x40000040, 0x0446);
//DACR0 SDRAM 地址和控制寄存器 0，开启自动刷新
write_data_long(0x40000048, 0x00001300);
//DMR0 SDRAM 屏蔽寄存器 0，设置 SDRAM 控制器使用的地址线，取消写保护
write_data_long(0x4000004C, 0x00FC0001);
//DACR0 SDRAM 地址和控制寄存器 0，预充电所有存储体
write_data_long(0x40000048, 0x00001308);

//往 SDRAM 空间中写数据，初始化 SDRAM
write_data_long(0x00000000, 0x00000000);
```

至此，SDRAM 已初始化完毕，可以将代码写入到 SDRAM 中了。代码写入完毕后，将 CPU 复位，因为此时已经将中断基址寄存器设置为了 SDRAM 的起始地址，所以 CPU 复位后，就会从 SDRAM 的中的代码开始执行。

4.4.3 监控程序

以上两种下载程序的方法都需要借助于 BDM 调试头。由于 BDM 调试头需要连接 PC 机并口，所以使用起来不是很方便。为此，特开发了基于串口下载代码的监控程序。

监控程序是运行在评估板 Flash 中的一段程序，该程序在系统上电或复位时自动运行，运行的过程中会通过串口与 PC 进行通讯，如果 PC 要往评估板中下载程序，那么 PC 机就会和监控程序进行握手，握手成功之后 PC 机就把要下载的用户程序发送到 CPU，监控程序会把从串口接收到的用户程序烧写到 CPU 的片外 Flash 中，下载完毕之后监控程序就自动跳转到外部 Flash 中的用户程序区执行。如果监控程序在运行了一小段时间之后没有和 PC 机实现握手，那么监控程序就直接跳转到外部 Flash 的用户程序区执行用户程序。

监控程序可完成很多功能，如调试程序、硬件检测、操作系统引导等。本课题中，只实现了监控程序通过串口将程序写入到外部 Flash 中，并执行 Flash 中的程序。监控程序首先设置中断向量表，初始化内部 SRAM，初始化堆栈，初始化系统模块，初始化串行通讯口，然后在 5 秒内监视串口，一旦接收到数据则判断指令，并跳转执行相应的功能，其具体运行流程如图 4-4 所示。以后，可以考虑在监控程序中移植一个 GDBStub(网上的开源项目)，配合 PC 方的软件，实现在线调试的功能。

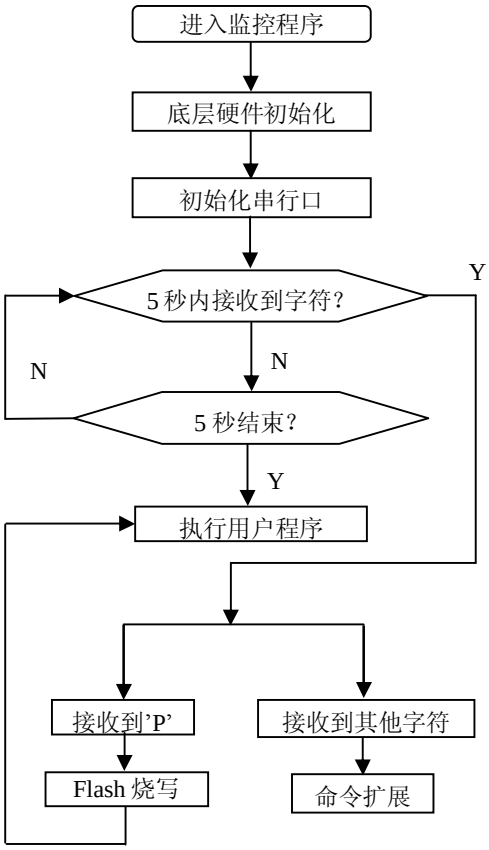


图 4-4 监控程序执行流程

4.5 本章小结

SdIDE for ColdFire 的开发是本课题的重点之一。本章给出了整个 IDE 的设计过程，并实现了 IDE 的硬件平台无关性。在扩展 IDE 对新架构 CPU 的支持时，不需要修改通用部分的源代码，只需要将准备好的交叉编译器放在指定目录并修改相应的配置文件，然后编写相应的代码下载软件即可。本章中以增加对 ColdFire 系列芯片的支持为例，详细的介绍了如何构建、使用 GCC 交叉开发工具。熟悉交叉开发工具的使用，对于以后在 Linux 平台做嵌入式开发也有极大的好处。本章最后给出了下载调试裸机程序的常用方法，为开发其他平台的类似软件提供了参考。

第五章 评估板应用编程示例

本章介绍了评估板的程序设计方法。首先阐述了嵌入式应用程序设计的基本知识，然后通过一个 IO 口的实验程序介绍了使用 SdIDE 进行开发的基本流程，最后讲述了 μCLinux 的相关知识以及移植方法。

5.1 嵌入式应用程序设计的基本知识

目前，嵌入式开发常用 C 语言。我们常说 C 语言的 main 函数是程序的入口点，其实 C 程序在真正开始运行之前，还需要完成初始化运行环境及传递参数给 main 函数等工作，这些都是在 crt0 即 the C run-time initialization object file 中实现的。crt0.o 是应用程序编译连接时需要的启动文件，在程序连接阶段被连接。主要工作是初始化应用程序栈，初始化程序的运行环境和在程序退出时清除和释放资源。在嵌入式程序中，crt0 中还包含初始化硬件的代码。

5.1.1 硬件系统初始化

由于不同的硬件系统之间差别很大，其初始化代码也不相同。虽然 crt0.o 中留有硬件初始化的入口，但是为了编程的灵活以及使程序流程更清晰，本设计中自己编写了 Start.S 文件，来实现和 crt0 类似的功能。

以下程序代码主要完成的功能有：

① 初始化片内外设基址寄存器；

② 使能内部 SRAM，并将堆栈设置在 SRAM 的高端；

③ 调用 MCF5271 的各个内部模块的初始化函数，主要包括初始化看门狗、设置系统时钟、设置片内 SRAM、设置片选、设置串口、设置 SDRAM 等；

④ 重新设置堆栈指针，使其映射在 SDRAM 空间。

```
/*程序入口*/
start:
    move.w  #0x2700,%sr/* 屏蔽所有中断，CPU 复位后执行的第一条指令*/

    /*初始化 IPSBAR */
    move.l  #(__IPSBAR + 1),%d0
```

```
move.l  %d0,0x40000000

/*初始化 RAMBAR1: locate SRAM and validate it */
move.l  #(__SRAM + 0x21),%d0
.long   0x4e7b0C05      /* movec d0,RAMBAR1 */

/*设置临时堆栈指针，使其指向 SRAM*/
move.l  #__SRAM,%d0
add.l   #__SRAM_SIZE,%d0
move.l  %d0,%sp

/*初始化 MCF5271 的各个模块*/
jsr     mcf5271_init

/*重新设置堆栈指针*/
move.l  #__SP_INIT,%sp
```

5.1.2 应用程序初始化

应用程序在运行之前必须对运行环境进行必要的初始化，应用程序的初始化主要包括下面两方面的内容：

① 将已经初始化的数据(如赋有初值的全局变量)搬运到可写的数据区，这个区也称为 data 段。在基于ROM 的嵌入式系统中，已经初始化的数据在映像文件运行之前保存在 ROM 中，在程序运行过程中这些数据可能需要修改。因而，在映像文件运行之前需要将这些数据搬运到可写的数据区，即 RAM 空间中。

② 在可写存储区建立 ZI 属性(Zero Initialize，如未初始化的全局变量，映像文件运行前，用0 初始化)的可写数据区，这个区也称为 bss 段。通常在映像文件运行之前，

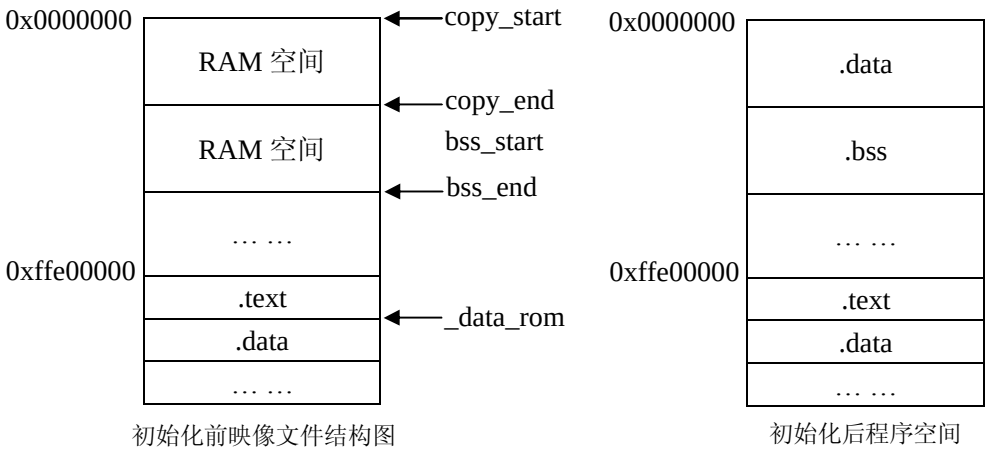


图 5-1 程序存储空间变化示意图

也就是保存在 ROM 中时，映像文件中不包含 ZI 属性的数据。运行映像文件时，在系统中可写的存储区域建立 ZI 属性的数据区。

在 Start.S 中，程序将保存在.data 段中的数据搬运到 RAM 中，并将.bss 段占据的存储空间全部清零。其主要实现代码如下，其中的 copy_start、copy_end、bss_start、bss_end、_data_rom 的值由连接器在连接的时候指定。执行后，程序的存储空间变化如图 5-1 所示。

