中文摘要

嵌入式集成开发环境 IDE 和硬件评估系统是嵌入式产品开发的必要工具。开发嵌入式集成开发系统涉及到的工具较多，过程较复杂，目标芯片的更新换代也较快，所以存在较大的难度。HCS12 系列 MCU 是 Freescale 公司于 2004 年前后推出的高性价比 16 位芯片，型号丰富，市场前景较好，国内目前尚未开发出针对该系列 MCU 的嵌入式 IDE，主要依赖国外进口。正是基于这样难得的机遇和挑战，本课题设计了一个通用的嵌入式集成开发系统，实现了对 Freescale HCS12 系列 MCU 源文件进行编辑、编译、下载和基本调试的功能，同时，在设计过程中也充分考虑了软硬件平台的通用性。

SdIDE12 的硬件平台包含 HCS12 系列 MCU 的最小系统模块、程序写入模块和通用扩展板模块，软件平台由通用 IDE 模块、程序写入模块和基本调试模块等组成。文章给出了硬件平台的通用性设计方法、相应模块的原理框图及测试流程，重点阐述了软件平台中通用 IDE 模块和通用 HCS12 系列 MCU 写入模块的设计要点、难点和细节，解决了交叉编译模块和写入模块不通用、写入代码大小受限制等问题。另外，文章还给出了 HCS12 系列 MCU 调试模块的初步设计与实现。最后，为了验证 SdIDE12 的稳定性和基本性能，进行了 MC9S12DG128 芯片的基础实验例程的编写和基于 CPU12 微处理器的 μC/OS-Ⅱ移植，实现了软硬件模块的综合测试，测试结果表明 SdIDE12 工作稳定可靠。

关键词： Freescale HCS12 系列 MCU，集成开发系统，SdIDE12，通用性，程序写入

Abstract

Embedded Integrated Development Environment(IDE) and hardware evaluation system are necessary tools for embedded product development. A mature, stable, and powerful embedded IDE can simplify the operation and shorten the development cycle. However, developing embedded IDE is complex and prone to cause error with a lot of tools, so there is big difficulty. HCS12 MCUs are high performance and low cost 16-bit chips of Freescale's company which has many types of chips and better market prospect . At present, the embedded IDE for HCS12 MCUs have not been developed at home. Based on the hard-won opportunity and challenge, the author has designed a generic embedded IDE called SdIDE12 with the functions of editing, compiling, downloading and basic debugging for Freescale HCS12 MCUs. At the same time, the universality is fully considered in the design of the hardware and software platform.

The hardware platform consists of chip support circuit, program downloading and general extend board modules. The software platform is composed of the general IDE, the program downloading and the basic debugging module. This paper shows the method to design the general hardware platform, the block diagram and test process of the hardware module. Furthermore, this paper analyzes the architecture of general software platform, and puts emphasis on describing the key points, difficulty and the detail of the general IDE and program downloading module, gives the methods to make the cross compiler and the program downloading module universality, and solves the problem about the restricted code size. This paper also introduces the design and the achievement of debugging module. Finally, for the purpose of testing the SdIDE12 's stability and basic performance , the paper shows the μC/OS-II transplant based on CPU12 and the basic experimental programs of MC9S12DG128, so that the comprehensive testing of the hardware and software modules can be realized, the test result shows SdIDE12 can work well with good stability and reliability.

Key words： Freescale HCS12 MCU, Integrated Development Environment, SdIDE12, Universality, Program Downloading

目　　录

第一章 绪论 1

1.1 嵌入式系统概述 1
1.2 SdIDE12 的开发背景 2
1.2.1 嵌入式 IDE 国内外发展现状 2
1.2.2 Freescal HCS12 系列 MCU 概述 4
1.3 开发 SdIDE12 的必要性及意义 5
1.4 课题设计目标 6
1.5 本文工作和论文结构 7
1.5.1 本文工作 7
1.5.2 论文结构 8

第二章 SdIDE12 的整体设计思路 9

2.1 需求分析 9
2.2 SdIDE12 的硬件环境设计思路 10
2.3 软硬件协同设计思路 10
2.4 SdIDE12 的通用性设计思路 11
2.4.1 硬件的通用性设计 11
2.4.2 软件的通用性设计 13
2.5 SdIDE12 的功能模块设计思路 16
2.5.1 SdIDE12 公有模块 17
2.5.2 SdIDE12 私有模块 18
2.6 本章小结 20

第三章 硬件设计 21

3.1 最小系统硬件设计 21
3.2 通用扩展板设计 23
3.2.1 键盘模块 23
3.2.2 LCD 模块 24
3.2.3 串口模块 25
3.2.4 A/D 转换模块 26
3.2.5 PWM 模块 26
3.2.6 USB 模块 26
3.2.7 CAN 模块 27
3.2.8 以太网模块 28
3.3 写入模块(BDM 头)硬件设计 28
3.3.1 写入模块硬件结构设计 28
3.3.2 写入模块原理图分析与设计 29
3.4 硬件平台测试及测试体会 31

3.4.1 测试方法和步骤 31
3.4.2 测试体会 32
3.5 本章小结 33
第四章 软件设计 35
4.1 SdIDE12 的通用 IDE 模块详细设计 35
4.1.1 SdIDE12 主界面结构设计 35
4.1.2 SdIDE12 的目录结构 36
4.1.3 SdIDE12 的代码结构 37
4.1.4 SdIDE12 的工程管理模块 38
4.2 通用 HCS12 系列 MCU 写入模块详细设计 43
4.2.1 通信程序接口设计 43
4.2.2 通用写入模块 PC 方程序设计 46
4.2.3 通用写入模块 MCU 方程序设计 48
4.3 HCS12 系列 MCU 内存扩展的实现 51
4.3.1 存储空间的扩展 51
4.3.2 线性地址转换为内存扩展地址 52
4.3.3 编译生成 S2 格式文件 53
4.4 调试模块详细设计 56
4.4.1 .lst 文件结构 57
4.4.2 设置断点的实现 57
4.4.3 调试环境的初始化 58
4.4.4 单步调试的实现 59
4.5 测试体会 60
4.6 本章小结 61
第五章 基于 CPU12 微处理器的 μC/OS-Ⅱ移植 62
5.1 μC/OS-Ⅱ移植过程 63
5.1.1 修改 OS_CPU.H 文件 63
5.1.2 修改 OS_CPU_C.C 文件 64
5.1.3 修改 OS_CPU_A.S 文件 65
5.2 μC/OS-Ⅱ移植的测试 67
5.3 本章小结 69
第六章 基础实验例程 70
6.1 编程规范 70
6.2 MC9S12DG128 芯片的模板程序 72
6.3 MC9S12DG128 各模块的实验例程 75
6.4 本章小结 80
第七章 总结与展望 81
7.1 总结 81
7.2 展望 82

参考文献 83
附录 A 硬件评估板原理图 85
附录 B 硬件评估板实物图 92
附录 C 规范程序实例 93

第一章 绪论

目前，计算机在社会的各个领域扮演着重要的角色，特别是嵌入式计算机系统在最近几年中呈现出兴起的趋势，已被广泛应用于工业控制、通讯、仪器、仪表、汽车、船舶、航空、航天、军事装备和消费电子等嵌入式领域。和通用计算机不同，嵌入式计算机系统的硬件和软件都必须高效率地设计、量体裁衣、去除冗余，力争在同样的硅片面积上实现更高性能，这就要求用户选择合适的处理器，可对其配置进行裁剪和添加。目前嵌入式系统中使用的处理器主要有：8 位、16 位、32 位、64 位和数字信号处理器(DSP)。其中，16 位 MCU 在近几年发展中呈现出高速增长的势头[1]，Freescale HCS12 系列 MCU 是目前市场影响力较大的 16 位 MCU，有近 70 个型号[2]，目前国内公司尚未开发出针对这一系列 MCU 的开发平台及开发环境。本文旨在开发一个通用嵌入式集成开发系统的框架，在该框架平台上，可以方便的添加各个系列 MCU 的交叉编译、程序下载和代码调试模块，以实现其通用性。目前已开发了该框架平台，并编写了 Freescale HCS12 系列 MCU 的编译、下载和调试等功能模块，且已成功的应用于实践(下文将该嵌入式集成开发系统简称为 SdIDE12)。其他系列 MCU 功能模块的编写将在后期逐步展开，并添加到该系统中。本章主要内容有：第一节介绍了嵌入式系统的基本概念，第二节给出课题的背景，第三节给出课题的意义，第四节给出 SdIDE12 的设计思路，最后列出本文的工作内容和论文组织结构。

1.1 嵌入式系统概述

随着后 PC 时代的到来，计算机技术、半导体技术及电子技术的飞速发展为社会各领域带来了巨大的生产效益，人们越来越多地接触到一个新的概念——嵌入式系统(Embedded System) 。但是对于何为嵌入式系统，什么样的技术可以归属于嵌入式技术，仍在讨论之中，有关嵌入式系统的概念也没有统一的定义。纵观嵌入式领域资深人士关于嵌入式系统的讨论[3][4]，可将其定义归结为广义和狭义两个方面。

广义上将其定义为：嵌入到对象体系中的专用计算机系统。狭义上则认为嵌入式系统是以应用为中心、以计算机技术为基础、软件硬件可裁剪、适应应用系统对功能、可靠性、成本、体积、功耗严格要求的专用计算机系统。应该说后者从功能应用特征

上比较好的给出了嵌入式系统的定义。

今天嵌入式系统带来的工业年产值已超过了 1 万亿美元，在嵌入式系统中占市场份额最大的是嵌入式微控制器(MCU)，它占整个嵌入式系统约 70%的市场份额。目前嵌入式系统除了部分为 32 位处理器外，大量存在的是 8 位和 16 位的嵌入式微控制器。嵌入式微控制器目前的品种和数量很多，比较有代表性的通用系列包括 8051、MIPS、R8C/TINY、PowerPC、MC68HC05/11/12/16、ColdFire/68k、ARM 等。

由于嵌入式微控制器可靠性高、功能强、体积小、使用方便，并随着相配套的成熟的集成开发平台和开发环境的相继出现，使其应用深入到社会的各个领域，对各行各业的技术改造、产品更新换代、加速自动化进程、提高生产率等方面起到了极其重要的推动作用。

1.2 SdIDE12 的开发背景

由于嵌入式软件自身的特点，其开发有一些特殊的要求，既要满足不同应用领域产品的功能要求，又要保证产品质量，缩短开发周期。因而，开发嵌入式系统产品，必须为嵌入式软件开发者提供一套使用方便、高效的集成开发环境 (Integrated Development Environment，IDE)，开发者可利用集成开发环境中提供的各种工具，高效的设计和研发市场所需的嵌入式产品。

1.2.1 嵌入式 IDE 国内外发展现状

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规模已经不再局限于原先的驱动程序和简单应用程序，目前形成的计算机应用模式带有明显的计算机的工程应用特点，即基于嵌入式系统软硬件平台，以网络、通信为主的非嵌入式底层应用。所以，构建完善的嵌入式集成开发系统已经逐渐成为嵌入式领域的一个新兴课题。

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

Tornado 和 CodeWarrior 是目前市场上影响力较大的 IDE 产品，其功能完善、性

能稳定，但都属于商业级软件，价格昂贵，操作界面国外化。国内主要使用国外引进的嵌入式 IDE，自主研究和开发的成果较少，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 IDE 产品，但它们适用的微控制器的系列较少，不具有良好的扩展性。

表 1-1 几种国内外 IDE 产品的发展状况

名称	目标 MCU	编辑	编译	调试	编译器	编程语言	供应商	价格	开发时间
Tornado	68xxx,MIPS, SPAR,ARM,NEC Vxx	√	√	√	GNU 编译器	C/C++ 汇编	Wind River	$6800	1995
CodeWarrior	683xx,ARM, PowerPC, MIPS, MC68HCxx	√	√	√	自己开发的	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	PowerPC,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

国外，嵌入式 IDE 的另一支重要的研发队伍是自由软件协会 GNU(GNU's Not Unix)组织，他们在 Internet 上免费提供有关研究和发展成果[5]，如嵌入式操作系统 eCos (Embedded Configurable Operating System) 、针对特定处理器的 GCC (本地编译器)和 CGCC(交叉编译器)等，这些编译器支持许多目前流行的系列微控制器，如 M680X0、MIPS、PowerPC、DSP、Mcore、AVR、MC68HC11/12/16、ColdFire/68k、ARM 等。现在已有很多公司在 GNU 软件的基础上，经过集成、优化和测试，推出了更加成熟、稳定的商业化版本的嵌入式 IDE。

苏州大学摩托罗拉实验室于 2002 年开发了 SD-1 嵌入式集成开发系统，功能较全，性能稳定，被国内 20 多所大学使用，但其有明显的缺点：只适用于 MC68HC908GP32 这一款芯片，且调试功能还需完善。从 2005 年开始我们又对该系统进行全新改版，使其适用于 Freescale HC08 系列 MCU，同时也完善了调试功能。

由以上对国内外嵌入式 IDE 的发展情况及其特性比较分析不难发现，目前国内外的嵌入式集成开发环境对用户来说，都存在以下问题：

① 要么使用方便且功能强大，但价格昂贵，且源代码不开放；

② 要么价格可以接受，但功能不强；

③ 要么提供源代码，但使用受限，操作不方便；

由此可见，开发适合广大国内用户使用的功能强大、性能稳定、价格低廉的通用嵌入式集成开发系统迫在眉睫。

1.2.2 Freescal HCS12 系列 MCU 概述

由于嵌入式系统应用需求的多样性，市场上的嵌入式处理器生产厂商的竞争越来越激烈，ARM、MIPS、68HC12/S12 等系列芯片销售额开始巨增，在消费电子、汽车电子和工业控制领域占有较大市场的飞思卡尔半导体，在 2004 年底取代飞利浦半导体，跻身全球半导体前十强。正是基于 Freescale 系列芯片所拥有的广阔前景，本文研制 SdIDE12 的首要目标就是实现对 16 位 HCS12 系列芯片的嵌入式产品开发。

Freescale 的 CPU12 系列单片机主要有 HC12 和 HCS12 两种类型。HC12 推出比较早，种类也比较多，针对不同的场合都可以选到合适的型号；HCS12 是最近推出的新型的 CPU12 系列单片机，性价比很高，是CPU12 系列单片机的发展趋势。HCS12 系列单片机型号有很多种。在这些不同型号的单片机中，资源各不相同，即使是同一种型号的单片机，也有多种封装形式，其 I/O 口数目也不相同。如 MC9S12DG128 就有 80 脚的 QFP、112 脚的 LQFP 两种封装形式。表 1-2 体现了 HCS12 系列单片机的基本资源差异情况。从表中可以看出 HCS12 系列单片机内置资源差异很大，内存容量最大的达到 14K 字节，最少的为 2K 字节；最多的 I/O 口数有 117 个，最少的只有 25 个；FLASH 最大的达到了 512K 字节，而最少的只有 16K 字节。下面给出几款典型的 HCS12 系列 MCU 的概况。

MC9S12UF32 是一款带 USB 接口的单片机，总线频率最高达 60MHz。片内有 32KB 的 Flash、3.5KB 的 RAM，片内集成了高速的 USB2.0 模块，通信速率 480Mb/s，有 SM(Smart Media) 接口、MS(Memory Stick) 接口、ATAPI接口等，有 100 引脚和 64 引脚两种封装。可制作多功能硬盘，与 MP3、数码相机结合，应用前景广阔。

带 10Mb/s 或 100Mb/s 以太网接口的 MC9S12NE64。片内有 64KB 的 Flash、8KB 的 RAM，8 路 A/D 转换，通信模块有 SCI、SPI、I^2C、以太网，内带 125MHz 的时钟发生器。封装形式有 80 引脚和 112 引脚两种。片内带符合 IEEE802.3 通信协议标准的以太网接口模块，还集成了 10Mb/s 或 100Mb/s 以太网发送接收驱动器(EPHY)。NE64 适用于网络接入设备、家庭网关、工控设备等。

表 1-2 HCS12 系列 MCU 的资源差异情况表

系列	产品型号	RAM	E^2PROM	FLASH	I/O 口数	通信	A/D
A 系列	MC9S12A32	2K	1K	32K	59	2 SCI 1 SPI	8 通道 10 位
	MC9S12A64	4K	1K	64K	91	2 SCI 1 SPI I^2C	2*8 通道 10 位
	MC9S12A128	8K	2K	128K	91	2 SCI 2 SPI I^2C	2*8 通道 10 位
	MC9S12A256	12K 4K		256K	91	2 SCI 3 SPI I^2C	2*8 通道 10 位
	MC9S12A512	14K 4K		512K	91	2 SCI 3 SPI I^2C	2*8 通道 10 位
B 系列	MC9S12B128	4K	1K	128K	91	SCI SPI I^2C	8 通道 16 位
	MC9S12B64	2K	1K	64K	91	SCI SPI I^2C	8 通道 16 位
C 系列	MC9S12C128	4K	0K	128K	60	SCI SPI	8 通道 10 位
	MC9S12C96	4K	0K	96K	60	SCI SPI	8 通道 10 位
	MC9S12C64	4K	0K	64K	60	SCI SPI	8 通道 10 位
	MC9S12C32	2K	0K	32K	60	SCI SPI	8 通道 10 位
D 系列	MC9S12D32	2K	1K	32K	59	2 SCI 1 SPI 8	通道 10 位
	MC9S12D64	4K	1K	64K	91	2 SCI 1 SPI I^2C	2*8 通道 10 位
	MC9S12DB128	8K	2K	128K	91	2 SCI 2 SPI 2*8	通道 10 位
	MC9S12DG128	8K	2K	128K	91	2 SCI 2 SPI I^2C	2*8 通道 10 位
	MC9S12DG256	12K 4K		256K	91	2 SCI 3 SPI I^2C	2*8 通道 10 位
	MC9S12DJ64	4K	1K	64K	91	2 SCI 1 SPI I^2C	2*8 通道 10 位
	MC9S12DJ128	8K	2 K	128K	91	2 SCI 2 SPI I^2C	2*8 通道 10 位
	MC9S12DJ256	12K 2K		256K	91	2 SCI 3 SPI I^2C	2*8 通道 10 位
	MC9S12DP256	12K 2K		256K	91	2 SCI 3 SPI I^2C	2*8 通道 10 位
	MC9S12DP512	12K 4K		512K	91	2 SCI 3 SPI I^2C	2*8 通道 10 位
	MC9S12DT128	8K	2K	128K	91	2 SCI 2 SPI I^2C	2*8 通道 10 位
	MC9S12DT256	12K 4K		256K	91	2 SCI 3 SPI I^2C	2*8 通道 10 位
E 系列	MC9S12E64	4K	-	64K	90	3 SCI SPI I^2C	16 通道 10 位
	MC9S12E128	8K	-	128K	90	3 SCI SPI I^2C	16 通道 10 位
GC 系列	MC9S12GC128	4K	0K	128K	60	SCI SPI	8 通道 10 位
	MC9S12GC96	4K	0K	96K	60	SCI SPI	8 通道 10 位
	MC9S12GC64	4K	0K	64K	60	SCI SPI	8 通道 10 位
	MC9S12GC32	2K	0K	32K	60	SCI SPI	8 通道 10 位
	MC9S12GC16	2K	0K	16K	60	SCI SPI	8 通道 10 位
H 系列	MC9S12H128B	12K 4K		128K	117	SCI SPI IIC	16 通道 10 位
	MC9S12H256B	12K 4K		256K	117	SCI SPI IIC	16 通道 10 位
NE 系列	MC9S12NE64	8K	-	64K	70	2 SCI SPI IIC	8 通道 10 位
T 系列	MC9S12T64	4K	-	64K	25	2 SCI SPI	8 通道 10 位
UF 系列	MC9S12UF32	3.5K	-	32K	75	2 SCI USB2.0	-

