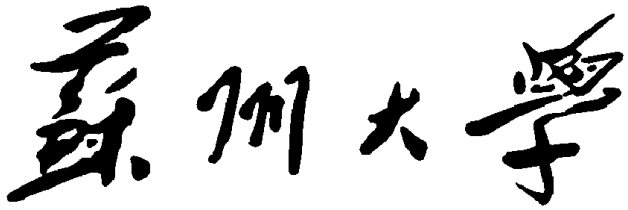


硕 士 学 位 论 文

（2008 届）

基于 ColdFire 的嵌入式调试系统的设计与实现

Design and Implementation of Embedded Debugging System based on ColdFire

研 究 生 姓 名　陈　祎

指导教师姓名　王宜怀（教授）

专 业 名 称　计算机应用技术

研 究 方 向　嵌入式系统

论文提交日期　2008 年 4 月

苏州大学学位论文独创性声明及使用授权声明

学位论文独创性声明

本人郑重声明：所提交的学位论文是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含其他个人或集体已经发表或撰写过的研究成果，也不含为获得苏州大学或其它教育机构的学位证书而使用过的材料。对本文的研究作出重要贡献的个人和集体，均已在文中以明确方式标明。本人承担本声明的法律责任。

研究生签名：_____日　期：_____

学位论文使用授权声明

苏州大学、中国科学技术信息研究所、国家图书馆、清华大学论文合作部、中国社科院文献信息情报中心有权保留本人所送交学位论文的复印件和电子文档，可以采用影印、缩印或其他复制手段保存论文。本人电子文档的内容和纸质论文的内容相一致。除在保密期内的保密论文外，允许论文被查阅和借阅，可以公布（包括刊登）论文的全部或部分内容。论文的公布（包括刊登）授权苏州大学学位办办理。

研究生签名：_____日　期：_____

导 师 签 名 ：_____日　期：_____

中文摘要

嵌入式软件的特殊性使得其开发过程比传统的通用计算机软件要复杂得多，而调试作为嵌入式系统开发中的关键环节，扮演着十分重要的角色。目前，国内在嵌入式调试技术方面所做的研究工作较少，一般嵌入式开发调试工具都使用国外产品。因此，深入研究嵌入式调试技术并研发自己的开发调试工具具有重要意义。

为了缓解国内嵌入式开发调试工具严重依赖进口、开发资料较少等一系列问题，本文针对国内外市场发展迅猛的 ColdFire 微处理器，使用基于 GDB 调试器的远程调试方式，设计并实现了一套 ColdFire 嵌入式调试系统，包括硬件调试平台 SDMCF52233EVB、调试桩 GDBStub for ColdFire 以及宿主机端集成调试软件 SD-IDE for ColdFire，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集成调试平台。

本文详细阐述了 ColdFire 调试系统的开发流程。首先分析了 GDB 相关调试技术并给出了本调试系统的实现结构框架；其次给出了硬件调试平台 SDMCF52233EVB 的最小系统设计、各硬件功能模块的实现方案以及硬件测试流程；随后在借鉴 GDBStub 的一般调试原理和工作机制的基础上，设计并实现了 GDBStub for ColdFire，以作为目标机端调试代理，配合 GDB 完成调试；最后按模块讨论了宿主机端集成调试软件 SD-IDE for ColdFire 的实现，包括添加工程模板、交叉编译、代码写入和代码调试。本课题的研究成果为开发基于其他型号微处理器的调试系统以及类似的嵌入式产品提供了借鉴和参考。

关键词：ColdFire，MCF52233，GDB，远程调试，调试桩

作　　者：陈　祎

指导老师：王宜怀

ABSTRACT

The speciality of embedded software makes its development much more complicated than the traditional general computer software's. Debugging as a critical process in the embedded development, plays a very important role. Currently, there is little domestic research in the area of embedded debugging. The developing and debugging tools are provided by foreign companies generally. So it is greatly meaning to study the embedded debugging technology deeply and develop the tools of ourselves.

To ease a series of problems such as too much dependency on import tools and lacking of development documents interiorly, we aim at ColdFire CPUs which are rapidly enlarged in the national market, adopt the remote debugging method based on GDB to design and implement a ColdFire embedded debugging system including hardware debugging platform SDMCF52233EVB, debugging stub GDBStub for ColdFire and integated debugging software SD-IDE for ColdFire in the host. It provides an integated debugging platform with full function, simple operation and low price to meet the needs of both study and development for the users.

The paper describes the development process of ColdFire debugging system in detail. Firstly, it analyses GDB debugging technology and shows the structure frame of ColdFire debugging system. Secondly, it shows the design of SDMCF52233EVB mininum system, the realization of each hardware module and the hardware testing process. Then, basing on the reference of GDBStub's debugging principle and working mechanism, we design and implement GDBStub for ColdFire as a debugging agent in the target to work with GDB together. At last, it discusses the realization of integated debugging software SD-IDE for ColdFire according to different modules, including adding project templates, cross compiling, downloading codes and debugging codes. The results of this research provide a reference for the development of debugging system based on other chips and similar embedded productions.

Keywords: ColdFire，MCF52233，GDB，Remote debugging，Debugging stub

Written by Chen Yi

Supervised by Wang Yihuai

目　　录

第一章 绪论

软件调试是程序开发流程中必不可少的环节，其主要作用就是帮助研发人员跟踪代码的执行，发现并纠正运行过程中遇到的潜在错误与漏洞，有效的提高开发效率。对于目前得到广泛应用的嵌入式系统而言，由于其功能专一、针对性强，在研发过程中往往缺乏键盘、显示器、硬盘、充足的内存等有效交互手段和基本硬件开发资源，因此在嵌入式软件的开发过程中调试就显得异常重要。为了缓解嵌入式开发调试工具发展相对滞后的状况，本文针对国内外市场增长迅猛的 ColdFire 系列微处理器，在分析 GDB(GNU DeBugger，GNU 调试器)相关调试技术的基础上，利用基于串行通信的 GDB 调试代理，设计并实现了 ColdFire 调试系统。

作为全文的导引，本章首先阐述了 ColdFire 系列微处理器的特点及应用，接着分析了各种常用的嵌入式调试技术，在了解当前国内外相关技术的发展状况后，给出了 ColdFire 调试系统的实现目标及意义，最后为本文的工作和组织结构。

1.1 ColdFire 系列微处理器

作为世界上最大单片机供应商的 Freescale 半导体(原 Motorola 半导体产品部)，早在 1974 年就开始生产 8 位微处理器，并在 1979 年推出了 MC68000(68K)系列微处理器。1996 年，在传统 68K 指令集的基础上，Freescale 以高性价比、高集成度推出了 32 位 ColdFire 系列产品。

68K 系列属于复杂指令集处理器(Complex Instruction Set Computer，CISC)，芯片内部的译码逻辑和控制逻辑占据了大部分内核空间，对于存储器的频繁访问也降低了 CPU(Central Processing Unit，中央处理单元)的有效执行速度。而 ColdFire CPU 对此进行了改进，采用指令长度可变的精简指令集技术(Reduced Instruction Set Computer，RISC)，精简掉部分 68K 指令，又与 68K 指令集兼容，使多数指令可以在 1 个周期内完成。这样 CPU 内核不仅可以做得很小，而且处理速度也可以达到 400MIPS(Million Instruction Per Second，每秒执行的百万指令数)，有效的降低了生产成本，更适合于嵌入式设备的开发和应用[1]。

ColdFire 系列产品由最初的 V2(V 代表 Version，V2 包括第一款 ColdFire 微处理

器 MCF5206 以及 MCF52xx)发展到 V3(MCF53xx)、V4(MCF54xx)，直到最新版超流水线结构的 V6，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 核心的几十种不同型号的芯片，后续将推出 V5、V6 核心的芯片。各版本间的比较如下[2][3]：

① V2 核心：两个独立、解耦(Decoupled)的两级指令获取流水线和两级指令执行流水线；单周期局部总线；具有统一的 Cache、RAM(Random Access Memory，随机存取存储器)和 ROM(Read Only Memory，只读存储器)。

② V3 核心：两个独立、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、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；大容量分支 Cache。

⑤ V6 核心：V6 为超流水线结构。

由于性价比优异、选型丰富，ColdFire CPU 在 Freescale 的 32 位微处理器产品中扮演着十分重要的角色，其应用领域也不断扩大。从工业自动化系统到低端网络设备、智能家电甚至安防系统等嵌入式产品，都喜欢选用 ColdFire 微处理器[3][4]。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方式的需求，Freescale 为 ColdFire 推出了多种可供选择的连接方式，包括以太网、USB(Universal Serial Bus，通用串行总线)、PCI(Peripheral Component Interconnect，外设部件互连)、CAN(Controller Area Network，控制器局域网)和 UART(Universal Asynchronous Receiver/Transmitter，通用异步收发)；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 架构上集成了增强型时间处理单元；为了适应市场对安全性的要求，Freescale 在 ColdFire 产品上提供了可选的加密加速器。

1.2 常用嵌入式调试方式分析

嵌入式系统自身的特点决定了其开发调试过程与通用计算机系统截然不同，需要涉及宿主机和目标机两方面的内容，即用户在宿主机端调试运行于目标机端的代码。从调试技术实现的途径及其应用两个角度，可以将嵌入式系统调试分为以下几种方式：在线仿真、片上调试、模拟调试、ROM 监控以及调试代理。其中在线仿真和片

上调试属于硬件调试；而 ROM 监控与调试代理属于软件调试。本文讨论的基于 ColdFire 的嵌入式调试系统选用调试代理 GDBStub 方式，通过远程调试协议与宿主机端调试器 GDB 交互调试信息。下文将一一阐述以上五种调试方式。

1．在线仿真

在线仿真是一种利用在线仿真器(In-Circuit Emulator，ICE)替代目标机上物理微处理器的调试方式。ICE 的功能与被代替的微处理器完全相同，但是它不仅能够产生控制目标机外围电路所需要的信号且接收外部输入信号，还可以允许用户查看 CPU 内部的数据或代码信息并控制 CPU 的运行[5]。

一个 ICE 通常由仿真探头和仿真主板组成。仿真探头通过一条电缆与仿真主板相连，里面包含了一颗功能与被代替微处理器完全相同的 CPU。该 CPU 能够执行被代替微处理器的所有指令，只是为了调试进行了特殊处理，其内部的某些信号连接到了芯片的引脚上，使得外部逻辑可以通过这些引脚监控 CPU 内部的状态[6]。由于 ICE 对目标机微处理器的代替完全是物理上的，因此用户通常要将目标机上的微处理器拔出，然后才能将 ICE 的仿真探头插入目标机的 CPU 插槽中。仿真主板提供了软件断点、硬件断点、复杂断点、触发、实时跟踪、重叠 RAM 和影子 RAM 等众多调试功能，它通过串口或者 USB 接口直接连接到调试宿主机上。其中实时跟踪是 ICE 提供的最有特色的调试手段，它可以在不占用运行时钟周期的情况下获得程序的执行情况，具有抗干扰性强的特点。特别是在强实时系统中，由于无法使用断点，实时跟踪就成了唯一有用的调试方式。

尽管 ICE 具有很多的优点，如不消耗目标机系统资源、支持硬件断点调试、实时性强等，但是存在价格非常昂贵、通用性不强、可扩展性差等缺陷，使得在线仿真方式的应用受到了极大限制。

2．片上调试

由于现代微处理器的封装越来越表贴化(贴近电路板表面)，仿真探头的实现也越来越困难。另外，根据统计：在大约 95％的调试过程中，用户仅仅使用了简单断点、单步以及访问微处理器资源、内存和外设等一些运行控制方面的基本调试手段。因此，一个很自然的发展趋势就是将实时跟踪和运行控制分开，将运行控制放到目标机系统的微处理器核内，由一个专门的调试控制逻辑模块来实现，并用一个专用的串行信号接口开放给用户，用户可以通过微处理器核内的调试控制逻辑模块来停止或者继续

CPU 的运行，并访问目标机上的各种资源。这种放弃实时跟踪功能，但是提供了大多数 ICE 特性的调试方式就是片上调试(On-Chip debugging，OCD)[7][8]。

宿主机与目标机微处理器的调试控制逻辑模块之间可以通过一块简单的信号转换电路板来匹配专用串行接口的通信信号。这块信号转换电路板称为“片上调试器”或“串行调试器”，信号转换只是其最基本的功能之一，而其它高级功能的实现由各个厂商在其发布的片上调试器产品中完成。常见的片上调试串行接口有以下三种：JTAG、OnCE 和 BDM。

① JTAG(Joint Test Action Group，联合测试行动小组)。JTAG 是 1985 年指定的检测 PCB(Printed Circuit Board，印刷电路板)和 IC(Integrated Circuit，集成电路)芯片的一个标准，1990 年被修改成为 IEEE(Institute of Electrical and Electronics Engineers，电气及电子工程师学会)1149.1 标准。通过该标准，用户可对具有 JTAG 接口芯片的硬件电路进行边界扫描和故障检测[9]。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引脚的连接是否可靠，提高测试效率；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。JTAG 接口是国际上常用的一种标准接口，现有的大部分微处理器都带有 JTAG 接口[10]。

② OnCE(On-Chip Emulation，片上仿真)。OnCE 是另一种常见的调试接口，在 Freescale 568xx 系列 DSP(Digital Signal Processing，数字信号处理器)和 M*Core 系列微处理器上都包含有该接口。值得注意的是：M*Core 里除了包含有 OnCE 接口之外，也包含有 JTAG 调试接口。

③ BDM(Backgroud Debug Mode，背景调试模式)。Freescale 最早认识到 OCD 技术的发展趋势，率先在 683xx 和 68HC16 微处理器上创造了 BDM 调试接口，并将其用于 ColdFire、PowerPC 等架构的 32 位微处理器中。

虽然片上调试功能强大，但它要求微处理器具有片上调试功能(现在有的微处理器就不支持片上调试功能)，并且实现起来非常复杂，通用性和可移植性较差。

3．模拟调试

通常使用的模拟调试方式是指令集的模拟，它相当于在宿主机上虚拟了一台目标机。该目标机可以和宿主机使用不同类型的 CPU。利用指令集模拟方式进行的调试是一种完全由软件模拟的调试方法，根本不需要硬件板卡的支持，就连输入输出等设备

也都是软件模拟的。而实际上软件模拟的结果有时与真实板卡还是有一些差别，硬件的信号、延迟以及对资源的竞争用纯软件的方法根本无法模拟。

由于指令集模拟调试不需要硬件开发板的支持，因此适合于嵌入式系统开发的初级阶段，或者硬件板卡不是批量生产，数量十分有限的情况。模拟调试方式也适合于应用程序的调试，因为应用程序与硬件和外围设备关系不是很大。虽然模拟调试功能有限，但是它无需硬件支持，节省了嵌入式系统开发的成本[8]。

4．ROM 监控

ROM 监控是指一段驻留在目标机 ROM 中的小程序，它可以在开发过程中辅助测试与调试用户所编写的嵌入式程序。采用 ROM 监控方式进行调试需要在目标机上运行 ROM 监控和被调试程序，宿主机的调试器通过远程调试协议与目标机上的 ROM 监控建立通信连接。当 CPU 复位时，将首先执行 ROM 监控程序。在执行完一些必要的初始化后，ROM 监控一般将等待来自宿主机的连接信息，以建立调试会话。ROM 监控能完成被调试程序的下载、目标机内存和寄存器的读写、设置简单断点以及单步运行等功能。一些高级的 ROM 监控能完成代码分析(Code Profiling)、系统分析(System Profiling)、ROM 空间的写操作，以及设置各种非常复杂的断点等功能。

ROM 监控方式不需要专门的调试硬件支持，明显提高了程序调试效率，降低了调试难度，有效的缩短了产品开发周期。但该调试方式也同时具有较多缺点：ROM 监控程序开发难度较大；要求目标芯片必须有一定的 ROM 空间；不便于调试有时间特性的程序。

5．调试代理

调试代理(Debugging agent)也是一小段驻留在目标机上的代码，可分为调试桩(Debugging stub)和调试服务器(Debugging server)两类。采用这种调试方式进行软件调试时也需要在目标机上运行调试代理和被调试程序，宿主机的调试器和目标机的调试代理也使用远程调试协议进行通信。与 ROM 监控调试方式不同的是，ROM 监控程序是驻留在目标机的 ROM 中的，普通用户无法改写，系统复位时首先执行 ROM 监控，然后下载被调试程序进行调试；而调试代理并不是固化在目标机上的，需要先通过某种工具将它们下载到目标机中。由于调试桩往往是设计用来独立运行于目标机上的，不需要操作系统软件环境的支持，因此它必须与被调试程序一起配合运行，一般用于裸机程序即底层程序的调试；而调试服务器通常作为目标机系统上的一个应用实

例运行，只能用于基于操作系统的应用程序调试。以典型的 GNU(GNU's Not Unix)调试器 GDB 为例，当使用 GDB 调试底层程序时，需要使用 GDB 的调试桩——GDBStub，而使用 GDB 调试基于操作系统的应用程序时，需要使用 GDB 的调试服务器——GDBServer。调试代理的职责就是在目标机上实现由宿主机上的调试器发送过来的调试命令，例如：读写内存、读写寄存器、设置断点以及运行被调试程序，并将结果返回，以配合宿主机的调试器完成调试。

自由软件协会组织 GNU 免费提供的 GDB 拥有强大的远程调试功能，它能使开发人员以远程调试的方式单步执行目标机平台上的程序代码、设置断点、查看内存，并同目标机系统交换信息。GDB 远程调试的实时、动态、方便、免费等优点使它逐渐成为嵌入式开发首选的调试方案[6]。

1.3 国内外发展状况

随着网络和通信技术的快速发展，嵌入式系统软、硬件技术有了很大的提升，使得嵌入式系统软件的规模已经不再局限于原先的裸机程序和简单应用程序。目前形成的嵌入式应用模式带有明显的计算机工程的特点，即基于嵌入式系统软硬件平台，以网络、通信为主的嵌入式非底层应用。所以，构建功能完善的嵌入式集成开发环境已经逐渐成为嵌入式领域的一个新兴课题。

目前，嵌入式集成开发环境主要由第三方工具公司提供，为不同操作系统的不同微处理器版本专门定制。国内外比较流行的嵌入式集成开发环境的发展状况如表 1-1[11]所示，其中，√表示拥有该功能，○表示可选功能。

表 1-1 几种国内外集成开发环境的状况

名称	目标微处理器	编辑	编译	调试	编译器	编程语言	供应商	价格	发布时间
Tornado	68xxx,MIPS, SPAR,ARM,NEC Vxx	√	√	√	GNU 编译器	C/C++ 汇编	Wind River	$6800	1995
CodeWarrior	683xx,ARM,PPC, MIPS,MC68HCxx, ColdFire	√	√	√	自主研发	C/C++ 汇编 JAVA	Metrowerks Inc	$4800	1993
Embest IDE	S3C44BOX,ARM, S3C2410	√	√	√	GNU 编译器	C/C++	深圳英蓓特信息技术有限公司	$1200	2001
Lambda	PPC,MIPS,ARM	√	√	√	GNU 编译器	C/C++ 汇编	北京科银京成技术有限公司	$1800	2001
Orion	SPARC,i386,ERC32	√	○	√	GNU 编译器	C/C++ 汇编	欧比特(珠海)软件工程有限公司	$900	2003

Tornado 和 CodeWarrior 是目前市场上影响力较大的嵌入式集成开发环境，其调试功能完善、性能稳定，但都属于商业级软件，价格昂贵，操作界面国外化，不具通用性。国内主要使用国外引进的集成开发环境，自主研发的成果较少，与国际先进水平相比尚存一定差距。表 1-1 所列出的 Lambda 和 Orion 是目前国内比较成功的产品，但它们适用的微处理器的系列较少，不具备良好的扩展性。

国外研发嵌入式集成开发环境的另一重要分支是 GNU 组织。他们通过 Internet 向全世界免费提供各种相关的研究成果甚至源代码。现在已有很多公司和个人在 GNU 软件的基础上，经过集成、优化和测试，推出了基于 Linux 平台的嵌入式集成开发环境和工具。虽然这当中有一部分是免费和开源的，其调试功能也很强大，但是由于基于 Linux 平台，普通用户和开发人员难以上手。这无形中提高了入门门槛，限制了其在国内的推广和应用。

由此可见，开发适合广大国内 ColdFire 用户使用的功能强大、性能稳定、价格低廉的嵌入式开发调试系统已迫在眉睫。

1.4 课题实现的目标和意义

Freescale 大力推广的 32 位 ColdFire 系列微处理器因其很高的性价比、丰富的外围设备控制接口、广泛的型号选择范围迅速站稳了市场，并具有很好的发展前景。与国内外嵌入式微处理器市场发展迅猛相比，嵌入式集成开发环境的发展则相对滞后。CodeWarrior 等商业软件虽然调试功能强大、性能稳定，但其昂贵的价格，英文的操作界面，使许多国内用户望而却步。而国内自主研发的产品则相对较少，与国际先进水平相比存在一定的差距。

本文实现的目标就是设计一种针对 ColdFire 系列微处理器的嵌入式调试系统，通过 GNU 提供的 GDB 调试工具获取 ColdFire CPU 的运行情况，实现对底层程序甚至基于操作系统的应用程序的监控和调试。研究基于 ColdFire 的嵌入式调试系统具有如下的使用价值和现实意义。

① ColdFire 系列是 Freescale 公司推出的 32 位微处理器，其性价比很高，应用前景广泛，且目前国内没有开发出针对该系列微处理器的功能完善的嵌入式集成开发环境。因而，设计和开发 ColdFire 系列嵌入式调试系统可以为国内广大嵌入式产品研发人员提供一套廉价实用的开发调试工具，促进 ColdFire 系列微处理器在国内的推广和

应用。

② 为国内高校利用国产化开发调试工具进行嵌入式系统教学提供方便，通过提供完备的调试功能和丰富的调试手段，为 32 位嵌入式系统初学者进行实验和开发提供帮助。

③ 为研发类似的嵌入式产品提供不可多得的借鉴和经验。本文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 平台上构建和执行 GDB 的方法，人机交互调试界面的开发，目标机调试代理程序的编写以及远程调试协议的实现都适合开发其他 CPU 架构的调试环境。硬件调试平台的设计、测试方法也可在设计其他电路时借鉴。

1.5 本文工作内容和结构安排

1.5.1 工作内容

本文在了解嵌入式调试技术的基础上，重点分析了基于 GDB 的远程调试方式，通过借鉴调试桩 GDBStub 的一般工作原理，设计实现了一套针对 32 位 ColdFire 系列微处理器的集成调试系统。主要工作内容包括：

① 设计并制作了 ColdFire 硬件调试平台。该平台为 ColdFire 调试系统提供了硬件支持，是调试系统不可或缺的部分，主要功能模块或接口包括：ColdFire 最小系统、串口通信模块、以太网通信模块、A/D(Analog to Digital，模数转换)模块与扩展板接口。实现硬件调试平台的工作流程为：根据本调试系统的功能需求，划分所需的硬件模块并完成芯片选型；阅读相关芯片手册和资料，确定各模块的外围电路；设计原理图以及绘制 PCB；焊接元件，按功能模块进行硬件测试。

② 设计并实现了目标机端调试桩 GDBStub for ColdFire。调试桩 GDBStub for ColdFire 作为调试代理运行于目标机端，配合宿主机端的 GDB，实现对于 ColdFire 程序的联合调试。本文借鉴了 GDBStub 的一般实现机制，设计并实现了该调试桩，主要内容包括：研究目标芯片的工作机制，设计启动模块；通过捕获及分析 GDB 远程调试时的通信字符，实现远程调试协议；设计中断模块并实现命令处理模块。

③ 开发了宿主机端集成开发调试软件 SD-IDE for ColdFire。SD-IDE for ColdFire 内部集成了 GDB 调试器，通过串口与 GDBStub for ColdFire 交互调试数据，完成远