```
/*初始化数据段：从 ROM 中向 RAM 拷贝数据*/
move.l  #_data_rom, %a0
move.l  #copy_start, %a1
move.l  #copy_end, %a2
cmp.l   %a1, %a2
beqs    1f     /*NOCOPY*/

COPYLOOP:
      move.l  (%a0)+, %d0
      move.l  %d0, (%a1)+
      cmp.l   %a1, %a2
      bhi     COPYLOOP   /*       大于转移*/

1:
      /*清零未初始化数据段*/
      movel  #bss_start, %d1
      movel  #bss_end, %d0
      cmpl  %d0, %d1
      jbeq  3f
      movl  %d1, %a0
      subl  %d1, %d0
      subql  #1, %d0
2:
      clrb  (%a0)+
      subql  #1, %d0
      jbpl  2b
3:
      /*其他代码*/
```

5.2 通用 IO 口的编程实例

通用 I/O 口实验是学习一款微处理器的第一步，类似于高级语言中的“Hello World”程序。通过这个实验，可以了解整个硬件的初始化过程、集成开发环境的使用和编程的基本框架。本节以使用 SdIDE for ColdFire 集成开发环境编写 I/O 口实验

程序为例，分析编写程序的基本思路和方法。

5.2.1 编程基础

早期的嵌入式应用主要是通过各种外围电路与接口电路，实现对应用对象的智能控制。面向这种应用的主要是一些 8 位、16 位的微控制器，它们提供了丰富的 I/O 口资源，用户可以通过这些 I/O 口实现对外接电路的控制。随着嵌入式应用领域的不断深入，人们需要对现实世界中大量的数据进行处理，因此出现了运算速度更快，处理能力更强的 32 位微控制器。这类芯片有很强的数据处理能力，速度一般在 30MHz～500MHz，因此也称为微处理器。这类芯片内部集成很多内部模块，用户使用芯片提供的外部接口引脚，就可以使用这些模块提供的功能。一般这些模块的引脚和其他模块引脚都是复用的，在不使用这些外部接口引脚时，为了使资源不浪费，这些引脚可以定义为通用 I/O 引脚，可以用来对外部电路进行控制。

在对 I/O 口编程前，首先应对整个系统的 I/O 资源数有所了解。本实验平台采用的主控 CPU 是 160 脚的 MCF5271，共有 55 个通用 IO 口，而 196 引脚的 MAPBGA 封装有 97 个通用 I/O 口，如表 5-1 所示。实验中选用 FEC 的 EMDC 和 EMDIO 作为 PFECI2C3、PFECI2C2 使用。

硬件连接：PFECI2C3 接一开关，PFECI2C2 接一小灯。

实验目的：一个 IO 口作为输入，读取开关的状态，然后根据开关的状态，通过另外一个 IO 口实现对小灯亮暗的控制。

表 5-1 GPIO 引脚的复用信号

复用模块	GPIO信号	复用引脚	数量
外部存储扩展	PADDR[7:5]、PDATAH[7:0]、PDATAL[7:0]、 PBS[7:4] 、PBUSCTL7 、 PBUSCTL6 、PBUSCTL4、PBUSCTL1	A[23:21] 、D[15:8]、D[7:0]、BS[3:0]、R/W、TS	27
片选信号	PCS[3:1]	CS[3:1]	3
SDRAM 控制器	PSDRAM[5:2]	SD_WE 、 SD_SCAS 、SD_SRAS、SD_CKE	4
外部中断口	PIRQ7、PIRQ4、PIRQ1	IRQ7、IRQ4、IRQ1	3
FEC(I2C)	PFECI2C3、PFECI2C2	EMDC、EMDIO	2
QSPI	PQSPI[3:0]	QSPI_CS0 、 QSPI_CLK 、QSPI_DIN、QSPI_DOUT	4
UARTs	PUARTL[7:0]	U1CTS、U1RTS、U1TXD、U1RXD、U0CTS、U0RTS、	8

		U0TXD、U0RXD	
DMA 定时器	PTIMER5、PTIMER3 PTIMER[1:0]	DT2IN、DT1IN、DT0IN、 DT0OUT	4

5.2.2 编程步骤

① 写引脚配置寄存器(PAR_x)。清零对应位，设置 PFECI2C3、PFECI2C2 引脚的功能为 GPIO。寄存器中的各位与引脚的对应关系如图 5-2 所示。这里的每一引脚对应着配置寄存器的 2 位，这是因为这些引脚不只 2 种功能。如 PAR_EMDC 还可以用作串口 2 的数据发送口(U2TXD)及 I^2C 的时钟引脚(I2C_SCL)。

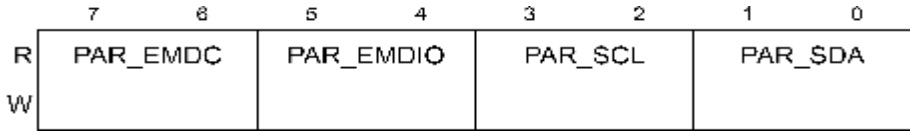


图 5-2 FEC/I2C 引脚配置寄存器(PAR_FECI2C)

```
// PFECI2C3、PFECI2C2 作为 GPIO 使用
MCF_GPIO_PAR_FECI2C    &=  0x0F;
```

② 写端口方向寄存器(PDDR_x)。设置 PFECI2C3、PFECI2C2 为输入还是输出。PDDR_FECI2C 的使用如图 5-3 所示。其中第 3 位对应 EMDC，第 2 位对应 EMDIO，第 1 位对应 SCL，第 0 位对应 SDA。

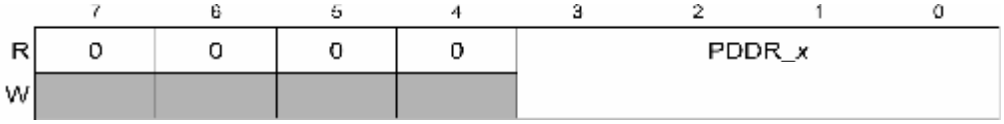


图 5-3 FEC/I2C 端口方向寄存器(PDDR_FECI2C)

```
MCF_GPIO_PDDR_FECI2C  &= 0xF7;   //        清对应位，PFECI2C3 作为输入
MCF_GPIO_PDDR_FECI2C  |= 0x04;   //        置对应位，PFECI2C2 作为输出
```

③ 通过端口数据/置位寄存器(PPDSDR_x)，读取输入端口上的数据。在控制输出引脚时，也可以使用该寄存器，但只能通过向该寄存器写 1，驱动对应引脚为高电平；向该位写 0 时，无影响。为了可以控制 PFECI2C2 输出状态，可以直接写端口输出数据寄存器(PODR_x)。这两个寄存器的位对应关系和 PDDR 的相同。