带 CAN 总线的 S12D 系列 16 位单片机，其主要特点可参见表 1-2 中 S12D 系列。S12D 系列 16 位单片机采用 5V 供电，总线频率高达 25MHz。主要用于工业控制，特别适合用于汽车上。该系列单片机的特点是拥有丰富的 I/O 模块和工业控制专用的通信模块。80 引脚封装的单片机有 59 个 I/O 口，112 引脚的有 91 个 I/O 口。通信模块有 SCI、SPI、I^2C、CAN、ByteFlight 等。

1.3 开发 SdIDE12 的必要性及意义

嵌入式应用软件的开发与通用计算机软件的开发有较大的差异，由于受资源约

束，嵌入式软件开发一般需要在专门的开发平台上进行交叉开发(cross developing)，软件开发难度大，对开发人员要求较高。同时，由于嵌入式系统所控制的外部设备受环境的复杂度、可靠性及实时性等限制，使得嵌入式软件的开发和调试非常复杂，迫切需要强有力的开发环境作为支持。如果有一套功能强大的嵌入式集成开发环境进行软件开发，则可以保证程序执行的实时性、可靠性，保障软件质量，并减少开发时间。因而，当今对嵌入式集成开发系统的研究倍受重视，出现了许多功能完善的嵌入式开发软件，如 ImageCraft 公司的 ICCAVR、Metrowerks 公司的 CodeWarrior 等，但是作为公司产品，这些软件价格昂贵，且不具有通用性。研究和开发基于 Freescal HCS12 系列微控制器的通用集成开发系统具有如下的使用价值和现实意义：

① Freescale HCS12 系列 MCU 是 Freescale 公司最近推出的新型的 CPU12 系列单片机，性价比很高，是 CPU12 系列单片机的发展趋势，且目前国内没有开发出针对该系列 MCU 的嵌入式集成开发环境，因而，设计和开发SdIDE12 嵌入式集成开发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工具。

② 为国内高校利用国产化的开发工具和实 验仪器进行嵌入式系统 教学提供方便，通过提供丰富的实验例程和教程，为 16 位嵌入式系统初学者进行实验和开发提供捷径。

③ 文中重点介绍通用 IDE 的技术要点，从而尽可能的避免针对不同 MCU 平台而需要重复开发 IDE 的情况，这一点是许多同类型 IDE 所不具有的功能，可为今后的研发人员开发类似产品提供参考。

1.4 课题设计目标

在传统的嵌入式系统开发中，主要是使用仿真器进行软件开发，用软件来模拟目标系统中 MCU 的运行情况，但是使用仿真器往往把目标硬件与软件独立开来。2000 年以来，许多公司推出的 MCU 都具有片内可重复擦写的 Flash 存储器，支持在线编程，不需要额外的设备就可以将程序写入到芯片中运行或者调试。在线编程(In-Circuit Program)允许单片机内部运行的程序去改写 Flash 存储器的内容，利用这个特点，不仅可以在运行过程中修改某些运行参数，也为研制新型嵌入式应用开发工具提供了技术基础。Freescale HCS12 系列单片机支持在线编程和调试，因此本课题设计的 IDE 兼顾了教学与研发的设计思想，集工程管理、编辑、编译、下载和调试工具于一体，

使用这些工具可以生成运行于 HCS12 系列 MCU 的二进制文件，并可对相应的源程序进行调试。SdIDE12 的主要功能特点：①提供目标硬件板、交叉开发工具 等；②拥有通用的 IDE；③集成程序写入和调试功能；④提供教学实验例程和教程，即可以用于高校嵌入式教学又可以为测试开发人员提供参考；图 1-1 为 SdIDE12 嵌入式集成开发系统的模型图。

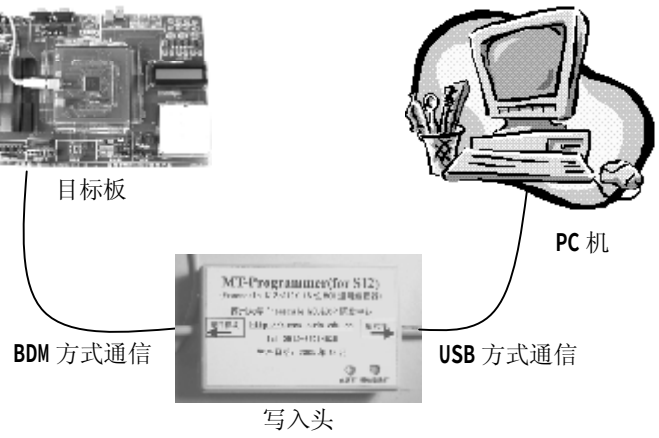


图 1-1 SdIDE12 嵌入式集成开发系统模型图

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) SdIDE12 的整体设计思路

对软、硬件设计的具体内容进行分析，确定软、硬件平台的通用性设计思路。

(2) 硬件设计

① 最小系统板设计

② 通用扩展板设计

③ 写入模块硬件板设计

(3) 软件设计

① 通用 IDE 的详细设计

② HCS12 系列 MCU 通用写入模块详细设计

③ HCS12 系列 MCU 内存扩展的实现

④ 调试模块详细设计

(4) 基于 CPU12 微处理器的 μC/OS-Ⅱ移植

① μC/OS-Ⅱ移植过程

② 移植的测试

(5) 实验例程编写

① 编程规范

② 模板程序的编写

③ 实验例程的编写

1.5.2 论文结构

第一章介绍了开发 SdIDE12 的课题背景、意义以及设计目标；

第二章阐述了 SdIDE12 的通用性设计思路，给出软硬件的整体设计框架；

第三章详细阐述了 SdIDE12 硬件的具体设计与实现，并给出测试心得；

第四章详细阐述了 SdIDE12 软件的具体设计与实现，包括集成开发环境 IDE 的软件设计、写入模块的 PC 方和 MCU 方软件设计、扩展内存的实现和调试模块软件设计；

第五章给出了基于 CPU12 微处理器的 μC/OS-Ⅱ移植；

第六章给出了用于教学和硬件测试的基础实验例程；

最后对全文进行了总结。

第二章 SdIDE12 的整体设计思路

嵌入式集成开发系统包含硬件平台和软件平台两部分：硬件平台以嵌入式处理器为中心，配置存储器、I/O 设备、通信模块、接口设备等必要的外设；软件平台以交叉编译、程序下载和代码调试模块为核心，向上提供应用程序编程环境，向下屏蔽具体硬件特性的板级支持软件包。嵌入式集成开发系统中软硬件紧密配合、协调工作，共同完成系统预定的功能。本章将从软件和硬件两方面阐述 SdIDE12 集成开发系统的整体设计思路。

2.1 需求分析

在一个项目开发过程中，需求分析扮演着很重要的角色，需求分析要明确给出一个项目开发的背景、将来的市场价值和应用前景。由于 16 位 MCU 在消费电子、汽车电子、工业控制及通信工程领域的广泛应用，其市场前景非常看好。Freescale HCS12 系列 MCU 是目前较有市场影响力的新款 16 位微控制器，因而开发 SdIDE12 就显得尤其必要。目前，16 位嵌入式集成开发系统无论是对高等院校教学还是对中小企业产品开发都具有一定的意义。他们对 16 位嵌入式集成开发系统都有以下需求：

① 有开放的功能完善的嵌入式集成开发工具，如编译器，编程器，调试器等；

② 有相应的软件源代码、芯片资料和硬件开发板(包括最小系统板和外围器件的扩展板)；

③ 有各个模块规范的实验例程和实验教程，有合适的实时操作系统内核；

根据以上要求，本课题的设计涵盖了以上所有内容，制定了通用的软件和硬件的设计方案，如表 2-1 所示。

表 2-1 SdIDE12 的软硬件模块

模块类别			功能描述
硬件	最小系统板	CPU	采用 HCS12 系列 MCU 为目标核，如 MC9S12DG128、NE64、UF32 和 S12XDP512 等
		存储器 2	～14KB RAM，32~512KB Flash
	通用扩展板	LCD 和 LED	LCD 液晶屏和 8 段数码管
		键盘	4 x 4 键盘接口
		SCI 和 SPI	串行通信接口
		定时器	定时器溢出中断和输入捕捉
		A/D 转换	模数转换模块

		传感器	常用传感器接口
		电机	PWM 模块应用
		CAN	CAN 总线接口
		I^2C	I^2C 总线接口
		NET	以太网接口
		USB2.0	USB2.0 模块
	写入模块硬件板	实现程序写入与调试	
软件	通用 IDE		具有工程管理、代码编辑、编译和出错处理等功能
	TBDML 通信接口		通过调用 TBDML 动态链接库函数,实现 PC 机与目标机的通信
	写入模块		支持代码的下载
	调试模块		支持代码的动态调试
	实验例程		MC9S12DG128 芯片各个模块的实验例程及文档
	μC/OS-Ⅱ移植		基于 CPU12 微处理器的 μC/OS-Ⅱ移植

2.2 SdIDE12 的硬件环境设计思路

SdIDE12 的硬件环境主要由宿主机、目标机和通信接口三部分组成。宿主机是指执行编译、链接、定址过程的计算机，编译过程由交叉编译器实现。所谓交叉编译器就是运行在一个计算机平台上并为另一个平台产生代码的编译器。编译过程产生的所有目标文件被链接成一个目标文件，称为链接过程。定址过程会把物理存储器地址指定给目标文件的每个相对偏移处。通过这三个过程便可将应用程序转换成可以在目标机上运行的二进制代码。本文选用普通的桌面 PC 机作为宿主机。目标机指运行嵌入式软件的硬件平台，一般而言，目标机可以是通用机，也可以是单片机，本开发系统是以 HCS12 系列 MCU 为目标机。通信接口是宿主机和目标机之间的桥梁，用于程序下载和调试。例如，对于 HCS12 系列、ColdFire 系列 MCU，其通信接口可选用 TBDML。对于 ARM 系列 MCU，可选用 JTAG 为通信接口，而 M*Core、C*Core 系列 MCU 可选用串行口作为通信接口。SdIDE12 硬件环境结构如图 1-1 所示。

2.3 软硬件协同设计思路

IDE 传统的设计方法主要是自上向下或模块化设计，但都是将软件和硬件独立设计，使系统中的一些潜在的优势不能发挥，系统资源不能充分利用，重复开发率高，调试困难。而软硬件协同设计[6][7]可以克服传统设计方法存在的缺点，有助于软硬件开发平台在设计过程中不断的相互测试，使得在设计开发过程的早期发现问题，并及早解决，从而避免在设计开发的后期反复修改系统以提高系统开发的效率，缩短开发

周期。软硬件协同设计的方法流程图如图 2-1 所示。软硬件开发平台的紧密结合，协同工作，是嵌入式集成开发环境 研制的重点工作。一个高质量的 IDE 开发系统，不仅要功能全面，而且还应该具有通用性、开放性和可靠性等特点，这样的 IDE 系统定会成为倍受用户青睐的交叉编译工具，也是将来嵌入式集成开发系统的发展方向。本文设计的 SdIDE12 集成开发系统就是采用软硬件协同设计的方法，及时发现各个模块中出现的问题，力争缩短开发周期，提高系统稳定性。

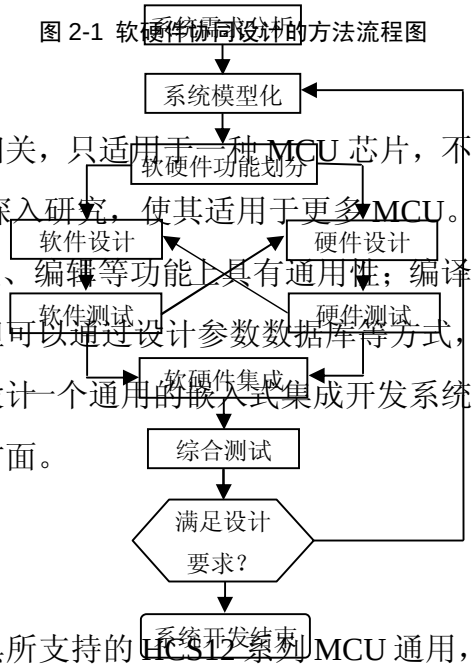


图 2-1 软硬件协同设计的方法流程图

2.4 SdIDE12 的通用性设计思路

一般嵌入式集成开发系统都与具体的单片机相关，只适用于一种 MCU 芯片，不具有通用性。本文在 SdIDE12 的通用性上做了较深入研究，使其适用于更多 MCU。嵌入式开发平台中运行于 PC 方的 IDE 在工程管理、编辑等功能上具有通用性；编译设置、程序下载和调试等功能虽然与硬件相关，但可以通过设计参数数据库等方式，用软件的方法避免其直接与硬件打交道。所以，设计一个通用的嵌入式集成开发系统具有可行性。在设计过程中，重点考虑以下两个方面。

2.4.1 硬件的通用性设计

为了使硬件平台尽可能对 GNU Tools 开发工具所支持的 HCS12 系列 MCU 通用，在设计时，把硬件分为核心板、扩展板和写入模块硬件板三部分。把每种芯片的最小支撑电路单独设计在一块核心板上，把所有的模块及 I/O 引脚引出来，与核心板接口相连，以便把相应的模块电路移到扩展板上。由于同一系列芯片的基本模块和功能引脚数基本相同，只是引脚所处的位置不同而已，所以，每种 MCU 基本模块完全可以移到扩展板上，在扩展板上设计这些模块的支撑电路。为了使扩展板能适用于多种核心板，则在设计核心板的 PCB 时，要把所有的引脚留出来，形成一个与扩展板相对

应的接口插槽，当然这些插槽要和扩展板上模块的相应接口对应起来；对于每种芯片的特殊模块(即别的芯片可能没有的模块)，也可以将其设计在扩展板上，相应的引脚接口连接到单独一排的插槽上。这样就可以实现一块扩展板适用于多个芯片的功能。由于考虑到不同 MCU 及不同模块的工作电压不同，则把单一输入电源设计在扩展板上，输入为 12V，再通过电压转换芯片(如 7809、7805 及 1085)则可输出 9V、5V、3.3V 等不同的电压。

另外，由于大多数嵌入式产品，均需要高端软件和低端软件进行通信，以前大部分采用比较简单的 RS232 串口通信，但随着 USB 技术的高[illegible]在许多设备都用 USB 接口替代串行口作为通信工具，例如，现在许多笔[illegible]没有串行口。因而，设计写入器时，既保留了串行接口，也加上了 USB 接口，进一步的提高了系

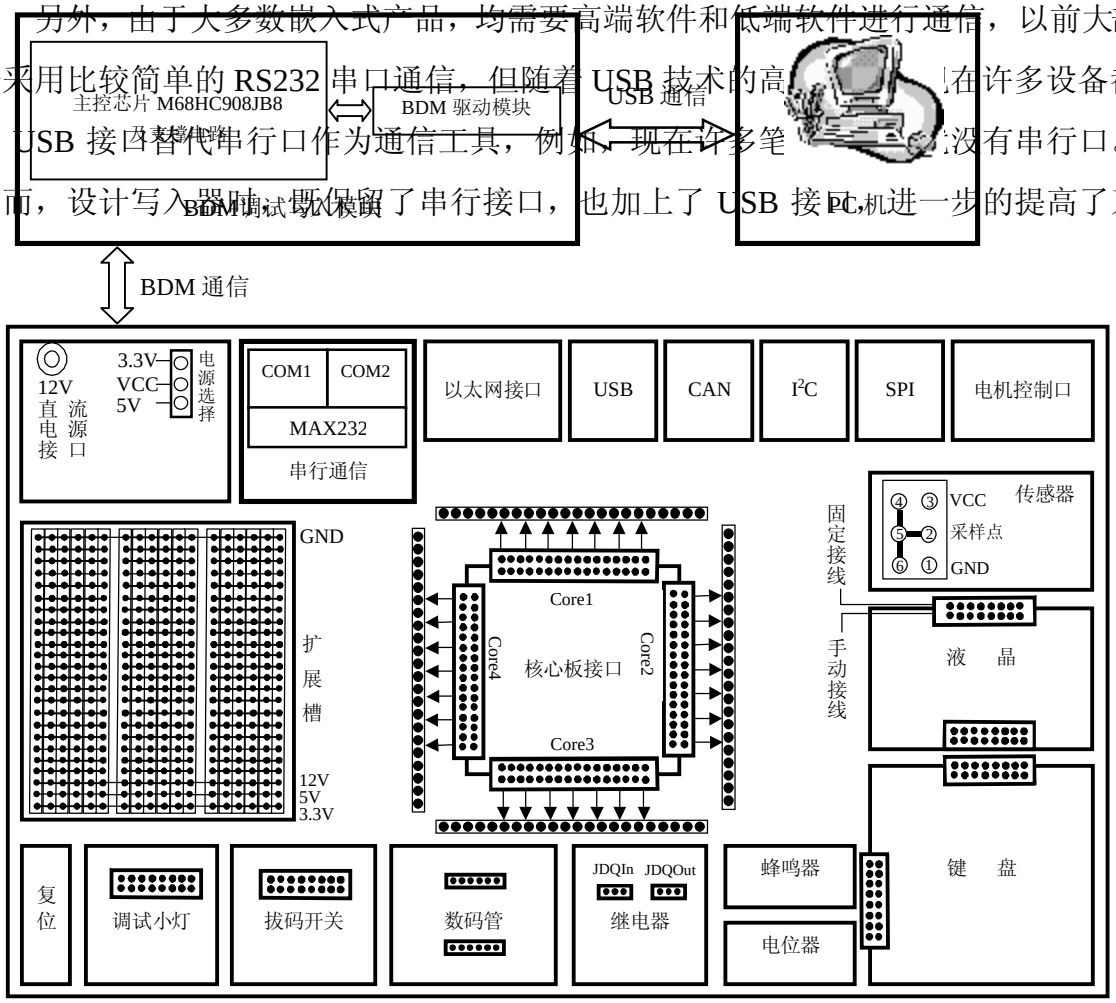


图 2-2 硬件系统框图

统在硬件上的通用性。

通用扩展板包含了 HCS12 系列 MCU 基本模块，主要包括：通用接口模块(4×4 键盘接口、定时器、串行和并行 I/O、AD/DA 等)、人机界面显示模块(LCD、LED 等)、通讯类模块(CAN、I^2C、USB、以太网等)、传感器类模块等，可以在扩展板上进行这些模块的实验开发；写入模块主控芯片是 M68HC908JB8[8]，写入模块硬件设计主要包括：MC68HC908JB8 芯片的支撑电路模块、BDM(Background Debug Model)驱动电路、BDM 调试头电路和 USB 通信电路。目前 SdIDE12 的硬件平台包括 MC9S12DG128[9]、MC9S12NE64[10]、MC9S12UF32[11]和 MC9S12XDP512[12]的最小系统板(即核心板)、外围器件的通用扩展板以及通用写入硬件板；硬件系统的框图如图 2-2 所示，硬件系统实物图可参见附录 B。