程调试功能，并提供人机交互调试界面。除代码调试外，用户还可通过 SD-IDE for ColdFire 实现针对 ColdFire 微处理器的代码编辑、代码编译与代码下载。SD-IDE for ColdFire 的工作主要分为以下几个部分：了解交叉编译原理，构建 ColdFire 交叉编译工具并创建连接脚本；分析 ColdFire 系列 BDM 接口的工作机制，实现 BDM 驱动程序，在此基础上实现代码写入模块；掌握 GDB 运行方式，实现调试过程中 GDB 的调度与重定向，开发与用户交互的调试界面。

1.5.2 结构安排

第一章首先概述了 ColdFire 系列微处理器的相关知识。其次分析与比较了嵌入式系统开发过程中的几种常见调试方法，并简要介绍了本文的实现思路。在讨论了当前国内外嵌入式开发调试工具发展状况的基础上，提出了本文的研究目标和实现意义。最后给出了本文的主要研究内容。

第二章分析了与本文相关的 GDB 调试技术。首先简单介绍了 GDB 的特征及总体结构。其次分析了与通信相关的 GDB 机器接口(GDB Machine Interface，GDB/MI)与远程调试协议。随后讨论了两种不同的调试代理 GDBServer 与 GDBStub。最后在阐述远程调试概念及特点的基础上，给出了本文实现的 ColdFire 调试系统的结构框架。

第三章讨论了硬件调试平台 SDMCF52233EVB 评估板的设计和实现。首先概述了相关硬件的选型原则，并对该评估板选用的主要芯片进行了简单介绍。随后给出了每个硬件功能模块的设计方案。最后详细叙述了硬件测试流程及测试心得。

第四章在借鉴调试桩 GDBStub 的基础上，按模块阐述了 GDBStub for ColdFire 的实现思路与方法。首先分析了 GDBStub 的一般结构及调试原理。随后详细介绍了每个模块的设计思路与实现方法。最后给出了基于 GDBStub for ColdFire 的调试实例。

第五章给出了宿主机端集成开发调试软件 SD-IDE for ColdFire 的详细实现，内容包括添加目标芯片的相关工程模板、交叉开发工具的构建与使用、代码写入模块的设计与实现、GDB 的调度与重定向以及人机交互调试界面的实现。

第六章对全文进行了总结，就进一步研究的问题进行讨论。

第二章 GDB 调试技术分析

在利用 GDB 调试器设计嵌入式调试系统之前，首先必须了解相关的 GDB 调试技术。本章首先介绍了 GDB 与其总体结构，随后分析了连接 GDB 与 SD-IDE for ColdFire 的 GDB/MI 接口、远程串行通信协议(Remote Serial Protocol，RSP)与 GDB 调试代理，在阐述 GDB 远程调试方式的基础上，给出了本文实现的 ColdFire 调试系统结构。

2.1 GDB 简介

GDB 是 GNU 组织免费提供的程序调试工具，一般和 GCC(GNU Compiler Collection，GNU 编译器)搭配使用，可以帮助程序开发人员了解程序运行的细节，达到调试程序的目的。

通过 GDB，用户可以查看和改变被调试程序的数据，随时停止或继续程序的运行，动态改变程序的执行环境，以查找程序中的错误。一般而言，其功能主要分为以下几部分：

① 设置断点，使被调试程序在指定的断点处停止，断点可以是条件表达式。

② 当被调试程序停止运行时，查看程序的当前状态。

③ 按照用户的自定义要求，运行被调试程序。

④ 动态改变被调试程序，以便用户修改当前错误后继续运行程序。

GDB 可以支持多种编程语言的调试，包括 GNU 所支持的所有微处理器汇编语言以及 C、C++、JAVA、PASCAL、FORTRAN 和一些其他的高级语言。在使用 GDB 调试程序之前，必须使用-g 或者-GDB 编译选项编译源程序，以生成可以供 GDB 使用的调试信息[12]。

2.2 GDB 的总体结构

GDB 可以划分为以下三个部分：用户接口，符号处理模块以及目标系统处理模块，其中符号处理模块和目标系统处理模块构成了 GDB 的内核。用户接口有命令行接口和图形界面两种，是用户与 GDB 交互信息的窗口。符号处理模块主要负责表示和提取调试信息，其功能包括读入目标文件，解释调试信息，管理符号表和输出表达

式或变量值等。而目标系统处理模块则是真正进行调试操作的部分，实现了控制代码运行，操纵目标芯片硬件，跟踪甚至分析堆栈等功能。

当用户使用 GDB 调试源程序时，用户接口首先响应某个命令请求，随后符号处理模块根据当前符号表将该命令请求转换为具体的操作，最后通过目标系统处理模块实现，并最终向用户返回源程序级的结果。

2.2.1 用户接口

用户接口不仅可以接收用户的调试命令，还能够通过调用内部算法将命令执行结果反馈给用户，因此用户接口包括输入和输出两个部分。常用的用户接口有：CLI(Command-Line Interface，控制台命令行接口)、GDB/MI 和 TUI(Text User Interface，文本用户接口)。CLI 是 GDB 默认的启动接口，它是面向用户而设计的交互接口，反馈的信息利于用户直接阅读，却不利于用软件进行解析以及信息的提取，因此 CLI 只适用于终端的命令行模式。与 CLI 不同，GDB/MI 是面向机器设计的接口，提供了更多更详细的控制命令，更利于使用软件分析 GDB 的输出信息，因此 GDB/MI 通常用于将 GDB 作为一个 GUI(Graphical User Interface，图形用户接口)或集成开发环境调试模块后端的情况。而 TUI 是一个使用 curses 库的终端接口，它可以在不同的文本窗口中显示源文件、汇编输出、寄存器值和 GDB 命令等。

2.2.2 符号处理

符号处理模块是 GDB 的主体，它由两部分组成。一是符号处理内核，包括完成符号表、段表的建立、维护与查询；另一部分是 GDB 的内部算法，即实现 GDB 各种调试命令的方法。有些内部算法可以在符号处理模块中直接完成，如计算和打印表达式，而有些则需要目标系统处理模块支持才能完成，如插入断点等。

通过目标文件符号表读取器，GDB 能够从各种不同格式的目标文件中读取符号表，如 out、ELF(Executable and Linking Format，可执行及连接格式)、COFF(Common Object File Format，通用对象文件格式)等。目标文件符号表读取器的实质是一个封装了多个函数的结构，其中每个函数支持一种目标文件格式。当需要读取目标文件符号表时，GDB 根据目标文件的格式在读取器中找到相应函数执行。

2.2.3 目标系统处理

GDB 将其能够调试的对象统称为目标，包括可执行文件、进程暂停时的内存映像文件和正在运行的进程[13]。有了目标的抽象，GDB 对各种调试对象的控制就归结为对目标的控制。为此，GDB 为目标定义了一种集合其调试控制函数的抽象数据结构——target_ops。target_ops 是一个由函数指针组成的结构体，每种类型的目标都有一个对应的 target_ops 结构。目标与其 target_ops 结构组成了一个目标系统。GDB 的调试功能正是通过多次调用其中的一个或多个函数指针指向的函数完成的。这些函数包含了 9 种基本操作：读寄存器、写寄存器、读内存、写内存、进入调试状态、退出调试状态、断点操作、单步执行和复位。通过调用不同 target_ops 结构的函数，目标系统处理模块真正实现了对于各种目标的调试操作。

2.3 GDB/MI 接口

MI 接口是针对调试器设计的文本行机器接口，其开发初衷是面向将调试器作为组件之一的复杂系统。FSG(Free Standards Group)组织的 DMI(Debugger MI)工作小组正致力于将 MI 接口变为调试器的标准机器接口，而 GDB/MI 接口的实现与应用为 MI 的标准化提供了参考。类似于 DDD(Data Display Debugger，数据显示调试器)，Insight 等以 GDB 为后端的 GUI 调试器就充分利用了 GDB/MI 接口。

GDB/MI 提供了一种面向机器的交互接口，能够更好的适应那些直接解析 GDB 输出的程序。简单地说，GDB/MI 接受字符串形式的命令输入，然后产生一行表示命令执行结果的输出。当然，这里的输入命令和输出记录都有严格的格式与内容定义，这种定义更利于软件的提取。与 CLI 接口相比，GDB/MI 不但包括了 CLI 所有命令，还具备一些 CLI 所不提供的功能，其最大的不足在于其输出内容比较复杂，必须很熟悉其输出格式才能理解，不如 CLI 输出直观。本节将依次介绍 GDB/MI 接口的输入命令与输出记录格式。

2.3.1 GDB/MI 命令简介

启动 GDB 调试器时，用户可以通过-i 或者-interpreter 调试选项打开 GDB/MI 接口。GDB/MI 的命令可分为以下几类：断点、程序环境、线程、程序执行、栈、变量、

数据、跟踪点、符号、文件和目标数据等。常用的命令如表 2-1 所示[14][15]，其中 break 命令支持通过行号和函数名等多种方式设定断点；exec 支持单步步入、单步步过、继续运行等调试指令；用户可以通过 stack 命令了解系统堆栈情况，使用 environment 设置目录，通过 thread 处理线程；var 和 data 命令则提供用户对于内存变量、寄存器等数据存储单元的操作。

表 2-1 GDB/MI 接口命令表

功能	命令格式	常用参数及含义		
断点操作	-break-参数	insert：插入断点	delete：删除断点	list：显示断点信息
执行程序	-exec-参数	run：开始执行程序，直到遇到断点或程序退出	next：单步步过执行一行代码	step：单步步入执行一行代码
		continue：继续执行程序，直到有断点或者程序退出	return：中止当前程序，立即返回	interrupt：中止正在运行的程序
目录设置	-environment-参数	pwd：显示当前工作目录	cd：设置当前 GDB 工作目录	
堆栈操作	-stack-参数	info-depth：显示堆栈深度	list-frames：列举所有的堆栈帧	
数据操作	-data-参数	list-register：显示寄存器信息	evaluate：计算表达式值	read-memory：读数据块信息
线程处理	-thread-参数	list-ids：产生一个 GDB 当前已知的线程列表	select：选择当前线程	
变量操作	-var-参数	create：创建一个变量	delete：删除一个变量	assign：给变量赋值

2.3.2 GDB/MI 的输出记录

在用户输入以上任一命令后，GDB 通过 MI 接口输出该命令对应的执行结果，其内容可分为以下三类：结果记录、流记录和 Out-of-band 记录。结果记录以“^”字符作为起始标志，反馈了当前调试系统的状态信息，包括远程目标机已连接“^connected”、系统正在运行中“^running”、成功执行命令“^done”、运行出错“^error”和系统已退出调试“^exit”等。流记录则结合了控制台输出、目标平台输出和内部日志三部分信息。控制台输出信息以“~”字符开始，由于内容与相应的 CLI 输出一致，用户可以方便的直接阅读输出结果；目标平台信息是正在进行调试的目标机的输出内容，首字符为“@”；以“&”开头的内部日志是 GDB 内部在调试过程中产生的输出信息。Out-of-band 记录用于提示用户由于某种原因 GDB 内部的状态发生了改变，其格式为“*stopped, reason=”。导致 GDB 状态变化的事件可以是程序执行到断点或观察点、程序执行完毕、收到外部信号、退出调试和单步执行完毕等。

2.4 RSP 通信协议

RSP 远程串行通信协议(简称 RSP 通信协议或远程调试协议)用于连接宿主机调试器 GDB 与被调试目标机。这是一种简单、高效、易于扩展的 ASCII(American Standard Code for Information Interchange)字符流协议，包括了建立连接、读写内存、查询寄存器、运行程序、单步执行等命令，适用于通过串口线、局域网或者其他通信方式进行信息交互的远程调试环境。在RSP通信协议中，所有的 GDB 调试命令及其反馈结果除确认标志外，都通过报文方式传输。报文由起始符“$”、数据、结束符“#”和校验码组成，即“$数据#校验码”[16]。其中数据部分各字段之间通过“,”、“:”或“;”分隔；而校验码为数据部分所有字符之和模 256 后的两位无符号十六进制数。

表 2-2 RSP 通信协议常用调试命令一览表

调试命令	功能	返回信息
H	与调试代理的握手命令	OK
q	查询段地址信息	十六进制数
g	读寄存器值	十六进制数
G	写寄存器值	OK
m	读内存数据	十六进制数
M	写内存数据	OK
c	继续执行命令	十六进制数
s	单步执行命令	十六进制数
k	结束调试进程	无
?	查询最近的信号	十六进制数

运行于目标机上的调试代理或其他监控程序在接收到宿主机发送的报文后，首先对其进行校验，判断该报文是否完整。如果检验正确，则回送标志“+”，随后解析报文得到调试命令并执行相应操作；否则回送标志“-”，要求重传报文。

RSP 通信协议支持“g”、“G”、“m”、“M”、“c”、“s”等调试命令，具体的含义如表 2-2 所示。“g”命令用于获取寄存器值，目标机在接收到此命令后，读取当前微处理器的寄存器值，按照指定的顺序以十六进制方式编码，返回给宿主机调试器 GDB；“G”命令用于写寄存器值，目标机将解析到的数据值，

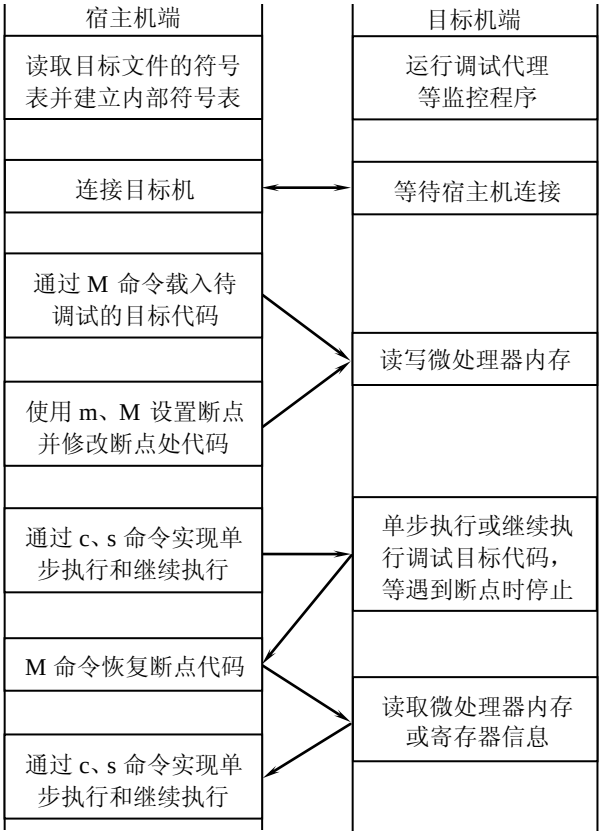


图 2-1 RSP 通信协议远程调试流程图

按顺序写入相应的寄存器中，操作成功后返回“OK”;“m”命令和“M”命令则用于读写目标机的内存数据，命令参数依次为读写内存起始地址、字节数和按照递增顺序排列的数据串(“m”命令无此参数)；GDB 通过“c”命令来继续目标机的运行，可选的参数是继续执行的地址，如果参数被省略，那么目标机就从当前程序计数器开始执行;“s”命令的参数与“c”命令类似，宿主机调试器 GDB 通过该命令要求目标机单步执行一条指令，执行完毕后返回目标机状态信息;“H”命令用于通信双方握手;“q”命令用于查询段地址信息；通过“k”命令退出调试;“?”命令则可以查询最近信号[17]。

使用 RSP 通信协议进行远程调试的流程如图 2-1 所示[13]。宿主机调试器 GDB 在建立与目标机的连接后，首先通过 M 命令载入待调试的目标代码，随后的每次断点调试过程都可分为以下几步：通过 m、M 命令设置断点并修改断点处代码；使用 c、s 命令实现单步执行与继续执行；利用 M 命令恢复断点处代码；通过 m、g 命令查看微处理器的内存数据和寄存器值。

2.5 调试代理

GDB 为目标机提供了两种不同的调试代理解决方案：调试服务器 GDBServer 和调试桩 GDBStub，而本文设计实现的调试系统选取了其中的调试桩 GDBStub 方案。调试服务器 GDBServer 建立在操作系统之上，即基于操作系统的应用程序，由于它对操作系统的依赖性，导致其无法实现操作系统级调试，只适用于应用程序级代码的调试；而调试桩 GDBStub 不依赖于操作系统，可以对整体系统甚至裸机程序进行调试，但必须与被调试程序竞争系统资源。下文简单分析了调试服务器 GDBServer 与调试桩 GDBStub 的工作原理。

2.5.1 调试服务器 GDBServer

GDBServer 主要应用于类 Unix 系统，通过 Unix 系统提供的 ptrace 调用来实现对被调试程序的访问和控制。虽然 GDBServer 有着和 GDB 同样的操作系统要求，能够运行 GDBServer 就意味着可以在目标机上移植 GDB 调试器，但是这并不能说明 GDBServer 毫无意义。与 GDB 相比，GDBServer 所需空间更少，更适合于作为 GDB 的调试代理在系统资源紧张的嵌入式目标机上运行。

GDBServer 工作流程分为启动初始化和命令处理两个阶段，如图2-2所示。在GDBServer初始化时，首先进行参数检测，判断启动命令所带的参数类型和个数；随后初始化目标机，初始化用于保存断点、寄存器和信号量的数据结构；最后，根据进程号采用相应方式启动被调试进程。完成初始化工作并建立连接后，GDBServer 进入命令处理的无限循环中：接收调试报文，解析报文执行相应命令，反馈调试结果，直到接收到终止命令报文[18]。

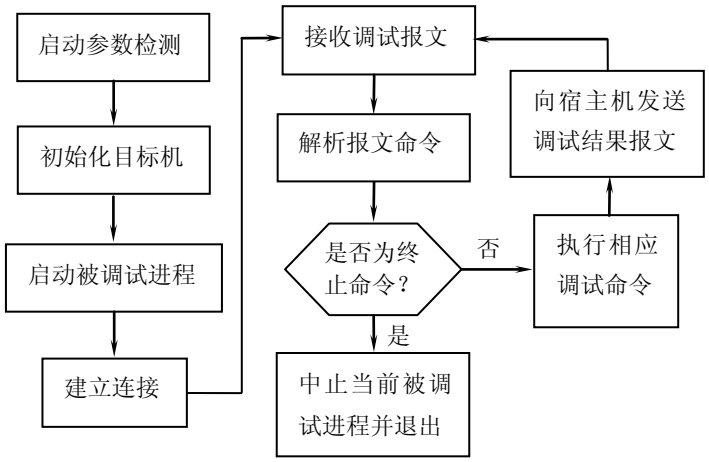


图 2-2 GDBServer 工作流程图

2.5.2 调试桩 GDBStub

调试桩 GDBStub 的实质就是用中断服务程序接管被调试程序的全部异常处理和部分中断处理，并负责与宿主机调试器 GDB 通信。其功能主要可以分为以下几点：设置中断向量，使得被调试程序发生中断时，进入设定的中断服务程序；实现中断服务程序，这是 GDBStub 的核心，被调试程序发生中断后进入此服务程序，主要完成与 GDB 的交互，处理 RSP 通信协议，读写内存数据和寄存器值等，直到接收命令继续执行被调试程序为止；设置初始断点，在目标机运行初期触发该断点，使 GDB 获得被调试程序的控制权。

与 GDBServer 不同，GDBStub 虽然不受应用级调试的限制，无需操作系统的支持，但调试基于操作系统的应用程序会显得非常复杂，因此它适用于嵌入式底层程序即无操作系统代码的调试。考虑到基于 ColdFire 系列微处理器的嵌入式调试系统主要面向调试底层代码，GDBStub 已能够满足非应用级程序的调试需求，故本文选取 GDBStub 方式实现目标机的调试代理。

2.6 远程调试

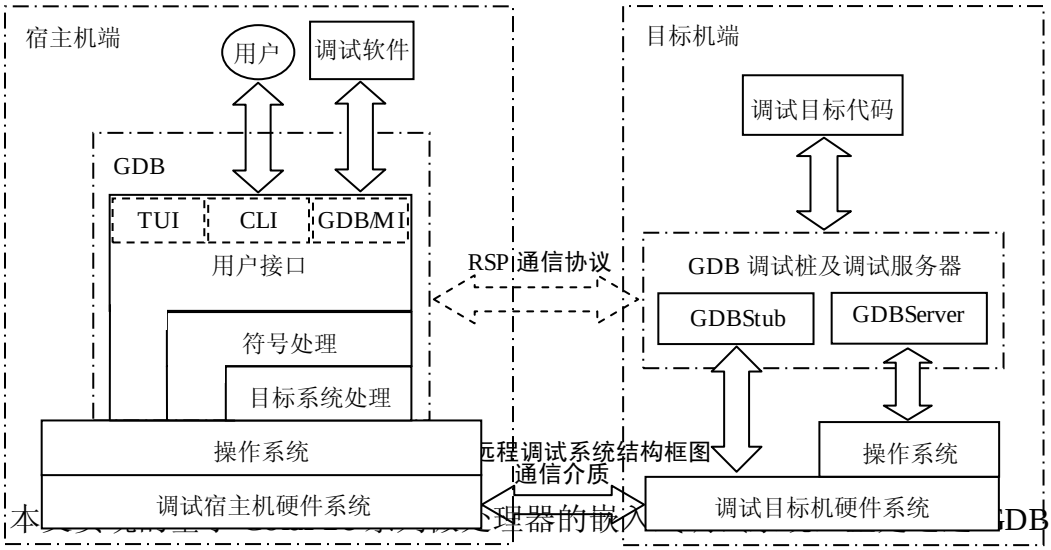
嵌入式系统由于资源限制等因素难以直接运行调试器，一般采用远程调试方式，

即用户通过一系列通信端口在宿主机上调试和运行目标机可执行代码[13]。在大多数情况下，调试器连接到目标机上，实现在线中止一个进程、插入断点或恢复进程运行等功能。调试器可以直接访问目标机微处理器的寄存器、内存数据和代码段，并且改变它们的内容。由此可见，远程调试具有以下特点：

① 调试器和被调试程序运行在不同的计算机平台上。调试器运行在一般的 PC(Personal Computer，个人计算机)或工作站上，而被调试程序运行在实际的嵌入式设备或专业的评估板上。

② 调试器通过某种通信方式与目标机建立联系。通信方式可以是串口、并口、网络等。

③ 一般在目标机上运行调试代理或其他监控程序，这些程序能与调试器配合，一起完成对目标机上运行代码的调试。



远程调试系统结构框图

本[illegible]理器的嵌入[illegible]GDB 提供的远程调试功能获取 ColdFire CPU 的内部信息，实现对目标代码的调试，GDB 远程调试结构框图如 2-3 所示。在具体的调试过程中，运行于宿主机端的 GDB 与目标机 ColdFire 微处理器通过串口建立连接。GDB 读取目标文件符号表等信息，建立被调试程序与目标代码的对应关系。ColdFire 微处理器启动目标代码运行后，用户可以通过基于 GDB/MI 接口的人机交互调试软件设置以及清除程序断点，查看内部数据，执行单步步入、单步步过、单步步出、继续执行等命令。GDB 根据 RSP 通信协议编码用

户的命令，通过串口发送给运行于 ColdFire 微处理器的调试桩 GDBStub for ColdFire。该调试桩通过查看、修改寄存器或内存单元值等行为完成相应的命令，并依据 RSP 通信协议编码处理结果，返回给 GDB。

2.7 本章小结

本章主要工作总结如下：

① 简要概述了 GDB 调试器的作用与功能，分析了 GDB 的总体结构，包括用户接口、符号处理模块与目标系统处理模块。

② 讨论了面向机器的交互接口 GDB/MI，并将其作为 GDB 与 SD-IDE for ColdFire 的通信接口。针对该接口的输入输出格式，介绍了 GDB/MI 输入命令和输出结果记录。