```
while(1)
{
    Data = MCF_GPIO_PPDSDR_FECI2C & 0x08; //对 PFECI2C3 的电平
    if ( Data == 0 )
    {
        MCF_GPIO_PODR_FECI2C  &= 0xFB;      //          输出低电平
    }
    else
    {
        MCF_GPIO_PODR_FECI2C  |=  0x04;     //          输出高电平
    }
}
```

5.2.3 程序测试与固化

为了验证以上编写的程序是否正确，下面将通过 SdIDE 编译源程序并下载到目标板上执行。选择“工程”－“新建工程”，新建一个 MCF5271 的模板工程，这个模板工程中已经包含了系统初始化的代码、一个空的 main 函数以及一个默认的连接脚本文件，用户可以直接在 main 函数里添加用户程序。将上述代码放在 main 函数里就可以编译了。

首先将代码映射在 SDRAM 空间，修改连接脚本，将 text 段中的内容定向到 SDRAM 中，然后编译生成的代码即可通过 IDE 提供的写入界面下载的评估板的 RAM 中并自动执行。

程序反复检测 PFECI2C2 所接开关的状态，当它为高电平时，PFECI2C3 使小灯亮，反之，则小灯熄灭。测试通过后，可以利用 SdIDE 提供的 Flash 写入功能将代码固化在评估板的 Flash 中，这样每次上电后，程序就会自动从 Flash 启动。修改连接脚本，将 text 段中的内容定向到 Flash 中，编译后将代码烧入 Flash 中。

5.3 μClinux 的板级移植

由于 32 位微处理器处理速度快，可以进行总线扩展等特点，所以在处理复杂任务时，可以考虑移植一个嵌入式操作系统。基于操作系统的应用系统具有多任务、响应快、应用程序编写简单、可移植性好、系统运行稳定可靠等优点。

操作系统的移植根据难度及工作量主要可分为两种：芯片级操作系统移植和板级操作系统移植。

芯片级操作系统移植就是将某种嵌入式操作系统应用在基于特定芯片的系统上。这种形式也可分为两种，一种是针对一款新的 CPU 架构，这种操作系统从来没有在此种构架的 CPU 上实现过；另一种是同种 CPU 架构下的不同芯片，如 ColdFire 系列有 MCF5206、MCF5249、MCF5271 等，如果已经有了 MCF5249 的移植版本，然后再去针对 MCF5271 做移植，将会容易的多。

板级操作系统移植是指已经有某种 CPU 的移植代码，但这只是对一种具体的硬件板而设计的，当硬件板改变了，操作系统也要作相应的改动，使之适合特定的应用。32 位微处理器一般可以通过总线扩展的方式增加很多外部资源，如存储器、液晶屏、各类板卡等，这也造就了同种型号的 CPU 的硬件电路板多样性。这种移植不需要对 CPU 的核心代码修改，主要是针对不同的外设编写或修改相应的驱动程序即可。

本节将通过在 SDEVB5271 移植 μClinux 为例，分析在移植 μClinux 时主要的工作以及方法。

5.3.1 bootloader 开发与移植

bootloader 引导程序是嵌入式开发很重要的组成部分，它是 CPU 加电后第一段开始执行 的代码，由它最终将操作 系统启动起来并将控制权交给操作 系统内核。bootloader 引导程序最基本的功能是进行硬件的初始化(包括 CPU 的主频，SDRAM，中断，串口等)和内核启动参数的设置并启动内核等。功能强大的bootloader 可以和主机进行交互，从串口、USB 口或者以太网口下载映像文件，并可以对 Flash 等存储设备进行管理。bootloader 可以利用已有的开放源代码的启动代码进行改动移植到自己的硬件板上。通常，bootloader 严重依赖于硬件，特别是在嵌入式系统领域。

Freescale 为自己的 ColdFire 设计了经典的 dBug 程序，可以完成 bootloader 的所有任务，甚至支持从网络上下载 μClinux。dBug 的源代码可以从 Freescale 的网站上下载。为了简化 bootloader 的功能，本课题重新设计了一个 bootloader。bootloader 完成的任务主要为：系统初始化、将操作系统镜像从 flash 拷贝到 RAM 中，然后从 RAM 启动操作系统。用到的文件可以从 dBug 中提取，主要文件包括：mcf5200.h、mcf5271.h、mcf5271evb.h、start.S、sysinit.c、board.c、main.c、printk.c。由于这些文件是针对 Freescale 提供的标准 MCF5271EVB 的，它使用的 MCF5271 是 196 脚 MAPBGA 封装，而本文使用的是 160 脚 QFP 封装，因此硬件电路并不完全相同，相应的初始化代码也要做

相应的改变，如 SDRAM 部分的代码。在 main.c 中主要实现引导操作系统的功能。其核心代码如下：

```
printf("Copy begin....! \r\n");
while(count)
{
    *dest++ = *src++;  /*  每次拷贝一个长字*/
    count -= 4;        /*       剩余字节数*/
}
printf("Copy end....! \r\n");
asm{jmp 0x20000}
```

5.3.2 μClinux 的移植

有了 bootloader 之后，就可以开始移植 μClinux。由于这个 μClinux 版本已经有针对 MCF5271 的移植，所以需要做的修改不会很多，它应该是最合适移植的原始版本。

(1) μClinux 的目录结构

μClinux 的源代码使用的是从 μClinux 官方网站 http://www.uclinux.org/Pub/uClinux/dist/ 下载的 uClinux-dist-20051014.tar.gz，μClinux 内核源程序的目录分布主要为[39][40]：

linux-2.0.x/，linux-2.4.x/，linux-2.6.x/：存放 μClinux 各个版本的内核源码，不同的内核版本支持的平台数量不同，对于 MCF5271 来说，2.4、2.6 的内核已经移植好了；

glibc/lib/uClibc/：库文件。glibc 是标准的 GNU 应用程序库，uClibc 是专门针对 μClinux 的应用程序库它比 glibc 小得多，但是对原功能有很好的支持，对于 MCF5271，内核不支持 glibc；

user/：放置一些在普通用户模式下的应用程序，这部分并不属于内核部分，而是文件系统中包含的常用程序，如 busybox 工具集、gdbserver、tftp 等；

vendors/：一些厂商及平台的详细信息及支持文档；

documents/：一些有用的帮助文档；

config/：目前支持的评估板的配置信息。

下面以内核 2.4.x 为例详细介绍内核源文件的结构，该版本的 μClinux 核心源程序安装在 uClinux-dist/linux-2.4.x 目录下。

arch/：包含多种架构 CPU 的代码，如 arch/m68knomcu 目录下面是专门针对没有 MMU 的 68K 系列的代码；

documentation/：与内核有关的各种文档；

drivers/：各种设备驱动；

fs/：内核支持的各种文件系统，如 JFFS2，ROMFS 等；

include/：各种头文件；

init/：这个目录下主要是 Linux 的主函数文件 main.c 文件，Linux 的主流程就是在这个文件里面实现的；

ipc/：主要是进程间通信的一些实现代码；

kernel/：这个目录下面是内核的一些核心文件；

lib/：这个目录下是一些通用的库函数；

mm/、mmnommu/：这两个目录下主要是内存管理的代码；

net/：这个目录包含了 linux 对网络的支持，各种协议的实现都在这个目录中；

scripts/：这个目录下主要是一些脚本文件，通过与这些文件的交互对内核进行配置。

(2) 修改内核代码

① 存储器

本课题使用的 μClinux 是基于 RAM 的，所有的代码必须加载到 RAM 中执行，这样在编译内核前就必须指定内核加载的位置。如果存储器地址和大小改了，那么主要应该 修改 uClinux-dist/linux-2.4.x/arch/m68knommu/platform/527x/M5271EVB/ram.ld，里面定义了连接时内核放置的地址，默认是从 0x20000 开始，之前的 8K 空间留给 bootloader 使用。还有 uClinux-dist/linux-2.4.x/arch/m68knommu/platform/527x/M5271EVB/crt0_ram.S，里面定义了内存的大小：

```
#define  MEM_BASE  0x00000000  /* Memory base at address 0 */
#define  MEM_SIZE   0x01000000  /* Memory size 16Mb */
```

需要根据板子的设计改变基址和大小。

② 串口

板子默认的波特率是 19200，N，8，1，无流控。波特率的定义在文件 /uClinux-dist/linux-2.4.x/drivers/char/mcfserial.c 中：