硬件平台设计和制作的主要步骤如下[13]：

① 使用 PROTEL DXP/99 软件，设计硬件平台，建立主要元器件库；

② 建立硬件平台元器件原理图符号库、PCB 符号库(主要是芯片和元件封装)；

③ 生成与导入网络表，完成 PCB 的布局和布线；

④ 检查 PCB 布线规则，修改 PCB 布线错误，完成 PCB 版图的设计；

⑤ 完成硬件平台印刷电路板的焊接和测试；

详细的硬件设计在第三章介绍。

2.4.2 软件的通用性设计

为了使该开发平台适用于多种 MCU 型号，在IDE 模块和写入模块的软件设计方面做了大量的研究工作，因为这两方面是使 SdIDE12 具有通用性的关键点。SdIDE12 的通用性设计在软件上主要体现在以下两个方面。

(1) IDE 模块的通用性设计

在 IDE 模块设计方面分为与 MCU 无关的部分(公有模块)和与 MCU 有关的部分(私有模块)，公有模块主要包括编辑、编译出错处理、makefile 文件编写[14]、工程管理、函数及全局变量列表、界面设计等；私有模块主要包括程序写入和代码调试模块；要使这些模块具有通用性就必须对模块的独立性和编译器的适应性进行研究和设计。

SdIDE12 通过建立线程管道的方式调用 GCC 编译器。先把 GCC 所支持的编译器目录放在开发环境的安装目录下，采用线程管道的方法捕获由 make.exe –f 命令对

makefile 脚本文件进行编译的输出信息流，并显示在 SdIDE12 的信息输出窗口中。如果捕获的编译输出信息出现任何错误，包括源文件语法错误和其他错误，编译、链接操作立刻终止，并在输出窗口视图中提示错误，具体的编译信息在日志窗口中给出。若是语法错误，则用户可以通过双击错误提示行，来定位引起错误的源文件行，且用红色背景标识。

为了实现 SdIDE12 的通用，其中一项关键的工作是研究如何编写较通用的 makefile 脚本文件。SdIDE12 的新建工程对话框中设计了一个选择 MCU 平台的选项，编译模块可根据该对话框中所设置的某一系列(如 HCS12、ColdFire、ARM、M*Core 和 C*Core 等)的 MCU 型号来决定调用 GNU 移植过来的相应编译器(如 m68hc11、m68k、arm、mcore 或 ccore 等)。自动更换编译器的功能，主要是通过设计 makefile 脚本文件来实现。makefile 是编程人员和 make 之间的接口， 它可以设定各模块的依赖关系。makefile 的内容大致可分成宏定义区和基于依赖关系的指令区两大部分。宏定义区主要是定义一些变量，可以在 makefile 的任何地方被引用；在 makefile 中所有使用编译器名的地方都用一个变量来代替，当调用一个新的编译器时，只要在变量定义的地方将该编译器名赋给它即可，其他所有引用这个变量的地方不用改变。在建立 makefile 之后，就可以使用 make 命令来完成所需的编译工作。

在 SdIDE12 中更换编译器的设计思路具体如下：

① 把各系列 MCU 平台相关的 GCC 交叉编译工具链放在安装目录下的相应编译器目录中，以便程序调用。暂时不用的编译器，则相应编译器目录可以为空，当用到时，由用户把该编译器的工具集复制过去，这样设计可以减少 SdIDE12 的安装文件的大小。

② 把 Cygwin.dll 复制到 System32 目录下。为了在 Windows 平台上构建 GNU 工具链，则需要安装 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 Bashe 外壳和 GNU 工具[15]，这样就有了建立交叉编译器工具的环境。

③ 在目标平台对话框中选定某种 MCU 型号时，系统就可以确定要调用的编译器，把编译器名赋给一个全局变量 CompilerName。

④ 更换 makefile 脚本文件中的编译器名。主要是把编译器工具集中用到编译器名的地方全部用变量$(CompilerName)来替换。其他的基本上可以不变动。

⑤ 编译参数更改。由于一般编译命令都需要定义一组编译参数选项，而且对于不同系列的 MCU，其编译参数也有差异，所以，采用对话框的方式，对不同系列 MCU 的编译参数进行设置，设置好后形成一组编译参数选项，并被多个规则(或编译器)引用。在软件设计中将这组参数选项自动赋值给一个变量，变量可在 makefile 文件的宏定义区定义，然后把所有引用这组参数选项的地方用这个变量替换。当需要改变参数选项时，只需要在对话框中选择和取消相应选项即可。在SdIDE12 的菜单“工程”→“工程设置”对话框中可以实现对警告、库文件路径和扩展内存等编译参数的设置[16]。

(2) 写入模块的通用性设计

写入模块主要是实现程序下载功能，写入模块的通用性一般只针对某一系列或某一芯片而言的，不可能做到完全通用。因而，在设计写入模块时，针对同一系列的芯片均要编写一个通用的程序下载模块，例如，由于 Freescale HCS12 系列芯片都有 BKGD 引脚，可以用 BDM 方式下载，所以编写了一个对该系列通用的下载模块，并打包成可执行文件，存放在 SdIDE12 的下载目录中，供系统执行下载时调用；另外，C*CORE 和 M*CORE 系列芯片可通过串口以监控的方式下载，ARM 系列芯片可通过 JTAG 下载，这些系列都可以编写一个较通用的下载模块，打包成可执行文件后，放在 SdIDE12 的下载目录中。下面以 Freescale HCS12 系列芯片的程序下载模块设计为例，阐述如何设计针对某个系列 MCU 通用的写入模块。

设计针对 HCS12 系列芯片通用的写入模块，其关键技术在于如何处理 HCS12 系列 MCU 的 FLASH 参数。要对不同芯片的 FLASH 参数进行深入理解、比较和总结之

表 2-2 HCS12 系列芯片擦写操作所需的参数

芯片名	用户数据起始地址	INITRAM 寄存器	标志位首地址	写入文件路径	擦除文件路径
MC9S12DB128B	0x1A00	0x19	0x19FE	.\DownLoad\DB128Bwrite.s19	.\DownLoad\DB128Berase.s19
MC9S12DG128	0x1800	0x19	0x16FE	.\DownLoad\DG128Write.s19	.\DownLoad\DG128Erase.s19
MC9S12DG128B	0x1800	0x19	0x16FE	.\DownLoad\DG128Bwrite.s19	.\DownLoad\DG128Berase.s19
MC9S12DJ128B	0x1800	0x19	0x16FE	.\DownLoad\DJ128Bwrite.s19	.\DownLoad\DJ128Berase.s19
MC9S12NE64	0x3000	0x20	0x2FFE	.\DownLoad\NE64Write.s19	.\DownLoad\NE64Erase.s19
MC9S12UF32	0x1800	0x19	0x16FE	.\DownLoad\UF32Write.s19	.\DownLoad\UF32Erase.s19
MC9S12XDP512	0x2800	0xfd	0x27FE	.\DownLoad\S12Xwrite.s19	.\DownLoad\S12Xerase.s19
…	…	…	…	…	…

后，才能对程序下载模块进行设计。找出这些芯片有差异的 FLASH 参数后，将其存放于数据库中，以便下载程序时，从该数据库中读取相关参数。设计好参数数据库之后，PC 方下载界面可以根据所选择的芯片型号从配置文件和数据库中获取相应芯片擦写操作所需的参数值。表 2-2 列出了 HCS12 系列 MCU 擦写时所需的参数，其值针对不同的 MCU 可能不同。

用户数据起始地址参数：是指一页用户数据存放在 RAM 区的起始地址。将一页用户数据写入相应空白芯片之前，要先将这一页数据通过 BDM 写入到 RAM 区，然后运行写入程序，再将其从 RAM 区写入到 ROM 区。但由于不同芯片的 RAM 区大小和起始地址有可能不同，把一页用户数据写入到 RAM 区时，其起始地址也会有所不同，所以将其设置为可变参数，使用时，从数据库中读取。

INITRAM 寄存器参数：是指 INITRAM 寄存器的地址。由于 HCS12 系列芯片的 RAM 区基本空间是 2K，但根据需要，可以通过设置 INITRAM 寄存器，使不同芯片的 RAM 空间扩展到 4K 或 8K，甚至更大，所以把 RAM 扩展值设置为可变参数，从数据库中提取后，再对 INITRAM 寄存器赋值。

标志位首地址参数：该参数是 RAM 区的一个地址，用于存放擦写成功或出错的标志，占两个字节。每写完一页用户数据之后，PC 机方可从该地址处读出一个字的内容，判断 Flash 写入操作是否成功，只有成功后，才允许写下一页用户数据。

写入文件路径和擦除文件路径参数：由于 HCS12 系列 MCU 中每款芯片的 FLASH 参数和空间有所不同，每款芯片的擦除和写入子程序也有差异，因而要编写每款芯片的 FLASH 擦除和写入子程序，经过编译生成 S19 文件后，存放于 SdIDE12 下载目录中。PC 机方根据当前所选的 MCU 型号以及当前是擦除还是写入操作来调用相应的擦除程序代码文件(*Erase.s19)或写入程序代码文件(*Write.s19)，以实现擦除或写入。

2.5 SdIDE12 的功能模块设计思路

开发通用的集成开发系统 SdIDE12，其难度和工作量都非常大。本文根据任务需求和当前已有的 SdIDE12 系统开发工具，采取以下开发方案：以自由的 GNU 工具软件为基础，参照国内外有代表性的嵌入式 IDE 产品[16][17][18]，搭建 SdIDE12 的基本架构，集成有关开发工具，构建基本的集成开发环境；针对 Freescale HCS12 系列 MCU 的特点，重点进行 SdIDE12 软件开发自动编译管理技术和自动生成程序模板等方面

的研究，实现 SdIDE12 嵌入式集成开发系统的自动编译管理功能，实现多平台(如 M*Core、C*Core、ColdFire 及 ARM 等系列 MCU 平台)的编辑、编译和下载等功能。

本文设计的通用集成开发系统 SdIDE12 是使用 Visual Studio 2005 [19][20]进行开发的，拥有标准的 Win32 界面，具有工程管理、编辑、编译、链接、程序写入与代码调试等功能。图 2-3 为 SdIDE12 的软件功能模块图。

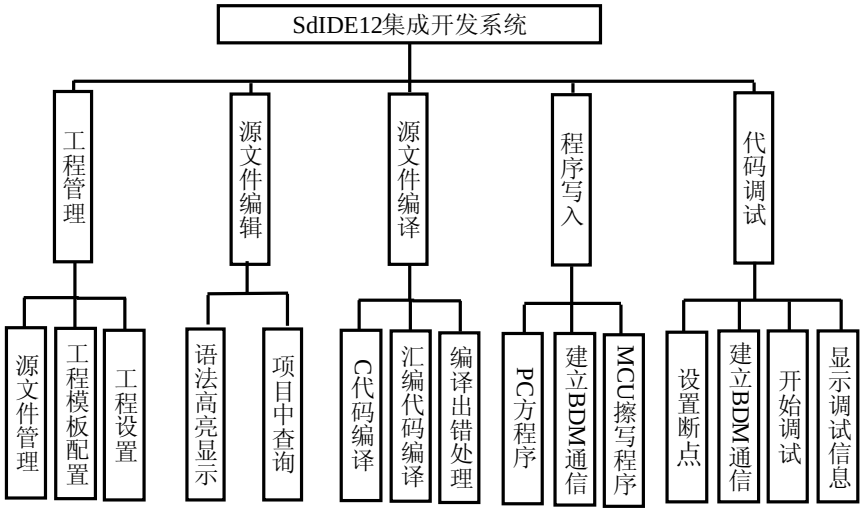


图 2-3 SdIDE12 的软件功能模块图

耦合性与内聚性是模块独立性的两个定性标准，对软件系统划分模块时，尽量做到高内聚，低耦合，提高模块的独立性。因而，为了使集成开发系统 SdIDE12 具有通用性，将其功能模块划分为公有模块和私有模块。各模块的功能及其设计思路如下。

2.5.1 SdIDE12 公有模块

这里公有模块是指与编译器和目标 MCU 无关的模块，完成工程管理、编辑操作。设计时，一定要把它们与整个系统切割开来。

(1) 工程管理模块

在 SdIDE12 中，工程是一个非常重要的概念，它是用户组织一个应用程序的所有源文件(本文使用目标树控件实现文件管理)、显示整个项目中函数及全局变量、对编译出错信息的定位处理，最终生成调试信息文件和目标文件的一个基本结构。提供对源文件、库文件及其他输入文件的管理，实现自动生成模板程序。此外，它还提供编译和链接参数的设置。

(2) 源文件编辑模块

提供源文件编辑，支持剪切、复制、粘贴、显示行号、在文件或整个项目中查找、替换，完全支持中文和语法高亮显示(Syntax Highlight)。此部分有两个主要的技术技巧：一是在整个项目中进行查找和替换操作，二是语法高亮显示，解决半个汉字问题。在整个项目中查找操作采用了创建新线程的方法，提高了响应速度，使用户在正常操作的同时，可以进行查找操作。因为全文查找可能需要较多时间，给系统增加很大的负担，甚至造成假死的现象。使用 MFC 的 RichEdit 控件可以解决半个汉字问题，但没有语法高亮功能，作者在 RichEdit 的基础上编写了语法高亮类，使用效果较好。

2.5.2 SdIDE12 私有模块

私有模块是指与编译器和目标 MCU 相关的模块，完成编译、链接、程序下载与调试等功能。

(1) 交叉编译模块

嵌入式 IDE 是一个支持编辑、编译、链接、下载和调试等功能的交叉集成开发环境[21]，嵌入式 IDE 应具有良好的开放性和可扩展性，这样才有利于其发展。嵌入式集成开发环境需要集成源码编辑器、项目管理器、程序编译器等功能，项目管理对整个开发过程提供支持。交叉开发系统的开发过程一般最先需要构建交叉编译环境，交叉编译环境首先由源码编辑器来实现基本的文本编辑、语法高亮显示、文本内查询，项目中查询等功能；其次，交叉编译环境要有系统配置文件对内核参数以及相关组件参数进行配置，实现工程运行环境的生成与维护；最后，交叉编译环境还要有功能强大的编译器支持，本开发环境使用的是开源软件 GCC 交叉编译器[22]，包括 C、C++ 编译器、汇编器、程序链接器以及文件格式转换工具等，另外，还有 make 程序，以实现编译的自动维护。通过交叉编译，把应用程序转换成可以在目标机上运行的二进制代码。本文开发的交叉编译环境具有通用性。

(2) 程序写入模块

SdIDE12 提供系列 MCU 通过串行接口或 BDM 或 JTAG 等方式进行程序写入与调试[23]。该集成开发平台的程序写入功能是根据所选的 MCU 型号，自动调用相应 MCU 写入模块的打包文件(即可执行文件：.exe)来实现的。例如，若对 AT91RM9200T 微控制器进行程序写入，则系统会调用其相应的写入模块可执行文件 JTAG..exe。下

面以 Freescale HCS12 系列微控制器的程序写入为例，阐述写入模块的功能设计。

HCS12 系列微控制器的程序写入模块包括 S19 文件分析模块、Flash 存储器的擦除模块和 Flash 存储器的写入模块。但采用 BDM 头下载时还需要 TBDML 通信模块，该模块负责通过 USB 接口将 PC 方的 S-record 机器码写入到空白的 Flash 存储器的指定区域。S19 文件分析模块则负责对 S-Record 标准的 S19 文件进行分析，重新组包成 PC 方和 MCU 方事先约定的数据包格式，以便双方的通信和传输。同时，该模块也对文件中的程序起始地址、页数、是否越界等进行判断。程序写入模块先将 Flash 擦写程序代码和用户程序数据写入到指定的 RAM 区，当一页用户程序数据写入到事先分配的 RAM 区后，则可通过调用 tbdml 动态链接库函数来实现擦除和写入。用 BDM 头写入程序的工作流程如图 2-4 所示。

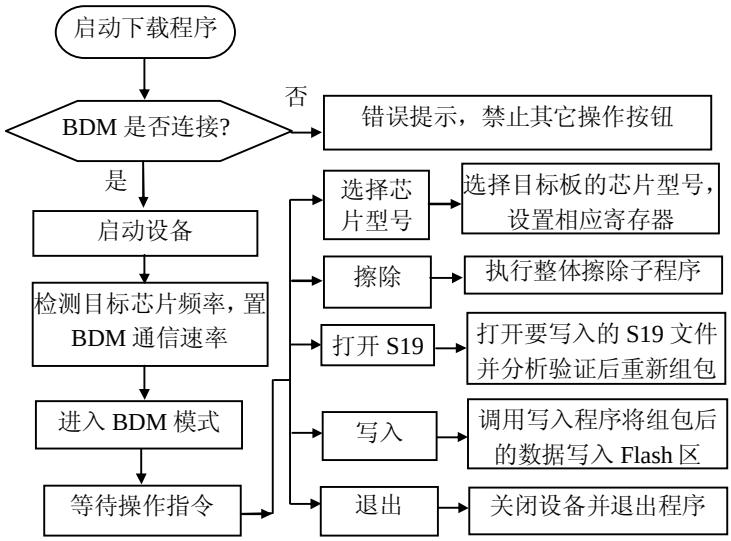


图 2-4 程序写入流程图

(3) 调试模块

与过去的仿真方式不同，SdIDE12 的调试是通过 BDM 头，直接对 MCU 中的程序进行调试。BDM 头除完成 FLASH 写入、擦除功能外，还可以在应用程序运行时，动态地获取 CPU、存储器等瞬态信息。BDM 调试工具与单片机的通信通过双向的 BKGD 引脚实现。下面简要介绍本开发系统所实现的单步调试功能[24]。

断点设置：在调试状态下，用户可以使程序的执行停留在设置了断点的代码行上，系统提供各寄存器的值、程序中定义的变量值及 Flash 中相应地址的值等信息。设置了断点的代码行将以红色高亮显示。

单步调试：设置断点后，用户首先要对调试环境和断点地址进行设置，然后才可以单步执行，实现单步调试。断点地址匹配成功后，当前即将执行的代码行将以红色高亮显示，同时在 SdIDE12 集成开发平台的右侧会出现三个用于调试的信息窗口：Register 窗口中显示的是各寄存器的当前值(各寄存器的值会自动显示)；Watch 窗口中

显示的是变量的值(在变量地址中输入变量名，例如 I1，按回车即可)；在 Memory 窗口中显示的是存储器中的内容(在地址中输入 Flash 地址，例如 0x8000，按回车即可)；如图 2-5 所示。在断点调试状态下，系统提供相关的命令：“设置断点”、“开始调试”、“单步执行”、“退出调试”。