③ 分析了用于远程调试的 RSP 通信协议，给出了基于该协议的常用调试命令和通信中的协议报文格式，随后叙述了使用 RSP 通信协议进行调试的流程。

④ 对比了两种不同类型 GDB 调试代理的工作原理与使用范围，结果表明，调试桩更符合本文设计的 ColdFire 嵌入式调试系统的需求。

⑤ 在给出远程调试含义与特点的基础上，提出了本文设计的调试系统的结构框架，并阐述了基于该结构的调试过程。

第三章 ColdFire 硬件调试平台设计

实现面向 ColdFire 架构的嵌入式调试系统，首先需要有基于 ColdFire 微处理器的硬件调试平台的支持。本文选取 Freescale 公司于 2006 年推出的 32 位 ColdFire 系列微处理器 MCF52233，设计了硬件评估板 SDMCF52233EVB，作为开发和调试程序的硬件载体。本章首先讨论了一般嵌入式系统的选型原则，接着对 ColdFire 硬件调试平台选取的主要芯片进行了简单介绍，随后详细给出了每个硬件功能模块的设计方案和电路连接方法，最后阐述了硬件测试方法以及体会。

3.1 硬件选型

在嵌入式产品的设计过程中，硬件选型是一个非常重要的环节，它不仅直接决定了产品的设计进度与性能，还可能会影响到产品成形后的生产。针对特定的用途，如果选择功能过少的硬件模块，则无法完成系统功能；选择功能过强的硬件模块，则会浪费资源，造成性价比下降。下面将从 CPU 以及外围器件两方面，阐述本调试系统选择芯片的几点考虑。

3.1.1 CPU 的选取

一般而言，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其特性来考虑的。因此，选择一个合适的 CPU 对于嵌入式系统尤为关键。影响 CPU 选择的因素有很多，除技术需求外，非技术原因也十分重要。表 3-1 列出了主要非技术因素。

本文的目的是设计一套基于 ColdFire 的嵌入式调试系统，因此需要从 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连接扩展和网络市场空间的飞速发展，现代嵌入式系统越来越倾向于选择高性能、低价位和网络化的解决方案。Freescale 推出的 MCF5223x 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 60MHz 的时钟频率下提供 57MIPS 的处理能力[19]。考虑到本文的实际实现情况，决定选用该系列中的 MCF52233，它有 LQFP80(Low rofile Quad Flat Package，薄型四侧引脚扁平封装)和

LQFP112 两种封装，可以选择较易焊接的 LQFP80 的封装形式，而且其样片可以直接从 Freescale 网站申请。

表 3-1 选择 CPU 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 内部有硬件加速单元，运算密集时应优先选用这样的 CPU。
3	封装	目前，QFP(Quad Flat Package，四侧引脚扁平封装)、BGA(Ball Grid Array，球栅阵列封装)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	电磁兼容	应优先选用对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	内存管理	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 CPU 之前，要对它的生命周期有很清楚的了解，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 CPU 的定位，低端还是高端，消费电子还是工业应用。

3.1.2 外围器件的选取

外围器件决定着 CPU 能否完成相应的功能扩展，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连接的芯片。

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风险。

分析 ColdFire 硬件调试平台的需求可知，评估板需要电源转换芯片为 CPU 提供稳定的工作电源；串口通信是本调试系统设计和实现的硬件基础，故串口通信模块必不可少；以太网通信是 MCF52233 的主要应用领域及其存在价值的体现；A/D 转换则为嵌入式系统中较常用的功能模块。

根据以上外围器件的选择原则和本硬件评估板的功能需求，本文选用了以下芯片，如表 3-2 所示：用于以太网通信的滤波变压器 PRJ005；用于 2 路串行

表 3-2 各模块芯片选型

模块名	芯片名称	备注
以太网	PRJ005	以太网通信滤波变压器
串口	MAX232	2 路串行通信的电平转换芯片
电源转换	LM1085-3.3	降压至 3.3V 的电压转换芯片
A/D 转换	TLC2543	11 路输入 10 位精度

通信的电平转换芯片 MAX232；支持 11 路输入 10 位精度的 A/D 转换芯片 TLC2543；降压至 3.3V 的电压转换芯片 LM1085-3.3。

3.2 主要芯片简介

3.2.1 MCF52233 微处理器

1．MCF52233 微处理器主要性能

MCF52233 是 Freescale 半导体公司于 2006 年推出的一款高性能、低成本的 32 位微处理器。该系列微处理器是业内第一款在单芯片解决方案中提供以太网控制器、以太网物理层收发器和 Flash 存储器的 32 位 CPU，也是目前以太网互连领域中最小形状因素(form-factor)的解决方案。它采用了 ColdeFire V2 内核，内部集成了 10/100Mbps(Million bit per second，每秒百万位数)快速以太网控制模块(Fast Ethernet Controller，FEC)；支持半双工和全双工模式的以太网物理层收发模块(Ethernet Physical Transceiver，EPHY)；具有 256KB 的内部 Flash 和 32KB 的 RAM 空间；支持增强乘法加法运算单元和硬件加密功能；提供了 8 通道的 12 位 A/D 转换器、4 通道的 16 位脉宽调制模块(Pulse Width Modulator，PWM)和实时时钟(Real Time Clock，RTC)；并集成了三个通用异步收发模块 UART、一个队列式串行外围接口(Queue Serial Peripheral Interface，QSPI)和一个内部集成电路(Inter-Integrated Circuit，I^2C)接口；其通用输入输出引脚数目最多可达 73 个[20]。它的内部功能模块框图可参见附录 A。

2．MCF52233 以太网模块简介

MCF52233 以太网模块包括快速以太网控制模块 FEC 和以太网物理层收发模块 EPHY，这两个模块都遵循 IEEE802.3 标准。下面将分别介绍它们的特性。

(1) FEC 模块的特性

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 线工业标准接口。

② IEEE 802.3 全双工流量控制。

③ 在 50MHz 的系统时钟下即可支持 200Mbps 吞吐量的全双工操作，在 25MHz 的系统时钟下即可支持 100Mbps 吞吐量的半双工操作。

④ 如果检测到冲突则无需经过 CPU 内部总线，直接从 FIFO(First In First Out，

先入先出)发送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，无需占用 CPU 内部总线。

(2) EPHY 模块的特性

① 支持半双工和全双工模式。

② 具有介质无关接口(Medium-independent interface，MII)：支持 10/100Mbps 的速度，提供独立的 4 位发送和接收通道以及简单的管理接口。

③ 支持自动协商和自动协商下一页的能力。

④ 支持基线漂移(Baseline wander，BLW)纠正功能和远端错误检测功能。

⑤ 支持数字化适应均衡。

⑥ 25MHz 的 MDC(Management Data Clock)频率，支持 MDIO(Management Data Input/Output)前导序列。

⑦ 125MHz 的时钟产生器和时序恢复功能。

⑧ 具有内部波形集成电路并且支持循环模式。

MCF52233 的 FEC 模块和 EPHY 模块的内部结构如图 3-1 所示。图中 FEC 模块和 EPHY 模块可以独立使用，也可以集成使用。如果分开使用则通过 MII 接口的 18 个引脚与外界 EMAC (Ethernet Medium Access Control)模块或者 EPHY 模块通信；如果使用内部模块，那么 MII 接口引脚对这两个模块均不可见。本文在设计时使用内部 FEC 模块和 EPHY 模块，这样以太网模块对外通信引脚只有 4 个：2 个输入(RXN、RXP)和 2 个输出(TXN、TXP)。RXN、RXP 接收一对双绞线的输入信号；TXN、TXP 向一对双绞线发送差分信号。

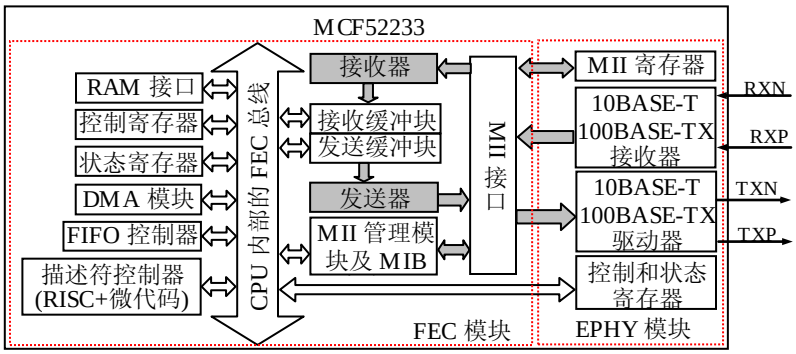


图 3-1 MCF52233 的以太网模块结构框图

3.2.2 PRJ005 以太网滤波变压器

PRJ005 是深圳市脉普特通讯技术有限公司推出的一款 10Mbps/100Mbps 以太网

滤波变压器，采用 RJ45 连接头底座的封装，遵循 IEEE802.3u 的规范。以太网滤波变压器又称为网络隔离芯片或网络隔离变压器，它在网络通信过程中所起的作用主要有两个：一是传输数据，将物理层输出的差分信号通过差模耦合的线圈进行耦合滤波以增强信号，并且通过电磁场的转换耦合到不同电平的连接网线的另外一端；二是隔离网线所连接的不同网络设备间的不同电平，防止网线传输不同电压时损坏设备。除此之外，以太网滤波变压器还能对设备起到一定防雷保护作用[21][22]。

图 3-2 PRJ005 的原理及实物图

以太网滤波变压器的选择根据耦合电路功能而定，具体可参考以太网滤波变压器资料[23]。图 3-2[24]给出了 PRJ005 网络变压器的原理图及实物图。引脚含义说明如下：

CT(4、5 脚)：一般接电源正极，本文接正 3.3V。

R-、R+(6、3 脚)：数据信号发送引脚，分别接 CPU 的 RXN、RXP。

T-、T+(2、1 脚)：数据信号发送引脚，分别接 CPU 的 TXN、TXP。

J6、J3、J2、J1：连接双绞线的 4 根传输线。

3.2.3 TLC2543 A/D 转换器

1．TLC2543 的主要性能

TLC2543 芯片是 TI 公司从 1998 年开始在我国推广的 12 位 A/D 转换芯片，使用开关电容逐次逼近技术完成 A/D 转换过程。其主要性能如下：

① 12 位分辨率 A/D 转换器。

② 在工作温度范围内的转换时间为 10μs。

③ 11 个模拟输入通道。

④ 可编程的 MSB(Most Significant Bit)或 LSB(Least Significant Bit)前导。

⑤ 可编程的输出数据长度。

2．TLC2543 的内部结构

TLC2543 内部包含一个 14 通道模拟多路器、采样与保持电路、12 位 A/D 转换器、

图 3-3 TLC2543 内部结构图

12 选 1 数据选择器及驱动器、输入寄存器、输出数据寄存器、控制逻辑及 I/O(Input/Output，输入/输出)计数器以及其他端口，其内部结构如图 3-3 所示[25][26]。

TLC2543 与微处理器芯片的接口部分有五个引脚，分别是：时钟输入(I/O Clock)、串行控制字输入(Data Input)、片选输入($\overline{CS}$)、A/D 转换串行数据输出(Data Output)、转换结束电平输出(End Of Conversion，EOC)。TLC2543 与具有 SPI 或相同功能接口的微处理器可以直连，对于没有 SPI 接口的微处理器可通过软件编程模拟 SPI 操作[27]。

3.3 硬件调试平台的设计

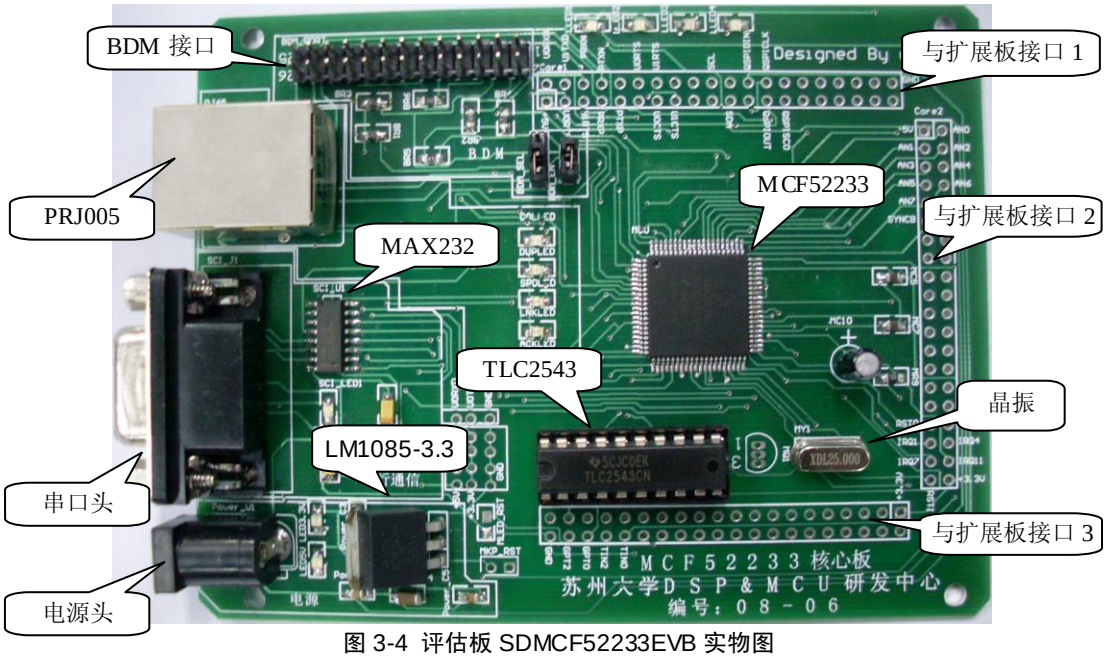


图 3-4 评估板 SDMCF52233EVB 实物图

硬件评估板 SDMCF52233EVB 采用 2 层 PCB 板设计，外部晶振频率为 25MHz，内部总线频率达到 60MHz，接口和外围设备丰富，实物如图 3-4 所示。本节将给出硬件评估板的最小系统与各外围设备的原理框图，并对外围设备接口进行说明，涉及的外围模块包括以太网通信、串口通信与 A/D 转换。附录 B 则给出了硬件评估板的详细原理图。

3.3.1 最小系统硬件设计

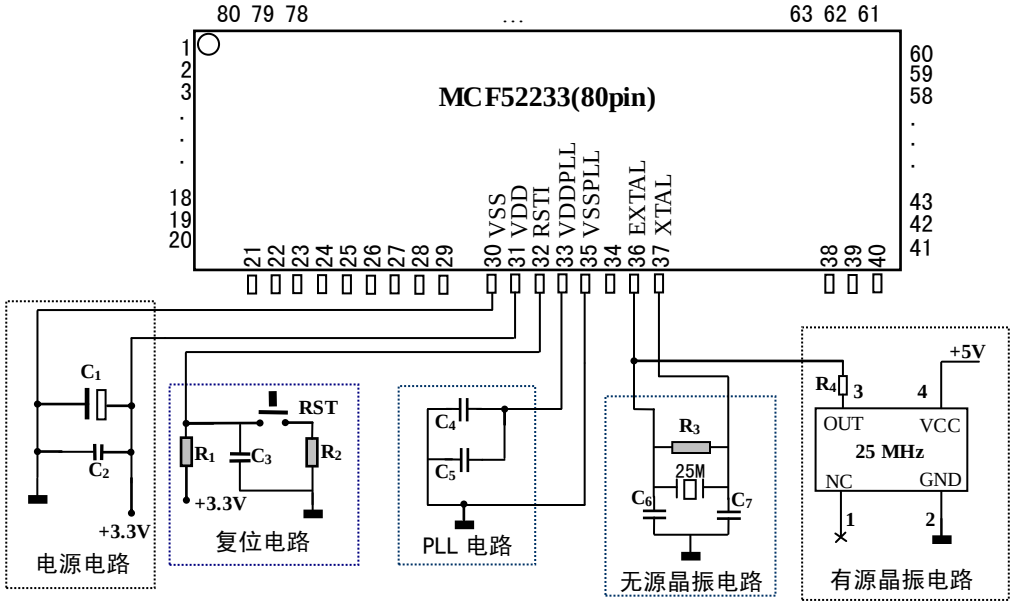


图 3-5 MCF52233 最小系统支撑电路

ColdFire 系列微处理器的硬件结构中仅有一个 CPU 是无法工作的，它必须结合其他相应的外围支撑电路，才能构成一个内部程序运行所必需的工作环境即最小系统。ColdFire 系列 CPU 的最小系统一般包括电源电路、复位电路、PLL(Phase Locked Loop，锁相环)电路、晶振电路、BDM 接口电路。MCF52233 芯片最小系统支撑电路示意图如图 3-5 所示，其中各个部分功能如下：

① 电源电路主要给 CPU 提供 3.3V 电源。

② 复位电路主要完成系统上电复位和系统在运行时用户按键复位。

③ PLL 电路主要用于倍频外部输入的时钟信号，以提高芯片内部工作频率。

④ 晶振电路用于给 CPU 提供外接的石英晶振输入，可支持有源和无源两种。

⑤ BDM 接口电路主要负责与 BDM 片上调试器(以下简称 BDM 头)相连，向

MCF52233 写入程序并可以完成一些简单的硬件调试功能。

1．电源电路

SDMCF52233EVB 硬件评估板采用单一的 5V 直流电源直接供电。使用了电压转换芯片 LM1085-3.3 来提供 3.3V 的直流电源。5V 电源可供 A/D 转换芯片 TLC2543 和串行通信电平转换芯片 MAX232 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 作为以太网滤波变压器 PRJ005 的电源，并为 MCF52233 内核提供工作电压。

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。如图 3-5 中的电源电路所示，其中 C_1 和 C_2 构成滤波电路，其电容值分别为 4.7 μF 和 0.1 μF，这样设计电源电路可以改善系统的电磁兼容性，降低系统对电源的高频干扰，增强电路工作稳定性。

2．复位电路

ColdFire 系列微处理器在响应各种外部故障或侦测到内部系统故障时，自动进行系统复位。一旦 CPU 检测到复位信号，立即将寄存器和控制位恢复成默认值，从而触发复位。系统复位的主要目的是进行错误修复，即当微处理器检测到内部故障后，尝试回到一个已知的、明确的工作状态。

芯片硬件复位电路如图 3-5 中的复位电路部分所示。正常工作时，由于 RSTI 引脚通过 4.7K 上拉电阻 R1 接到电源正极，所以应为高电平。若按下复位按钮，则 RSTI 脚通过 100Ω 电阻 R2 接地，芯片复位。

3．PLL 电路

片内 PLL 电路兼有频率放大和信号提纯的功能。通过内部 PLL 电路，系统可以通过较低的外部时钟信号获得较高的工作频率，以降低因高速开关时钟所造成的高频噪声。图 3-5 中的 PLL 电路主要起到滤波作用，VDDPLL 引脚由芯片内部提供电压。

4．晶振电路

晶振电路用于向微处理器及其他硬件模块提供工作时钟。晶振电路的设计非常关键，无论设计过程中出现何种偏差即便是辅助元器件参数选用不当，都会造成时钟电路工作不稳定，导致整个系统无法正常工作。因此初次设计时钟电路时，建议使用有源晶振作为外部时钟源。

外部有源时钟电路的接法如图 3-5 中的有源晶振电路所示：有源晶振的 4 脚接 5V

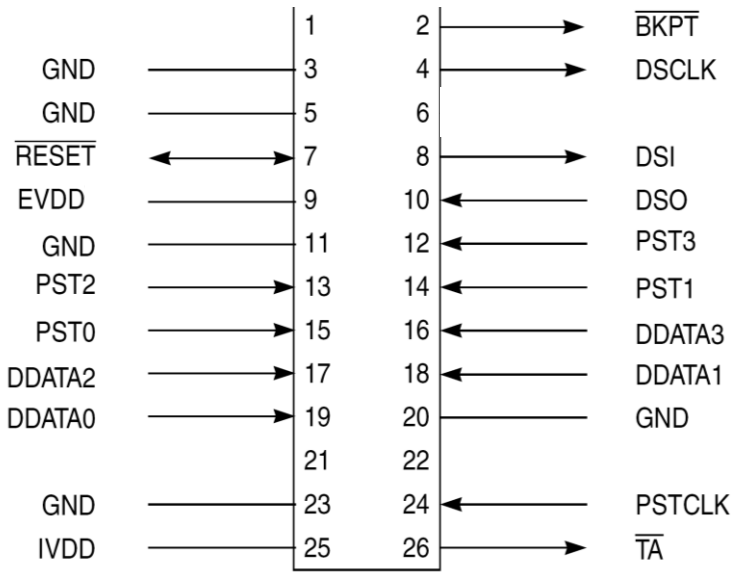
电源，1 脚悬空，2 脚接地，3 脚为晶振的输出，可通过一个小电阻(此处为 22 欧姆)接芯片的 EXTAL 引脚。

外部无源晶振需要接在芯片的外部晶振输入引脚 EXTAL 和 XTAL 上，外接无源晶振的接口电路如图 3-5 中的无源晶振电路所示：其中元器件 C_6 和 C_7 起到滤波作用，电容值均为 15pF，而 R_3 为 10M 欧姆的电阻。

5．BDM 接口电路

在 Freescale 的众多微处理器中，有着多种不同类型的 BDM 接口标准。HCS08 和 HCS12 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。PowerPC 系列内部有专门用于调试的微指令，它能够暂停正常的机器码，转而执行设定的 BDM 命令。ColdFire 的 BDM 接口模式类似于 PowerPC，并在其基础上新增了实时监控功能[28]。

BDM 接口功能的实现需要 BDM 头的协助。BDM 头负责完成宿主机并口与 BDM 调试端口之间操作时序、操作逻辑及电压的转换。而通过宿主机端的 BDM 驱动程序，宿主机建立与目标机的通信，实现对 Flash 存储器、RAM 区域及 CPU 内部寄存器的读写及程序执行流程的控制。本文使用了苏州大学 Freescale 实验室自主研发的 ColdFire BDM 头，其一端通过并口与宿主机连接，另一端通过 26 针 IDE(Integrated Drive Electronices interface)插座与SDMCF52233EVB硬件评估板相连。该 26 针插座对应着标准 BDM 接口，其引脚定义如图 3-6 所示。



3.3.2 以太网通信

MCF52233 集成的 FEC 模块及 EPHY 模块主要用于实现 MCF52233 与以太网之间的数据通信。MCF52233 以太网接口硬件连接如图 3-7 所示。

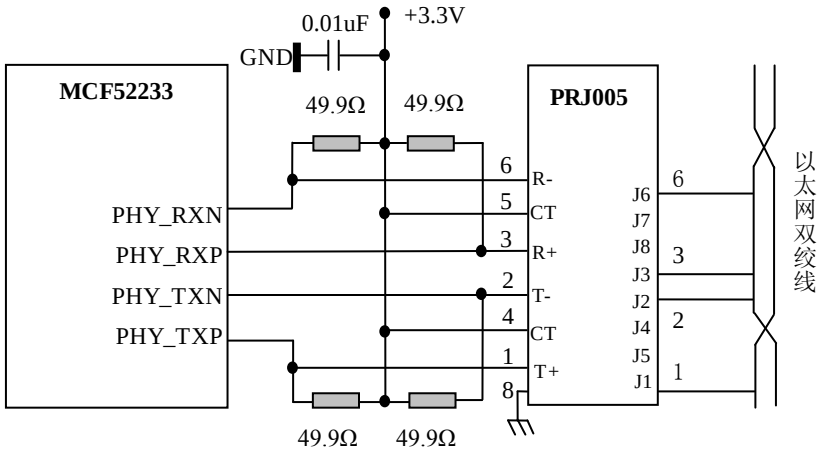


图 3-7 MCF52233 以太网接口电路原理图

MCF52233 的以太网接口模块实现了 TCP/IP 协议模型中的网络接口层功能。结合图 3-1 和图 3-7，可以分析出 MCF52233 的网络接口模块的主要功能。接收时，首先 EPHY 模块的 10BASE-T/100BASE-TX 接收器接收从外部双线引脚 PHY_RXN/PHY_RXP 输入的信号，然后进行曼彻斯特解码或 4B/5B 解码，并将解码值通过 MII 接口自动传送给 FEC 模块；FEC 模块中的接收器将从 MII 接口接收到的以太帧数据存放至 RAM 区域的多个接收缓冲块中，CPU 根据接收缓冲块描述符的具体信息，获取每个缓冲块的起始地址和数据长度，从而访问接收缓冲块中的详细数据。发送时，把需要发送的以太帧数据存放至发送缓冲块，并将各个缓冲块的详细描述信息写入发送缓冲块描述符中，通过发送控制寄存器启动发送命令，然后 FEC 模块的发送器根据发送缓冲块描述符，自动将发送缓冲块内的数据通过 MII 接口传送给 EPHY 模块；EPHY 模块对接收到的数据信号进行曼彻斯特编码或 4B/5B 编码，最终通过 10BASE-T/100BASE-TX 驱动器，将编码信号从双线引脚 PHY_TXN/PHY_TXP 输出。