```
#define  CONSOLE_BAUD_RATE    19200
#define  DEFAULT_CBAUD        B19200
```

可以根据自己的需要修改波特率的设定。

(3) 配置、编译内核

在编译 μClinux 内核之前，首先要对内核进行配置，可以用 make menuconfig 来进行内核的配置。该命令执行完毕后生成文件.config，它保存这次的配置信息。此时会出现菜单配置对话框，要求进行目标平台的选择，输入回车后，出现供选择的具体的供应商和产品列表，选择:Motorola/MCF5271EVB。在库的选择上，选择 uClibc。用户还可以自定义内核配置和应用程序配置，根据具体的硬件板设计及系统功能将必要的选项选中，将没有必要的选项去掉，以减少内核的大小。

运行 make dep 建立源文件的依赖关系。

运行 make all 编译内核。

如果编译成功，在 uClinux-dist 下创建一个 images 目录，下面有四个文件：

- image.elf：含调试信息和 romfs 的 μClinux 可执行文件格式，可以用 GDB 装载调试运行

- romfs.img：:romfs 的二进制文件

- linux.bin：不含 romfs 的 μClinux 二进制文件

- image.bin：linux.bin 和 romfs.bin 合并而成，并多了 4 个字节的校验

默认生成的文件格式中不包含 S Record 的文件，可以使用下面的命令生成，也可以将此句增加到 Makefile 中。