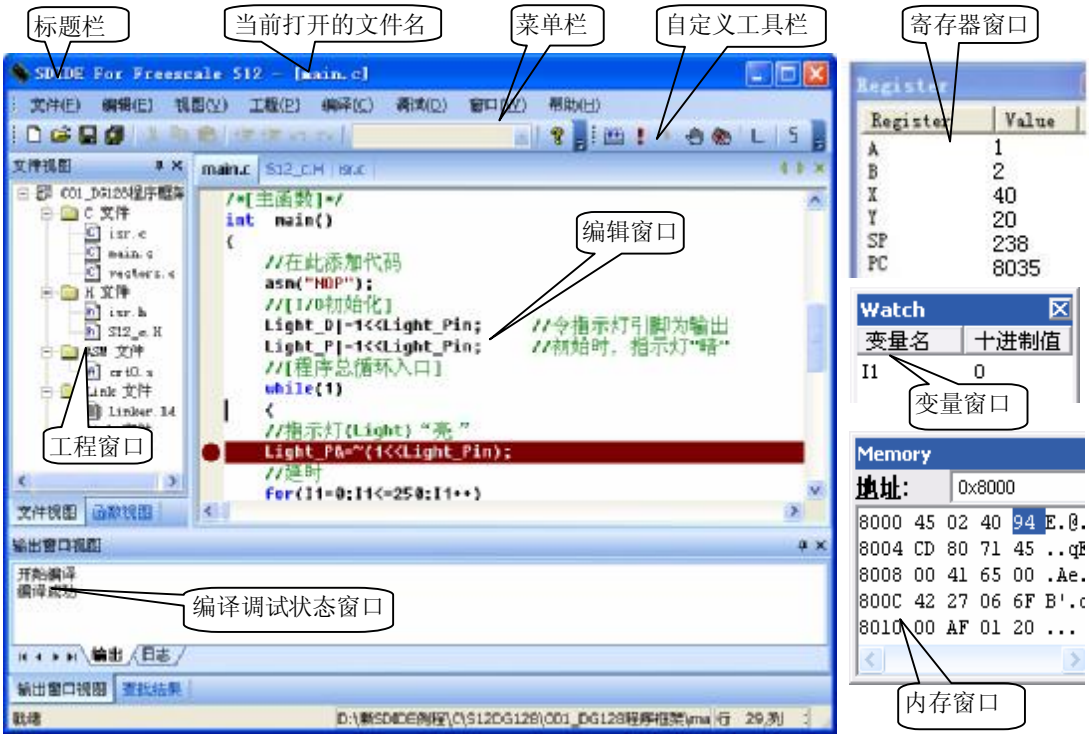


图 2-5 SdIDE12 中的断点的调试状态

2.6 本章小结

本章重点阐述了 SdIDE12 的硬件和软件通用性设计思路。硬件平台采用“核心板＋写入板＋通用扩展板”的积木式结构进行设计，以便于更换核心板，实现硬件平台的通用性。软件平台分割成公共模块和私有模块两部分来设计，公共模块设计了一个通用的交叉编译模块，实现对各个系列 MCU 源文件的编辑和编译，私有模块将各个系列 MCU 的程序下载和调试部分均开发成独立运行的模块，然后内嵌到系统中，以供系统根据所选的 MCU 型号来调用执行。

本章只给出了软、硬件平台的通用性设计思路，明确下一步工作的具体方向和研究重点。以后章节中将按照上述的设计思路，给出各模块的详细设计和具体实现。

第三章 硬件设计

任何一种嵌入式集成开发系统都是通过集成功能强大的软件，实现对特定的硬件对象进行操作。嵌入式集成开发系统是以硬件平台为基础，在硬件平台上进行软件开发。可见，硬件设计在整个嵌入式集成开发系统的设计中扮演着很重要的角色。为了使硬件开发平台对 HCS12 系列 MCU 通用，本章将 SdIDE12 硬件平台分为三部分：最小系统模块、通用扩展板模块和写入模块。下面将详细阐述每个模块硬件电路设计。

3.1 最小系统硬件设计

HCS12 系列 MCU 的硬件结构中仅有一个 MCU(微控制器)是无法工作的，它必须结合其他相应的外围电路(即 MCU 支撑电路)，才能构成一个最小系统。HCS12 系列 MCU 的最小系统一般包括电源电路、时钟电路、复位电路、BDM 调试头电路，MC9S12DG128 芯片最小系统支撑电路示意图如图 3-1 所示。其中各个部分功能如下：

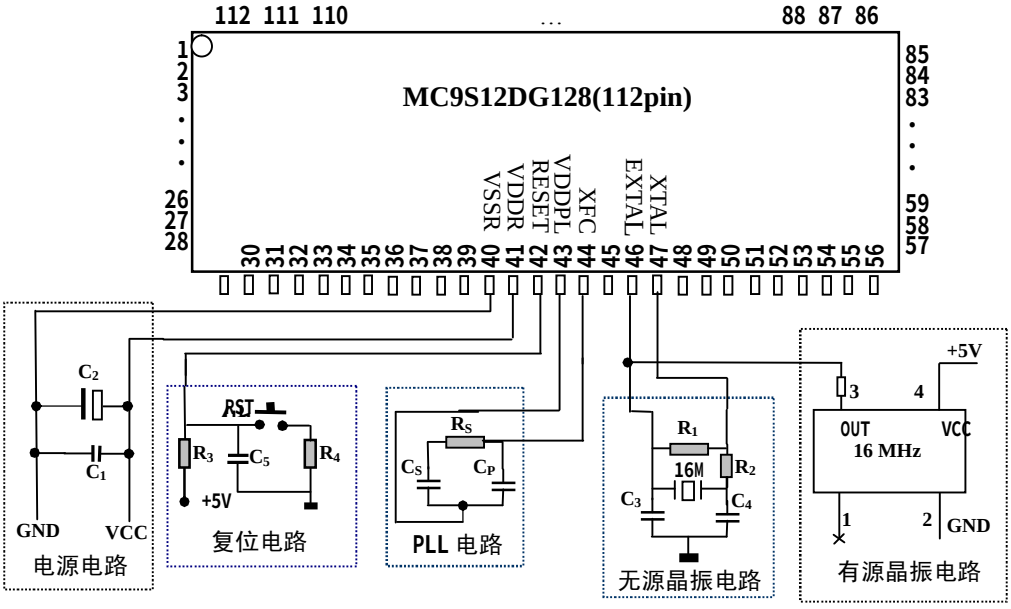


图 3-1 MC9S12DG128 最小系统支撑电路

① 电源电路主要给 MCU 提供＋5V、+12V 和+3.3V 电源。

② 时钟电路给 MCU 提供一个外接的石英晶振。

③ 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。

④ BDM 接口电路主要完成与 BDM 调试工具相连，向 HCS12 单片机写入和调

试程序。

(1) 电源电路

HCS12 系列单片机的芯片内部使用+3V 电压，I/O 端口和外部供电电压为+5V，有些模块甚至需要+12V。在电源的电路设计中，使用电压转换芯片实现单一输入电源，输出几种工作电压。系统电源电路如图 3-1 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 4.7uF 和 0.1uF，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。另外，为了标识系统已通电，设计一个电源灯。

(2) 时钟电路

时钟电路用于向 MCU 及其他电路提供工作时钟。时钟电路的设计非常关键。如果设计过程中出现严重错误，则会造成时钟电路不起振；而时钟电路设计不合理或辅助元器件参数选用不当则会导致造成时钟电路不稳定。因此初次设计时钟电路时，建议使用有源振荡器作为外部时钟源。它不同于常用的无源晶振，主要是在电路接法上有差异。外部有源时钟电路的接法如图 3-1 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单片机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 3-1 中的无源晶振电路所示。其中元器件 C_3 和 C_4 起到滤波作用，电容值均为 22pF；R_1 为 10M 欧姆电阻，R_2 为 0 欧姆电阻。

片内的 PLL 电路兼有频率放大和信号提纯的功能，因此，系统可以以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 3-1 中的 PLL 电路主要起到滤波作用，VDDPLL 引脚由单片内部提供 2.5V 电压。C_S、C_P 和 R_S 的取值与晶振、REFDV 寄存器和 SYNR 寄存器有关，需要通过计算得出。在数据手册上有一个计算的流程图，可以计算出它们的值。当 f_{OCS}=4MHz，总线时钟为 25MHz 时，通过计算得出 C_S、C_P 和 R_S 的值分别为 4.7nF、470pF 和 10K 欧姆。

(3) 复位电路

HCS12 系列 MCU 在响应各种外部故障或侦测到内部系统故障时，系统进行复位。当 MCU 检测到复位信号时，它将寄存器和控制位设置成默认值。系统复位的目的是进行错误恢复，即当 MCU 检测到内部故障时，尝试回到一个已知的、明确的状态。

芯片硬件复位电路如图 3-1 中的复位电路部分所示。正常工作时，由于 RESET 引脚通过 4.7K 上拉电阻 R_3 接到电源正极，所以应为高电平。若按下复位按钮，则 RESET 脚通过 100Ω 电阻 R_4 接地，芯片复位。

(4) BDM 接口电路

BDM 接口用于连接 BDM 调试工具。目前常用的 Freescale 标准的 BDM 调试头如图 3-2 所示，各个引脚信号的定义如表 3-1 所示。由于引脚 VDD 和 GND 在 BDM 调试头两端，操作中容易插反导致目标芯片烧坏。所以作者在对 BDM 调试头进行设计时，做了一些改进。由于标准的 6 芯插头中有两芯线没有使用，实际使用到的只有四根线，所以决定采用如图 3-3 所示的 B 型 USB 连接头来代替 BDM 调试头,这样不仅方便用户使用，而且可以有效的防止反插。

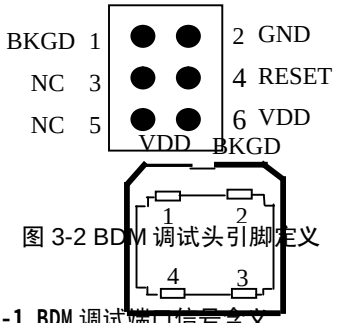


图 3-2 BDM 调试头引脚定义

表 3-1 BDM 调试端口信号含义

BKGD	BKGD 是单线背景调试模式引脚，它用来接收和发送背景调试指令
GND	接地
VDD	电源
RESET	目标机复位引脚

最小系统的核心板原理图如附录 A.1 MC9S12DG128 核心板原理图所示。其他芯片的最小系统原理图与 MC9S12DG128 最小系统原理图相类似。

图 3-3 USB 连接头引脚

3.2 通用扩展板设计

3.2.1 键盘模块

键盘是嵌入式应用中常见的外部设备之一。识别键盘是否有键被按下的方法有查询法、定时扫描法与中断法等。在设计 HCS12 系列 MCU 的核心板之前，一定要先掌握芯片的所有 I/O 口特性，然后，再确定选用哪 8 个 I/O 口作为 4×4 键盘的输入端，并将这 8 个口与通用扩展板的 4×4 键盘接口相对应。因为，作为键盘的 I/O 口，必须具有内部上拉的特性，若用中断法，则该 I/O 口要有中断功能，一旦在设计时没有考虑这些情况，随意将某个 I/O 口作为键盘输入端，则有可能造成键盘模块无法工作。为了避免设计失误或 I/O 口的使用冲突，在扩展板上设计键盘接口电路时，引出 8 个备用的键盘连接口，可以手动连线，随时更改与键盘相连的 I/O 口。具体设计原理图如附录 A.2 中(2)扩展板输入模块原理图中的键盘模块所示。

3.2.2 LCD 模块

通用扩展板上的液晶选用点阵字符型 LCD，该种 LCD 专门用于显示数字、字母、图形符号及少量自定义符号。这类显示器把 LCD 控制器、点阵驱动器、字符存储器、显示体及少量的阻容元件等集成为一个液晶显示模块，其电气特性及接口特性基本规范化和统一化，在硬件上，只要设计出一种型号的接口电路，在指令上稍加修改即可适用各种规格的字符型液晶显示模块。通用扩展板上设计的是日立公司生产的 HD44780 型号[25]的液晶接口电路，其外部接口信号一般有 14 条，也有 16 条的，与 MCU 的接口有 8 条数据线、3 条控制线。见表 3-2。

在设计 MCU 与 LCD 相连的显示数据传送口和控制信号传送口时，要注意 MCU

表 3-2 HD44780 引脚信号

管脚号	符号	电平	方向	引脚含义说明
1	Vss			电源地
2	Vdd			电源(+5V)
3	V0			液晶驱动电源(0～5V)
4	RS	H/L	输入	寄存器选择：1-数据寄存器 0-指令寄存器
5	R/W	H/L	输入	读写操作选择：1-读操作 0-写操作
6	E	H/L H→L	输入	使能信号：R/W=0，E 下降沿有效，R/W=1，E＝1 有效
7～10	DB0～DB3		三态 8	位数据总线的低 4 位，若进行 4 位传送时，此 4 位不用
11～14	DB4～DB7		三态 8	位数据总线的高 4 位，若进行 4 位传送时，只用此 4 位
15～16	E1～E2		输入	上下两行使能信号，只用于一些特殊型号

的 I/O 口的复用特性。在初次设计 MC9S12DG128 核心板时，把其 PTE 口设计成 LCD 显示数据传送口，在编程测试过程中，程序代码总是不能正确写入到目标芯片，通过多方面的调试，一步步的缩小出错的范围，最终确定是硬件设计上的问题，把 PTE 口作为 LCD 显示数据传送口，犯了两个致命错误：首先，PTE0 和 PTE1 口只能作为输入引脚，因而，这两个引脚不能设计成 LCD 的显示数据传送口(输出)；其次，PTE 口是与 BDM 模式选择、时钟选择和中断等复用的引脚，因而，当液晶接上时，则写入软件不能正常工作，同时液晶显示也不稳定。若把 PTB 口作为 LCD 显示数据传送口，则问题迎刃而解。在后期设计中，为了避免硬件设计上的失误或 I/O 口复用冲突，在扩展板的液晶接口电路部分增加了一排引线插孔，其分别与 LCD 的数据传送口和控制信号传送口相连通，用户可以手动插线，随时改变与 MCU 相连的 I/O 口。具体设计原理图如附录 A.2 中(3)扩展板输出模块原理图中液晶模块所示。

3.2.3 串口模块

串行通信接口(Serial Communication Interface，SCI)是 MCU 重要方式之一，所有串行通信接口都具有发送引脚 TxD 和接收引 L 电平引脚。在 MCU 中，若用 RS-232C 总线进行串行通信，则需外接电平转换电路。在发送端需要用驱动电路将 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。目前使用 MAX232 芯片[26]较多，该芯片使用单一+5V 电源供电实现电平转换。图 3-4 给出了 MAX232 的引脚。

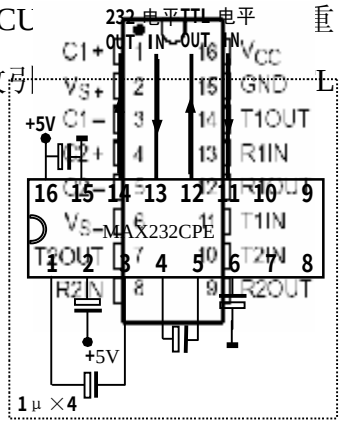


图 3-4 MAX232 引脚

引脚含义简要说明如下：

Vcc(16 脚) ： 正电源端，一般接+5V

GND(15 脚) ： 地

VS+(2 脚) ： VS+=2Vcc-1.5V

VS-(6 脚) ： VS-=-2Vcc-1.5V

C2+、C2-(4、5 脚) ： 一般接 1μF 的电解电容

C1+、C1-(1、3 脚) ： 一般接 1μF 的电解电容

输入输出引脚分两组，基本含义见表 3-3。在实际使用时，若只需要一路 SCI，可以使用其中的任何一组。

图 3-5 SCI 电平转换电路

表 3-3 MAX232 芯片输入输出引脚分类与基本接法

组别	TTL 电平引脚	方向	典型接口	232 电平引脚	方向	典型接口
1	11 12	输入 输出	接 MCU 的 TxD 接 MCU 的 RxD	13 14	输入 输出	通过 232 其它设备相接
2	10 9	输入 输出	同上	8 7	输入 输出	同上

考虑到，串口模块的标准化和统一化，很容易从通用角度进行串口外围电路的设计。SCI 外围硬件电路的主要功能是将 MCU 的发送引脚 TxD 与接收引脚 RxD 的 TTL 电平，通过 RS-232 电平转换芯片转换为 RS-232 电平。图 3-5 给出一个基本 SCI 电平转换电路。具体设计原理图如附录 A.2 中(1) 扩展板外围接口模块原理图中的 SCI 模块所示。

3.2.4 A/D 转换模块

以 MC9S12DG128 单片机为例，其 8 路 10 位 A/D 转换模块引脚为 PAD15/AN15 /ETRIG1～PAD0/AN0，每一个引脚都可以实现模拟量(ANx)/数字量(PADx)的复用输入。为了与外部信号同步进行 A/D 转换，A/D 转换模块有一个外部触发转换通道即 ETRIG，用户可以选择触发方式(沿触发或电平触发)。A/D 转换有个参考电压问题，在使用 A/D 转换器的输入引脚时，芯片的VRH、VRL 引脚分别接参考电压的正、负，通常就是接电源的正、负端。VDDA、VSSA 引脚作为 A/D 转换器的电源提供引脚，分别接电源的正、负端。其电路设计时，不需要特殊的外围电路，只需把相应的引脚引出来即可，具体设计原理图如附录 A.2 中(2)扩展板输入模块原理图中的传感器组模块所示。

3.2.5 PWM 模块

脉宽调制器(Pulse Width Modulator，PWM)是[illegible]一，以 MC9S12DG128 芯片为例，其 PWM 模块有 8[illegible]引脚为 PWM0~PWM7，每个通道都可以单独设置左对齐[illegible]PWM 信号，也叫脉宽调制波。通过指定所需的时钟周期和占[illegible]持续时间。通常定义占空比为信号处于高电平的时间(或[illegible]的百分比。脉冲宽度是指脉冲处于高电平的时间。通用扩展板上设计了利用 PWM 控制电机的模块，其电路设计时，不需要特殊的外围电路，只需把相应的引脚引出来即可，具体设计原理图如附录 A.2 中(3)扩展板输出模块原理图中的 PWM 接口模块所示。

3.2.6 USB 模块

USB 接口已经在无线接入设备、电子商务、PDA 及计算机便携式外设上有了广泛的应用。通用扩展板上设计了 MC9S12UF32 芯片所支持的高速、全速 USB2.0 接口。可以说这款单片机是专门为

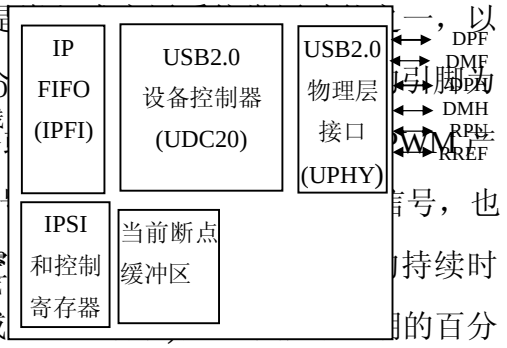


图 3-6 USB2.0 模块框图

高速 USB 存储设备接口设计的。这款芯片的主要模块是集成队列控制器(IQUE)、USB2.0 接口和众多存储控制器接口(如 ATA5)中的一个或多个。CPU 本身起到了对系统的配置和控制作用，IQUE 模块控制一条专门的内部总线，真正的数据传输不需要 CPU 的干预。这条总线使得从存储接口到 USB 模块可以保持每秒 60M 字节的传输率。USB2.0 模块依靠片上 USB 设备控制器 (UDC20) 和 USB 物理层接口 (UPHY)，支持 USB 全速和高速协议。USB2.0 模块的示意框图如图 3-6 所示。U[illegible]中所制定[illegible]接口有两种不同大小的 USB 连接头，如图 3-7 所示[illegible]接头内拥[illegible]其中 2 个用来传递差动数据，另 2 个是给 USB 设备[illegible]4 条内含[illegible]线规以及颜色如表 3-4 所示。其中 D+信号引脚与 MC9S12UF32 的 DPF(USB 全速 D+数据线)或 DPH(USB 高速 D+数据线)相连，D-信号线引脚与 MC9S12UF32 的 DMF(USB 全速 D-数据线)或 DMH(USB 高速 D-数据线)相连。其电路设计时，不需要特殊的外围电路，具体设计原理图如附录 A.2 中(1)扩展板外围接口模块原理图中的 USB 方口和扁口模块所示。