图 3-8 串行电平转换电路

3.3.3 串行通信

通用异步收发器 UART 是用于处理 CPU 与串行设备通信的控制逻辑，也可称为串行通信模块。串行通信接口是微处理器与外界进行通信的重要方式，所有串行通信接口都具有发送引脚 TxD 和接收引脚 RxD，它们是 TTL 电平引脚。在 CPU 中，若用

RS-232C 总线进行串行通信，则需外接电平转换电路，在发送端要用驱动电路将 TTL 电平转换成 RS-232C 电平，在接收端用接收电路将 RS-232C 电平转换为 TTL 电平。

电平转换器可以由晶体管分立元件构成，也可以直接使用集成电路。目前业内使用 MAX232 芯片[29]较多，该芯片使用单一正 5V 电源供电，实现 TTL 电平到 RS-232C 电平的转换，基本电平转换电路如图 3-8 所示。MAX232 芯片支持两路电平转换，在实际使用时，若只需要一路串行通信，可以选择其中任一组。

3.3.4 A/D 转换模块

SDMCF52233EVB 的 A/D 转换模块在电路设计时可采用两种方式：使用 MCF52233 内部集成的 8 通道 12 位精度 A/D 转换模块；利用内部 QSPI 模块连接扩展的 11 通道 12 位精度 A/D 转换芯片 TLC2543。为了支持更多的采集对象(包括各种温度、光敏传感器等物理量输入器件)，本文选用了一片 TLC2543，与 CPU 的 QSPI 接口相连，并占用 QSPI 片选 CS0 位，其引脚分配如表 3-3 所示。

表 3-3 TLC2543 引脚描述

引脚	名称	描述	MCF52233 引脚
15	CS	片选信号	QSPI_CS0
18	I/O CLK	输入输出时钟	QSPI_CLK
17	DIN	数据输入	QSPI_DOUT
16	DOUT	数据输出	QSPI_DIN
1-9，11-12	AIN[11:0]	模拟信号输入	无
14	REF+	参考电压正	无
13	REF-	参考电压负	无

3.3.5 与扩展板接口

SDMCF52233EVB 评估板设计了 3 排 DIP(Dual-In-line Package，双列直插封装)接口，通过这些扩展接口可以和苏州大学 Freescale 实验室开发的扩展板对接。扩展板主要集成了嵌入式系统实验中常用的各种模块的接口电路，这样评估板只需设计 CPU 的最小工作电路、基本外围模块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：串口通信、以太网通信、I^2C 通信、A/D 采样、键盘输入、液晶显示等。

3.4 测试及体会

3.4.1 测试方法

在制作硬件电路板时，电路原理设计、制板、焊接以及元件质量等问题都会引起

硬件电路板无法正常工作。因此，在进行硬件测试时，必须遵守一定的方法。硬件测试方法主要有如下几种：

① 利用万用表测量引脚的电压是否正常，以及两个引脚间是否短接或虚焊。

② 通过示波器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 头测试 CPU 是否正常运行。

④ 编写基本的模块测试程序，确定每个硬件模块是否能够稳定工作。

3.4.2 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

焊接好电压转换芯片 LM1085-3.3、电源接头以及滤波电容后，可以测量 LM1085-3.3 输入电压是否为正 5V，输出电压是否为正 3.3V，并且输出稳定无波动。

(2) 主控芯片测试

主控芯片 MCF52233 的基本工作电路包括晶振电路、PLL 电路、复位电路以及芯片电源滤波电路。根据各模块的功能，首先应焊接晶振电路。如果使用的是有源晶振，则不需要 CPU 的干预就可以直接用示波器测出产生的晶振是否为 25MHz。随后就可以焊接主控芯片、PLL 电路和复位电路了。由于 MCF52233 的内部存储器中没有固化程序，因此必须借助 BDM 头来测试 CPU 是否能够正常工作。将 BDM 头与评估板上的 26 针 BDM 接口连接，运行宿主机端的测试软件，即通过 BDM 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 头写入到内部的 RAM 中运行。

(3) 以太网测试

由于 MCF52233 以太网接口包括 FEC 和 EPHY 两个模块，因此以太网接口的硬件测试可以按如下几步进行：

① FEC 模块支持循环模式，在不加任何外围电路的情况下，可禁止 MII 接口，使 FEC 发送器和接收器在芯片内部直接通信。因此先对 FEC 模块在自循环模式下进行测试，观察 FEC 模块工作指示灯，以确定 FEC 模块能正常工作。

② 在 FEC 模块正常工作的基础上，进行了 EPHY 模块的测试。同样 EPHY 模块也支持数字循环模式，在与外界通信介质断开的情况下，可使 EPHY 的 TXD 数据直

接发送给 RXD。因此采用该模式并观察 EPHY 模块工作指示灯，可确保 EPHY 模块正常工作。

③ FEC 和 EPHY 模块都正常了，接下来用交叉网线将评估板与宿主机连起来，并给评估板上电，看宿主机的“本地连接”图标是否能检测到有 10M 或 100M 以太网卡存在。若没有表示网络接口硬件上有故障。参考电路图，检查各引脚的电平状态以及接收发送引脚是否接反，确定相关引脚是否满足电路图的要求。

④ 在 MAC(Medium Access Control，介质访问控制)地址、通信速度以及双工模式等初始化内容完成后，给评估板配置 IP(Internet Protocol，互联网络协议)地址，然后从 PC 机 Ping 该 IP 地址，观察 EPHY 的接收指示灯是否闪烁，闪烁表示正在接收数据。

⑤ 向 FEC 的发送缓冲块中写入数据并启动发送命令，观察 FEC 的发送信包指示灯是否闪烁，闪烁表示正在发送数据。

⑥ 在网络发送和接收都正常的情况下，硬件评估板发送 ARP(Address Resolution Protocol，地址解析协议)请求报文，同时接收宿主机发来的 ARP 响应报文，并将响应报文通过串口发往宿主机，从而检测以太网接口一次完整的收发操作，如果报文格式分析结果正确，则说明以太网接口完全正确工作起来了。

(4) 其他模块的测试

QSPI、A/D 转换、串口以及与扩展板接口的测试主要是要编写相应的测试子程序，其硬件连接相对简单，测试相对容易。

3.4.3 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，就开始进行绘制 PCB 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 MC9S12NE64 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通信模块需要跳线。

(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

通过后，才可以继续焊接。因为一旦有错误，就可以很快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 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 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 0.1 μF 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，PCB 更美观。

3.5 本章小结

主要工作总结如下：

① 探讨了嵌入式系统的选型原则，对 ColdFire 硬件调试平台选取的主要芯片进行了简单介绍，包括 MCF52233 微处理器、网络滤波变压器 PRJ005 与 A/D 转换芯片 TLC2543。

② 依次讨论了 MCF52233 最小系统、以太网通信模块、串行通信模块和 A/D 转换模块的设计与实现，给出了 SDMCF52233EVB 与扩展板的接口。

③ 结合实际开发经验，阐述了硬件调试平台的测试方法与步骤，并给出了测试过程中的一些体会。

第四章 GDBStub for ColdFire 的设计与实现

ColdFire 嵌入式调试系统包括 ColdFire 硬件调试平台、目标机调试代理与运行于宿主机的 GDB 调试软件三部分。其中调试代理通过通信端口接收宿主机调试器 GDB 发送的调试命令，并将命令执行结果反馈给 GDB，实现对于被调试程序的控制和访问。本文借鉴一般 GDBStub 的调试原理与工作机制，设计并实现了针对 ColdFire 微处理器的调试桩 GDBStub for ColdFire，作为目标机端的调试代理，通过基于串口的 RSP 通信协议与 GDB 交互调试信息。本章首先分析了 GDBStub 的一般结构和调试原理，然后按各模块详细阐述了调试桩 GDBStub for ColdFire 的设计思路与实现方法，最后给出了使用该调试桩的调试实例。

4.1 GDBStub 的结构

GDBStub 作为运行于目标机系统上的调试代理，提供了被调试程序与 GDB 调试器之间的联系。这种联系通常屏蔽了目标机的具体硬件特征，使运行于宿主机端的 GDB 能够更有效可靠的获取目标机微处理器的内部信息并监控被调试程序。通常情况下，GDBStub 可以分为以下四部分：启动模块、中断模块、通信模块和命令处理模块，如图 4-1 所示。其中启动模块主要完成对目标机系统的初始化，包括初始化基本硬件模块，设置堆栈，填写中断向量表，分配 RAM 空间等。在目标机系统成功启动后，GDBStub 通过不断执行陷阱指令，触发 CPU 进入相应的陷阱中断服务程序。中断模块是 GDBStub 的核心，也是其最复杂最关键的部分，主要用于处理被调试程序运行过程中产生的异常(包括陷阱)，并调用通信模块与命令处理模块。基于 RSP 协议的通信模块则负责实现与宿主机端 GDB 交互数据。最后，作为 GDBStub 执行单元的

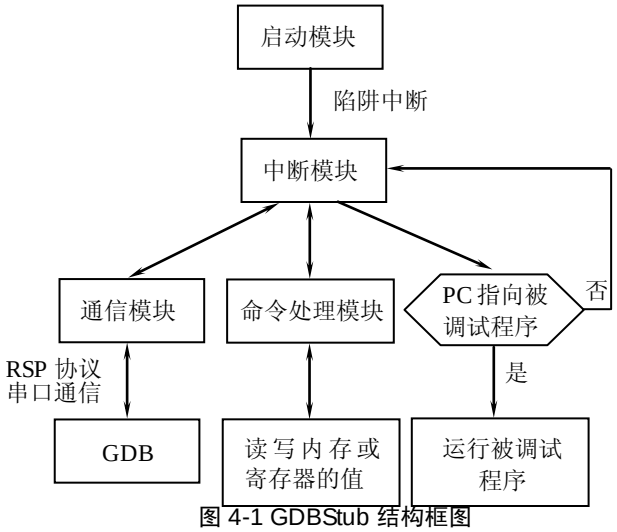


图 4-1 GDBStub 结构框图

命令处理模块通过修改 CPU 内部存储器与寄存器(包括程序计数器 Program Counter，简称 PC)值，实现对于程序的调度与调试。

4.2 GDBStub 调试原理分析

使用 GDBStub 调试程序时，应首先将 GDBStub 可执行代码写入到目标机系统非易失性存储器的特定单元中，使得目标机微处理器每次上电后都能立即运行 GDBStub。在确保 GDBStub 启动运行和串口通信正常后，调试者可通过 RSP 协议实现宿主机端 GDB 与 GDBStub 间的串行通信，将被调试程序的目标代码装载到目标机 CPU 的设定 RAM 区域，这样就可以通过对 RAM 空间数据的读写实现对程序的跟踪和监控。GDBStub 支持的调试功能主要有设置断点、运行调试程序、继续执行、单步执行、查看变量和寄存器值等，下面将逐一阐述其实现原理。

4.2.1 设置断点

断点是调试器控制程序执行的基本手段，可分为硬件断点和软件断点。硬件断点方式利用 CPU 内部断点寄存器存储断点处地址，并监视地址总线，一旦地址匹配则强制 CPU 进入调试模式，因此硬件断点方式可用于任意地址，但断点个数有限；而软件断点方式则利用 CPU 内部存储器存放断点信息，且监视从存储器取出的数据，因此软件断点理论上可以设置任意多个，但只适用于 RAM 等便于读写的存储区域[30]。

GDBStub 采用软件断点方式，其断点设置方法就是用陷阱指令替换断点处的程序指令，使得程序运行到断点处时，执行的是陷阱指令而不是原有程序代码，通过触发相应的陷阱中断进入 GDBStub 的中断模块。由于 GDBStub 进入中断模块后，暂停了被调试程序的执行，并调用通信模块等待 GDB 发送下一步调试命令即将当前的 CPU 控制权交给了宿主机，因此 GDB 就可以通过 RSP 协议获取此时断点处的 CPU 内部信息，包括内存数据和寄存器值等，实现了被调试程序在设置断点处暂停同时反馈调试信息的功能。例如，在调试基于 ColdFire 微处理器的代码时，调试者希望在源程序的第 10 行 movel 指令(其目标代码为 0x202e)处设置断点，GDBStub 则可以使用陷阱指令 Trap15(其目标代码为 0x4e4f)取代 movel，具体做法是将两字节数据 0x4e4f 覆盖存储 movel 指令目标码 0x202e 的内存区域。当调试者取消在第 10 行设置的断点或调

试过程中需要执行该 movel 指令时，GDBStub 则恢复目标码 0x202e。

需要注意的是GDBStub对于断点的操作是在继续执行或单步执行过程中实现的。即当调试者设置断点后，GDBStub 并不立刻修改内存中的断点指令，而将断点设置操作推迟到下一次继续执行或单步执行的起始阶段；当继续执行或单步执行结束后，再将原断点处指令改回，以保证内存数据的正确性和一致性。

4.2.2 运行调试程序

目标机系统运行 GDBStub 完成对各硬件模块的初始化后，通过主函数中的陷阱指令(可以是设置断点时使用的陷阱指令)进入中断模块，使 GDB 获得当前目标机系统的起始控制权。由于使用 RAM 存放被调试程序对应的目标代码，因此 GDB 可以通过读写内存单元操作将目标代码按字节顺序逐个写入指定 RAM 区域中，同时记录该 RAM 区域首地址。完成加载调试代码后，如果没有进一步调试命令，GDBStub 将返回被中断程序处继续执行。

当调试者发送开始调试命令后，GDB 向调试桩发送写 PC 程序计数器命令，将其值修改为目标代码 RAM 首地址。这样程序在中断返回时，并不跳回原中断处继续运行，转而执行已加载到内存中的调试目标代码，实现了对于调试目标代码的运行调度。

4.2.3 继续执行与单步执行

继续执行是指 CPU 从当前代码运行处开始，向下持续执行源程序，直到遇到断点或运行出错为止。其实现过程如图 4-2 所示：GDBStub 首先通过替换断点处指令实现设置断点操作；随后从当前程序计数器 PC 中加载程序地址运行，直到遇到断点处陷阱中断指令；最后在中断模块中 GDBStub 将断点处指令恢复，并将当前中断处地址写入程序计数器 PC。

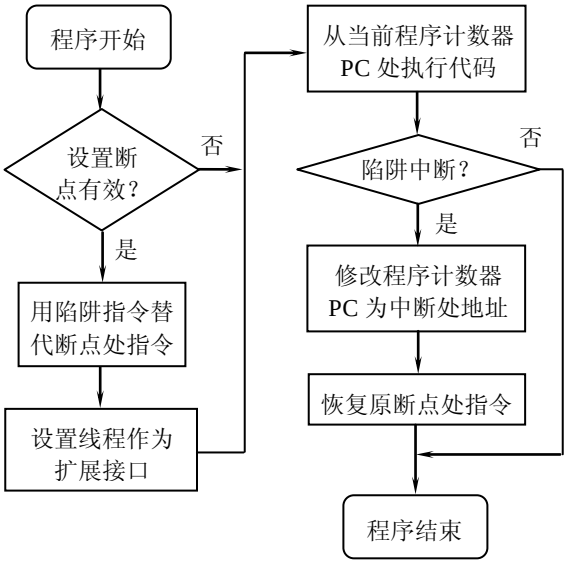


图 4-2 继续执行流程图

单步执行在整个调试过程中，具有非

常重要的作用。调试者可以通过单步执行观察被调试程序的当前状态，在程序运行遇到错误时分析出现异常的过程和原因。所谓“单步”是指 CPU 从当前代码运行处向下逐行执行源程序，每次只执行一行源代码，随后即停止源程序的运行。单步执行存在多种方式，如单步步入、单步步过等，其区别在于对函数的处理不同。单步步过将调用的函数看作原子性操作即单行代码，运行时直接跳过函数调用直到下行源程序；而单步步入则进入调用的函数内部，逐行执行其代码。

单步步过的实现方法与继续执行类似，只是 GDB 在用于记录当前断点状态的链表中加入了内部临时断点并使其指向下一行源程序(把函数调用看作单行代码)，促使 CPU 执行完每一行源程序后进入陷阱中断，以暂停当前调试程序的运行。而单步步入则利用了微处理器的指令级单步中断，在继续执行的基础上加入了单步中断功能，使微处理器每执行完毕一条指令就触发中断。

4.2.4 读写变量和寄存器值

GDBStub 进入中断模块后，调用通信模块与宿主机 GDB 交互信息，其中包括当前微处理器内部的 RAM 区域、Flash 存储器和寄存器值。对于 Flash 存储器中的数据值，GDBStub 可以通过地址访问方式对其进行读操作；而对于 RAM 区域中的数据值，GDBStub 则可以简便、快速的进行读写操作；但对于 CPU 寄存器，GDBStub 无法通过地址进行直接访问，其读写必须通过内存映像。

GDBStub 读写 CPU 寄存器的过程可分为以下几步：首先在进入中断服务程序后，GDBStub 利用系统堆栈的部分 RAM 空间，将 CPU 寄存器值复制到设定的变量数组中；随后 GDBStub 将对于 CPU 寄存器的一切读写操作都转移到该变量数组上，即将变量数组作为目标对象进行读写操作；最后在中断模块进入中断返回状态时，将变量数组的改动写回 CPU 内部寄存器。

4.3 启动模块软件设计

由于启动模块主要完成对目标机系统的初始化，包括初始化基本硬件模块，设置堆栈，配置中断向量基址，分配 RAM 空间等，因此其代码与微处理器硬件体系架构关系密切，一般使用汇编语言实现。本文针对 MCF52233 微处理器设计并编写了

GDBStub for ColdFire 启动模块程序，其实现步骤主要如下：

① 屏蔽所有中断。

② 允许使用内部 RAM。

③ 设置堆栈指针，使其映射到 RAM 空间。

④ 初始化片内外设基址寄存器(Internal Peripheral System Base Address Register，IPSBAR)。

⑤ 允许使用内部 Flash 存储器。

⑥ 调用 MCF52233 的各个内部模块的初始化函数，主要包括设置 IRQ(Interrupt ReQuest)中断，初始化看门狗，配置系统时钟，设置片内 RAM，初始化串口，设置通用 I/O 口，初始化以太网口，配置中断向量基址，初始化程序运行环境等。

⑦ 跳转到主函数执行。

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

    /* 初始化RAMBAR1，允许使用内部RAM */
    move.l  #__RAM,d0
    add.l   #0x21,d0
    movec   d0,RAMBAR1

    /* 设置堆栈指针 */
    move.l  #__SP_INIT,sp

    /* 初始化IPSBAR */
    move.l  #__IPSBAR,d0
    add.l   #0x1,d0
    move.l  d0,0x40000000

    /* 初始化RAMBAR0，允许使用内部Flash */
    move.l  #__FLASH,d0
    add.l   #0x21,d0
    movec   d0,RAMBAR0

    /* 初始化MCF52233的各个模块 */
    jsr     mcf52233_init

    /* 跳转到主函数执行 */
    jsr     main
```

GDBStub for ColdFire 在运行之前必须对程序运行环境进行必要的初始化，主要包括以下三方面的内容：

① 将位于 Flash 区域起始地址的中断向量表复制到 RAM 空间中，以加快访问中断向量表的速度。

② 将已经初始化的数据(如赋有初值的全局变量)搬运到 RAM 区域，这个区域也称为 data 段。在基于 ROM 的嵌入式系统中，已经初始化的数据在程序运行之前保存在 ROM 中，在程序运行过程中这些数据可能需要修改。因而，在运行之前需要将这些数据搬运到可读写的 RAM 空间中。

③ 在可读写存储区建立 ZI(Zero Initialize，初始化为零)属性的数据区，这个区也称为 bss(Block Started by Symbol)段。bss 段包括了所有在程序中定义却未初始化的全局变量，在使用前需要将其统一清零，以防止数据值的不确定导致程序运行异常。通常当程序保存在 ROM 中时，目标代码中不包含 ZI 属性数据，因此在运行程序前，应在系统可读写存储区域建立 ZI 属性的数据区，将 bss 段占据的存储空间全部清零。

```
/* 将位于Flash区域的中断向量表复制到RAM空间 */
if (__VECTOR_RAM != VECTOR_TABLE)
{
    for (n = 0; n < 256; n++)
    __VECTOR_RAM[n] = VECTOR_TABLE[n];
}

/* 将ROM区域的已初始化数据复制到RAM区域 */
if (__DATA_ROM != __DATA_RAM)
{
    dp = (uint8 *)__DATA_RAM;
    sp = (uint8 *)__DATA_ROM;
    n = (uint32)(__DATA_END - __DATA_RAM);
    while (n--)
    *dp++ = *sp++;
 }

 /* 将未初始化数据清0 */
 if (__BSS_START != __BSS_END)
 {
    sp = (uint8 *)__BSS_START;
    n = (uint32)(__BSS_END - __BSS_START);
    while (n--)
    *sp++ = 0; }
```

4.4 RSP 通信模块的实现

宿主机调试器 GDB 通过 RSP 通信协议与 GDBStub for ColdFire 协同工作，从而控制当前 ColdFire 微处理器的运行，并与其交互数据信息。常用的 RSP 协议调试命令包括：读寄存器值 g、写寄存器值 G、读内存数据 m、写内存数据 M、继续执行 c、单步执行 s 和结束调试 k。RSP 通信模块负责解析宿主机调试器 GDB 发送的报文信息，并从中提取调试命令交予命令处理模块执行；当 GDBStub for ColdFire 需要向宿主机反馈调试信息时，通信模块则将数据封装成报文，调用串口驱动程序发送给宿主

机。本节给出了串口驱动程序的设计与 RSP 协议的实现流程。

4.4.1 串口驱动程序设计

本文设计了基于 MCF52233 微处理器的串口驱动程序，实现了串口字节数据的发送与接收，作为通信模块的底层子模块调用。在启动模块完成对于串口的初始化(包括设置收发方式，配置波特率，设定通信格式等)后，通信模块就可以通过调用串口驱动程序收发 RSP 报文了。

发送一个字节的操作如下：通过判断串口状态寄存器(UART Status Register，USR)的第 2 位 TxRDY 是否为 1，等待 CPU 的发送器为空，为空则将一字节待发送数据放入发送缓冲寄存器(UART Transmit Buffer，UTB)中。而接收一个字节的过程如下：通过判断串口状态寄存器 USR 的第 0 位 RxRDY 是否为 1，等待 CPU 的接收器不为空，不为空则从接收缓冲寄存器(UART Receive Buffer，URB)中读取一字节数据；若等待一段时间后，没有读到数据就置失败标志。下文给出了串口收发一字节数据的子程序：