```
m68k-elf-objcopy –o srec image.elf image.srec
```

要运行 μClinux，只要通过 bootloader 将 image.bin 下载到 RAM 的 0x20000 处，然后跳转到 0x20000 处执行就可以启动操作系统。如利用 dBug 通过串口将 image.srec 将内核下载到 RAM，然后执行 go 0x20000 开始操作系统的执行。本课题中，首先将 image.bin 通过 BDM 调试头写入到从 0xFFE0 8000 开始的 Flash 中，然后由 bootloader 在完成系统初始化后，将内核搬运到 RAM 的 0x0002 0000 处并执行。

5.4 本章小结

本章以 SDEVB5271 评估板为例，首先介绍了嵌入式应用程序的基本知识，包括

如何初始化 MCF5271，如何初始化程序的运行环境等；接着以编写通用 IO 口的实验为例，分析了怎样考虑并实现这样一个基本的程序；最后本文在 SDEVB5271 上运行 μClinux 的移植实例，阐述了嵌入式操作系统的有关知识。

第六章 总结与展望

6.1 总结

本文针对目前广泛使用的 32 位微处理器 ColdFire 设计了一套评估系统，为开发人员提供了一套操作简单、价格低廉、功能强大的嵌入式开发环境，降低了嵌入式软件开发的难度，使用户不需要将主要精力花费在如何使用开发工具上，而可以专注于产品的功能设计、程序的流程控制等。本文主要工作总结如下：

(1) 采用 Freescale ColdFire 系列中的 MCF5271 微处理器，设计并制作了一块硬件评估板 SDEVB5271，它集成了目前 32 位嵌入式应用中常用的外围设备，主要资源包括 64K 片内 SRAM、16M SDRAM 、2M Flash 、2 个串口、1 个以太网口等，可运行 μC/OS、μCLinux 等嵌入式操作系统。

(2) 分析了 ColdFire 的 BDM 调试模式的原理以及串行通信时序、逻辑，制作了适用于 ColdFire 系列的 BDM 调试头，并编写了基本的驱动程序。

(3) 开发了一套适合 ColdFire 的嵌入式集成开发环境——SdIDE for ColdFire。在设计集成开发环境时，充分考虑了平台的芯片无关性和可扩展性。将与芯片关系不是很紧密的编辑、编译、工程管理部分作为 SdIDE 的通用部分独立出来，并参与了其代码的编写；针对ColdFire 系列编写了与芯片密切相关的程序写入部分，并通过修改一些配置文件实现了 SdIDE 对 ColdFire 的支持。

(4) 使用已开发的 ColdFire 的软硬件开发工具，在 SDEVB5271 硬件板上完成一些基本程序，包括系统初始化、I/O 操作、串口操作等，并在评估板上移植了 μCLinux 操作系统。

6.2 展望

本课题是基于 Freescale 半导体公司的 32 位微处理器 ColdFire 进行开发的，作者以及所在的研究室以前并没有使用过该系列的 CPU，而且其中文资料很少，作者必须研读 Freescale 公司提供的英文手册进行软硬件的设计和开发。本文设计并实现了一套基于 ColdFire 的评估系统，但在设计实现过程中存在一些不尽完善之处，下面给出几点在后续工作中需要改进的地方：