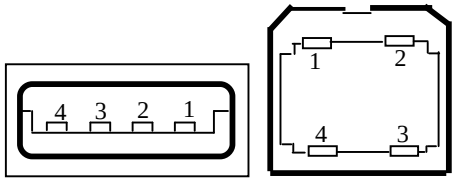


图 3-7 USB 连接头图

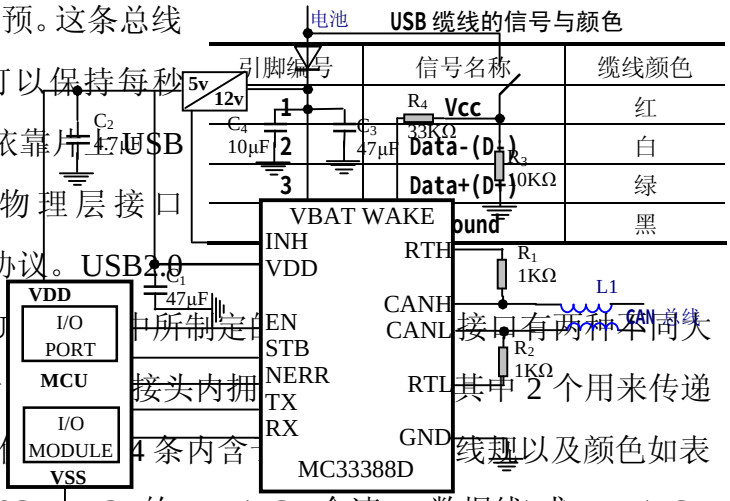


图 3-8 MC33388 与 MCU 连接电路图

3.2.7 CAN 模块

CAN 是控制器局域网络[27](Controller Area Network, CAN)的简称，是由德国 BOSCH 公司开发的。目前已是国际上应用最广泛的现场总线之一。扩展板上的 CAN 通信接口适用于 MSCAN12 模块，可在 HCS12 系列 MCU 上实现。它服从 CAN2.0A/B 协议，集成了除收发器外 CAN 总线控制器的所有功能。CAN 的收

发器种类很多，扩展板使用 Motorola 公司的 MC33388 芯片[28]，MC33388 与 MCU 应用电路见图 3-8。

MC33388 的 INH 引脚接外部调压器；EN、STB、NERR 引脚分别与 MCU 的 I/O 口相连，以实现 MC33388 的工作状态及差错控制；信息接收及发送引脚 TX、RX 与 MCU 的 CAN 通信模块中的 TX、RX 引脚相连，以实现数据传输。CANL、CANH 分别接 CAN 的物理通信线路。R1 与 R2 电阻取值相同，范围为 500 欧姆~16k 欧姆，但该阻值除以结点数应该大于 100 欧姆。

3.2.8 以太网模块

通用扩展板上的以太网接口设计，是以集成以太网 MAC 层和物理层的 16 位单片机 MC9S12NE64 为构架来实现的。该以太网接口可以方便地实现单器件以太网连接，构成一个完整的终端节点。利用 MC9S12NE64 芯片可以构成不同功能的网络终端节点，如网络服务器、带因特网功能的设备、远程监控(数据采集，诊断)、对现场设备的远程控制、远程设备通过电子邮件或文字寻呼机发送消息等。

基于 MC9S12NE64 以太网连接的硬磁隔离模块(即 RJ45 以太网接口)是该 PHY_TXN 为发送线，PHY_RXP 和 PHY 物理端口与隔离变压器连接时必须符合 IEEE802.3 对物理层规范的要求，如 RJ45 的插孔与隔离变压器(即图 3-9 中的 S1 部分)的间隔应尽量小，输出和输入差分信号对的走线要很好的隔离，确保电源的额定负载电流不小于 300mA。

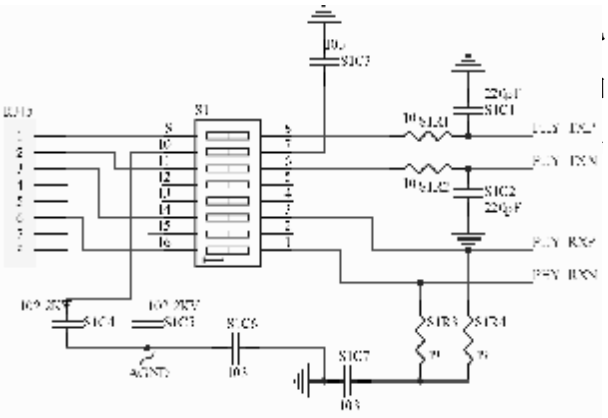


图 3-9 MCU9S12NE64 以太网接口原理图

3.3 写入模块(BDM 头)硬件设计

3.3.1 写入模块硬件结构设计

写入模块采用 M68HC908JB8 为主控芯片，与 PC 方通过 USB 接口以 USB 方式

进行通信，与目标板则通过 BDM 驱动模块以 BDM 方式进行通信，系统由 USB 提供电源。主控芯片 M68HC908JB8 引脚及其功能如表 3-5 所示。

表 3-5 M68HC908JB8 芯片引脚

引脚类型	MCU 引脚	功能
MCU 支撑电路的控制与信号引脚	VSS	接地引脚
	OSC1	外部晶振引脚
	OSC2	外部晶振引脚
	VREG	输出 3.3V 电压
	VDD	电源引脚
	RST	芯片复位引脚
与 PC 机进行 USB 通信的控制与信号引脚	PTE3	差动数据信号引脚 D-
	PTE4	差动数据信号引脚 D+
MCU 与 BDM 驱动模块连接的引脚	PTE1	BDM 输入信号引脚，接收目标板的数据信号，接 74HC125 中的引脚 6
	PTA6	
	PTA7	BDM 输出信号引脚，向目标板发送数据信号，接 74HC125 中的引脚 2
	PTC0	BDM 驱动信号引脚，接 74HC125 中的引脚 1
	PTA4	
	PTA5	目标板复位信号引脚，从目标板收复位信号，接 74HC125 中的引脚 8
	PTA1	目标板复位控制引脚，向目标板发复位信号，接 74HC125 中的引脚 13

3.3.2 写入模块原理图分析与设计

写入模块硬件设计原理图主要包括：MC68HC908JB8 芯片的支撑电路、BDM 驱动电路、BDM 调试头电路和 USB 通信电路。

(1) MC68HC908JB8 芯片的支撑电路模块

MC68HC908JB8 为写入模块的主控芯片，可实现 USB 接口通信。该芯片的支撑电路如图 3-10 所示。

(2) BDM 驱动电路模块

BDM 驱动电路模块主要是通过 74HC125 芯片实现的。74HC125 是一种带控制端的三态缓冲门芯片，它本身具有一定的驱动能力。在写入模块电路设

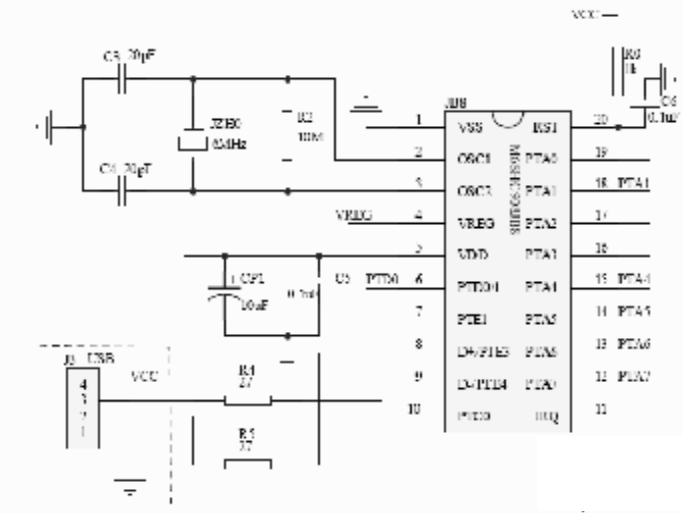


图 3-10 MC68HC908JB8 芯片支撑电路

计中，该芯片主要是 用于控制以 BDM 方式通信 送，它可以在 3.3V~5V 电压范围内低功耗的传输 BDM 信号，其电 。通过 74HC125 芯片电路设计可以实现 BDM 信号的 发控制和复 具体实现过程如下。

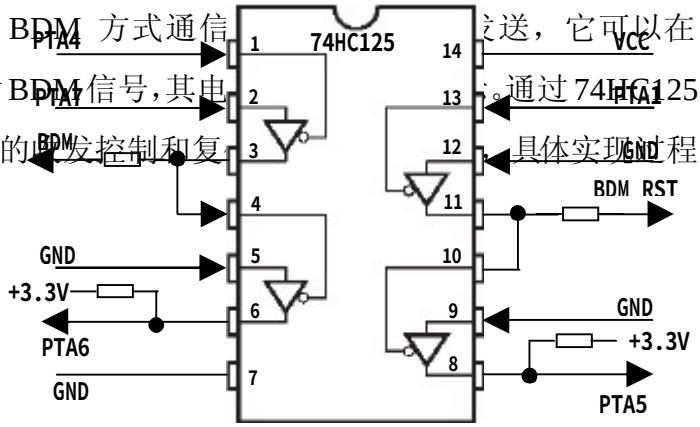


图 3-11 74HC125 芯片驱动电路

① BDM 复位。如图 3-11 所示，74HC125 芯片的右边两路三态门用于驱动 BDM 的复位信号。首先由 PTA1 引脚输入一个低电平信号，使三态门处于工作状态，此时接地的第 12 脚将低电平送至 BDM_RST 引脚输出到目标板，而 PTA5 引脚用于验证 BDM_RST 引脚信号的正确性，如果检测到该引脚输出的也是低电平，则在 PTA1 引脚的控制下将低电平延时一段时间，以产生一个作为对目标芯片进行复位的信号。

② 数据输出。如图 3-11 所示，74HC125 芯片的左边两路三态门用于驱动 BDM 的数据信号。首先由 PTA4 引脚输入一个低电平信号，此时，三态门处于工作状态，则控制引脚 PTA7 将来自 PC 方的输入信号送至 BDM 引脚，并输出到目标芯片，而 PTA6 引脚则用于验证 BDM 引脚信号的正确性。

③ 数据输入。如图 3-11 所示，仍然由 74HC125 芯片的左边两路三态门用于驱动 BDM 的数据信号。此时 PTA4 引脚保持高电平信号，三态门处于高阻状态，PTA7 引脚与 BDM 引脚为截止状态。当目标芯片有数据信号输入时，由 BDM 引脚作为输入，当 BDM 引脚输入信号“0”时，则第 5 引脚与 PTA6 引脚导通，直接输出一个低电平信号；而当 BDM 脚输入信号“1”时，则第 5 引脚与 PTA6 引脚截止，但 PTA6 引脚平时被上拉到 3.3V 的高平，所以直接输出一个高电平信号。

74HC125 中各引脚的连接说明及信号输入输出控制情况见表 3-5 所示。

(3) BDM 调试头模块

HCS12 系列芯片都采用 BDM 方式进行程序写入和调试。用户可以通过 BDM 调试头向目标 MCU 的 Flash 区写入程序，也可以通过它进行应用程序的在线更新，在线动态调试和编程，读取 CPU 各个寄存器的内容，单片机内部资源的配置与修复，

程序的加密处理等操作，而这些仅仅是通过向 CPU 发送几个简单的指令就可以实现，显然这样使得程序写入和调试的软件编写变得相对简单。BDM 调试头接口的设计也不难，关键是要满足通信时序关系和电平转换要求。目前常用的 BDM 调试头可见 3.1 节 CPU 支撑电路中的 BDM 接口电路部分。

(4) USB 通信电路模块

USB 可作为 MCU 与外界进行数据通信的接口，其速度快，抗干扰能力强，在目前的嵌入式产品中被广泛使用。目前大多数写入调试器使用串行接口与 PC 机相连，但该系统使用 USB 接口。电路设计中选用含有 USB 模块的 MC68HC908JB8 作为主控芯片。同时，通过计算与测试，写入模块最大所需电流为 100mA，考虑到一般的 USB 接口具有此供电能力，所以写入模块的电源可以通过 USB 端口取得。MC68HC908JB8 负责 USB 通信的是 PTE3/D+、PTE4/D-两个引脚，当允许 USB 时，它们作为 USB 通信引脚。USB 通信的接口电路如图 3-12 所示。

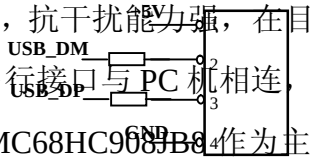


图 3-12 USB 通信电路接口

3.4 硬件平台测试及测试体会

3.4.1 测试方法和步骤

(1) 测试方法

嵌入式产品基本上包括硬件和软件两大部分，嵌入式硬件和软件的集成过程是调试与探索的过程，经常需要软件和硬件并行开发与调试。硬件测试的常用方法如下：

① 利用示波器和万用表检测电路板走线是否正确，芯片引脚是否有虚焊；

② 借用第三方软件(如 Codewarrior 等)，对硬件进行测试；

③ 软硬件协同测试；

(2) 测试步骤

硬件测试的步骤如下：

1) 最小系统板的测试

最小系统板(即核心板)的主要电路模块包括电源模块、时钟模块、复位模块、BDM 接口模块。电源模块可以由 USB 供电，也可以外接电源供电，连接 BDM 头后，通过万用表测量核心板上 B 型连接头的 Vcc 引脚线的电压即可知道供电是否正常；复

位电路可通过万用表测量，复位按钮设计在扩展板上，当核心板与扩展板连接好后，用万用表测量 RESET 引脚在复位按钮按下时是否变为低电平，若是，则复位模块电路正常，否则，可通过万用表逐步确认错误出现在复位电路的哪个环节；时钟模块可通过示波器对晶振引脚进行测量，通过显示的起振周期确认晶振电路是否工作，及起振频率是多少，如果晶振没有起振，则要检测晶振电路设计及连接的正确性。在设计晶振电路时，尽量避免在晶振下方和周围布线，特别是电源和地线，否则，会影响晶振工作的稳定性；BDM 接口，主要是四根信号线：电源线 (Vcc) 、地线(GND)、复位信号(REST)、背景调试(BKGD)，可通过万用表确认这些接口是否分别与 MCU 的对应引脚正确相连。这些模块电路检测均正确后，可以结合芯片进行核心板的综合调试。首先，编写一个小灯闪烁的程序，用 Codewarrior 软件编译后，写入到目标板运行，若发光二极管能够正常闪烁，则最小系统的各个模块电路设计基本上没有问题。

2) 通用扩展板测试

当核心板测试正确后，则可以把核心板与通用扩展板连接起来，进行扩展板上的各个模块电路测试。首先，通过万用表确保扩展板上各个模块的接口与核心板上的相应引脚接口是连通的，也就是说，不能出现虚焊、断线和错连的情况；其次，编写相应模块的基本程序，编译下载后，看运行现象，若现象不正确，则很可能是电路设计上出现了错误，要排除这类错误，一般要仔细查阅芯片手册和有关这些模块比较成熟的参考设计，经过更换电容、电阻，反复调试，最终一定可以调试通过的，关键是要掌握正确的调试方法，逐个排除错误，缩小纠错范围，缩短开发周期。编写这些基本模块的实验程序时，最好借助第三方成熟稳定的软件(如 Codewarrior 等)，这样可以保证在实验程序正确的前提下，进行硬件的相应模块测试。

3.4.2 测试体会

由于硬件设计的经验不足和调试方法的不正确，在第一版硬件设计和测试过程中，遇到许多问题，也获得了宝贵的经验和教训。下面给出作者在整个硬件开发和调试过程中遇到的问题和解决的方法，希望对做类似开发的读者起一定的参考作用。

(1) 核心板、通用扩展板及相应模块接口的设计体会

设计第一版 PCB 时，由于没有考虑到核心板与扩展板之间的通用性，因而，在设计 HCS12 系列 MCU 核心板时，其引脚没有完全与通用扩展板上的接口相对应，

造成部分 HCS12 系列 MCU 的某些模块不能用通用扩展板来测试。更为严重的是，在没有完全掌握 MCU 的 I/O 口特性的情况下，就开始设计核心板，在接口选择上欠考虑，致使设计后的系统出现严重错误。例如，在设计MC9S12DG128 芯片核心板时，与通用扩展板上的液晶(LCD)模块相连的 I/O 引脚，选择了 PTE 口，但在测试液晶模块时，发现程序无法下载，经过仔细查阅手册和反复调试，最终发现问题发生在 PTE 口的特性上，因为 PTE 口是复用口，主要是和 BDM 模式的设定和中断复用，因而，液晶插在 LCD 模块接口上时，写入软件非常不稳定，液晶影响了 MODA、MODB 和内外部晶振选择这些引脚的信号，从而影响了 BDM 写入的稳定性，经常出现写入失败情况，同时在手册中明确注明 PTE0 和 PTE1 只能作为输入口，因而，这两个引脚根本就不可能为 LCD 提供信号。在第二版 PCB 中更改了与 LCD 相连的 I/O 口后，再调试，则一切正常。在设计键盘时，也是犯了类似的错误，主要原因是选择了 PTA 口，作为其输入口，PTA 口没有内部上拉，且也不具有产生中断的功能，因而无法实现键盘扫描和中断功能。在第二版 PCB 中也对其进行了 I/O 口的更换，同时也多留出一排键盘接口，可手动与任意 I/O 口相连。出现以上设计错误的主要原因在于没有透彻理解芯片手册。希望读者在今后的电路设计时，能吸取同样的经验教训。

(2) 核心板晶振电路设计体会

通过多次的设计和调试核心板，发现其最不稳定的模块是时钟电路模块，因为设计过程中有许多因素都可能引起时钟电路的工作不稳定。从经验教训中，总结出单片机系统时钟电路的 PCB 设计需要注意以下几个方面，只有按照这些要求设计，才能使得系统的电磁兼容性得到保证[29]。