```
//UARTSend1:串行发送1个字节---------------------------------------*
//功能:串行发送1个字节                                           *
//参数:要发送的数据                                             *
//返回:无                                                       *
//-------------------------------------------------------------*
void UARTSend1(INT8U o)
{   while (!(ReSendStatusR & SendTestBit));
    /* Send the character */
    SendDataR = o;
}

//UARTRe1:串行收一个字节数据---------------------------------------*
//功能: 从串口接收1个字节的数据                                   *
//参数: 标志指针p                                               *
//返回: 接收到的数据(若接收失败，返回0xff)                        *
//说明: 参数*p 带回接收标志. *p = 0, 收到数据; *p = 1, 未收到数据 *
//-------------------------------------------------------------*
INT8U UARTRe1(INT8U *p)
{   INT16U k;INT8U i;
    // Wait some time until character has been received
    for(k=0; k < 0xfbbb; k++)
    if (ReSendStatusR & ReTestBit)
    {   i = ReDataR;
        *p = 0x00;
        break;
    }
    // receive fault
    if (k >= 0xfbbb)
    {   i = 0xff;
        *p = 0x01;
    }
    return i;}
```

4.4.2 RSP 协议的实现

RSP 通信模块按照“$数据#校验码”的格式，实现了对于接收数据的解包和发送数据的封包。其接收流程如图 4-4 所示：程序首先等待串口接收到起始标志“$”；随后将读取到的数据逐个写入缓冲区中，直到遇到结束字符“#”；最后判断该报文校验码是否正确，如果正确向 GDB 回送字符“+”，并将缓冲区数据作为调试命令处理，否则发送字符“-”要求重传报文。发送流程如图 4-3 所示：程序首先发送报文起始标志“$”；随后按字节逐个发送缓冲区内数据；在发送完结束标志“#”与报文校验码后，等待接收字符“+”，否则重传该报文。

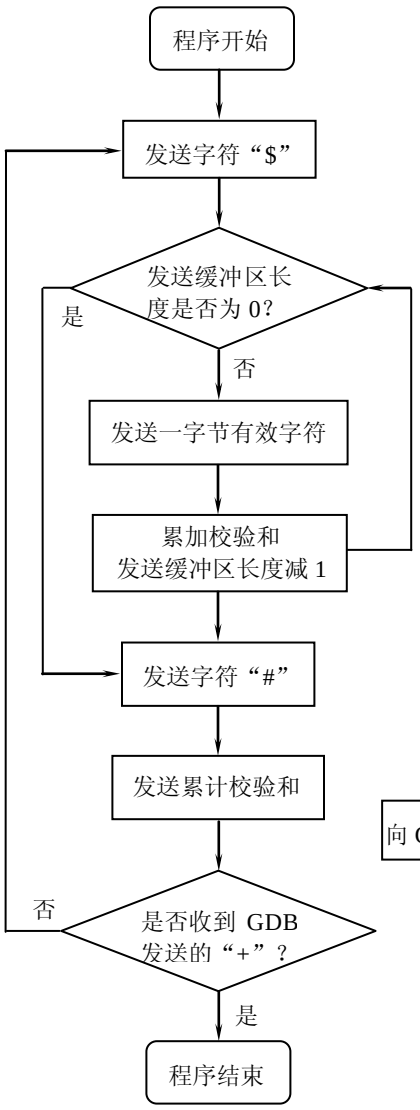


图 4-3 通信模块发送流程图

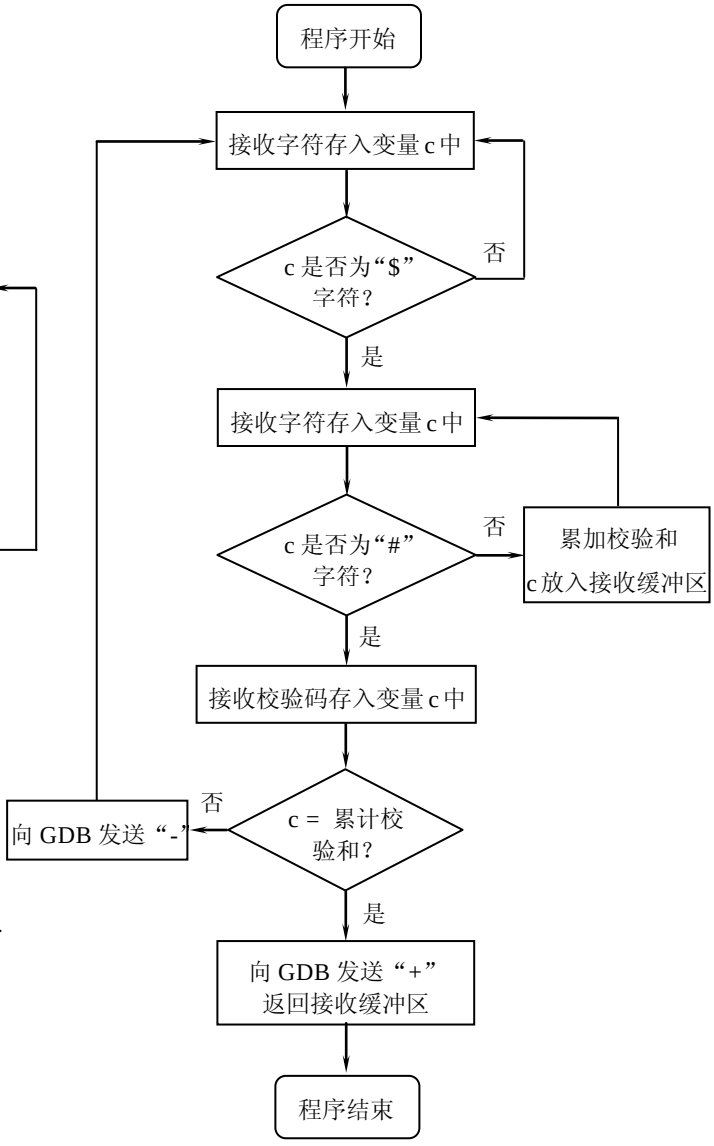


图 4-4 通信模块接收流程图

4.5 中断模块软件设计

中断模块可分为中断向量表与中断服务程序两部分，是 GDBStub for ColdFire 调度执行的核心，主要负责处理运行过程中的异常，包括调用通信模块与 GDB 交互信息，调用命令处理模块执行各种调试指令。本节将从中断向量表的设置、中断模块相关数据结构描述以及中断服务程序的编写三方面阐述中断模块的设计实现。

4.5.1 填写中断向量表

GDBStub for ColdFire 使用到的中断包括单步中断(trace)、陷阱中断 0(trap0)、陷阱中断 15(trap15)与串口接收中断(uart0)。单步中断用于实现调试过程中的单步步入功能，即 CPU 每执行完当前指令后触发中断，停止调试程序的运行；陷阱中断 0 使 GDBStub for ColdFire 在完成启动模块初始化后立刻进入中断模块，等待接收 GDB 发送调试命令；陷阱中断 15 配合替换了断点处代码的陷阱指令实现了被调试程序在设定断点处暂停；串口接收中断对于 GDBStub for ColdFire 而言并不是必须的，但利用此中断，调试者可以通过串行通信随时暂停被调试程序的运行，以防止程序进入不可预知的死循环。虽然触发 GDBStub for ColdFire 进入中断模块的条件即中断类型各不相同，但它们对应的中断服务程序却只有一个——gdb_interrupt_handler。该中断服务程序在判断当前的中断类型后，实现了读写寄存器值、保存中断时的系统信息、调用通信模块与命令处理模块等中断模块功能。下文给出了 MCF52233 中断向量表设置：

```
VECTOR_TABLE:                                    /* 中断向量表 */
INITSP:    .long  ___SP_INIT            /* 初始化时的堆栈指针SP值 */
INITPC:    .long  start               /* 初始化时的程序计数器PC值 */
vector02:  .long  _asm_exception_handler     /* 默认中断服务程序 */
vector09:  .long  _gdb_interrupt_handler        /* 单步中断trace */

vector19:  .long  _irq_handler            /* 默认IRQ中断服务程序 */
vector20:  .long  _gdb_interrupt_handler        /* 陷阱中断trap0 */

vector2F:  .long  _gdb_interrupt_handler       /* 陷阱中断trap15 */

vector4D:  .long  _gdb_interrupt_handler   /* 串口0接收中断uart0 */
```

4.5.2 数据结构

为了实现调试桩 GDBStub for ColdFire 访问 MCF52233 微处理器内部的编程模式寄存器，本文在中断模块中定义了静态内存数组 gdb_register_file，实现了寄存器值与

RAM 区域之间的映射。下文给出了 gdb_register_file 结构的相关定义：其中 A7 既可以作为通用地址寄存器，又可以当作堆栈指针使用；而 FV 并不是 MCF52233 CPU 内部真实存在的寄存器，其中存放的中断信息是中断模块判断当前中断类型的依据，包括 4 位中断格式与 12 位在中断向量表中的偏移地址。

```
/* 枚举当前的编程模式寄存器 */
enum
{
  D0, D1, D2, D3, D4, D5, D6, D7,       /* 通用数据寄存器 */
  A0, A1, A2, A3, A4, A5, A6, A7,       /* 通用地址寄存器 */
  SP = A7,                              /* 硬件堆栈指针寄存器SP */
  PS,                                   /* 状态寄存器SR(低16位有效) */
  PC,                                   /* 程序计数器PC */
  FV,            /* 4位中断格式与12位中断向量表中的偏移量 */
};

/* 静态变量定义 */
static long gdb_register_file[FV - D0 + 1];
```

4.5.3 中断服务程序的实现

中断服务程序 gdb_interrupt_handler 完成了几乎所有的中断模块功能，主要分为以下四部分：实现寄存器到 RAM 区域的映射、根据中断类型修改系统寄存器并改写相应功能标志位、调用通信模块收发调试信息、调用命令处理模块执行调试命令。其详细执行如图 4-5 所示，步骤如下：

① 屏蔽所有中断，防止中断嵌套的发生。

② 利用 unlk 指令，取消通过 C 函数调用进入中断服务程序时压栈的参数，使堆栈指针指向压栈的系统信息。

③ 将通用寄存器数据按顺序批量导入静态内存数组 gdb_register_file。

④ 通过当前堆栈指针导入系统信息到 gdb_register_file 中，包括进入中断时的状态寄存器(Status Register，SR)、程序计数器 PC、中断类型信息 FV。

⑤ 根据中断类型 FV，修改状态寄存器 SR 与程序计数器 PC，并设置特殊功能标志(如置单步执行中断标志、清除串口中断标志)。

⑥ 向 GDB 发送当前 CPU 内部编程模式寄存器值。

⑦ 调用通信模块接收调试命令。

⑧ 调用命令处理模块执行调试命令。

⑨ 调用通信模块反馈命令执行结果。

⑩ 将当前 gdb_register_file 中的数据值写入寄存器。

⑪ 打开所有中断，确保下一中断的发生。

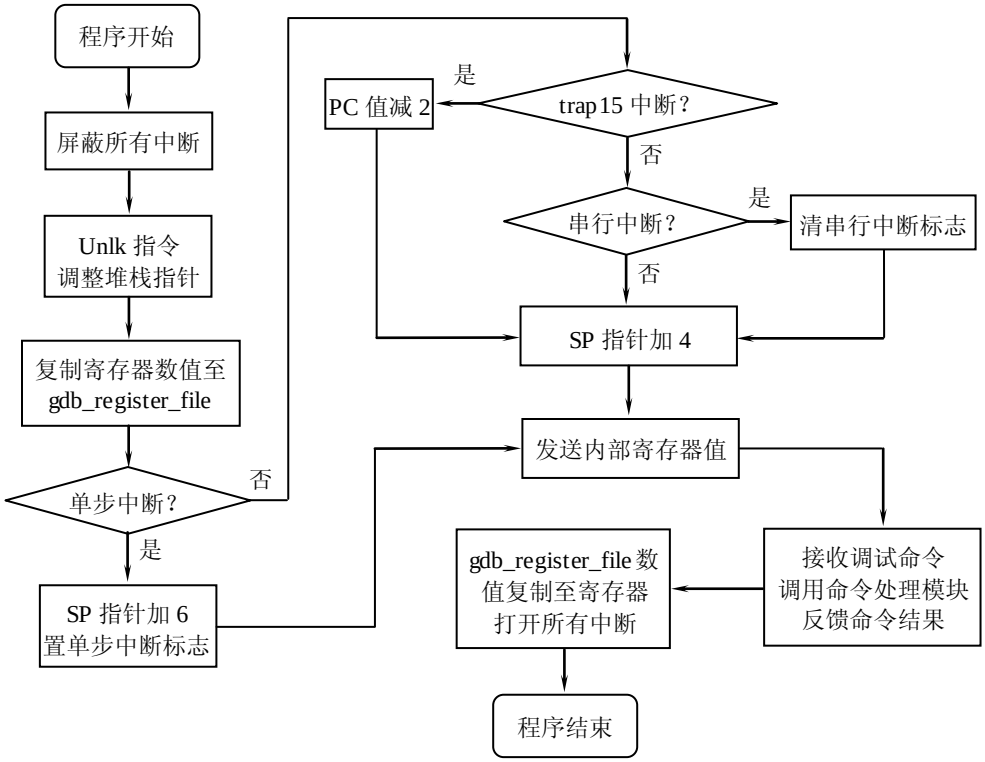


图 4-5 中断服务程序流程图

4.6 命令处理模块的实现

命令处理模块是 GDBStub for ColdFire 的真正执行单元，负责完成与调试命令相关的各种调试功能，并通过通信模块向调试器 GDB 报告产生的各种事件。该模块的主要功能是依据 GDBStub for ColdFire 通信模块解析到的 RSP 命令，执行对应的调试操作，包括读寄存器数据、写寄存器值、继续执行、单步执行、读内存数据、写内存数据、查询最近的信号、查询段地址信息、执行握手命令、结束调试等。表 4-1 列出了命令处理模块的主要功能函数，其中读写寄存器值函数 gdb_read_registers、gdb_write_registers 与 gdb_write_register 利用静态数组 gdb_register_file 实现与 RAM 区域交互数据；读写内存函数 gdb_write_memory 和 gdb_read_memory 通过动态调整数据格式实现对于 RAM 存储器的快速访问;继续执行或单步执行可以带有地址参数，其实现关键是对于程序计数器 PC 的修改；退出调试函数 gdb_kill 在发送标志“＋”后，将程序复位。

表 4-1 命令处理模块主要功能函数一览表

功能描述	函数名	对应调试命令	实现方法	反馈信息
发送最近信号	gdb_last_signal	?	利用内存变量记录最近的信号代码，封装成报文后发送，若没有收到“＋”，则重传数据报文。	最近的信号代码
继续执行	gdb_continue	c	首先判断调试命令后是否带有地址参数，若包含有效地址数据则将该值赋给程序计数器 PC 使程序从该地址处继续执行，否则从原有 PC 值处继续执行。	CPU 内部信息(各寄存器值与信号代码)
读寄存器的值	gdb_read_registers	g	从静态内存数组 gdb_register_file 中批量读取寄存器数据值，封装成报文后发送，若没有收到“＋”，则重传数据报文。	当前 CPU 内部寄存器值
写寄存器的值	gdb_write_registers	G	按顺序计算每一内部寄存器长度，并从收到的数据报文中读取指定长度字节依次写入数组 gdb_register_file 中。	OK
写单个寄存器值	gdb_write_register	P	从数据报文中解析单个寄存器数据写入数组 gdb_register_file 中。	OK
退出调试	gdb_kill	k	发送标志字符“＋”，等待一段时间后，程序跳转到调试桩起始地址执行。	+
写内存数据值	gdb_write_memory	M	读取写入起始地址和写入数据长度，根据长度字节数调整写入数据的格式。若长度字节数%4 为 0，按长整型格式写入数据；若长度字节数%2 等于 0，按整型格式写入数据；否则按字符型格式写入数据。	OK
读内存数据值	gdb_read_memory	m	实现方法与写内存数据 gdb_write_memory 函数类似。	内存数据
单步执行	gdb_step	s	设置 CPU 单步中断标志，调用继续执行函数 gdb_continue。	CPU 内部信息(各寄存器值与信号代码)

4.7 GDBStub for ColdFire 调试实例

本小节给出了一个基于 MCF52233 平台的 GDBStub for ColdFire 调试实例，在设计并编写了一个简单调试样例的基础上，利用便于机器提取信息的 GDB/MI 接口，详细叙述了 GDBStub for ColdFire 进行实例调试的过程，并列出了本次调试过程中的 RSP 报文。

4.7.1 被调试样例程序设计

被调试样例程序运行于 MCF52233 硬件平台之上，用于模拟被调试程序在 MCF52233 目标机系统上的运行状况。为了保持被调试样例程序的通用性，本文设计的样例函数 StubTest 并不包含与硬件相关代码。

由于 GDBStub for ColdFire 已占用了一部分 RAM 空间，而调试目标代码又必须

下载到内存运行，所以在利用 GCC 工具链编译被调试样例程序时，将其目标代码连接到 0x20000800 处(RAM 起始地址为 0x20000000，而 GDBStub for ColdFire 占用的内存空间小于 2KB，故被调试样例程序的起始地址可以为 0x20000800)。

下面给出了被调试样例程序的源代码以及编译后对应的 S19 格式目标代码：

```
void StubTest(void)
{
  int i, j, k;
  while (1)
  { k = 0;
    for (i = 0; i < 10; i++)
      for (j = 0; j < 10; j++)
      { if((k%8)==0)
     {j++;i++;}
        k++;}}}
```