(1) 增加源代码调试功能。集成开发环境一般支持软件调试功能，可以让用户了解程序的执行流程并可以快速定位到出错的地方。由于采用 32 位微处理器的硬件板的 RAM 空间都比较大，将来可以考虑移植一个 GDBStub。GDBStub 是一个开源项目，其支持多种 CPU 架构，可以在没有操作系统的裸板上运行。文中在第四章设计程序写入的监控程序时已经留有可供扩展的命令接口，可以将 GDBStub 放到监控程序中，进一步扩展监控程序的功能。高端软件按照 GDB 的串行通信协议和 GDBStub 进行通信。采用这种方式，要调试的程序必须装载到 RAM 中，宿主机可以采用串口或以太网口与目标板通信。

(2) 丰富评估板的种类。由于硬件条件的限制，本课题只在 SDEVB5271 平台上完成了软件的编写、测试及相关的实验程序编写。如果更换了其他 ColdFire 系列的芯片，SdIDE 的通用部分的源码不需要改变，主要是程序写入部分。不同的芯片在写入时，基本的操作是相同的，只是寄存器的地址可能不一样，本课题中只是在程序中定义了一些芯片的寄存器地址，将来可以把这些信息存放在一个配置文件中，这样不需要修改源码就可以支持所有 ColdFire 系列的芯片的写入，增强了系统的灵活性。

(3) 完善 SDEVB5271 的实验程序。由于时间的关系，本课题中只完成了部分实验程序的编写，如 I/O 口实验、串口实验、QSPI 实验、FEC 的 MII 接口通信等；还需要进一步完成 I^2C、DMA、eMAC 等模块的实验以及复杂的网络应用程序。同时，还有基于 μCLinux 的一些应用程序的开发。

开发工具需要在实际的使用中根据用户的反馈进一步完善，开发板的种类还需要进一步丰富。基于ColdFire 的评估系统的开发必将促进 ColdFire 系列芯片的推广与应用，丰富嵌入式产品市场。

参考文献

[1] 李晶皎,王爱侠,张广渊.ColdFire 系列 32 位微处理器与嵌入式 Linux 应用[M].北京:北京航空航天大学出版社,2005.

[2] 邵贝贝.单片机嵌入式应用的在线开发方法[M].北京:清华大学出版社,2004.

[3] 申忠如,陶慧斌,曹建安.ColdFire 嵌入式系统设计[M].西安:西安电子科技大学出版社,2006.

[4] 杜春雷.ARM 体系结构与编程[M].北京:清华大学出版社,2003.

[5] 朱红星.基于 ColdFire 的嵌入式 IP 摄像机的方案研究[D].西北工业大学硕士学位论文,2005.

[6] Richard M Stallman. GNU Compiler Collection Internals[EB/OL]. http://www.gnu.org,2003.

[7] ColdFire MCF527x 系列单片机 [EB/OL]. http://www.freescale.com.cn/coldfire,2007.

[8] MCF5271 Reference Manual Rev.2[DB/OL].http://www.freescale.com，2006.