① 时钟产生器尽量靠近用到该时钟的器件。

② 尽量让时钟信号回路周围电场趋近于零。用地线将时钟区圈起来，时钟线要尽量短。

③ 石英晶振下面和对噪声特别敏感的器件下面不要布线。

④ 滤波电路要尽量靠近 MCU，单片机的每个电源端和接地端都要接一个去耦电容，去耦电容要尽量靠近 MCU。

3.5 本章小结

本章详细给出了 SdIDE12 硬件平台设计，在设计HCS12 系列 MCU 支撑电路时，

充分考虑了电磁兼容性问题，以确保系统能和谐、正常地工作，力争把电磁干扰降低到最小。通用扩展板和写入模块的硬件设计均是严格按照芯片手册上的要求进行设计，同时也参考了市场上一些成熟产品的设计方案，吸取其设计优点。最终，作者完成了 MC9S12DG128 核心板、通用扩展板和写入硬件板的原理图设计，并对相应的 PCB 进行布局和布线。由于硬件平台开发的工作量较大，MC9S12NE64、MC9S12UF32 和 MC9S12XDP512 芯片的核心板是让师弟和师妹们帮助设计和制作的。在所有硬件 PCB 板上的元器件焊接完成后，对整个硬件平台进行了测试。

本章只给出了硬件平台各个模块的具体设计方案和设计框图，比较详细的硬件电路原理图可参见附录 A。

第四章 软件设计

第三章详细阐述了硬件平台的设计，而对于一套成熟的嵌入式集成开发系统，只有稳定的硬件平台远远不够，还需要与操作便捷、功能强大和运行稳定的软件平台相融合，才能实现整个开发系统的通用性和完备性。在第二章已经介绍了 SdIDE12 的软件通用性设计思路。本章将对 SdIDE12 的通用集成开发环境 IDE、程序写入和代码调试等功能模块的详细设计进行阐述。

4.1 SdIDE12 的通用 IDE 模块详细设计

4.1.1 SdIDE12 主界面结构设计

考虑到工作量等因素，如果采用封闭自主的开发模式，完全自主开发 IDE 的全部功能显然是不合适的。在分析了我们实验室开发的 Freescale HC08 系列 MCU 的开发系统和开源软件(如 GNU 等)以及现有 IDE 开发工具的基础上，本系统使用 BCGControlBar 界面库[30]进行二次开发，BCGControlBar 是一套非常好的 MFC 扩展界面库，可以制作出非常漂亮专业的自定义界面，快速实现 SdIDE12 界面的开发。在 IDE 的设计中，各个模块的功能最终都由用户界面提供给用户使用，而用户界面也可反应出 IDE 的总体结构。如何将系统的功能展现给用户并提高用户友好性是用户界面设计时应考虑的问题。参考有关的 IDE 产品，并考虑用户的使用习惯，最终的 SdIDE12 主界面如图 2-5 所示，其中左方为工程管理窗口，右方为代码编辑，下方为编译输出及调试窗口。具体的体系结构设计如下：

主框架={菜单栏, 工具栏, 树目录视窗, 代码编辑窗, 编译调试信息窗, 状态条}

菜单栏={文件，编辑，视图，工程，编译，调试，窗口，帮助}

文件={{新建，打开，关闭，关闭工程}，{保存，另存，全部保存}，打印，退出}

编辑={撤消，重做，剪切，复制，粘贴，全选，查找，替换，在项目中查找}

视图={函数列表视窗，工具栏，状态栏，外观，LST 文件显示，S19 文件显示}

工程={工程设置，添加新文件，添加已有文件，从工程中移除}

编译={编译，下载}

调试={设置断点，开始调试，单步执行，退出调试}

帮助={SdIDE12 用户手册}

工具栏={新建，打开，新建工程，打开工程，保存，复制，粘贴，编译，下载，设置断点，显示 LST 和 S19 文件}

状态栏={状态信息，当前打开的文件所在路径，当前光标所在的行和列}

输出窗口视图={输出--编译调试结果信息，日志--编译的详细信息}

4.1.2 SdIDE12 的目录结构

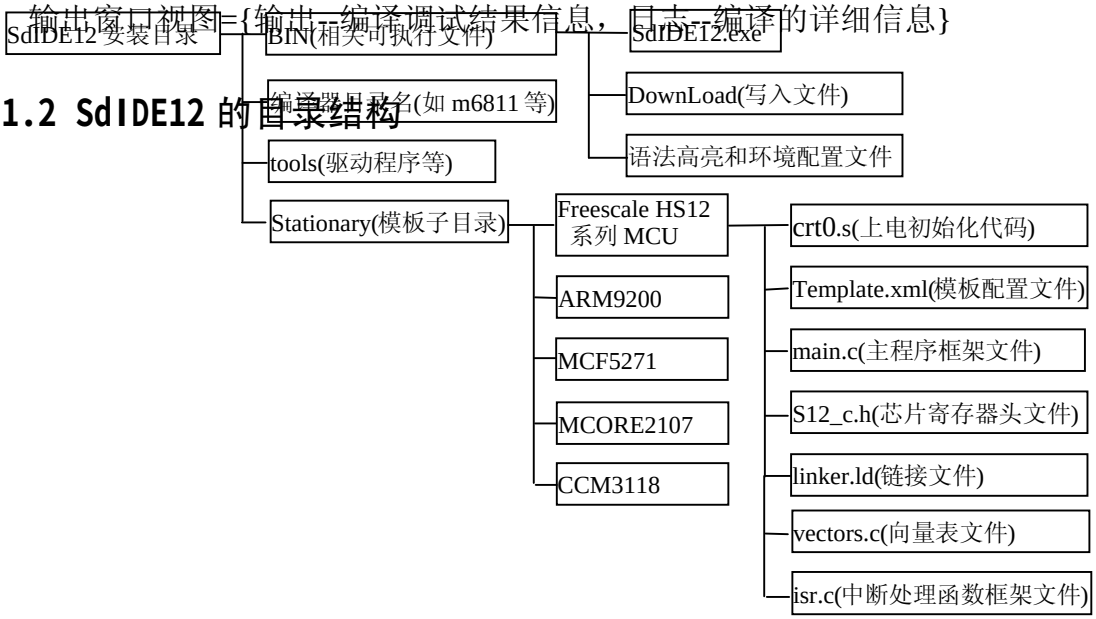


图 4-1 SdIDE12 的目录结构

SdIDE12 的目录结构如图 4-1 所示。

在 SdIDE12 的安装目录的 BIN 目录中有 SdIDE12 的可执行文件、DownLoad 文件夹(存放每一系列芯片 写入模块的可执行文件)、语法高亮和环境 配置文件。Stationary 为程序模板目录，SdIDE12 根据该目录中的系列芯片模板新建一个工程，并自动在该工程中生成相应系列芯片的通用文件及框架文件。以编译器名命名的文件夹中存放的是每一系列芯片的编译器工具集，如 Freescale HCS12 系列、AT91RM9200、MCF5271、MMC2107 和 CCM3118 微控制器分别调用 m6811、arm、m68k、mcore 和 ccore 编译器工具集来实现编译、汇编、链接等操作，并最终生成可执行的二进制代码或 Motorola S_record 代码。

4.1.3 SdIDE12 的代码结构

SdIDE12 嵌入式集成开发系统的功能结构可划分为五个部分：

① 工程管理，实现模板配置、工程设置、源文件组织等功能；

② 代码编辑，实现语法高亮显示、调试行高亮、在项目中查找等功能；

③ 信息输出，实现编译信息输出、错误信息输出、错误行跳转源文件行，函数列表等功能；

④ 编译，后台调用 GCC 编译器编译工程，生成目标代码；

⑤ 写入和调试，实现对目标芯片的 Flash 进行编程和调试；

SdIDE12 的代码按类(Class)的作用可分为 3 个部分：

(1) MFC 框架类

主框架、视图、文档类不属于任何模块，是 MFC 的窗口应用程序必需的类。如表 4-1 所示。

表 4-1 主框架、视图、文档类

类名	功能	隶属模块
CMainFrame	主框架类，**MDI** 程序必需	其它
CChildFrame	子框架类，**MDI** 程序必需	
CIDEDoc	文档类，**MDI** 程序必需	
CIDEView	视图类，**MDI** 程序必需	
CIDEApp	实例类，**Windows** 程序必需	

(2) 界面类

表 4-2 给出了界面类的功能和隶属功能模块。

表 4-2 界面类

	类名	功能	隶属模块
对话框类	**CAboutDlg**	描述程序的开发人员等信息	工程管理
	CAddNewToPro	添加文件到项目中	
	CFindAllWordDlg	在项目中查找对话框	
	CMakeSettingDlg	设置编译参数选项	
	CMakeSettingGeneDlg	设置编译参数选项中的基本选项	
	CMakeSettingWarnDlg	设置编译参数选项中的 **C** 编译参数	
	CNewProDlg	新建工程对话框	
容器类和控件类	**CFileViewBar**	文件视图列表管理	信息输出
	CFindViewBar	查找结果列表管理	
	CFucViewBar	函数视图列表管理	
	COutputViewBar	输出视图窗口列表管理	
	CFileTreeWnd	目录树控件类，管理工程中文件列表	工程管理
	CFindList	全局查找所需的字符	

	CFucListWnd	显示工程中.c 文件中的函数和变量	
	COutputList	记录编译输出信息	信息输出
	CTabList	有两个成员函数，分别记录编译输出的原始信息和处理后的编译输出信息	
	CCrystalEditor	语法高亮显示编辑器	代码编辑
	CCustomEditCtrl	自定义 CBCGPEditCtrl，使得可以画出断点标志等	

(3) 逻辑处理类

逻辑处理类主要包含工程管理、编译、写模块的一些算法，其中 CprojectSettings 是 SdIDE12 的核心类，工程管理由该类实现，如表 4-3 所示。

表 4-3 逻辑处理类

类名	功能	隶属模块
CIDEOptionSettings	保存系统的一些设置信息	工程管理
CProcStruct	函数声明及全局变量的结构	
CProjectSettings	保存当前项目的一些设置信息	
CXmlNode	主要是将工程文件(.IDESln)写入到缓冲区，供其他的类调用，一个 XML 文件节点有一个父节点，n 个子节点	
CRedirect	提供与编译器的链接，通过管道捕获编译器的输出信息	编译
CMakeFile	产生编译所用的 makefile 文件，并编译整个工程	

4.1.4 SdIDE12 的工程管理模块

SdIDE12 集成开发系统的核心部分是工程管理模块，该模块负责一个工程的文件管理、工程模板配置、文件编译和程序写入等操作。

(1) SdIDE12 工程管理的体系结构

SdIDE12 工程管理的体系结构如图 4-2 所示。当用户新建一个工程时，需要从模板目录的配置文件中获取 MCU 的相关参数信息。保存工程时会把有关的信息存放到工程文件中，以便打开工程时读取。

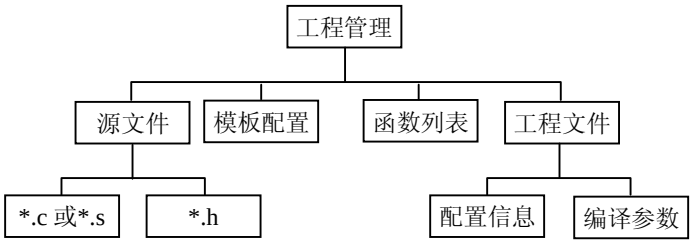


图 4-2 工程管理体系结构图

(2) 工程管理模块的主要子函数

工程管理核心模块为 CProjectSettings，该模块主要包含与工程文件的处理有关的子函数，限于篇幅，此处只给出这些子函数的列表，如表 4-4 所示。

表 4-4 工程管理模块的子函数列表

函数名	参数描述	
CreateProject(CString &Name, Cstring &Path, Cstring &McuType, CString &McuName, BOOL bCProject)	Name：工程名，Path：工程路径，McuTy 编译器名称， McuName：MCU 名称，bCProject：是 C 工程还是 Asm 工程	
OpenProject(CString strPath)	strPath:工程.IDESln 文件的路径	
SaveProject()	无	
CloseProject()	无	
OpenTemplate(Cstring TemplatePath, BOOL bCopy = FALSE)	TemplatePath:工程文件的路径，bCo 若是新建工程，则要把模板中的文件拷到当前目录中，若是打开工程，则不需	
OpenFilesFromTemplate(CXmlNode* pFileXmlNode, BOOL bCopy = FALSE)	PFileXmlNode:工程文件的的根节点，bCopy:若是新建工程，则把模板中文件拷贝到当前目录中，若是打开工程，不需要	根据工程文件打开工程，并保存到变量 m_listFiles 中
BOOL WriteXmlSettings(CXmlNode* pFileXmlNode)	PFileXmlNode:工程文件的的根节点	把生成的工程设置文件保存到工程文件中
GenerateSettingXml()	无	根据工程的设置生成工程文件

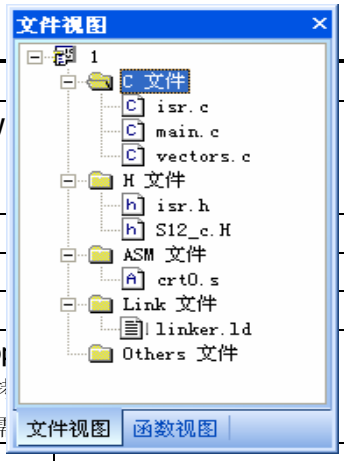


图 4-3 文件管理窗口

(3) 文件管理

工程管理模块提供完善的文件管理器，文件视图如图 4-3 所示。一个项目可包含任意数目的文件并按照源文件(*.s, *.c) 、头文件(*.h)和其它类型文件进行分类管理。在文件视图对话框内点击右键可以弹出相应菜单，对工程中的文件进行新建、添加和删除等操作。当添加文件时，系统可根据文件类型自动将其添加到相同类型的文件序列中,且文件按照字母顺序排列。

(4) 工程模板的结构

SdIDE12 为用户提供了工程模板[31]，其中包含了最基本的处理器初始化代码和每一系列微处理器的通用文件和框架文件，用户可以方便的从模板工程中新建一个新的工程，并自动生成与模板中完全一样的初始化代码、通用文件和框架文件，类似 Visual Studio 中的工程向导。图 4-4 是 SdIDE12 新建工程中的工程模板对话框。列表控件中所列出的是工程模板中相应系列的 MCU 型号。

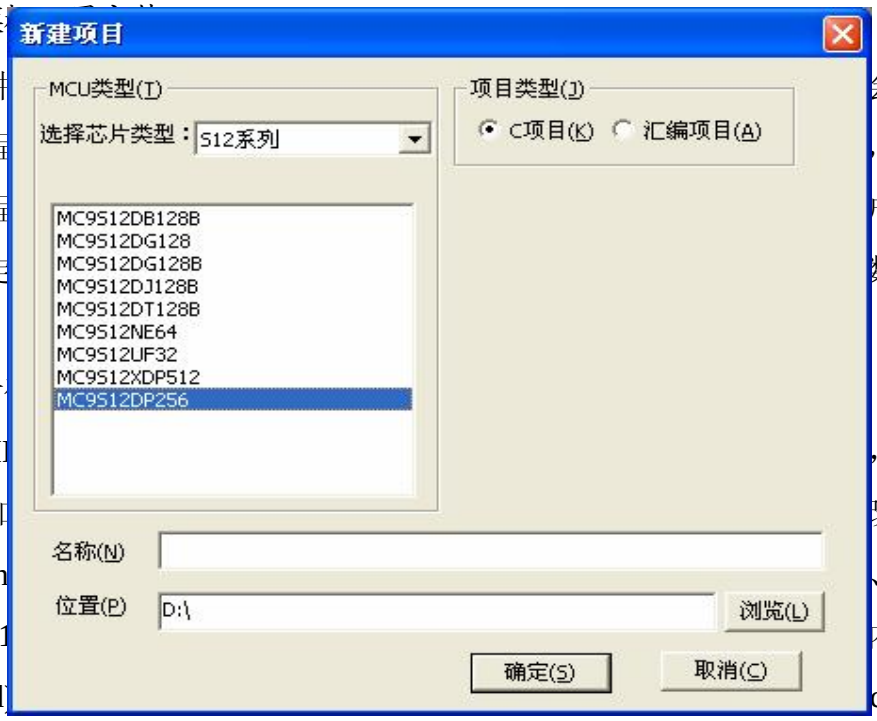
工程模板存放于 SdIDE12 安装目录下的 Stationary 目录中，工程模板向导根据该目录下的子目录名来判断模板名。例如，Freescale HCS12 系列 MCU 的模板名为 M6811，存放于 SdIDE12 安装目录\Stationary\目录下。M6811 目录中存放的是该系列

MCU 的通用源文件、配置文件和框架文件。这些文件在用户新建工程后会自动拷贝

图 4-4 工程对话框

到用户工程目录中。

(5) 模



该文件 会把它拷贝到新的工程 ，这样就生成了该工程 户添加了新文件并确定 数据结构会发生改变。

(6) 添

在 SdI ，但如何向系统中添加 改。

Station 、MMC2107 和 CCM311 内存配置文件(linker.ld ctors.c)、主程序框架文件(main.c)、中断处理程序框架文件(isr.c)和编译链接及芯片基本参数配置文件(Template.config)等。下面以加入 MC9S12DP256 芯片为例，阐述添加一个新型号

的 MCU 平台的主要步骤。

1) 编写配置文件

SdIDE12 工程模板中的每一款 MCU 都有一个相应的芯片配置文件 Template.config，在新建工程时，先在工程对话框中选定 MCU 平台，然后，系统则根据该 MCU 型号的模板目录中的配置文件信息生成一个对工程管理来说比较重要的文件，即扩展名为.IDESln 的工程文件，该文件内容是从 Template.config 文件中复制过来的。Template.config 文件中包含芯片的基本信息(如编译器名、MCU 名称及 C/ASM 工程类型)、环境变量(Cflag、ASFlag 和 LDFlag)、源文件列表。配置文件的内容可分为若干区块，每个区块中有若干字段，这些字段存储了工程的信息，表 4-5 给出了配置文件的区块，表 4-6 列出了所有区块中的字段及其含义。

表 4-5 区块的定义

区　块	含　义
<IDE12>	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

表 4-6 字段的定义

字　段	属于区块	含　义
<Version> …</Version>	<IDE12>	版本号
<McuType>…</McuType>		编译器名
<McuName>…</McuName>		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>		链接器参数
<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>		链接文件列表起止标志
<OtherFile>…</OtherFile>		其他文件列表起止标志

以下为 MC9S12DP256 芯片的配置文件内容示意。

```
<IDE12>
 <Version>"1.0.0"</Version>
 <McuType>"m6811"</McuType>
 <McuName>"MC9S12DP256"</McuName>
 <ProType>"C"</ProType>
 <Flags>
  <CFlag>" -nostartfiles -c -m68hc12"</CFlag>
  <ASFlag>"-m68hc12"</ASFlag>
  <LDFlag>""</LDFlag>
 </Flags>
 <Files>
  <ASMFile>
   <Value>"crt0.s"</Value>
  </ASMFile>
  <CFile>
   <Value>"isr.c"</Value>
```

```
        <Value>"main.c"</Value>
        <Value>"vectors.c"</Value>
      </CFile>
      <HFile>
        <Value>"isr.h"</Value>
        <Value>"S12_c.h"</Value>
      </HFile>
      <LinkFile>
        <Value>"linker.ld"</Value>
      </LinkFile>
      <OtherFile>
      </OtherFile>
    </Files>
  </IDE12>
```