```
S01800002E2F4F424A2F4744425374756254657374 2E7331396A
S315200008004E56FFF442AEFFFC42AEFFF4603842AED5
S31520000810FFF86024202EFFFC7207C0814A80660CF8
S315200008207001D1AEFFF87201D3AEFFF47001D1AEE4
S31520000830FFFC7201D3AEFFF87009B0AEFFF86CD49E
S315200008407201D3AEFFF47009B0AEFFF46CC060B491
S315200008504E710000000000000000000000000000B3
S70520000800D2
```

4.7.2 GDBStub for ColdFire 调试实例分析

表 4-2 给出了使用 GDB/MI 接口调试样例程序 StubTest 的具体步骤以及调试过程中的 GDBStub for ColdFire 的输入输出报文。GDB 首先打开 GDB/MI 接口，读取.elf 文件并加载被调试样例程序对应的详细调试信息，具体命令为“m68k-elf-gdb －interpreter mi －f StubTest.elf”；其次 GDB 使用设置的串行通信方式远程连接目标机端 GDBStub for ColdFire；随后写入被调试样例程序目标代码，利用设置断点、继续执行与单步执行功能完成对被调试样例程序的运行控制；最后通过-var 命令打印出局部变量值，使用-data 显示寄存器信息。

4.8 本章小结

本章工作总结如下：

① 分析了调试桩 GDBStub 的一般结构和调试原理，为 GDBStub for ColdFire 的设计实现提供了借鉴与参考。

② 按照 GDBStub for ColdFire 的结构模块，详细给出了该调试桩的实现思路与方法，内容涉及启动模块、RSP 通信模块、中断模块和命令处理模块。

③ 设计并编写了基于 MCF52233 平台的调试实例程序，并针对通信过程中的 RSP 报文信息，分析了该实例基于 GDBStub for ColdFire 的调试过程。结果表明：GDBStub for ColdFire 能够作为 MCF52233 等 ColdFire 微处理器的调试代理，配合宿

主机端 GDB，实现对该系列 CPU 程序的控制与调试。

表 4-2 GDBStub for ColdFire 调试实例流程表

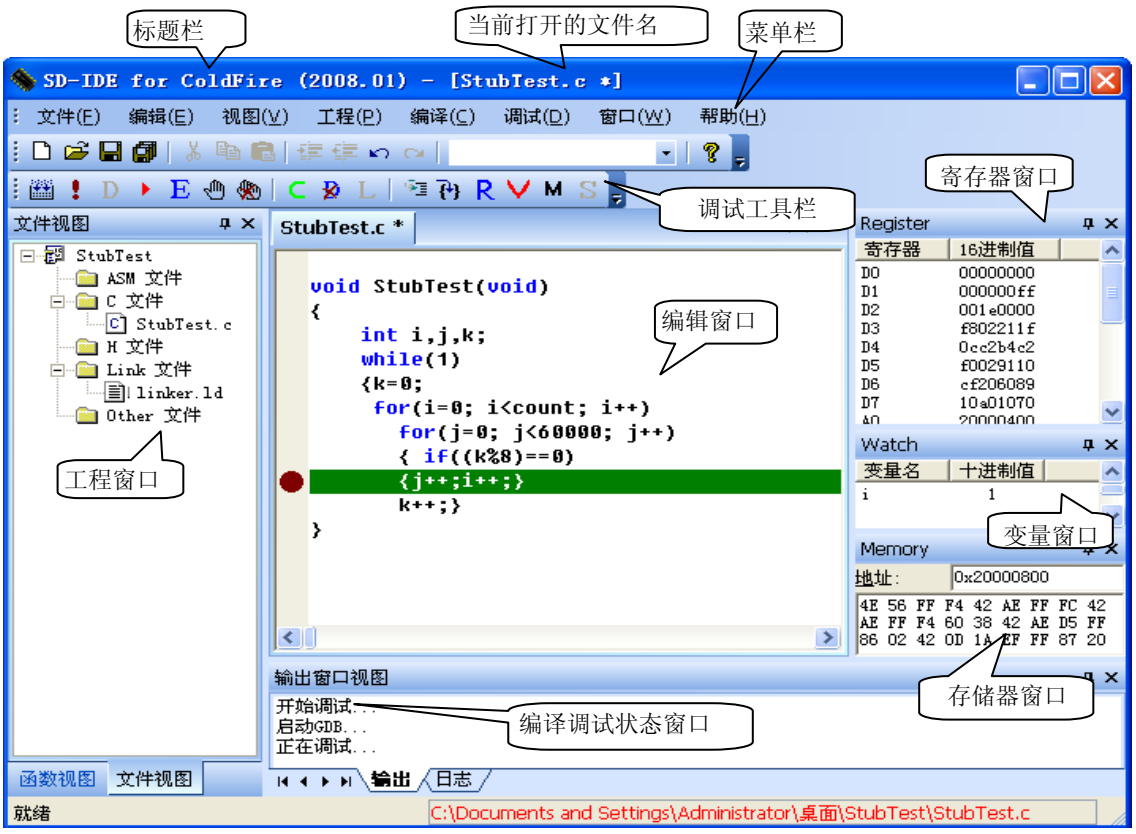
通过 GDB/MI 接口输入的调试命令	命令解释	GDBStub for ColdFire 收到的 RSP 协议报文	GDBStub for ColdFire 发送的 RSP 协议报文
-target-select remote COM1	通过宿主机 COM1 口远程连接目标机	$qPacketInfo#55 $Hc-1#09 $qC#b4 $qOffsets#4b $?#3f $Hg0#df $g#67 $qSymbol::#5b	$Text=0;Data=0;Bss=0#04 $OK#9a $Text=0;Data=0;Bss=0#04 $Text=0;Data=0;Bss=0#04 $S05#b8 $OK#9a $00000000000000ff001e0000f8022 11f0cc2b4c2f0029110cf20608910a 0107020000400000000c7a5481842 0d1f48edcfa01490426fe1120007ff 820007ff80000271400000416#97 $Text=0;Data=0;Bss=0#04
-target-download	加载被调试目标代码到目标机系统 RAM 区域并将程序计数器 PC 指向该段代码的起始地址	$M20000800,3f:4e56fff44 2aefffc42aefff4603842ae fff86024202efffc7207c08 14a80660c7001d1aefff872 01d3aefff47001d1aefffc7 201d3aefff87009b0aefff8 6c#aa $M2000083f,21:d47201d3a efff47009b0aefff46cc060 b44e7100000000000000000 00000000000#f5 $P11=20000800#79	$OK#9a $OK#9a $OK#9a
-break-insert 10	在被调试样例程序的第 10 行设置断点	无	无
-exec-continue	继续执行到断点处	$m20000814,2#5a $M20000814,2:4e4f#a7 $Hc0#db $c#63 $P11=20000814#7e $M20000814,2:202e#6d	$202e#f9 $OK#9a $OK#9a $T050:00000009;1:000000ff;2:00 1e0000;3:f802211f;4:0cc2b4c2;5 :f0029110;6:cf206089;7:10a0107 0;8:20000400;9:0000004c;a:7a54 8184;b:20d1f48e;c:dcfa0149;d:0 426fe11;e:20007ff4;f:20007fe8; 10:00002710;11:20000816;#b5 $OK#9a $OK#9a
-exec-step	单步执行	$Hc0#db $$s#73 $m20000814,2#5a M20000814,2:4e4f#a7 $Hc0#db $s#73 $M20000814,2:202e#6d	$OK#9a T050:00000000;1:000000ff;2:001 e0000;3:f802211f;4:0cc2b4c2;5: f0029110;6:cf206089;7:10a01070 ;8:20000400;9:0000004c;a:7a548 184;b:20d1f48e;c:dcfa0149;d:04 26fe11;e:20007ff4;f:20007fec;1 0:00002714;11:20000818;#dd $202e#f9 $OK#9a $OK#9a $T050:00000000;1:00000007;2:00 1e0000;3:f802211f;4:0cc2b4c2;5 :f0029110;6:cf206089;7:10a0107 0;8:20000400;9:0000004c;a:7a54 8184;b:20d1f48e;c:dcfa0149;d:0 426fe11;e:20007ff4;f:20007fec; 10:00002710;11:2000081a;#9d $OK#9a
-var-evaluate-expression i	打印局部变量 i 的值	$m20007fe8,4#c9	$00000001#81
-gdb-exit	退出调试	$k#6b	+

第五章 宿主机调试平台的实现

前面章节详细阐述了硬件调试平台的设计与 GDBStub for ColdFire 的实现。对于一套成熟的嵌入式调试系统而言，只有调试硬件和目标机代理是远远不够的，还需要与宿主机端调试软件相配合，才能确保整个调试系统便捷、高效、可靠的运行。本文在苏州大学自主研发的 SD-IDE for HCS12 集成开发环境的基础上，二次开发了针对 ColdFire 微处理器的 SD-IDE for ColdFire，作为宿主机端的集成调试环境。本章首先简单介绍了 SD-IDE for ColdFire，在添加 MCF52233 工程模板的基础上，重点阐述了与 MCF52233 相关功能模块的实现，包括交叉编译的完成，代码写入模块的设计，GDB 的重定向与代码调试模块的实现。

5.1 SD-IDE for ColdFire 概述

SD-IDE 是苏州大学 Freescale 实验室自主研发的针对 Freescale 公司微处理器产品的集成开发环境系列。它是一个集代码编辑、编译、下载、工程管理功能于一身的



MDI(Multiple Document Interface，多文档界面)多窗口应用程序，采用了统一的主界面风格，包含菜单、工具栏、状态栏、文件管理窗口、信息输出窗口等人机交互接口，极大方便了用户的使用。

到目前为止，苏州大学 Freescale 实验室已陆续开发了 SD-IDE for HC08、SD-IDE for HCS08、SD-IDE for HCS12、SD-IDE for M*Core 与 SD-IDE for ColdFire 等 SD-IDE 系列的集成开发环境。其中 SD-IDE for ColdFire 主要支持 MCF52233、MCF5271 等 ColdFire 架构的微处理器，其运行界面如图 5-1 所示。

由于 SD-IDE 系列集成开发环境的设计充分考虑到软件平台的芯片无关性，将与芯片无关的通用部分独立出来，并为芯片相关的内容留出了统一的编程接口，因此本文在开发 SD-IDE for ColdFire 时，无需对软件进行整体重新编码，而只需要修改若干配置文件以及编写独立的接口部分软件，就可以完成对新增 CPU 架构 ColdFire 微处理器的支持。图 5-2 给出了 SD-IDE for ColdFire 的模块结构图。其中工程管理、文件

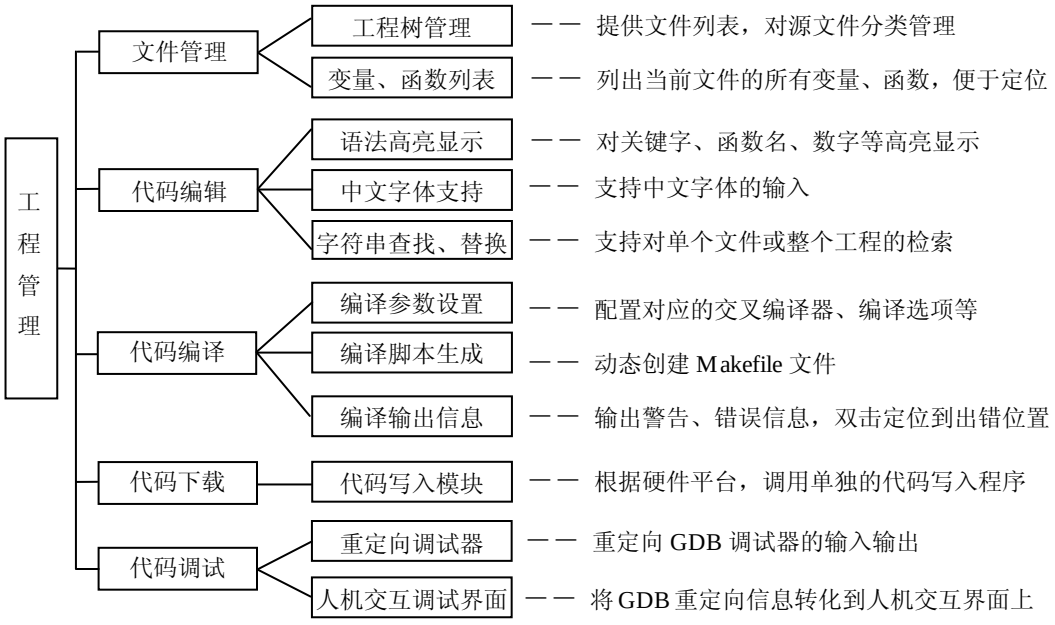
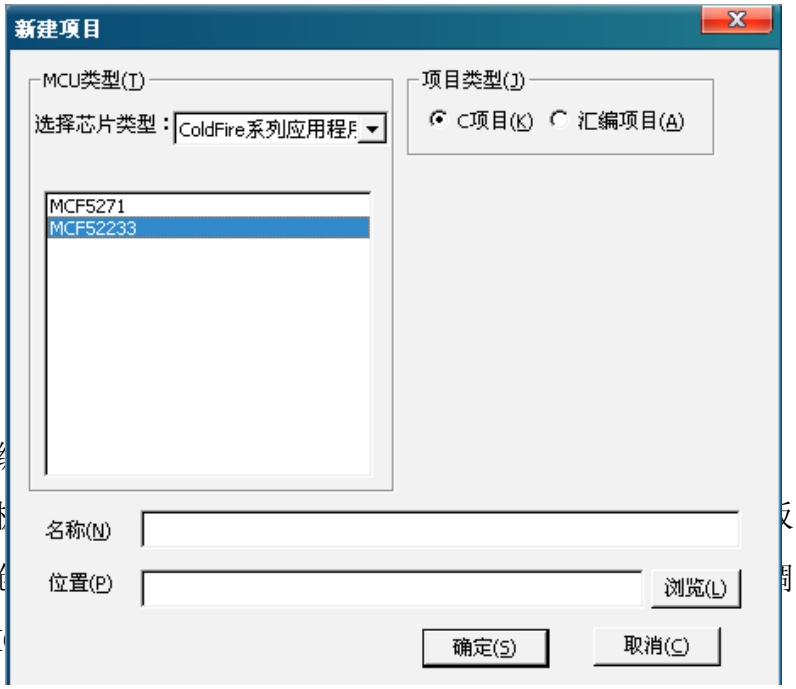


图 5-2 SD-IDE for ColdFire 模块结构图

管理、代码编辑属于 SD-IDE 的通用模块，其功能甚至代码对于任何 SD-IDE 系列集成开发环境都适用；而代码编译与代码下载模块则需要针对 ColdFire 微处理器进行改写。由于本文实现的 ColdFire 调试系统利用了基于 GDBStub for ColdFire 的 GDB 运程调试，而在此之前的 SD-IDE 并未涉及 GDB 相关内容，因此 SD-IDE for ColdFire 的代码调试模块需要本文设计与实现。

5.2 添加 MCF52233 工程模板

SD-IDE 的通用模块大多类似，其主要差别在于对不同型号的 CPU 需要配置不同的工程模板。工程模板为编程用户提供了每一系列微处理器的通用框架文件，包括最基本的微处理器初始化代码和中断向量表等。用户可以方便的从工程模板中创建一个针对该型号微处理器的新工程，并自动生成与模板中完全一样的初始化代码、中断向量表等通用框架文件，类似 Visual Studio 中的工程向导。图 5-3 是 SD-IDE for ColdFire 的新建工程对话框，列表控件中所列出的是与 SD-IDE for ColdFire 工程模板相对应的 CPU 类型。工程模板存放于 SD-IDE 安装目录下的 Stationary 文件夹中，而工程模板向导根据该文件夹下的子文件夹名来判断模板名。例如，MCF52233 CPU 的模板名为 MCF52233，其通用框架文件存放于安装目录\Stationary\MCF52233 子文件夹下，在



添加一个 CPU 工程模板文件、修改系统配置文件、创建调试平台，详细阐述了添加 MCF52233 工程模板的过程。

5.2.1 编写芯片配置文件

SD-IDE for ColdFire 工程模板中的每一款微处理器都有一个相应的芯片配置文件 Template.config。在新建工程时，用户须先在新

表 5-1 区块的定义

区　块	含　义
<IDEMCF>	包括所有的工程基本信息区
<Flags>	参数标志信息区
<Files>	工程文件列表区

建工程对话框中选定微处理器平台，然后系统会根据该微处理器型号打开相应工程模板中的配置文件 Template.config，复制其中内容生成扩展名为.IDESln 的工程文件。

表 5-2 字段的定义

字　　段	属于区块	含　　义
<Version>…</Version>	<IDEMCF>	版本号
<GCCType>…</GCCType>		编译器类型
<CPUName>…</CPUName>		芯片名称
<ProType>…</ProType>		工程类型
<CFlag>""</CFlag>	<Flags>	C 编译器参数
<ASFlag>""</ASFlag>		汇编编译器参数
<LDFlag>""</LDFlag>		连接器参数
<ASMFile>…</ASMFile>	<Files>	汇编文件列表起止标志
<CFile>…</CFile>		C 文件列表起止标志
<HFile>…</HFile>		头文件列表起止标志
<LinkFile>…</LinkFile>		连接文件列表起止标志
<OtherFile>…</OtherFile>		其他文件列表起止标志

Template.config 文件包含了芯片的基本信息(如编译器名、微处理器名称及 C 或 ASM 工程类型)、环境变量(如 Cflag、ASFlag 和 LDFlag)、源文件列表。配置文件的内容可分为若干个区块，每个区块中有若干个字段，这些字段存储了工程的信息，表 5-1 给出了配置文件的区块定义，表 5-2 列出了所有区块中的字段及其含义。

以下为 MCF52233 芯片的配置文件内容示意。

```
<IDEMCF>
  <Version>"1.0.0"</Version>  <McuType>"m68k"</McuType>
  <McuName>"MCF52233"</McuName>  <ProType>"C"</ProType>
  <Flags>  <CFlag>""</CFlag>  <ASFlag>""</ASFlag>  <LDFlag>""</LDFlag>  </Flags>
  <Files>
    <ASMFile>  <Value>"setup.s"</Value>   <Value>"vectors.s"</Value>  </ASMFile>
    <CFile>
      <Value>"isr.c"</Value>  <Value>"LED.c"</Value>
      <Value>"sysinit.c"</Value>  <Value>"main.c"</Value>
    </CFile>
    <HFile>
      <Value>"isr.h"</Value>  <Value>"LED.h"</Value>  <Value>"config.h"</Value>
      <Value>"Type.h"</Value>  <Value>"mcf52233.h"</Value>
    </HFile>
    <LinkFile>  <Value>"linker.ld"</Value>   </LinkFile>
    <OtherFile>  </OtherFile>
  </Files>
</IDEMCF>
```

5.2.2 编写芯片模板文件

每款芯片的模板文件主要是与该微处理器体系结构有关的通用框架文件，主要包括芯片的启动文件(setup.s)、模块初始化文件(sysinit.c)、头文件(mcf52233.h)、连接文件(linker.ld)、中断向量表文件(vectors.s)、中断处理函数框架文件(isr.c)和主程序框架文件(main.c)。这些文件功能和代码在此就不一一阐述了。

5.2.3 修改系统配置文件

在 SD-IDE for ColdFire 软件安装目录下的\Bin 文件夹中有一个系统配置文件 option.xml，该文件存储了 SD-IDE for ColdFire 所支持的系列芯片型号、相应的库和包含文件。在加入 m68k 编译器支持的 MCF52233 芯片时，需要在该文件的相应区中加上 MCF52233 相关内容，以便软件在运行时，从中读取信息，对界面进行初始化操作。option.xml 文件的主要内容示意如下：

```
<IDEMCF>
<Version>"1.0.0"</Version>
<GlobleOption>
…
<m68k>
<Stationary> <Value>"MCF5271"</Value> <Value>"MCF52233"</Value> </Stationary>
<IncPath>
<Value>"d:\MYPROJ~1\IDE\code\200611~3\m68k\m68k-elf\include"</Value>
<Value>"D:\MYPROJ~1\IDE\code\200611~3\m68k\m68k-elf\include"</Value>
</IncPath>
<LibPath>
<Value>"d:\MYPROJ~1\IDE\code\200611~3\m68k\m68k-elf\lib"</Value>
<Value>"D:\MYPROJ~1\IDE\code\200611~3\m68k\m68k-elf\lib"</Value>
</LibPath>
</m68k>
…
</GlobleOption>
</IDEMCF>
```

5.2.4 创建芯片相关子文件夹

在安装目录\Stationary 文件夹下创建一个以 MCF52233 命名的子文件夹，随后将上述编写的文件全部放入 MCF52233 子文件夹中。当用户再次打开新建工程对话框，并选择芯片类型为 ColdFire 系列时，就会发现 ColdFire 系列芯片中新增了一个 MCF52233 选项，如图 5-3 所示。在设定工程文件名、工程路径和工程类型后，单击“确定”，一个基于 MCF52233 芯片的新工程将出现在指定的路径下。工程文件夹中包含前面创建的通用框架文件，用户可在此基础上继续对该工程代码进一步编辑。

5.3 交叉编译的实现

不同系列的芯片在编译时使用的交叉编译器也是不相同的。如果要构建基于 ColdFire 的集成调试平台，首先必须要有 ColdFire 的交叉编译器。本节首先介绍了交叉编译器的基本知识，随后给出了构建交叉编译器的方法，最后讨论了交叉编译过程

中必不可少的 Makefile 文件和连接脚本。

5.3.1 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，生成可以在目标机芯片执行的可执行代码。目前，GCC 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微处理器，主要包括 ARM、MIPS、PowerPC、ColdFire/68K、M*Core 以及 x86 等[31][32]。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：GNU GCC C/C++编译器、汇编器、连接器、嵌入式系统的标准 C 库、GDB 调试器[33]。

5.3.2 构建 GCC 工具链

ColdFire 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程序时就会编译不过。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 微处理器上运行的代码[34]。

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(Dynamic Link Library，动态链接库)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[35]。值得注意的是使用 Cygwin 生成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。下文将简单介绍构建 GCC 工具链的准备工作及相关步骤。

1．Cygwin 配置

Cygwin 目前由 Redhat 负责维护，可以到其网站上下载。由于本文只是使用 Cygwin

编译交叉编译器，所以只需要安装 Cygwin 的最小组件：

① GCC——用来编译交叉编译器，主要是用到了 GCC-CORE。

② bintuils——GCC 用到的工具集，当选择 GCC 时，这一项会自动选中。

③ make——工程编译管理器。

④ perl——一种脚本语言。

⑤ flex——用于模式识别(Used for pattern recognition)。

⑥ 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，还能够使用其它 C 程序库。libc 可以有多种选择，在此选用 newlib。

3．GCC 交叉编译器的构建

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++编译器。

制作交叉编译器的步骤为：编译 binutils；生成只能编译 C 语言的最简 GCC；用刚才编译好的 GCC 和 binutils 编译 newlib；利用生成的 newlib 再次编译 GCC，生成功能完整的 GCC。其每一步骤中的编译细节可参考附录 C.1。

5.3.3 Makefile 文件

make 命令执行时，需要一个 Makefile 文件，以告诉 make 命令如何去编译和连接程序。Makefile 带来的好处就是“自动化编译”，一旦写好，只需一个 make 命令，整个工程就可以自动编译。编译时编译器会根据文件的依赖性规则决定是否重新编译某些源程序，从而极大的提高代码编译效率[36]。下面将分别介绍 Makefile 规则与文件依赖性关系。

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。如下所示：

```
main.o: main.c mcf52233.h uart.h io.h
m68k-elf-gcc –c main.c –o main.o
```

上面的 main.o 依赖于 main.c，mcf52233.h，uart.h 和 io.h 这 4 个文件。当这 4 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。

在交叉编译的过程中，要生成某个.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后，重新编译，能够得到更新后的目标代码。GCC 通过-M 选项，为每个 C 文件输出一个符合 make 语法的规则。这个规则将输出该 C 文件依赖的所有头文件，包括被角括号(<>)和双引号(“”)所包含的文件。如果确定系统头文件不会被更改，可以用-MM 来代替-M 传递给 GCC 编译器，角括号包含的系统头文件将不会被输出。未生成各文件依赖关系前，Makefile 文件的内容示意如下：

```
#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 后，将生成各文件之间的依赖关系。方法如下：首先使用字符串编辑程序 sed 对 Makefile 文件进行处理，输出 Makefile 文件中到“### Dependencies”行为止的所有内容，并生成 tmp_make 临时文件；随后通过-MM 标志对当前目录下的所有 C 文件执行 GCC 预处理操作，为每个源文件输出一个 make 规则，并将预处理结果都添加到临时文件 tmp_make 中，其结果形式为相应源程序文件的目标文件名加上其依赖关系即该 C 文件中包含的所有头文件列表；最后将该 tmp_make 复制成新的 Makefile 文件，并将其删除[37]。执行后，Makefile 文件在最后增加了 io.o 和 main.o 的依赖性规则，如下所示：

```
io.o: io.c mcf52233.h
main.o: main.c mcf52233.h uart.h io.h
```

5.3.4 连接脚本

连接过程由使用连接器命令语言组成的连接脚本控制。如果没有提供指定的脚本文件，连接器则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[38]。连接脚本使用的最重要的命令是“SECTIONS”，该命令用于描述输出文件的内存布局。

在 SD-IDE for ColdFire 集成开发环境中，连接器只需完成一些基本功能，所以连接脚本并不复杂，下文描述了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 系列微处理器上电复位后，首先会从 Flash 的起始位置读取 4 字节数据放入 SP 堆栈寄存器，接着再读取 4 字节的数据放入程序计数器 PC 中，然后执行 PC 寄存器所指向的指令。

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 空间。初始化代码中使用的地址变量值在连接时由连接器分配。

③ 为一些用户可配置寄存器提供参数。用户可以通过片内外设基址寄存器

IPSBAR 决定 ColdFire 系列微处理器的片内外设起始地址，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即重新编译源代码，只要重新连接一次，就可生成不同功能的代码。

SECTIONS 告诉编译器怎样将输入段映射到输出段，并指定输出段的存储位置[39]。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 命令来告诉连接器可用的存储区域。附录 C.2 给出针对 MCF52233 微处理器的连接文件，连接器将生成适合在 Flash 中执行的代码。其中的 data 数据段既占据 RAM 空间，也占据 Flash 空间。

5.4 基于 BDM 的代码写入程序设计

程序编译完成之后，要将生成的目标代码下载到芯片内运行。不同的芯片由于其内部寄存器以及存储器类型不同，代码下载的方式有所不同；甚至采用同种芯片的电路板因为存储扩展的不同导致下载方式差异很大。考虑到 SD-IDE 的平台无关性及可扩展性，有必要为该模块单独编写子程序，来实现代码的下载写入功能。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，通过 SD-IDE 提供的接口进行调用。同时，代码写入程序也可以单独发布供用户使用。

SD-IDE for ColdFire 通过支持 ColdFire 微处理器的 BDM 头，实现了对 MCF52233 内部 Flash 存储器的烧写操作，其支持的目标代码文件的格式为 S-Records[40]或 bin，详细实现过程可分为以下几个步骤：

① SD-IDE for ColdFire 通过 BDM 头与 MCF52233 建立连接，并使其进入 BDM 模式，在该模式下 SD-IDE for ColdFire 可读写内存数据和寄存器值。

② SD-IDE for ColdFire 编译 MCF52233 内部 Flash 擦除和写入程序，将生成的

S19 格式目标代码文件存放于安装目录下，供代码写入模块调用。

③ 由于宿主机端代码写入程序无法直接通过 BDM 头把用户程序下载到内部 Flash 中，所以只能通过把 MCF52233 的擦除程序、写入程序及用户程序目标代码全部下载到 RAM 区域，在 RAM 区域运行 MCF52233 擦写程序，从而把用户程序目标代码逐页写入到指定的 Flash 空间中。

本节从 BDM 通信接口程序设计、宿主机端写入程序设计和 MCF52233 内部 Flash 擦写程序设计三方面，阐述了基于 BDM 的代码写入模块的设计与实现。

5.4.1 BDM 通信接口程序设计

表 5-3 P&E BDM 动态链接库主要函数列表

函数名	参数	返回值	功能描述
bool load_dll(void)	无	加载成功返回 True 否则为 False	将动态链接库加载到宿主机内存，允许调用库函数
void init_port(unsigned short lpt_port)	lpt_port：并口号	无	初始化 BDM 并口
void set_io_delay_cnt_value(unsigned short new_io_delay_cnt)	new_io_delay_cnt：速度参数	无	设定 BDM 通信速度
bool go_to_background(void)	无	进入 BDM 模式返回 True，否则为 False	使目标芯片进入 BDM 模式
unsigned int get_control(unsigned int address)	address：读取地址	16 位寄存器值	读取控制寄存器值
void put_control(unsigned int address, unsigned int data)	address：写入地址 data：写入数据	无	向控制寄存器写值
unsigned int get_pc_value(void)	无	当前 PC 值	读取程序计数器 PC 值
void put_pc_value(unsigned int newpc)	Newpc：新 PC 值	无	修改程序计数器 PC 值
void write_data_byte(unsigned int address, unsigned short data)	address：写入地址 data：写入数据	无	向目标芯片 RAM 区域地址处写入一个字节
unsigned char read_data_byte(unsigned int address)	address：读取地址	一字节值	从目标芯片 RAM 区域地址处读取一个字节
void write_data_long(unsigned int address, unsigned int data)	address：写入地址 data：写入数据	无	向目标芯片 RAM 区域地址处写入一个字
unsigned int read_data_long(unsigned int address)	address：读取地址	两字节值	从目标芯片 RAM 区域地址处读取一个字
void write_data_word(unsigned int address, unsigned short data)	address：写入地址 data：写入数据	无	向目标芯片 RAM 区域地址处写入一个长字
unsigned short read_data_word(unsigned int address)	address：读取地址	四字节值	从目标芯片 RAM 区域地址处读取一个长字
bool load_srec_file(char *filename, unsigned int offset)	Filename：S 格式文件名 Offset：行偏移量	加载成功返回 True，否则为 False	加载 S 格式文件代码到目标芯片 RAM 区域
void resume(void)	无	无	从程序计数器 PC 处继续运行代码
void single_step(void)	无	无	单步运行