[9] 王宜怀,刘晓升.嵌入式应用技术基础教程[M].北京:清华大学出版社,2005.

[10] 李庆诚,孙明达.基于 NAND 型闪存的嵌入式文件系统设计[J].计算机应用研究,2006,4:231-239.

[11] AM29LV160DB Datasheet Rev2.0[DB/OL]. http://www.amd.com，2003.

[12] HY57V641620HG 4Bank × 1M × 16Bits synchronous DRAM Rev.0.7[DB/OL]. http://www.hynix.com，2002.

[13] DM9161E DataSheet[DB/OL]. http://www.davicom.com.tw,2003.

[14] TLC2543-EP 12-BIT ANALOG-TO-DIGITAL CONVERTERS WITH SERIAL CONTROL AND 11 ANALOG INPUTS[DB/OL]. http:// www.ti.com, 2002.

[15] 胥静.嵌入式系 统设计 与开发 实例详解 [M]. 北京 :北京航空航天大学出版社,2005.

[16] 黄红燕.嵌入式系统调试技术的分析与设计[D].浙江大学硕士学位论文,2006.

[17] 杨峰,张根宝,田泽等 .基于 JTAG 的 ARM 芯片系 统调试 [J].微计算 机信息.2005,22:87-89.

[18] Pavel Pisa. BDM Interface for Motorola 683xx MCU Usage with GDB Debugger [Z]. 2000.7.

[19] 简敬元,龙占超.针对Motorola微处理器的BDM调试系统的设计[J].现代电子技术,2004,6(173):46-56.

[20] 马彧,王丹利,王丽英.CPLD/FPGA 可编程逻辑器件实用教程[M].北京:机械工业出版社,2006.

[21] 王诚,薛小刚,钟信潮.FPGA/CPLD设计工具 Xilinx ISE 5.x使用详解[M].北京:人民邮电出版社,2003.

[22] FastFLASH XC9500XL High-Performance CPLD Family[DB/OL].http://xilinx.com,1999.

[23] CPLD Schematic Design Guide[DB/OL]. http://xilinx.com,1999.

[24] W. Mohat.Design and Construction of a High-Speed BDM Pod for ColdFire CPUs[Z].2000,3.

[25] 马义德.微型计算机原理与接口技术[M].北京:机械工业出版社,2005.

[26] 张宏林.Visual C++ Visual Basic串并口开发技术工程应用实例导航[M].北京:人民邮电出版社,2006.

[27] ColdFire Family Programmer 's Reference Manual Rev.3[DB/OL]. http://www.freescale.com,2005.