2) 编写芯片的模板文件

每款芯片的模板文件主要是与该微处理器的体系结构有关的通用文件和框架文件，主要包括芯片的程序代码启动文件(crt0.s)、头文件(S12_c.h)、链接文件(Linker.ld)、中断向量表文件(vectors.c)、中断处理函数框架文件(isr.c)和主程序框架文件(main.c)。这些文件的功能和代码编写可参见 6.2 节。

3) 修改 SdIDE12 系统配置文件 option.xml

在 SdIDE12 软件的安装目录\Bin 文件夹中有一个 option.xml 文件，该文件中存储 SdIDE12 软件所支持的系列芯片型号及相应的库和包含文件。现在新加了 m6811 编译器所支持的 MC9S12DP256 芯片，因而要在该文件中相应区加上该信息，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 文件的内容示意如下：

```
<IDE12>
  <Version>"1.0.0"</Version>
  <GlobleOption>
    <m6811>
      <Stationary>
        <Value>"MC9S12DB128B"</Value>
        <Value>"MC9S12DG128"</Value>
        <Value>"MC9S12DG128B"</Value>
        <Value>"MC9S12DJ128B"</Value>
        <Value>"MC9S12DT128B"</Value>
        <Value>"MC9S12NE64"</Value>
        <Value>"MC9S12UF32"</Value>
        <Value>"MC9S12XDP512"</Value>
        <Value>"MC9S12DP256"</Value>
      </Stationary>
      <IncPath>
        <Value>"F:\PROGRA~1\SdIDE\SdIDE12~1\m6811\m6811-elf\include"</Value>
      </IncPath>
      <LibPath>
```

```
            <Value>"F:\PROGRA~1\SdIDE\SdIDE12~1\m6811\m6811-elf\lib"</Value>
        </LibPath>
    </m6811>
…
</GlobleOption>
</IDE12>
```

4) 创建 MC9S12DP256 文件夹

在安装目录\Stationary\HCS12\C 目录下创建一个文件夹，文件夹以芯片名命名。在创建一个名为 MC9S12DP256 的文件夹后，把上述编写的文件全部放在 MC9S12DP256 文件夹中。再打开新建工程对话框，选择芯片类型为 S12 系列，则会发现 S12 系列芯片中新增了一个 MC9S12DP256 选项，如图 4-4 所示。当设定工程文件名、工程路径和工程类型后，单击“确定”，则一个 MC9S12DP256 芯片的新工程在指定的路径下生成，工程文件夹中包含前面创建的通用文件及框架文件，用户可在此基础上继续对该工程进一步编程。

4.2 通用 HCS12 系列 MCU 写入模块详细设计

写入模块的主要功能是实现程序下载，在第二章已经阐述过写入模块的通用性设计思路，一般针对某一系列芯片编写一个通用的程序写入模块，PC 方可通过选定的 MCU 型号，决定调用相应写入模块的可执行文件。限于篇幅，本节只阐述通用 Freescale HCS12 系列 MCU 写入模块的软件设计。

HCS12 系列芯片的程序写入模块的软件设计主要包括：通信程序接口设计、PC 方写入程序设计和 MCU 方擦写程序设计。程序写入的主要过程如下：

① 编写各款芯片 MCU 方的擦除和写入子程序，并编译生成 S 格式机器码；

② PC 方先把 MCU 方擦写程序的机器代码发送到目标 MCU 的指定 RAM 区；

③ PC 方把编译生成的用户程序机器代码分页发送到目标 MCU 的指定 RAM 区；

④ PC 方调用 BDM 动态链接库的相应函数执行发送到 RAM 区的擦写程序，把用户程序写入到 Flash 中去；

4.2.1 通信程序接口设计

写入模块以 USB 方式与 PC 机通信，以 BDM 方式与目标板通信。通信过程要先约定通信的数据包格式，然后根据约定的数据包格式，由通信程序调用 Freescale 公

司提供的 TBDML 动态链接库函数[32]来实现数据传输和程序的执行。下面给出一页用户数据包在写入过程中需要约定的通信参数和 BDM 动态链接库中主要包含的几类函数。

(1) 约定程序写入的通信参数

要实现以上功能，则 PC 方和 MCU 方进行一页用户数据包传输时，要约定其通信接口参数如下：

① 擦写程序代码存放在 RAM 区的起始地址，在编译擦写程序时，由 linker.ld 文件指定编译后代码存放的区域；

② 一页用户程序代码放在 RAM 区的起始地址。该参数可设定在表 2-2 所示的参数数据库中；

③ 一页用户程序代码的长度；

④ 一页用户程序代码的目标 FLASH 起始地址；

⑤ 擦除或写入成功与否的标志信息存放在 RAM 区的起始地址，占两个字节。该参数可设定在表 2-2 所示的参数数据库中；

③和④这两个参数是与每一页用户数据有关，存放在每一页数据包的前 5 个字节中，前三个字节是这一页用户数据的起始线性地址，后两个字节是这一页数据的长度。

②和⑤这两个地址参数值可以固定，MCU 方可直接到这两个地址处存取数据。

由此可见，只要 PC 机方和 MCU 方知道以上参数，就可以实现程序写入过程的通信。但一定要从总体上对可用的 RAM 区进行划分和安排，确保不会相互重叠，造成约定的数据丢失。例如，MC9S12DG128 芯片的 RAM 区程序空间基本上是按照图 4-5 所示的布局分配的。

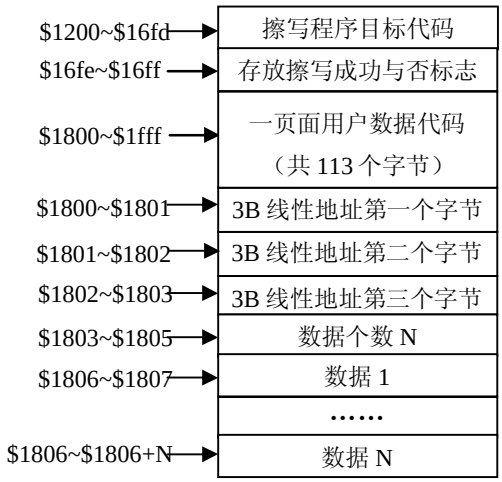


图 4-5 目标 MCU RAM 分配图

(2) 动态链接库函数列表

PC 方用户程序与底层 MCU 方程序进行通信时，其通信程序直接使用动态链接库文件 TBDML.DLL。TBDML 动态链接库的主要函数如表 4-7 所示。

表 4-7 tbdml 动态链接库函数列表

函数名	参数描述	功能描述
tbdml_init(void)	无	检测当前连接的设备数量
tbdml_open(unsigned char device_no)	设备号	启动 S12 写入设备
tbdml_target_sync(void)	无	自动检测目标芯片频率，并设置 BDM 通信速率
tbdml_get_speed(void)	无	取得当前目标芯片的晶振频率
tbdml_set_speed(float crystal_frequency)	晶振频率(MHz)	置当前目标芯片的晶振频率
tbdml_target_reset(target_mode)	=1 表示复位到用户模式，=0 表示复位到 BDM 模式	目标芯片复位后，进入 BDM 或用户模式
tbdml_write_reg_pc(unsigned int ddress)	PC 指针要指向的地址	设置当前 PC 指针的值
tbdml_target_go(void)	无	转向当前 PC 指针所指地址处开始执行程序
tbdml_read_byte(unsigned int address)	需要读取的地址	从目标芯片相应 RAM 区地址处读取一个字节
tbdml_write_byte(unsigned int address, unsigned char data)	address：需要写入的地址，data：需要写入的数据	向目标芯片 RAM 区地址处写一个字节
tbdml_read_bd(unsigned int address)	BDM 相应寄存器的地址	读 BDM 相应寄存器的值
tbdml_read_regs(registets_t * registets)	指向 MCU 寄存器的指针	读取 MCU 相应寄存器的值,如：PC、SP、D、X、Y、CCR
tbdml_target_step()	无	步过一条指令
tbdml_read_word (unsigned int address)	需要读取的地址	从目标芯片相应 RAM 区地址处读取一个字
tbdml_write_word(unsigned int address, unsigned int data)	address：需要写入的地址，data：需要写入的数据	向目标芯片 RAM 区地址处写一个字

(3) 通信程序接口的实现

写入软件的 PC 方程序通过调用动态链接库函数与 MCU 方通信，以实现对目标芯片的 RAM 区相应地址进行读写操作。首先，要把 TBDML.DLL 文件放在 PC 机系统的\WINDOWS\SYSTEM32 文件夹下，然后在写入软件的 PC 方程序代码中声明所使用的 TBDML 动态链接库函数。

通信程序接口的主要流程：首先需要检测并初始化通信设备，然后启动设备以保证设备可以通过 USB 来进行正常数据通信。在通信设备启动后，可设置目标板晶振，然后复位使目标芯片进入 BDM 模式。至此，设备的初始化过程完成。初始化后，就可以直接调用动态链接库函数实现对目标芯片的读写操作。

PC 方读写指定 RAM 区的一个字节子程序代码如下：

```
//01 初始化设备
initstatus =tbdml_init();          //          初始化 USB 并返回发现的设备号
tbdmlstatus = tbdml_open(0);     //     打开所给的设备，成功：0；失败：1
if(tbdmlstatus== 0)
strMsg.Format("已检测到 TBDML 设备!");
```

```
else
  strMsg.Format(" 未检测到 TBDML 设备,请复位或者重新连接设备!");
//02 检测目标频率，进入 BDM 模式
tbdml_set_target_type(0);       //       设置目标 MCU 类型：0：HS12,1：HS08；
z=tbdml_set_speed(9.83);        //       设置目标晶振为 9.83MHz
if(targetsync== 0)
  strMsg.Format(" 芯片频率检测成功");
x=tbdml_target_reset(0);        //       进入 BDM 模式
//03 读取目标芯片 Flash_address 地址处的一个字节数据,返回给变量 val
val = tbdml_read_byte(Flash_address);
'04 向目标芯片的 Flash_address 地址处写入一个字节数据(data)
tbdml_write_byte(Flash_address, data);
```

4.2.2 通用写入模块 PC 方程序设计

写入软件启动界面如图 4-6 所示，功能流程见图 2-4 所示。

图 4-6 写入软件的启动界面

写入软件的 PC 程序启动后，首先进行初始化操作：检测设备是否已经连接，如果没有连接，系统会返回错误信息，则用户在图 4-6 所示的操作区域中单击“复位到程序下载模式”按钮，可重新连接设备。程序如果检测到设备后则启动设备，检测并设定目标芯片工作频率，然后进入 BDM 模式。在初始化过程完成后，系统开始等待用户的操作指令，接收到相应的指令后执行相应模块的功能。下面详细阐述这些操作模块的功能及其实现。

(1) 选择芯片型号和设置目标晶振

将代码写入到指定 Flash 区之前，一定要设置芯片的 Flash 参数并保证该区域没有数据，因而写入软件启动面上的操作区有“选择芯片型号”和“设置目标晶振”按

钮，其主要功能是初次对该 MCU 进行擦写时，将其一些环境参数从表 2-2 所示的数据库中读出来，并写入注册表，目的是下次再对该芯片擦写时，不需要再通过这两个按钮来设置这些参数，由程序自动到注册表中去读取。但是，若出现写入失败时，需要再重新执行这两步操作。单击“选择芯片型号”按钮后，会弹出选择芯片型号的设置界面，界面中列表显示出 HCS12 系列 MCU 的型号，这些型号是目前系统可实现对其进行擦写的，后面还会陆续加入其他的型号。然后选定与目标板芯片相同的型号后，单击“确定”即可。单击“设置目标晶振” 按钮后，弹出设置目标板晶振的界面，在文本框中输入目标板上的晶振频率，单击“确定”，则目标板将以该频率与写入模块进行通信。但是，当核心板更换后，则对新的芯片进行首次擦写时，都要执行这两步操作。因为注册表中存储的芯片信息还是原来的，并没有更换。

(2) 擦除

PC 机方擦除程序的流程图如图 4-7 所示。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重新组成若干个数据包，再将这些数据包逐一写入到目标芯片的 RAM 区，然后通过调用 tbdml_write_reg_pc(addr)和 tbdml_target_ go()两个动态链接库函数实现转向擦除程序代码所在的 RAM 区，执行该擦除程序，实现对目标FLASH 区的擦除。其中 tbdml_write_reg_pc(addr)函数中的 addr 参数是擦除程序代码在 RAM 区的起始地址。

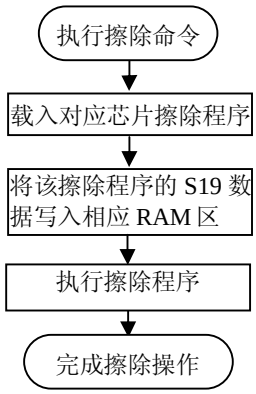


图 4-7 擦除操作程序流程图

(3) 打开 S19 文件

执行该操作，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，每读入一行时，都要验证该行的校验和是否正确，数组中每个元素存放 S19 的一行数据。然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 字节的数据前附加 5 个字节数据，前3 个字节用来存放一页用户数据写入目标 FLASH 区的起始线性地址。后 2 个字节用于存放一页用户数据长度，其值为 64。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，一页用户数据包的总长度为 133 字节。

(4) 写入

PC 方写入程序流程图如图 4-8 所示。在执行写入命令后，程序首先获取组装好

的一页用户数据包，将数据写入到 RAM 区相应区域，然后，判断写入程序代码是否已经写入到 RAM 区，若没有，则要将对应芯片的写入程序代码写入 RAM 区。通过调用动态链接库函数 tbdml_write_reg_pc(addr)设置当前的 PC 指针指向存放在 RAM 区的写入程序代码的起始地址，再调用 tbdml_target_go()函数来执行该写入程序代码。该写入程序的功能就是将存放在 RAM 区的一页用户数据写入到指定的 ROM 区。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 区的一个固定的地址处，占两个字节。PC 方根据读取的标志值，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，则可以接着进行下一页用户数据的写入操作，直到最后一页。

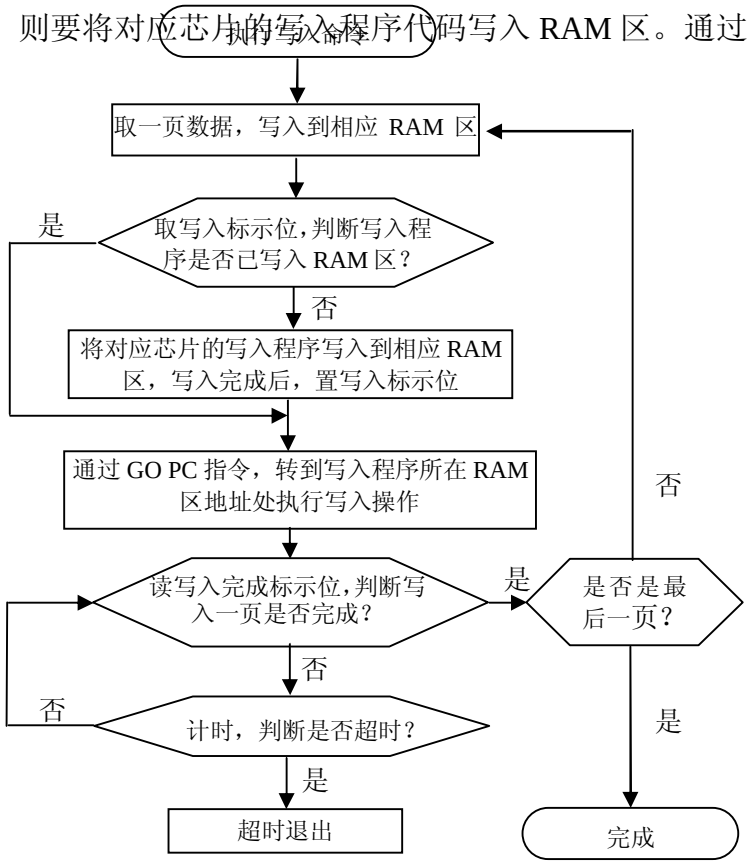


图 4-8 写入操作程序流程图

4.2.3 通用写入模块 MCU 方程序设计

MCU 方的擦写程序用于完成对目标芯片 Flash 的擦除和写入操作。由于在 Flash 的擦除和写入过程中，Flash 是不能读的，故擦除和写入 Flash 的程序要放在 RAM 中，也就是说，在 Flash 的擦除和写入前，要把擦除或写入的可执行代码复制到 RAM 中去，并让程序在 RAM 中执行，这一部分操作是由 PC 方程序完成的。

对于 HCS12 系列 MCU 而言，其 Flash 的擦写操作流程基本相同，只是每款芯片的 Flash 空间和寄存器地址会有差异。对于有差异的部分，可以把它设置成可变的参数，放在数据库中，供 PC 方和 MCU 方的程序存取，从而实现写入模块对 HCS12 系

列 MCU 通用。部分有差异的参数设置如表 2-2 所示。但需要编写每款芯片的擦除和写入子程序，把其编译生成 S19 文件后，放在 SdIDE12 系统的安装目录中，供 PC 方执行 FLASH 擦除和写入操作时调用。Flash 的擦除和写入操作应按以下步骤进行：

① 清除 Flash 状态寄存器 FSTAT中的标志位 ACCERR 和 PVIOL。目的是清除上一次 Flash 操作中出错的标志，方法是向状态寄存器 FSTAT的这两位写零。

② 写 Flash 配置寄存器 FCNFG 的 b1 和 b0 位。以 MC9S12DP256 为例，这两位表示选择 256KB 的 Flash 中的哪一个 64KB。

③ 写 PPAGE 寄存器，即选择擦写的页面。

④ 检查上一次 Flash 处理命令是否执行完毕。判断 Flash 状态寄存器 FSTAT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，要等待，直到可以使用为止。

⑤ 将要写入的数据字写到相应的地址中，地址必须为偶数地址。

⑥ 向 FCMD 命令寄存器写命令字。如 0x41 表示整体擦除，0x20 表示写一个字。

⑦ 向 Flash 状态寄存器 FSTAT中的命令缓冲区的标志位 CBEIF 写 1 清零。这时状态寄存器中的 CCIF 位将置位，说明操作成功。

(1) MCU 方擦除程序

擦除程序的主要步骤：首先调用 Flash 初始化子程序对部分 Flash 寄存器初始化，然后调用整体擦除子程序对 Flash 区进行擦除，最后调用写入一个字子程序，向 0xFF0E 地址写入保密字。对于HCS12 系列 MCU，其整体擦除子程序的流程基本相同，只需要修改主程序中有关寄存器定义的头文件即可。图 4-9 给出了整体擦除 Flash 的子程序流程图，MCU 方擦除主程序包含的子函数头说明如下：