SD-IDE for ColdFire 利用并口 BDM 头与目标机 MCF52233 通信，其通信程序调用了 P&E 公司提供的 BDM 动态链接库 unit_cfz.dll。表 5-3 列出了 unit_cfz.dll 动态链接库 3.17.0.0 版本的主要函数。代码写入模块通过调用动态链接库函数与 MCF52233 通信，实现对目标芯片的操作和控制，其主要流程如下：首先加载动态链接库，使用户可以调用其中的库函数；其次初始化连接 BDM 的并口，设定 BDM 通信速率，通过 BDM 头迫使芯片进入 BDM 模式；随后使用读写寄存器或 RAM 区域的函数实现对于 MCF52233 内部存储单元的读写操作；最后通过库中的执行控制函数使芯片退出 BDM 模式，并从程序计数器 PC 值处恢复运行。

5.4.2 宿主机写入程序设计

写入程序显示界面如图 5-4 所示。用户首先选择当前 CPU 类型为 MCF52233，并设定使用芯片内部 Flash 存储器即 Internal ROM；其次选定执行擦除写入操作的扇区，如果用户选中“全选”框后，目标扇区范围为整个 Flash 区域；在执行 Flash 写入操作前，必须先将待写入扇区的数据擦除，通过界面右上方的打开文件按钮选择写入的 S19 格式文件；校验按钮则用于验证选定的 Flash 扇区是否为空。以上操作的完成主要通过调用擦除程序、分析 S19 文件程序和写入程序，下面将详细阐述这些功能程序的流程和实现方式。

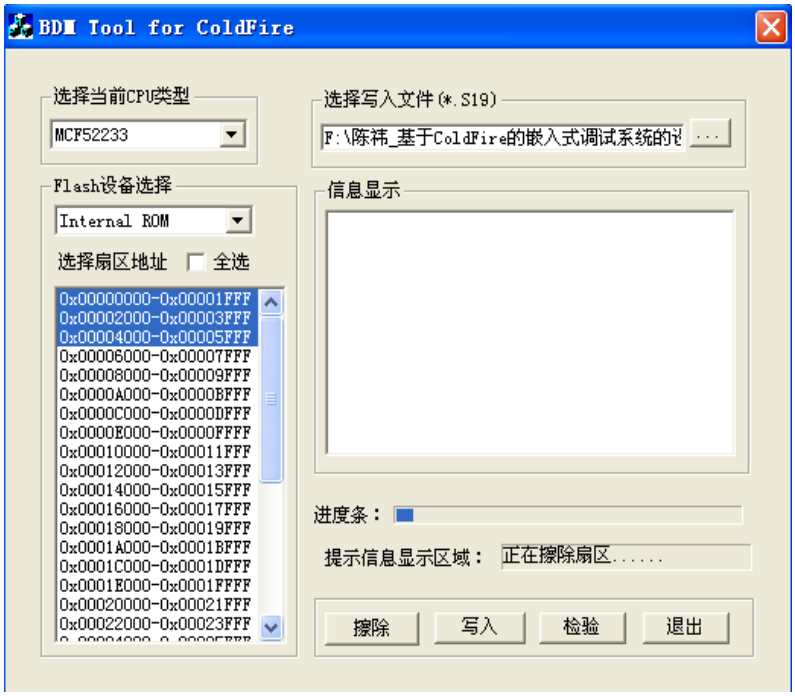


图 5-4 写入程序界面

(1) 擦除

擦除程序的流程如下所述：在取得擦除命令后，程序首先载入对应芯片的擦除程序代码并将其重新组装成若干个数据包；随后把这些数据包逐一写入到目标芯片的

RAM 中；最后通过调用动态链接库函数 put_pc_value 与 resume 调整程序计数器，使 CPU 转而运行 RAM 中的擦除程序代码，以实现擦除目标 Flash 扇区。

(2) 分析 S19 文件

该程序首先按行读入打开的 S19 文件，并将 S19 数据保存到字符串数组中，数组中每个元素存放 S19 的一行数据。每读入一行时，都要验证该行校验和是否正确。随后对数组中每个元素的数据进行重组，以一页 128 个字节为单位，并在 128 字节数据前附加 5 个字节数据。前 4 个字节用来存放一页用户数据写入 Flash 区域的起始地址，后 1 个字节用于记录一页用户数据长度。由这些数据组成的新字符串数组称为一页用户数据包，由此可见，一页用户数据包总长度为 133 字节。

(3) 写入

写入程序流程如图 5-5 所示。在执行写入命令时，程序首先获取组装好的一页用户数据包，将数据写入到 RAM 中的相应区域，然后，判断芯片写入程序代码是否已经存放到 RAM 中，若没有则写入对应芯片的写入程序代码。通过调用动态链接库函数 put_pc_value 设置当前的 PC 指针，指向存放在 RAM 区的芯片写入程序代码的起始地址，再调用 resume 函数来执行该芯片写入程序代码。该代码的功能就是将存放在 RAM 区的一页用户数据写入到指定的 Flash 区域中。每写入一页用户数据后，都需要读取写入成功与否的标志，该标志存放在 RAM 中的一个固定的地址处，占两个字节。程序根据读取的标志值，判断该页用户数据在写入过程中是否出现错误，若出现错误，则需要重新对该页数据进行写入；若未出现写入错误，

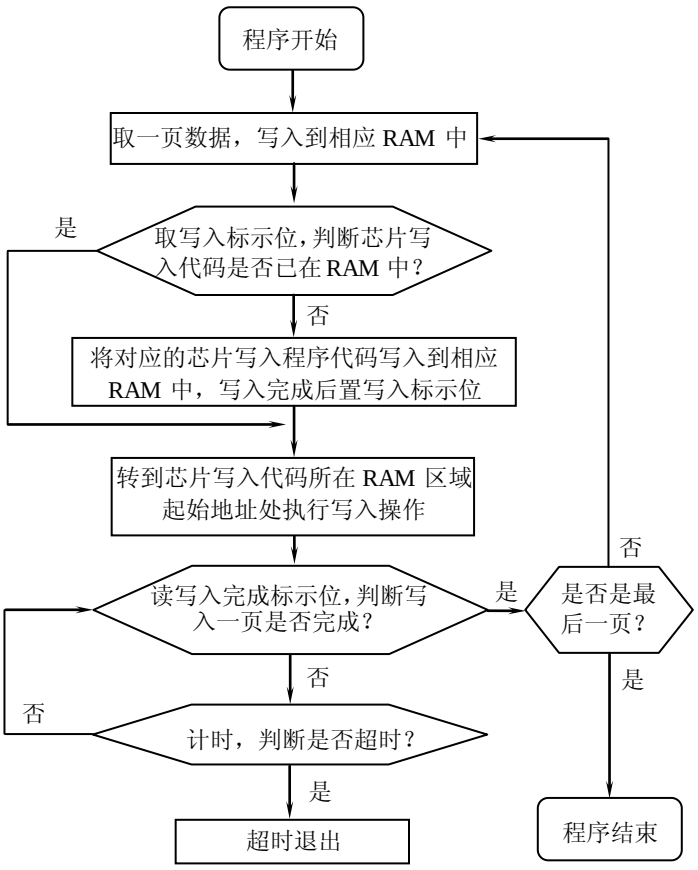


图 5-5 写入程序流程图

则可以接着进行下一页用户数据的写入操作，直到所有数据写入完毕为止。

5.4.3 内部 Flash 擦写程序设计

芯片内部 Flash 的擦除操作与写入操作过程相似，其执行步骤如下：

① 检查上一次 Flash 处理命令是否执行完毕。判断 Flash 状态寄存器(CFM User Status Register，CFMUSTAT)中的命令缓冲区标志位 CBEIF 是否为“1”，若为“1”，则命令缓冲区可以使用，否则等待到其可以使用为止。

② 清 Flash 状态寄存器 CFMUSTAT 中的标志位 ACCERR 和 PVIOL。目的是清除上次 Flash 操作的出错标志，方法是向 CFMUSTAT 寄存器的这两位写零。

③ 将要写入的数据长字写到目标地址中，目标地址必须为偶数地址，因为 Flash 存储器起始地址为偶数，且写操作是一次写一个长字即四个字节。

④ 向命令寄存器(CFM Command Register，CFMCMD)写命令字节。如 0x41 表示整体擦除，0x40 表示扇区擦除，0x20 表示写一个长字。

⑤ 向 Flash 状态寄存器 CFMUSTAT 中的命令缓冲区标志位 CBEIF 写 1 清零。如果此时状态寄存器中的 CCIF 位置 1，说明操作成功。

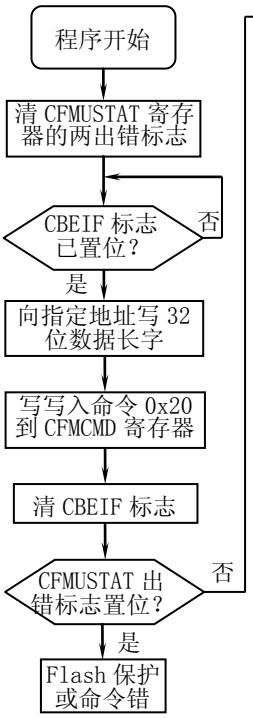


图 5-6 写入一个长字流程图

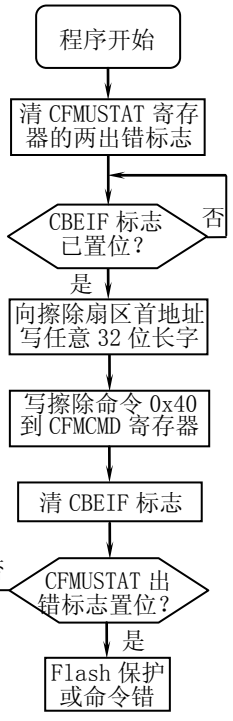


图 5-7 擦除一扇区流程图

MCF52233 内部 Flash 扇区擦除的操作流程如图 5-7 所示，而写入一个长字操作如图 5-6 所示。整体擦除操作流程与扇区擦除相同，修改命令字节为 0x41 即可。写入一页数据操作则需要在写入一个长字的基础上，重复执行③～⑤步。

5.5 调试器 GDB 的调度与重定向

Cygwin 将基于 ColdFire 架构的 GDB 移植到了 windows 平台，创建了在宿主机端可直接运行的.exe 文件。由于该.exe 程序运行过程中的输入输出都是基于 DOS 环境下的字符流，普通用户很难掌握，因此 SD-IDE for ColdFire 的代码调试模块在调度 GDB 调试时，需将其输入输出重定向，使用户通过人机交互调试界面与 GDB 通信。

SD-IDE for ColdFire 通过建立子进程的方式调用 GDB 调试器，并利用进程间的管道通信实现内存缓冲区与 GDB 输入输出间的映射。管道作为宿主机调试模块与 GDB 的通信介质，不但帮助 SD-IDE for ColdFire 向 GDB 输入用户调试命令，而且便于截获 GDB 的字符输出信息，转而显示在代码调试模块的窗口界面中。创建进程与管道的方法如下：

```
/* 创建输入、输出和错误管道，用于连接子进程 */
VERIFY(::CreatePipe(&m_hStdIn, &hStdInWriteTmp, &sa, 0));    // 输入
VERIFY(::CreatePipe(&hStdOutReadTmp, &m_hStdOut, &sa, 0));   // 输出
VERIFY(::CreatePipe(&hStdErrReadTmp, & m_hStdErr, &sa, 0));  // 错误
/* 建立子进程，并连接管道，lpszCmdLine为调用GDB调试器的字符控制命令 */
PrepAndLaundRedirectedChild(lpszCmdLine, m_hStdOut, m_hStdIn, m_hStdErr,
 bShowChildWindow);
```

通过建立读取相应管道内容的子线程来获取 GDB 调试器的输出内容包括错误提示，方法如下：

```
// 创建子线程读取子进程的输出信息
// staticStdOutThread为读取输出管道信息的子线程代码
m_hStdOutThread = ::CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)
staticStdOutThread, (LPVOID)this, 0, &dwThreadID);
VERIFY(m_hStdOutThread != NULL);
// 创建子线程读取子进程的错误信息
// staticStdErrThread为读取错误管道信息的子线程代码
m_hStdErrThread = ::CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)
staticStdErrThread, (LPVOID)this, 0, &dwThreadID);
VERIFY(m_hStdErrThread != NULL);
```

向 GDB 调试器发送调试命令时，只需向相应输入管道发送数据，方法如下：

```
// m_hStdInWriteTmp写输入管道句柄，lpszInput发送缓冲区，Length发送字符数
// WriteFile向管道传送数据，nBytesWrote写入的字符数
if (!::WriteFile(m_hStdInWriteTmp, lpszInput, Length, &nBytesWrote, NULL))
{
    if (::GetLastError() == ERROR_NO_DATA); // 管道关闭
 else  ASSERT(FALSE);} // 出错
```

GDB 调度与重定向过程中用到的主要 Windows API 函数如表 5-4 所示。

表 5-4 GDB 调度与重定向中的主要 API 函数

函数名称	功能	入口参数说明(按从左到右的顺序)	返回值
CreatePipe	创建管道	读管道端句柄、写管道端句柄、安全属性、管道大小	创建成功返回非零否则为零
DuplicateHandle	复制句柄	源句柄、副本句柄、接收复制进程句柄、目标副本句柄、访问副本句柄属性、句柄可继承属性标志、其它选项	复制成功返回非零否则为零
CreateProcess	创建进程	可执行程序名、命令、进程属性、线程属性、句柄可继承属性、进程类型、进程运行环境、当前目录、进程初始运行信息、进程其他信息	创建成功返回非零否则为零
CreateThread	创建线程	线程安全属性、线程堆栈深度、线程起始地址、传给线程的参数、线程创建的附加标志、线程 ID、	返回线程句柄或者零

5.6 人机交互调试界面的设计与实现

SD-IDE for ColdFire 调试界面如图 5-1 所示，其中寄存器窗口给出了芯片内部寄存器值，变量窗口显示了程序中定义的变量值，存储器窗口则用于显示 RAM 与 Flash 中的相关数据，一次显示 64 字节。由于代码调试模块通过 GDB/MI 接口(输入命令和输出记录格式可参见 2.3 节)与 GDB 通信以实现各种调试功能，故与用户交互信息的调试界面实现起来相对简单。首先需要向 GDB 发送 GDB/MI 命令，随后按照 GDB/MI 格式解析反馈的调试信息，最后将调试信息更新到调试界面之中。表 5-5 列出了 SD-IDE for ColdFire 各种调试功能调用的 GDB/MI 命令；而图 5-8 则给出了解析反馈调试信息的流程。反馈的调试信息是以“(gdb)”作为结束标志的。根据不同信息内容，解析操作可分为置相应标志、提取显示信息和执行“^done”参数命令

表 5-5 SD-IDE for ColdFire 调试功能对应 GDB/MI 命令

功能描述	相关 GDB/MI 命令调用
开始调试	-target-select remote COM1 -target-download
设置断点	-break-insert
取消断点	-break-delete
继续执行	-exec-continue
单步步入	-exec-step
单步步过	-exec-next
读存储器	-data-read-memory
读寄存器	-data-list-register-values
读写变量	-var-evaluate-expression/-var-assign
结束调试	-gdb-exit

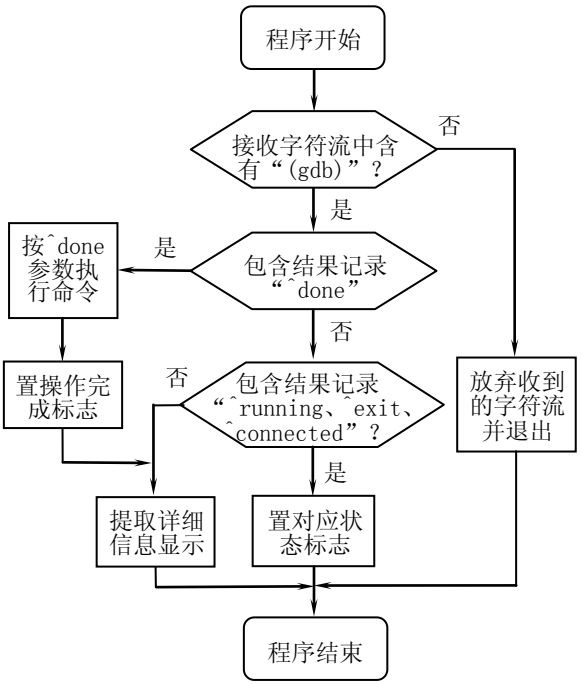


图 5-8 反馈调试信息解析流程图

三部分。其中当结果记录为“^running、^exit、^connected”时，只需修改当前状态标志；其他结果都需提取信息显示。

5.7 本章小结

本章工作总结如下：

① 简要介绍了宿主机调试平台 SD-IDE for ColdFire 及其模块结构，给出了添加 MCF52233 工程模板的方法。

② 详细阐述了实现交叉编译的相关内容，包括 GCC 工具链的构造，Makefile 文件和连接脚本的创建。

③ 利用动态链接库驱动并口 BDM 头完成与目标芯片的通信，在此基础上，通过运行内部 Flash 擦除写入程序，实现了针对 MCF52233 微处理器的代码写入功能。

④ 通过重定向调试信息，将 GDB 与人机交互调试界面联系在一起。利用 GDB/MI 接口与后端运行的 GDB 调试器进行交互，实现针对 MCF52233 微处理器的代码调试功能。

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 设计了一套调试系统，为开发人员提供了一个操作简单、价格低廉、功能完善的嵌入式集成调试平台，降低了嵌入式软件开发的难度，使用户无需将主要精力花费在开发调试工具上，而可以专注于产品的功能设计和程序的流程控制等。本文主要工作总结如下：

① 采用 Freescale ColdFire 系列中的 MCF52233 微处理器，设计并制作了一块硬件评估板 SDMCF52233EVB。它集成了目前嵌入式系统应用领域中常见的功能模块，主要硬件资源包括 32KB 片内 RAM、256KB 片内 Flash、串口、以太网口以及 A/D 转换芯片，并为连接扩展板提供了接口。

② 在借鉴 GDBStub 一般结构和调试原理的基础上，设计了针对 MCF52233 微处理器的调试桩 GDBStub for ColdFire，实现内容涉及启动模块、RSP 通信模块、中断模块和命令处理模块。该调试桩通过串口配合宿主机端 GDB，实现对于目标机系统的联合调试。编写了基于 GDBStub for ColdFire 的调试实例，通过分析远程调试过程中的 RSP 报文信息，给出了该实例的详细调试过程。

③ 开发了一套适用于 ColdFire 的 GDB 集成调试环境 SD-IDE for ColdFire。在设计 SD-IDE for ColdFire 时，充分考虑了平台的芯片无关性和可扩展性。针对 ColdFire 系列微处理器，实现了与芯片密切相关的交叉编译部分、程序写入部分和代码调试部分，并通过修改一些配置文件创建了 MCF52233 工程模板。交叉编译的实现需要构建 GCC 工具链，编写 Makefile 文件以及代码连接脚本。基于 BDM 的代码写入模块包括 BDM 通信接口程序、宿主机端写入程序和 MCF52233 内部 Flash 擦写程序三方面内容。代码调试模块的主要工作为重定向 GDB 调试器并实现人机交互调试界面。

实践证明，本文讨论的基于 ColdFire 微处理器的嵌入式调试系统能够满足 MCF52233 等 ColdFire 用户的调试需求，有效的降低了嵌入式程序的开发难度，提高了开发效率。

6.2 展望

课题是基于Freescale半导体公司的32位微处理器ColdFire进行开发的，作者以及所在的苏州大学 Freescale 实验室以前并没有使用过该系列的 CPU，而且其中文资料较少，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的调试系统，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 丰富评估板的种类。由于硬件条件的限制，本文只在SDMCF52233EVB平台上完成了芯片相关代码的编写、测试以及GSBStub for ColdFire的实现。如果更换了其他ColdFire系列的芯片，SD-IDE for ColdFire针对通用部分的设置以及代码调试模块的源程序都不需要改变，主要修改内容为代码写入模块。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，本文只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系列芯片的写入，增强了系统的灵活性。

② 扩展GDBStub for ColdFire的通信方式。本文实现的用作目标机端调试代理的GDBStub for ColdFire目前只能使用串行通信方式与GDB连接，将来可以考虑实现基于网络通信与BDM头通信的调试桩。其方法为更换通信时调用的相应介质驱动程序。

③ 增加 GDBStub for ColdFire 的 Flash 烧写功能。本文实现的 GDBStub for ColdFire 并不支持 Flash 存储器的擦除和写入，今后可以将 SD-IDE for ColdFire 代码写入模块中的 Flash 擦写功能移入 GDBStub for ColdFire 中并增加对应的 RSP 协议命令，以实现该调试桩对于目标机 Flash 的操作。

调试平台需要在实际使用过程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调试系统的开发必将促进ColdFire系列芯片的推广应用与嵌入式调试方式的发展完善，丰富嵌入式产品市场，推动嵌入式技术的进步。

参考文献

[1] 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.

[2] ColdFire Embedded Controllers[DB/OL].http://www.freescale.com,2007.

[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] Jack Ganssle.Introducion to In-Circuit Emulators[EB/OL].http://www.embedded. com/story/OEG20011126S0065,2001.

[6] 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.

[7] Amold berger,Michael Barr.Introducion to On-Chip Debug[EB/OL].http://www.embedded.com/story/OEG20030205S0032,2003.

[8] 龚伟.基于 gdb 的嵌入式系统调试器的设计与实现[D].电子科技大学硕士学位论文,2006.

[9] IEEE1149.1,IEEE standard test access port and boundary-scan architecture[s].2001.

[10] 郭继伟.基于 ColdFire 的评估系统的设计与实现[D].苏州大学硕士学位论文,2007.

[11] 聂章龙.Freescale HCS12 系列 MCU 嵌入式 IDE 的设计与实现[D].苏州大学硕士学位论文,2007.

[12] GDB Internals[EB/OL].http://sources.redhat.com/gdb/download/onlinedoc/gdb.html,2005.

[13] 单开涛.嵌入式远程调试中目标机模块若干关键技术的研究与实现[D].浙江大学硕士学位论文,2006.

[14] Debugging with GDB:GDB-MI[DB/OL].http://www.gnu.org,2007.

[15] 赵民栋.嵌入式软件集成开发环境中的调试器的设计与实现[D].西北工业大学硕士学位论文,2004.

[16] Gatliff Bill.Embedding with GNU:the gdb Remote Serial Protocol[J].Embedd Systems Programming,1999,9(11):109-110.

[17] 王伟波.嵌入式系统软件 IDE 设计与实现[D].浙江大学硕士学位论文,2005.

[18] 邹楚雄.交叉编译和交叉调试工具的研究和实现[D].电子科技大学硕士学位论文,2006.

[19] ColdFire MCF5223x 系列单片机[EB/OL].http://www.freescale.com.cn/coldfire,2007.

[20] MCF52233 Reference Manual Rev 2[DB/OL].http://www.freescale.com,2007.

[21] 曾立志.推广贴片式、薄膜式变压器是必然趋势[J].国际电子变压器,2004,(6):55-56.

[22] HQY.网络基础必修课图解网卡硬件篇[EB/OL].http://www.52blog.net/user1/3257/archives/2005/466045.shtml,2005.

[23] RJ45 Integrated Magnetics,Pulse Port Technology,PPT 深圳市脉普特通讯技术有限公司[EB/OL].http://www.pptchina.com,2005.

[24] 郑红静.基于嵌入式 Web 服务器的测控系统的开发[D].苏州大学硕士学位论文,2006.