[28] 帅辉明.SD9200-ARM9 嵌入式教学实验平台的设计与实现[D].苏州大学硕士学位论文,2006.

[29] 孙纪坤,张小全.嵌入式 Linux 系统开发技术详解 [M].北京:人民邮电出版社,2006.

[30] RobertMecklenburg. Managing Projects with GNU Make[M].南京:东南大学出版社,2005.

[31] Mario Camou, Aaron Von Cowenberghe著.Debian GNU/Linux高级应用大全[M].北京:清华大学出版社,2002.

[32] 康涌泉,桑楠,邹楚雄等.嵌入式 Linux 交叉开发环境[J].计算机应用.2006,Vol 26:261-263.

[33] 田军营,韩建海,马志荣.μClinux 源代码中 Make 文件完全解析[M]. 北京:机械工业出版社,2005.

[34] 赵炯.Linux 内核完全注释[M].北京:机械工业出版社,2004.

[35] 刘春祥.基于 RISC 核的系统软件研究与应用[D].浙江大学硕士学位论文,2002.

[36] Using LD,the GNU linker[DB/OL]. http://www.gnu.org, 2000.

[37] 张和君,张跃.基于 GNU 工具的嵌入式 Bootloader 设计与开发[J].计算机工程.2006,32(15):277-279.

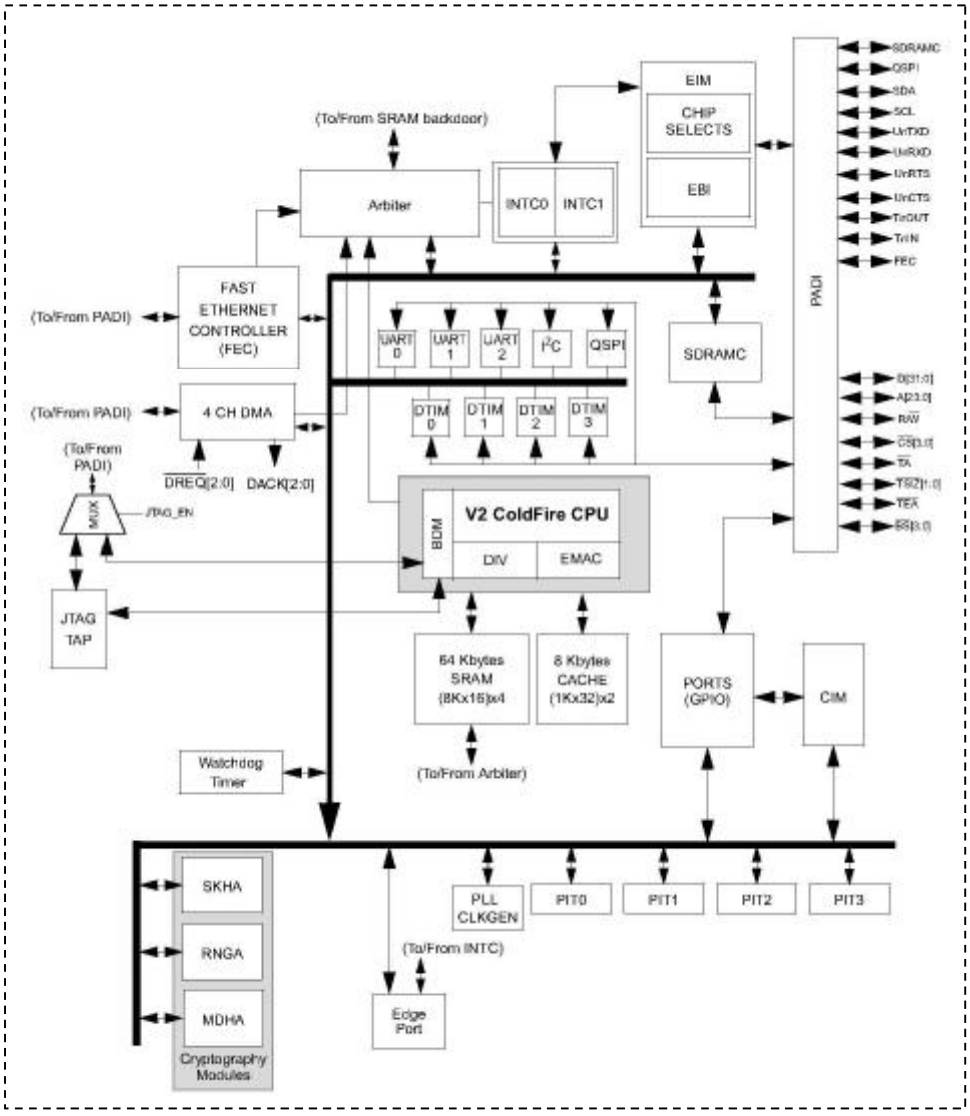
[38] Motorola S-record description[EB/OL].http://www.freescale.com, 1999.

[39] 周立功,陈明计,陈渝.ARM 嵌入式 Linux 系统构建与驱动开发范例[M].北京:北京航空航天大学出版社,2006.

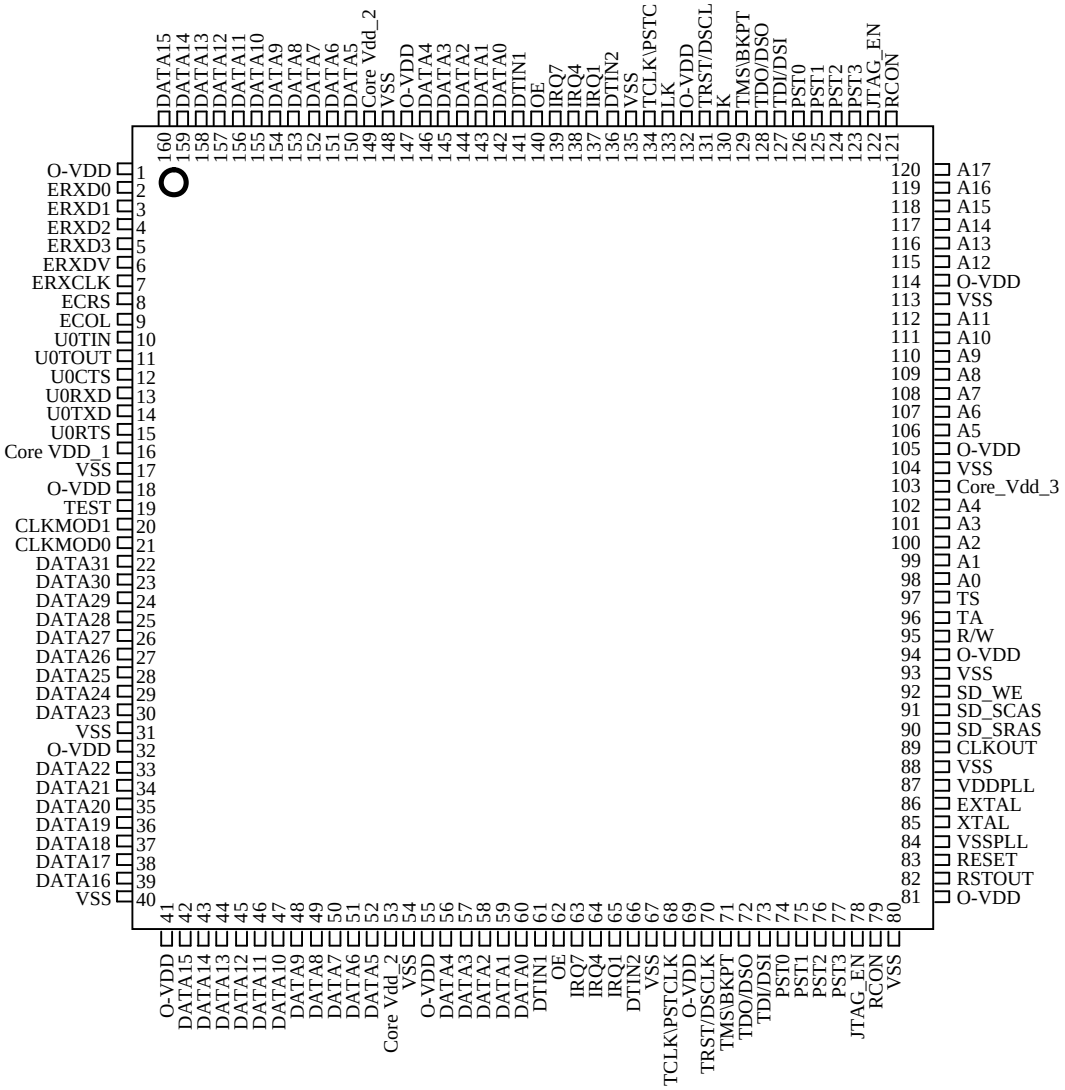
[40] 李善平,刘文峰,王焕龙.Linux 与嵌入式系统[M].北京:清华大学出版社,2006.

附录 A MCF5271 相关资料

A.1 MCF5271 模块框图

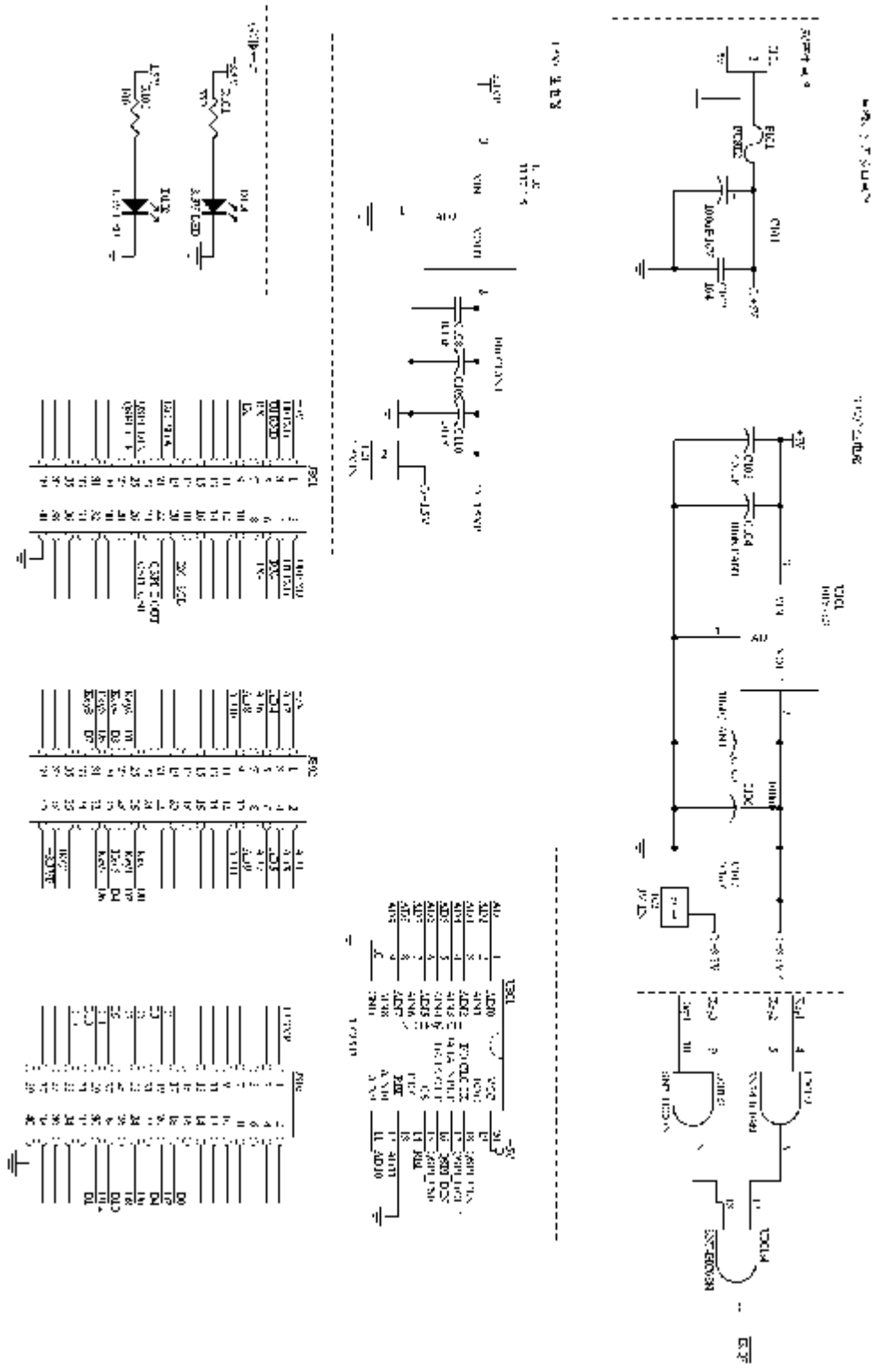


A.2 MCF5271 芯片引脚图

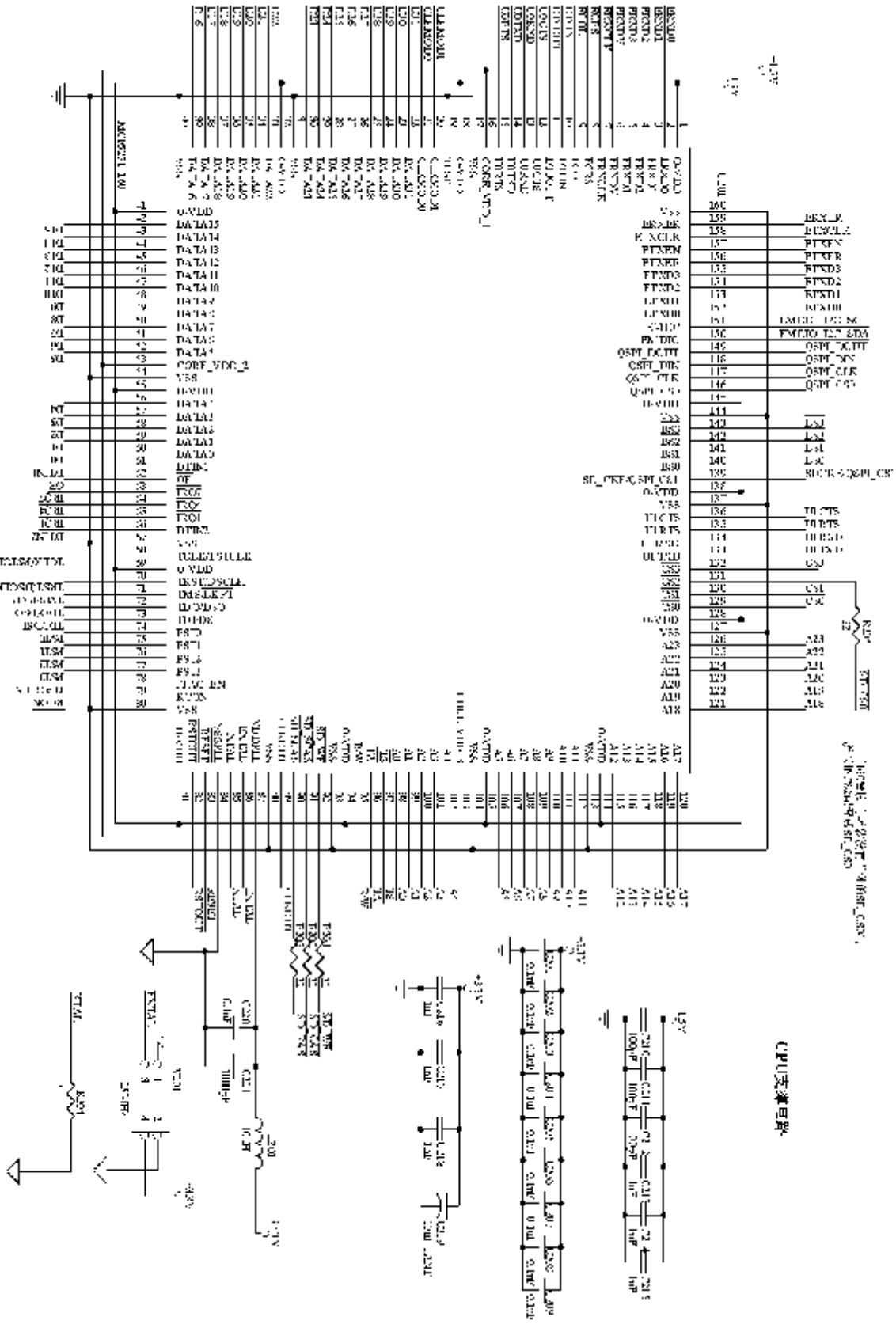


附录 B SDEVB5271 硬件原理图

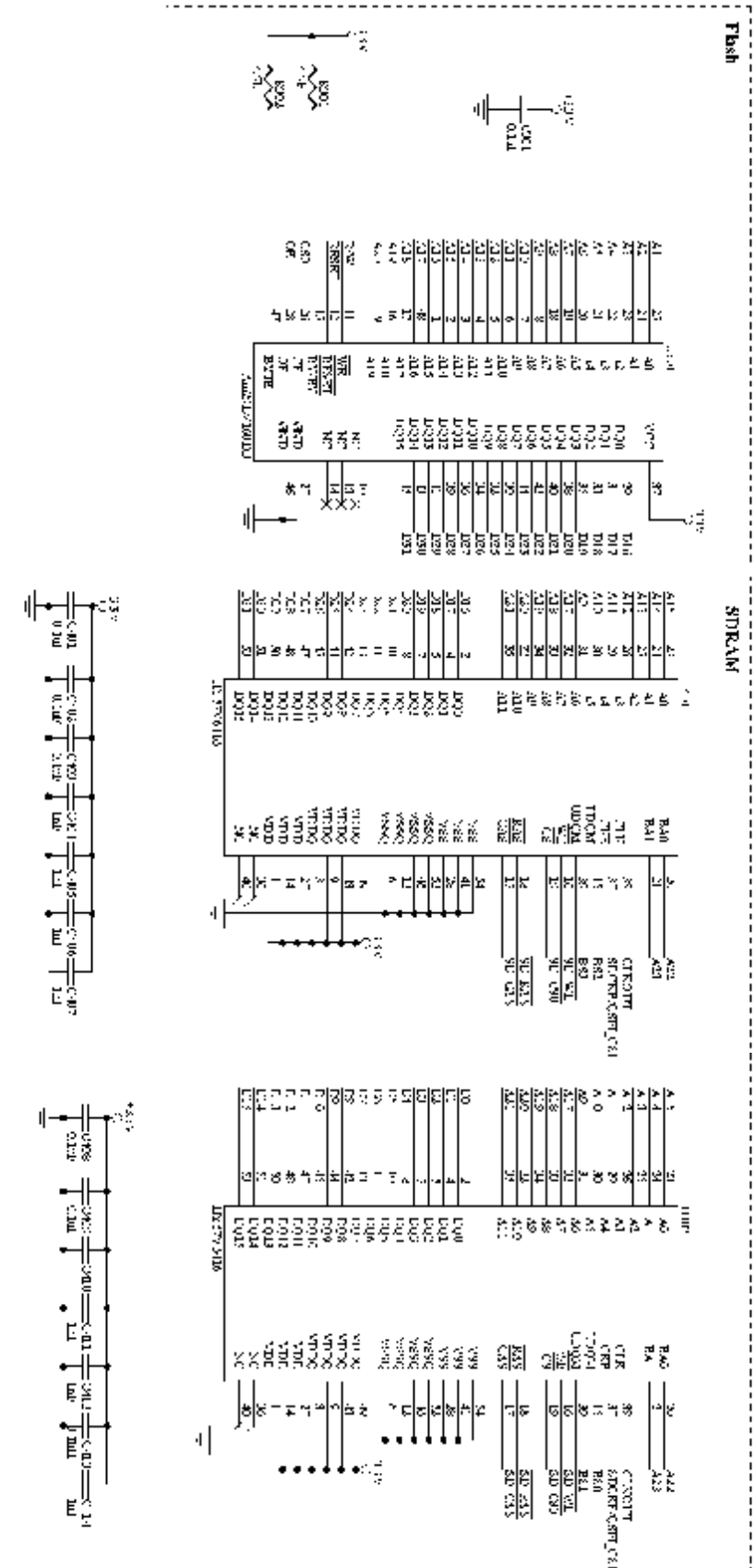
B.1 电源电路及扩展接口电路



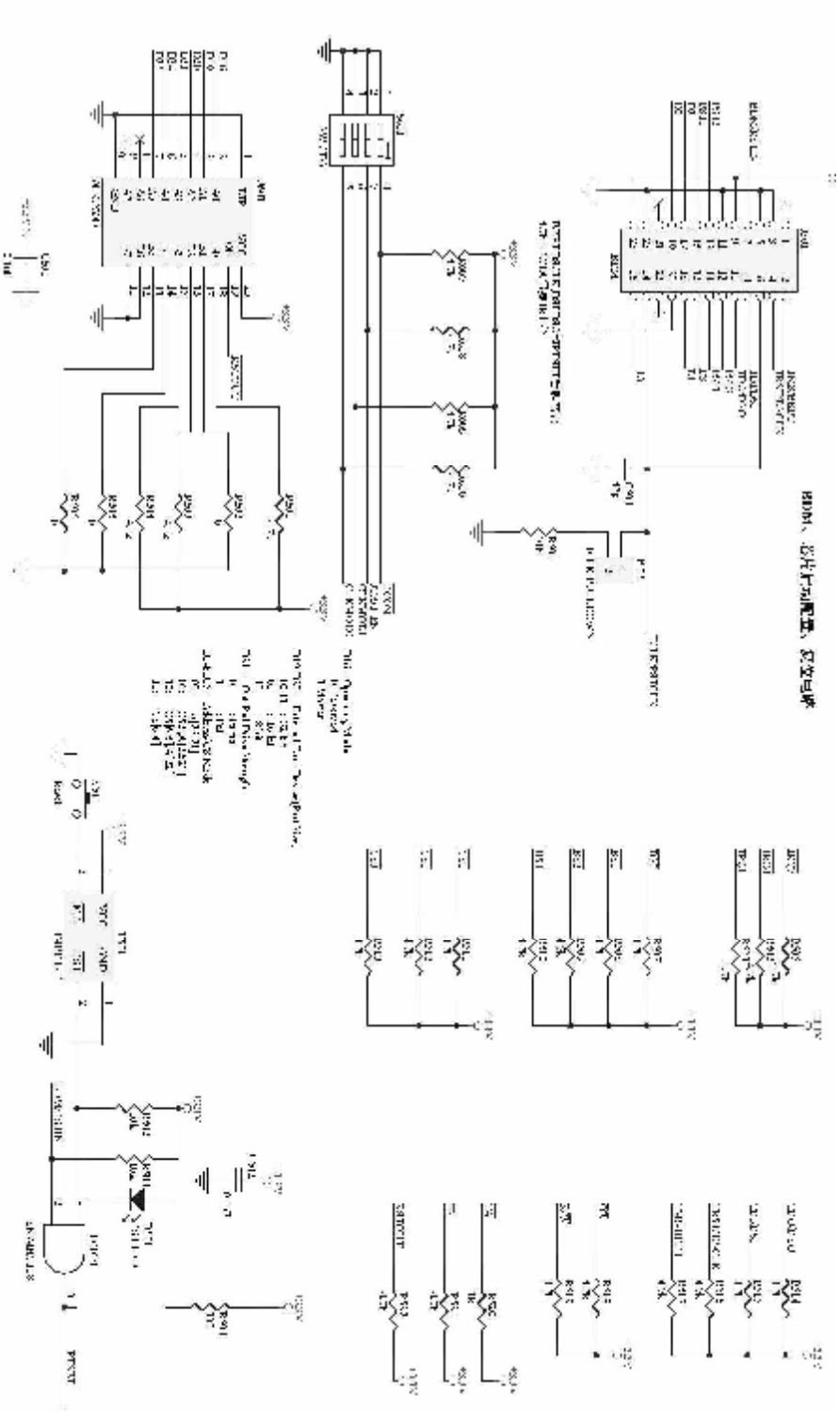
B.2 MCF5271 支撑电路



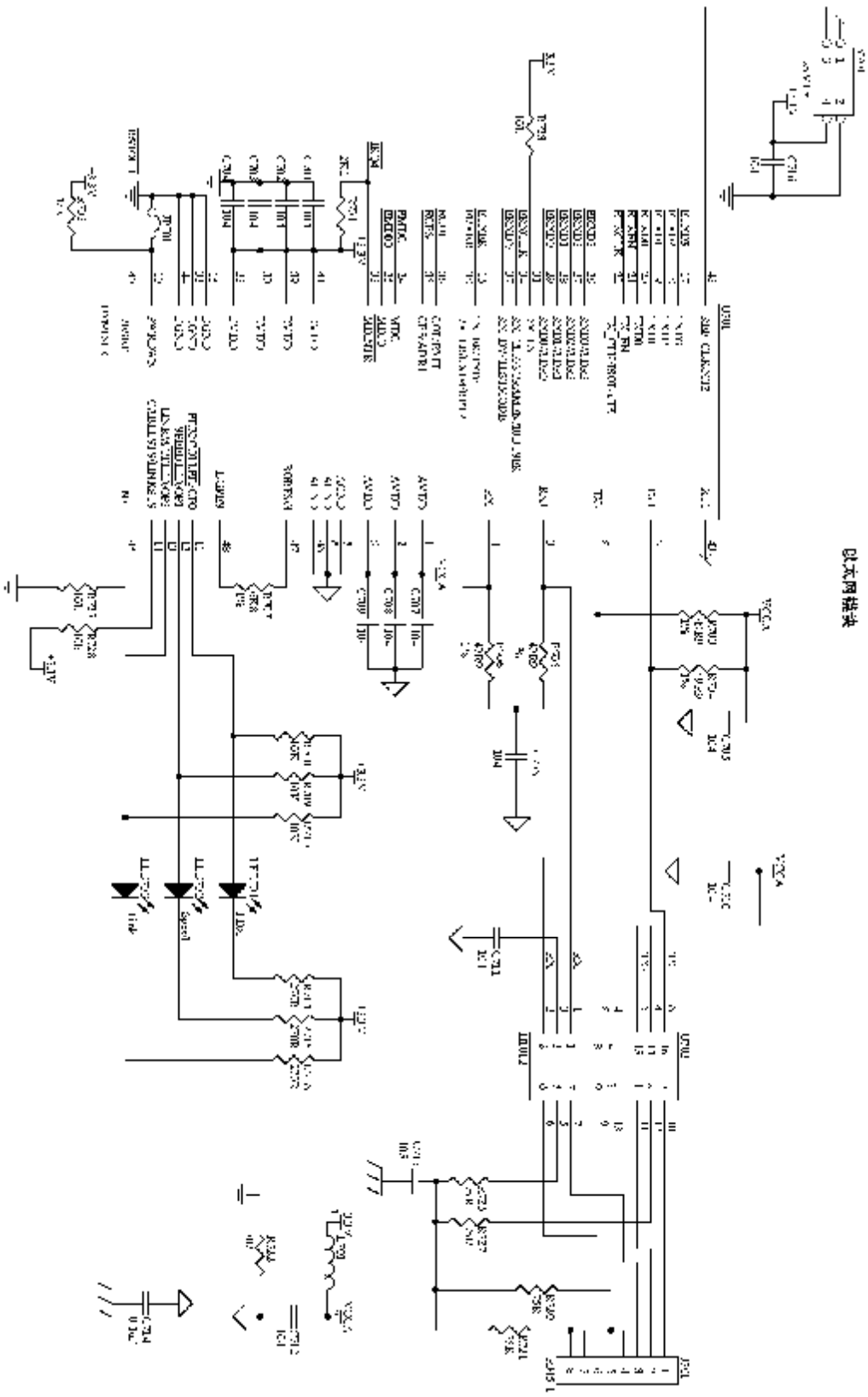
B.3 存储扩展电路



B.4 CPU 配置电路及 BDM 接口

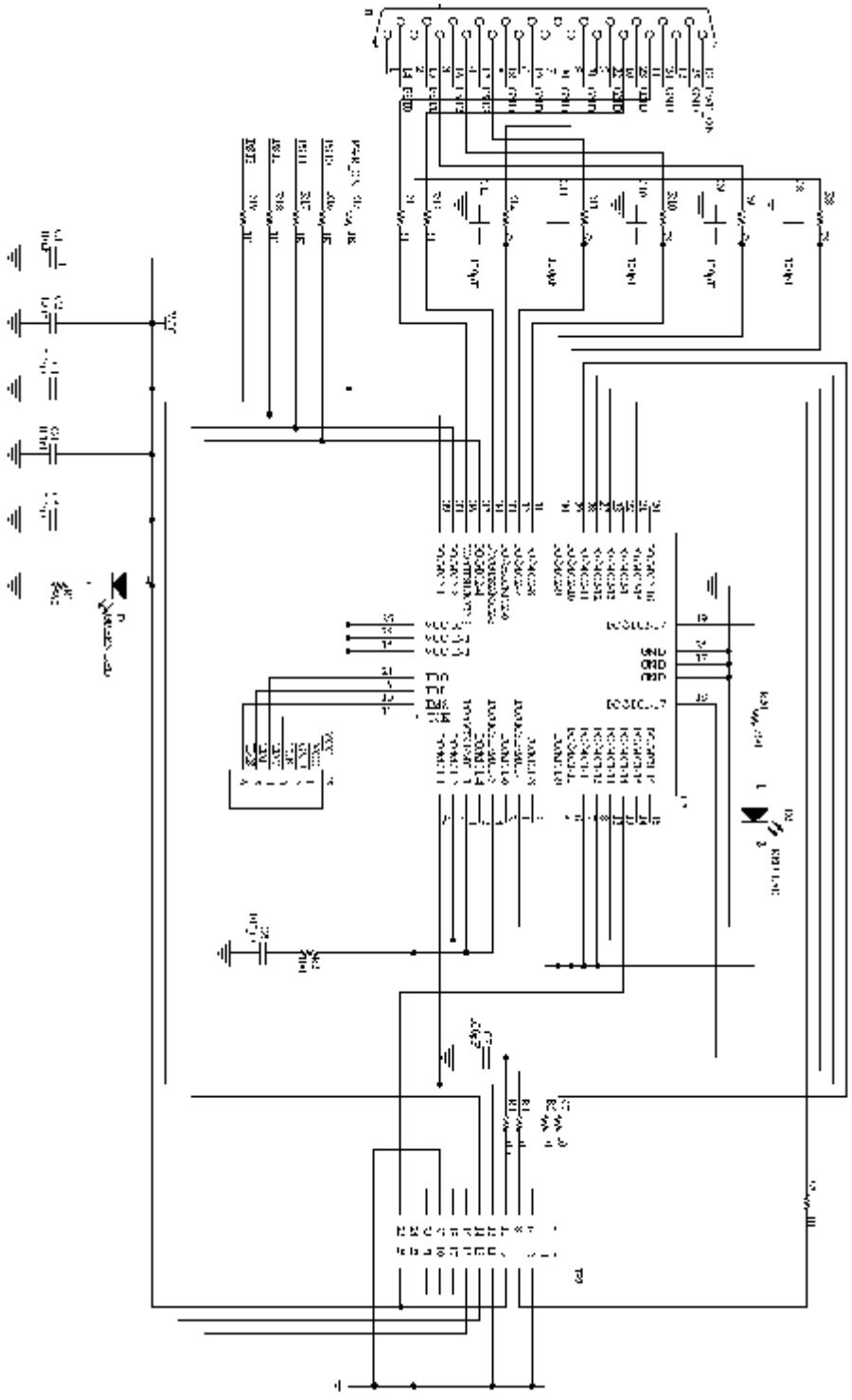


B.5 以太网接口



附录 C BDM 调试头原理图

C.1 BDM 调试头硬件原理图



C.2 XC9536XL 内部程序原理图