[25] 张南南,尤一鸣.恶劣环境下的高性价比数据采集系统[J].天津工业大学学报,2003,22(1):81-83.

[26] TLC2543-EP 12-BIT ANALOG-TO-DIGITAL CONVERTERS WITH SERIAL CONTROL AND 11 ANALOG INPUTS[DB/OL].http://www.21icsearch. com/searchpdf,2004.

[27] 王宜怀,刘晓升.嵌入式技术基础与实践[M].北京:清华大学出版社,2007.

[28] Pavel Pisa.BDM Interface for Motorola 683xx MCU Usage with GDB Debugger[Z].2000.

[29] 王宜怀.嵌入式应用在线编程开发系统的研制[J].计算机工程,2002,(12):22-24.

[30] 陆晗.基于 GNU 的 JTAG 调试器的集成与设计[D].浙江大学硕士学位论文,2005.

[31] Alfred V.Aho,Ravi Sethi.Compilers:Principles,Techniques and Tools[M].北京:人民邮电出版社,2002.

[32] 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.

[33] Robert Mecklenburg.Managing Projects with GNU Make[M].南京:东南大学出版社,2005.

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

[35] 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,2006,26(z1):261-263.

[36] 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.

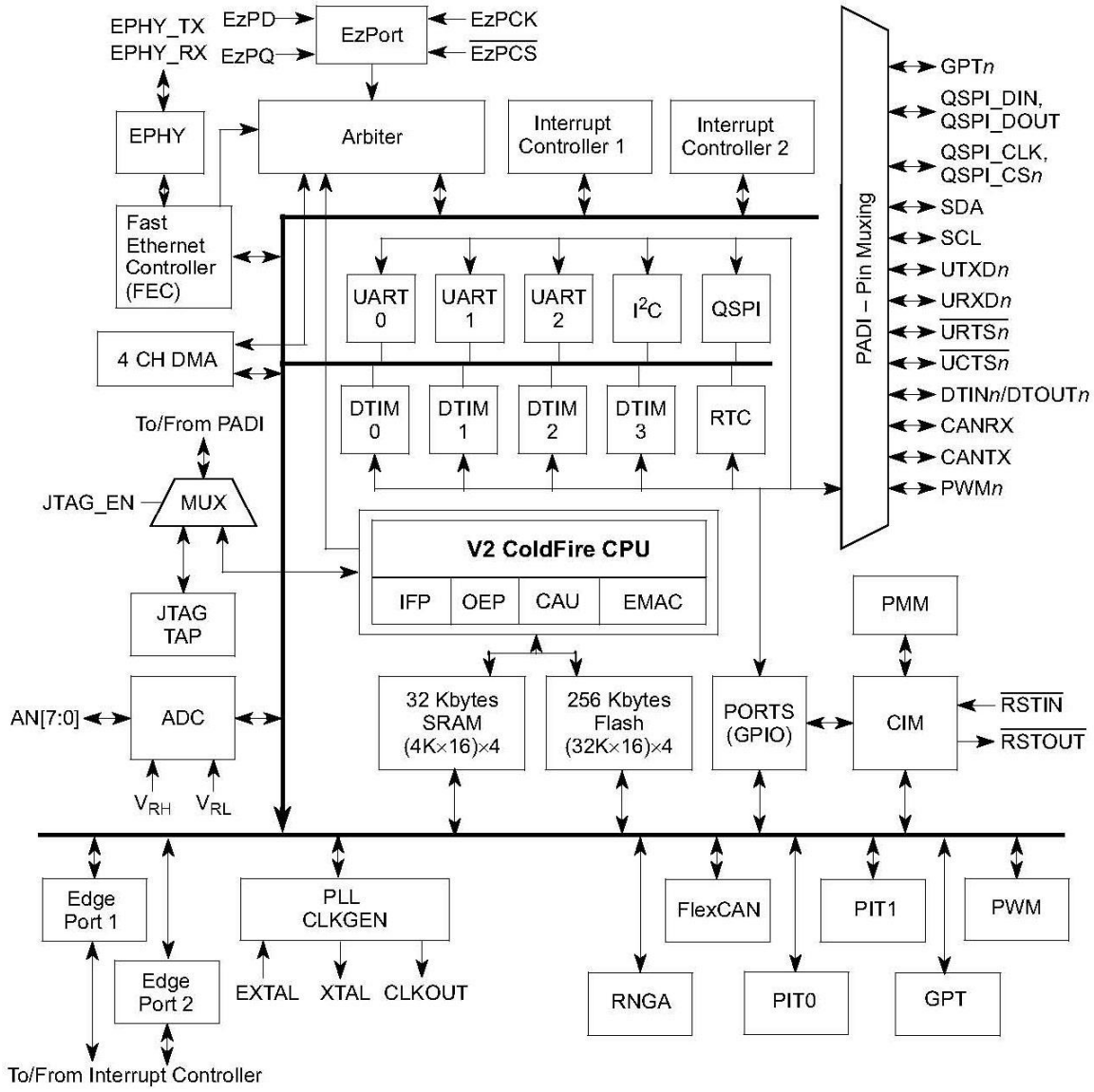
[37] 赵炯.Linux 内核完全注释[M].北京:机械工业出版社,2004.

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

[39] 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.

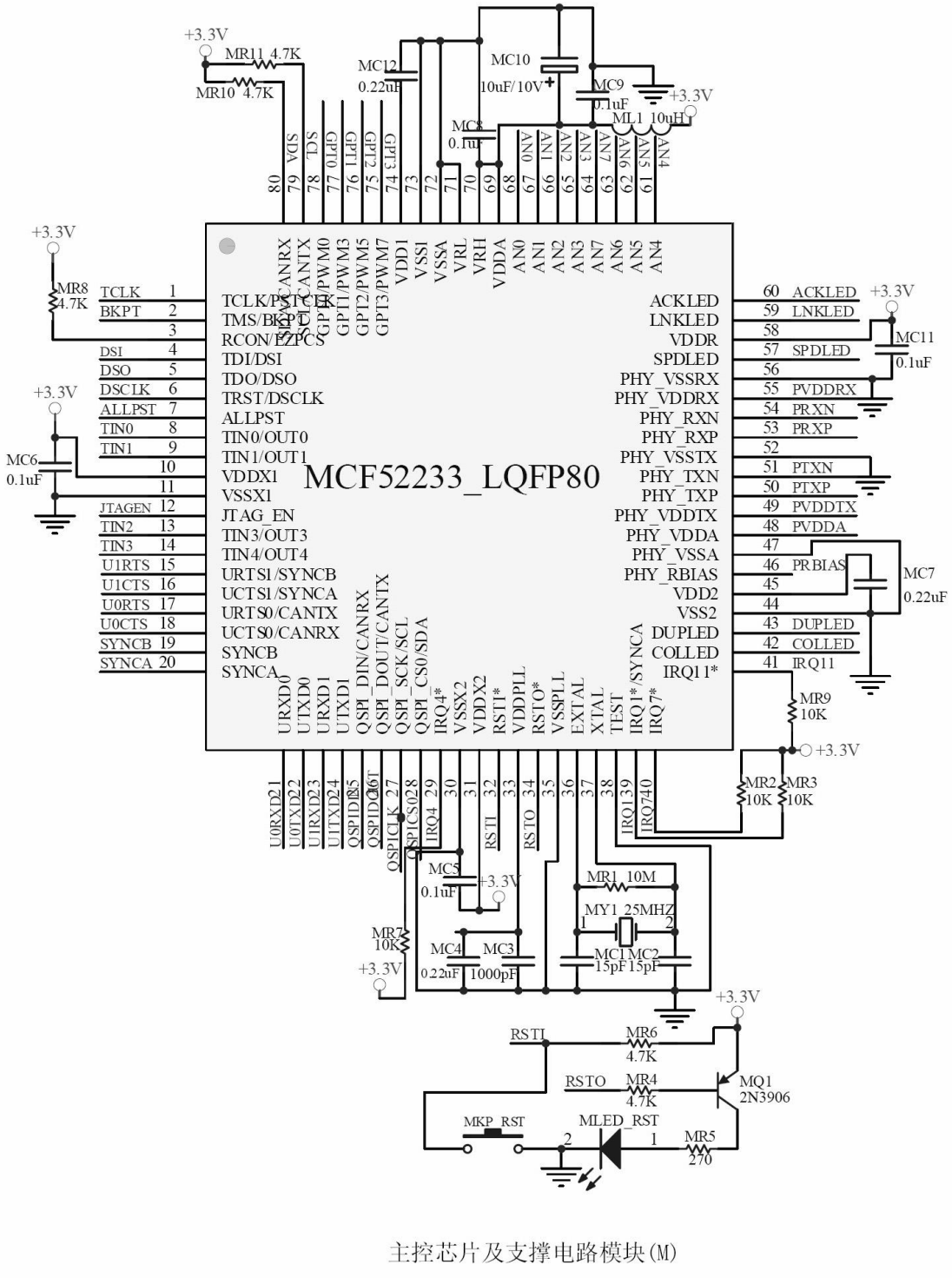
[40] Motorola S-record description[EB/OL].http://www.freescale.com,1999.

附录 A MCF52233 内部功能模块框图



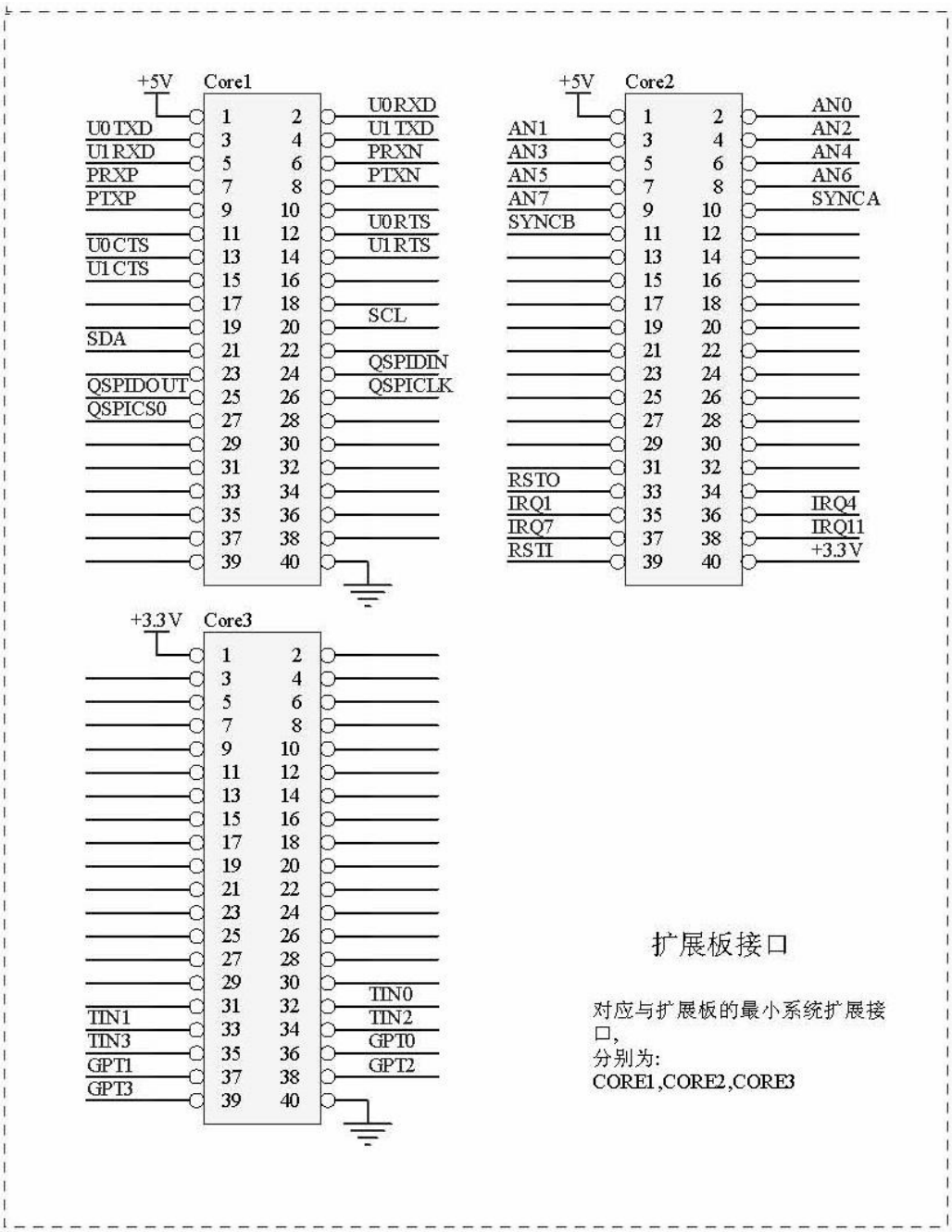
附录 B SDMCF52233EVB 原理图

B.1 MCF52233 芯片及支撑电路

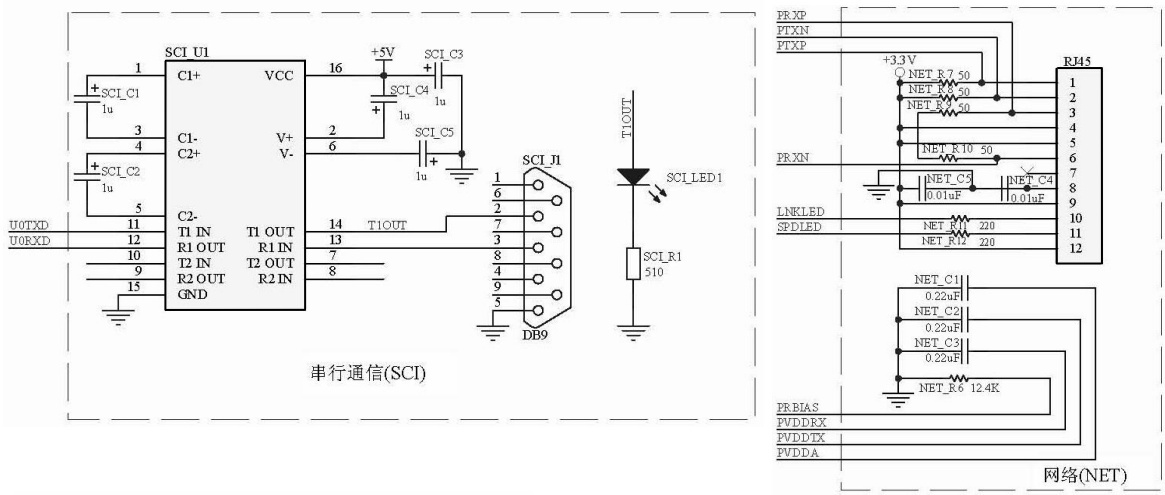


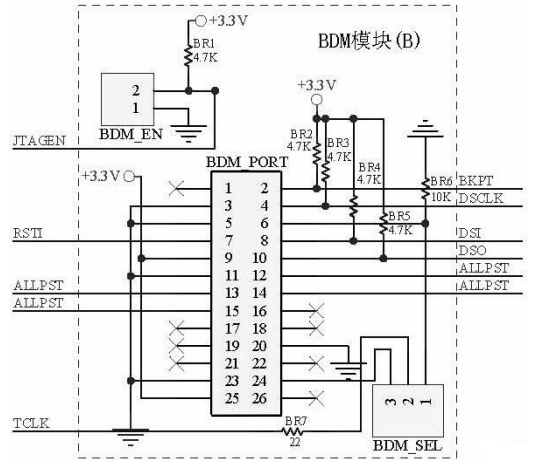
主控芯片及支撑电路模块(M)

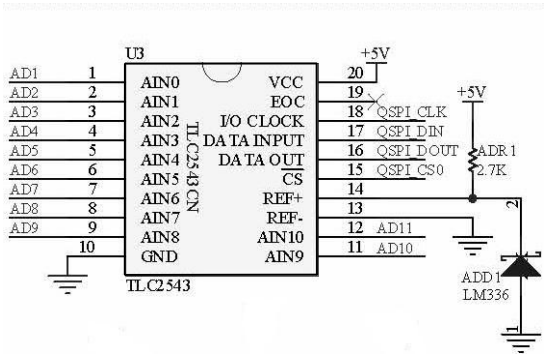
B.2 与扩展板的接口

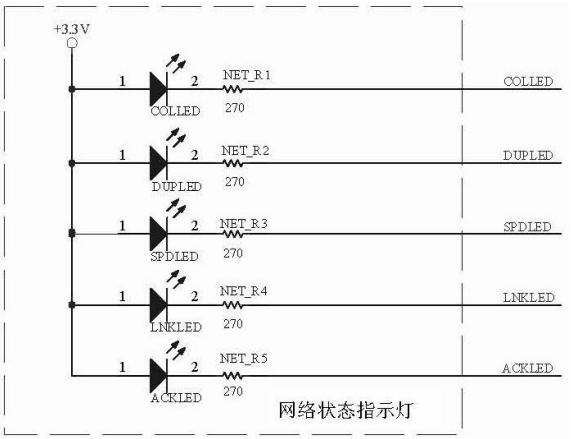

+5V
Core1
U0RXD
U0TXD
U1TXD
U1RXD
PRXN
PRXP
PTXN
PTXP
U0RTS
U0CTS
U1RTS
U1CTS
SCL
SDA
QSPIDIN
QSPIDOUT
QSPICLK
QSPICS0
+5V
Core2
AN0
AN1
AN2
AN3
AN4
AN5
AN6
AN7
SYNCA
SYNCB
RSTO
IRQ1
IRQ4
IRQ7
IRQ11
RSTI
+3.3V
+3.3V
Core3
TIN0
TIN1
TIN2
TIN3
GPT0
GPT1
GPT2
GPT3
扩展板接口
对应与扩展板的最小系统扩展接口,
分别为:
CORE1,CORE2,CORE3

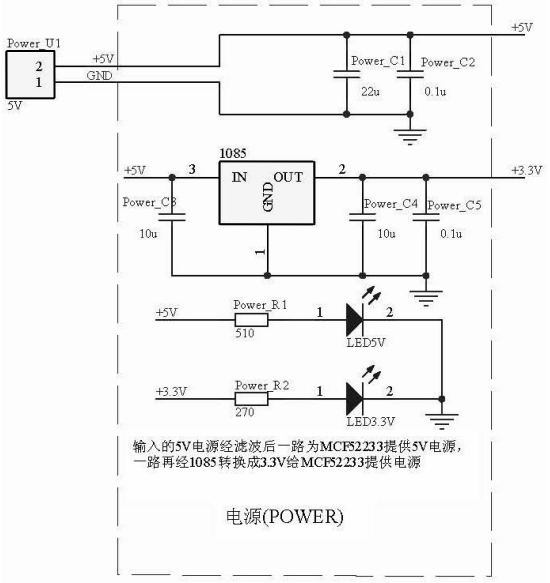
B.3 BDM 及各功能模块











附录 C 交叉编译相关资料

C.1 构建 GCC 编译器的详细命令

```
mkdir c:/build
cd c:/build
tar -jxf binutils-2.17.tar.bz2
tar -zxf gcc-4.1.0.tar.gz
tar -zxf newlib-1.14.0.tar.gz
mkdir build-bin
mkdir build-gcc
mkdir build-newlib

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

cd /cygdrive/c/build/build-gcc
/cygdrive/c/build/gcc-4.1.0/configure \
--target=m68k-elf \
--prefix=/cygdrive/c/coldfire \
--with-newlib --without-headers \
--enable-languages=c --disable-threads --nfp
make all install
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

cd /cygdrive/c/build/build-gcc
/cygdrive/c/build/gcc-4.1.0/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
```

C.2 MCF52233 芯片连接文件

```
OUTPUT_ARCH(m68k) /*定义输出目标文件CPU的体系结构*/
MEMORY /*存储器地址空间*/
{   flash(RX): ORIGIN = 0x00000000, LENGTH = 0x00040000     /* 256K */
    vectorram(RWX): ORIGIN = 0x20000000, LENGTH = 0x00000400 /* 1K */
    sram(RWX): ORIGIN = 0x20000400, LENGTH = 0x00007C00      /* 31K */
    ipsbar(RWX): ORIGIN = 0x40000000, LENGTH = 0x0
}
SECTIONS /*输入文件中的段的存储器映射*/
{   ipsbar: {} > ipsbar
   .flash: /* 代码段 */
    {
        *(.romvec)             /* vectors.o的代码段 */
        . = ALIGN(0x10);        /* 与下一个0x10对齐 */
        *(.text)
        . = ALIGN(0x10);        /* 与下一个0x10对齐 */
        *(.rodata)
        __DATA_ROM = .;  /* __DATA_ROM为当前地址值 */
    } > flash
    data: AT(__DATA_ROM) /* 数据段 */
    {
        __DATA_RAM = .;  /* 数据段在SRAM的起始地址 */
        *(.exception)
        . = ALIGN(0x10);        /* 与下一个0x10对齐 */
        *(.data)
        . = ALIGN(0x10);        /* 与下一个0x10对齐 */
        __DATA_END = .;  /* 数据段在SRAM的结束地址 */
        *(.sdata)
        . = ALIGN(0x10);        /* 与下一个0x10对齐 */
    } > sram
   .bss: /* 未初始化数据段 */
    {
        . = ALIGN(0x10);        /* 与下一个0x10对齐 */
        __START_SBSS = .;
        *(.sbss)
        *(SCOMMON)
        __END_SBSS = .;
        . = ALIGN(0x10);        /* 与下一个0x10对齐 */
        __START_BSS = .;
        *(.bss)
        *(COMMON)
        __END_BSS = .;
        __BSS_START = __START_SBSS;
        __BSS_END   = __END_BSS;
        . = ALIGN(0x10);        /* 与下一个0x10对齐 */
    } > sram
}
/*定义初始化时用到的系统参数*/
__FLASH = ADDR(.flash);          /* Flash的起始地址 */
__FLASH_SIZE = 0x00040000;             /*Flash的大小*/
__VECTOR_RAM = 0x20000000;
__SRAM = 0x20000000;              /* SRAM的起始地址 */
__SRAM_SIZE = 0x00008000;              /* SRAM的大小 */
__IPSBAR = ADDR(.ipsbar);/* 设置片内外设基址寄存器 */
__SP_INIT = __SRAM + __SRAM_SIZE;    /* 初始化堆栈 */
```

攻读学位期间公开发表的论文及参与的鉴定项目

[1] 陈祎、王宜怀，μC/OS-Ⅱ在应用系统中任务划分方法的研究，现代电子技术，2008，18(9)(已录用)

[2] 陈祎，基于 Freescale S12 系列 MCU 的 μC/OS-Ⅱ操作系统移植及应用，苏州大学学生科协基金项目三等奖。

[3] 陈祎、舒胜强等，串励他励叉车电机控制器，2005 年 12 月通过苏州大学科研处鉴定。

[4] 参与王宜怀、刘晓升编著的《使用 HCS12 微控制器的设计与应用》第 13 章的撰写，该书已于 2008 年 3 月由北京航空航天大学出版社出版。

致 谢

三年时间飞逝而过，在这即将结束我学生生涯之际首先感谢我的导师王宜怀教授和他的夫人张建英老师，感谢他们在学习上和生活上对我的关心和帮助。在王老师的言传身教下，我学会了如何独立思考问题、分析问题、解决问题，以及做事认真负责的态度。

感谢刘晓升老师，在这三年的研究生生活和相关的项目开发中给了我无私的帮助，让我在专业技术领域学到了很多知识。

三年中大部分时间都是在实验室度过的，这是一个温馨融洽的集体，感谢实验室的兄弟姐妹们。

感谢我的家人，这么多年来一直为我提供了良好的家庭环境，毫无保留的物质支持和精神鼓励，使得我能够全身心的投入学习。

最后，再次向所有的老师、同学、朋友所给予的关心、帮助与鼓励致谢！