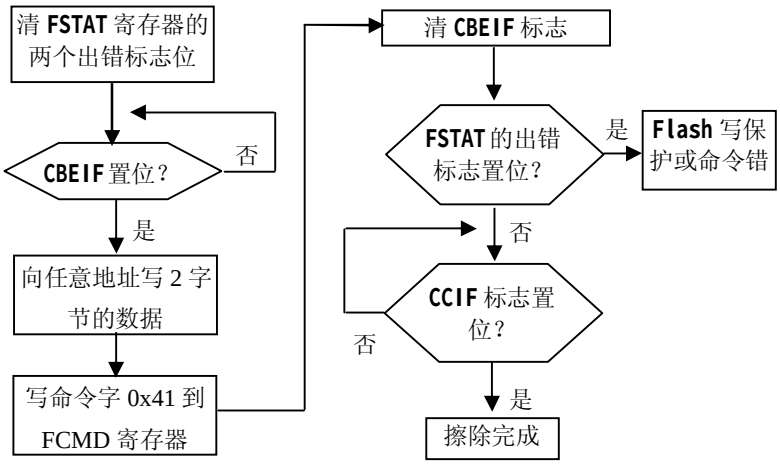


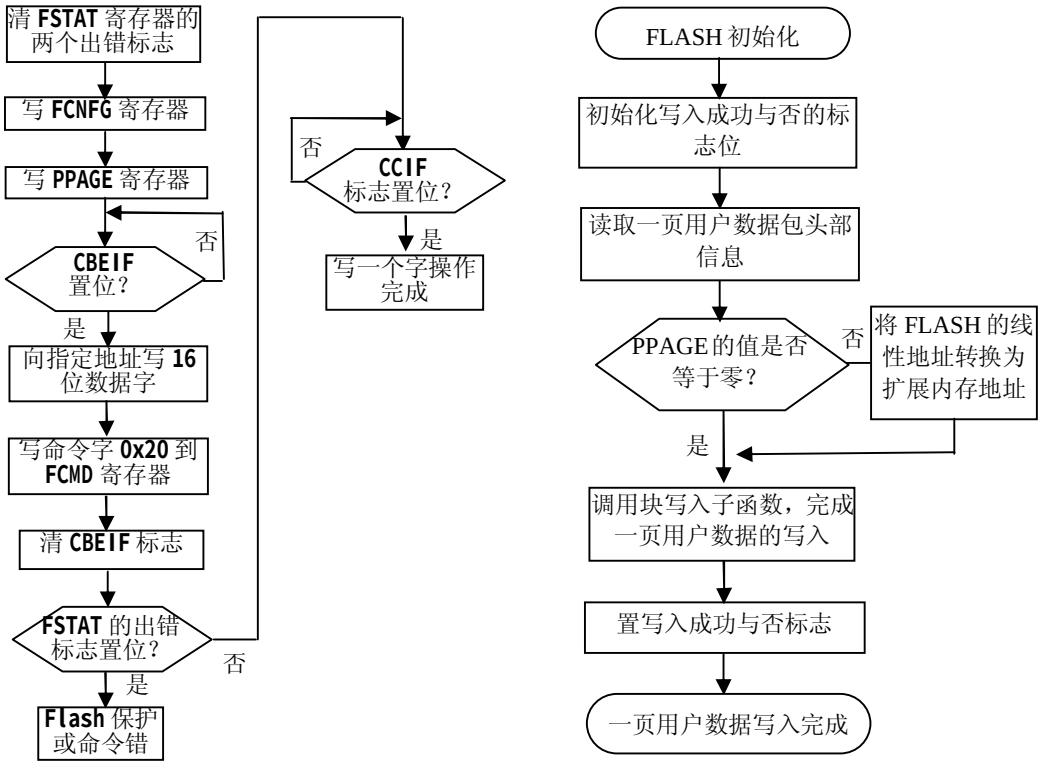
图 4-9 整体擦除操作的程序流程图

```
//FLASH 初化子函数*************************************************
函数：Flash_Init(void)
作用：清 FSTAT寄存器出错标志位，禁止 FPROT 的 Flash 保护
参数：无
返回：无
*****************************************************************************/

//FLASH 整体擦除子函数*********************************************
函数：Flash_Erase_MASS(unsigned int addr,unsigned int *pflag)
作用：整体擦除 FLASH
参数：addr 任意 FLASH 地址, *pflag：指向擦除成功与否标志的内存地址
返回：=0：整体擦除失败；=1：整体擦除成功
*****************************************************************************/
```

(2) MCU 方写入程序

写入程序的主要操作步骤如下：调用 Flash 初始化子程序对部分 Flash 寄存器进行初始化；写入成功与否的标志位初始化；读取一页用户数据的头部信息，即分别读出 PPAGE 寄存器的分页值、线性起始地址、一页的数据的长度；若用到扩展内存，则要将线性地址转换为扩展内存地址，否则不需转换；调用块写入子程序；设置写入成功与否的标志位。图 4-10 为写入一个字的子程序流程图和写入程序的主流程图。



写一个字子程序流程图　　写入程序主流程图

图 4-10 写入程序的流程图

MCU 方写入程序包含的子函数头说明如下：

```
//FLASH 写一个字子函数************************************************
函数：Flash_Write_Word(unsigned int addr, unsigned int data)
作用：向指定的 Flash 地址写入一个字
参数：指定的 Flash 地址 addr，需要写入的两个字节的数据 data
返回：写入成功与否的标志
*********************************************************************/

//FLASH 块写入子函数************************************************
函数：Flash_Write_Block(unsigned int address_source,unsigned int address_destination,
unsigned int count)
作用：将 address_source 指针所指向的地址中的数据内容写入到 address_destination
      的目标地址中，调用 COUNT 次 Flash_Write_Word子函数，写入 2*COUNT
      个字节的数据，每调用一次 Flash_Write_Word子函数，则 address_source 和
   address_destination    指针加 2，即地址均向后移 2 个字节的位置
参数：address_source:在 RAM 区中用户代码所在的起始地址, address_destination:要
      写入到 FLASH 中的地址, count:要写入的字数
返回：错误标志或成功标志
内部调用：写一个字子程序
*********************************************************************/
```

4.3 HCS12 系列 MCU 内存扩展的实现

4.3.1 存储空间的扩展

HCS12 系列单片机的寻址空间允许超过 64KB，达到 128KB、256KB 或 512KB，其内存扩展是基于分页来实现的。CPU12 内部有一个存储器页面寄存器 PPAGE，用于选择要写入或擦除的页面，对于 HCS12 系列 MCU 而言，其 PPAGE 是 5 位的寄存器。CPU12 规定每一页的大小固定为 16K 字节，因而 CPU12 的最大寻址空间就是 2^5 ×16KB，为 512KB。

下面以 MC9S12DG128 为例说明 Flash 在 64KB 以上的单片机是如何进行内存扩展的。其中扩展 Flash 中标注的阴影部分，如 0x3A，0x3B 页面中的阴影表示的是可保护的 Flash 区域，具体的保护地址可参见芯片手册。图 4-11 说明了如何将 64KB 寻址空间扩展到 128KB。

MC9S12DG128 的 128KB Flash 先被分成 2 块 64KB 空间：Block0 和 Block1，每块还可以细分为 4 个 16KB 的页。每个存储器页的页面编号为 0x38～0x3F 的某个值，在 64KB 的寻址空间的 0x8000～0xBFFF 这一段开了一个窗口，该窗口中永远只能看到页面寄存器 0x38～0x3D 中的某一页。128KB 的 Flash 中，0x3F 永远定位在 0xC000～

0xFFFF 这一段，0x3E 永远定位在 0x4000～0x7FFF 这一段。另外 6 页只能通过 0x8000～0xBFFF 窗口来访问，每一页对应一个 PPAGE 页面编号，Flash 换页是通过向 PPAGE 寄存器(地址为 0x30)写入页面编号实现的。

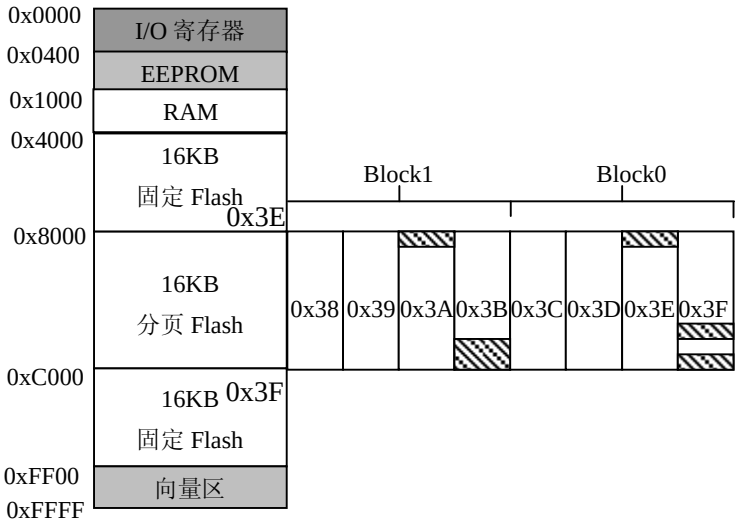


图 4-11 DG128 存储空间的扩展

4.3.2 线性地址转换为内存扩展地址

由于 16 位 HCS12 系列单片机的寻址能力被限制在 64KB 以内，为了增加片内 Flash ROM 的容量，采用了页面 Flash ROM 的寻址技术，即当应用程序做得很大，超过了 64KB 寻址空间以后，S1 格式已经不能适应应用的要求，需要按照 S2 格式下载程序。S2 格式与 S1 格式的区别仅仅在于 S2 格式使用 3B 表示线性地址，而 S1 格式使用 2B 表示线性地址(注：S1 格式的线性地址就是 Flash ROM 的内存地址，不需要进行地址转换)。根据 CPU12 用户手册[33]，编译器编译时，将线性地址转换为内存扩展地址的原理如下：所有 M68HC12 系列 MCU 的内存地址至少都是 16 位线性地址，当使用到扩展地址时，则要在 16 位线性地址之前再附加一个 6 位地址，组成一个 22 位的线性地址，其高 8 位与 PPAGE 寄存器的值复用，即可将其转换为 PPAGE 寄存器的值(页面编号)；低 14 位经过映射后可与内存扩展地址的低 16 位相对应。转换方法为：首先，将线性地址的 ADDR[21:14]位的值作为页面编号，如线性地址为 0x0f0000，则其 ADDR[21:14]位为“00111100”，其值为 0x3c，所以 PPAGE 寄存器的值应该是 0x3c；其次，线性地址的低 14 位 ADDR[13:0]可映射为扩展内存中的低 16 位地址 ADDR[13:0] +0x8000。

例如，线性地址为 0x0f5808，则其转换为内存扩展地址的步骤如下：

① 转换页面编号

ADDR[21:14]= 0b00111101=0x3d

② 线性地址的低 14 位转换为内存扩展地址的低 16 位

ADDR[13:0]+ 0x8000=0b01100000001000+0b1000000000000000=0x9808

③ 把 8 位页面编号值作为 24 位扩展内存地址的高 8 位，这样就形成了 24 位扩展内存地址 0x3d9808(即程序装载的 FLASH ROM 地址)

注：因为所有的内存扩展地址都存放于某个页面内的 0x8000~0xbfff 范围内，因而要把线性地址的低 14 位加上 0x8000 才能映射到相应页面的内存扩展地址 0x8000~0xbfff 内。

4.3.3 编译生成 S2 格式文件

使用 SdIDE12 生成 3B 线性地址的 S2 格式文件，编译器编译时，需要进行以下设置修改。

(1) 集成开发环境的编译器参数的修改

将 C 编译器编译参数中的“-mshort”改为“-mlong-calls”。

CFLAGS = "./m6811/m6811-elf/include -mlong-calls -m68hc12 -c ";

将汇编编译器编译参数中的“-mshort”改为“-mlong”。

ASFLAGS =" ./m6811/m6811-elf/include -m68hc12 -mlong ";

修改这两个参数后，编译器的汇编指令将使用 CALL 和 RTC 来调用和返回相应的子程序，而不再使用 JSR 和 RTS。因为在扩展内存中跳转时，使用 CALL 指令类似与 JSR，但 CALL 指令会把分页窗口的返回地址放到堆栈中，同时还会在 CALL 指令把新的 8 位数据写到 PPAGE 之前，把 PPAGE 的当前值放到堆栈中。调用 CALL 指令时，需要用 RTC 指令返回，程序继续执行时，把 PPAGE 的值和分页窗口的地址从堆栈中调出。

(2) 编写 Linker.ld 脚本文件，指定代码段

一般 GCC 在编译过程中会将指定的代码直接放入 Linker.ld 文件所指定的.bank 段中，下面介绍针对 MC9S12DG128 芯片页面窗口（Page Window ）的脚本文件 Linker.ld 的编写，128KB 的 Flash ROM 空间分成 8 页，每页 16KB，0x38~0x3D 的 6 页的寻址空间都一样(0x8000~0xBFFF)，靠 PPAGE 寄存器来选择使用哪一页。用扩

展内存时，编译器编译生成的是 S2 格式文件，其使用 3 位装载地址，对于 MC9S12Dx128 单片机，片内有 128KB 的 Flash 存储器，均按线性地址生成对应的内存扩展地址，以MC9S12DG128 单片机为例，其线性地址、页面编号和内存扩展地址的关系如表 4-8 所示。

表 4-8 DG128 线性地址、页面编号和内存扩展地址的关系

线性地址(链接地址)	页面编号	内存扩展地址
0x0E0000~0x0E3FFF	0x38	0x388000~0x38BFFF
0x0E4000~0x0E7FFF	0x39	0x398000~0x39BFFF
0x0E8000~0x0EBFFF	0x3A	0x3A8000~0x3ABFFF
0x0EC000~0x0EFFFF	0x3B	0x3B8000~0x3BBFFF
0x0F0000~0x0F3FFF	0x3C	0x3C8000~0x3CBFFF
0x0F4000~0x0F7FFF	0x3D	0x3D8000~0x3DBFFF
0x0F8000~0x0FBFFF	0x3E	0x4000~0x7FFF
0x0FC000~0x0FFFFF	0x3F	0xC000~0xFFFF

通过链接器(linker)，可以将浮动的 C 语言目标文件按照给定的线性地址进行链接，即编译生成 S2 格式文件。下面介绍该链接器的一些属性和使用方法。

在整个链接过程中，代码和数据的基本单位是“段”。用户将不同属性的内容放入不同的段中，链接器识别这些段，按照用户指定将各个段放入相应的存储单元中，完成链接。带页面窗口的链接脚本文件与普通的链接脚本文件有所不同，Page Window 的脚本文件主要有 MEMORY 和 SECTIONS 这两个基本命令构成[34]。命令功能及其代码编写如下。

① MEMORY 命令。MEMORY 命令是一种描述方式，它告诉链接器在整个可用的空间中哪些区域是可以为哪一类段使用。每个段名称后面的括号里是该段的属性标识，其中“r”表示该段存储单元中的数据可读取，“w”表示该段存储单元可写入，“x”表示该段存储单元中放入的数据是可执行代码。MEMORY 命令代码编写如下：

```
MEMORY
{
 /* 用户程序存放的第一块 ROM 区*/
 bank1(rx) : ORIGIN = 0x0e0000, LENGTH = 0x3fff
 /* 用户程序存放的第二块 ROM 区*/
 bank2(rx) : ORIGIN = 0x0e4000, LENGTH = 0x3fff
  …
 /* 用户程序存放的第七块 ROM 区*/
 bank7 (rx) : ORIGIN = 0x004000, LENGTH = 0x3fff
 /* 用户程序存放的第八块 ROM 区*/
 bank8(rx) : ORIGIN = 0x00C000, LENGTH = 0x3fff
  /*程序的矢量区，存放不会修改的常量数据*/
 rom  (r) : ORIGIN = 0x00ff80, LENGTH = 0x0080
```

```
 /* 程序的 RAM 区，用来存放初始化全局和静态变量*/
 data  (rwx) : ORIGIN = 0x1000, LENGTH = 0x0200
}
/*程序堆栈指针*/
PROVIDE (_stack = 0x1800-1);
```

bank1~bank8 是用户程序可存放的区域，bank1 段中存放的用户程序从线性地址 0x0e0000 开始，长度是 0x3fff 个字节，经过地址转换后，用户程序代码将从内存扩展地址 0x388000 处开始载入，程序代码长度被限制在 0x3fff 个字节内。bank2～bank6 段与 bank1 段类似，均需要进行地址转换。bank7 段中存放的用户程序实际是从 0x004000 开始，长度是 0x3fff 个字节，bank8 段中存放的用户程序实际是从 0x00C000 开始，长度是 0x3fff 个字节。bank7 和 bank8 分别始终对应 0x4000~0x7fff 和 0xc000~0xffff 这两段，不需要进行地址转换。

Rom 段是只读数据段，用来存放一些不会修改的常量数据，该段数据只可以读。

Data 段是从 0x1000 开始，长度是 0x0200 个字节的片内 RAM，全局变量和静态变量存放在该段，该段数据可读可写也可执行。

PROVIDE 是定义初始化时的堆栈指针值。由于 HCS12 的堆栈是向着低地址增长，即数据入栈后，堆栈指针减小，本例中将堆栈指针初始化为 RAM 区的 0x17ff。一般将堆栈指针初始化为 RAM 区的最大地址值 0x3fff，由于 RAM 区未进行映射时，0x2000~ 0x3fff 区域是不能用的，所以要用到这一块区域时，事先要对 RAM 区进行映射初始化，即对内存映射寄存器 INITRM 进行相应设置。

② SECTIONS 命令。SECTIONS 命令是脚本文件中最基本的命令，它的功能非常强大。下面通过一个程序段说明其用法。程序中包括代码段、数据段、向量表段和未初始化数据段，而代码段需要放在 0x0e0000 起始的线性地址处，数据段在 0x001000 起始的线性地址处，未初始化数据段放在数据段之后，向量表段放在 0x00FF80 起始的线性地址处。SECTIONS 命令的代码编写如下：

```
SECTIONS {
.bss 0x1000: {
   *(.bss)
 } > data
bank1 0x0e0000: {
   *(.init)
   *(.bank1)
      ./OBJ/setup.o(*.text)
} > bank1
…
```

```
bank7 0x004000: {
   *(.init)
   *(.bank7)
 } > bank7
bank8 0x00C000: {
   *(.init)
   *(.bank8)
 } > bank8
vectors 0xff80: {
   ./OBJ/vectors.o(*.rodata)
 } > rom
}
```

首先，SECTIONS 将当前指针设为 0x001000，随着程序运行，当指针指到 bank1 段时，在该段中指定了用户程序的目标文件 setup.o 放在该段从线性地址 0x0e0000 开始处。若用户程序的目标文件没有指定在任何一个代码段，则系统默认从 0x004000 起始处装载。当指针指向向量表段时，向量表目标文件 vectors.o 放在该段 0x00FF80 起始处。

(3) 线性地址转换为内存扩展地址

完成以上两步设置之后，则应用程序经编译后，会生成 S2 格式文件，但 S2 格式文件中的目标 FLASH 地址是扩展内存对应的线性地址，程序写入时，不能直接按照线性地址装入内存。在MCU 方写入程序中加入了线性地址转换为内存扩展地址的模块，转换方法如 4.3.2 节所示。其实，编译器编译时，会生成相应跳转语句的机器码，该机器码 是扩展 内存 地址 。如在 Linker.ld 文件中 指定把 Setup.c 文件代 码放在“0x0f0000~0x0f3fff”这一段中，Setup.c 文件中只有 MCUInit()函数，在主函数 main()中被调用，查看编译生成的 LST 文件时，发现调用 MCUInit () 函数的反汇编的语句“CALL 0f0000”对应的机器代码是“4a800038”，机器代码的最后一个字节是页面编号，即 PPAGE 寄存器的值：0x38，这一页的内存起始地址是 0x8000，也就是说 Setup.c 文件编译后的机器代码要放在扩展内存 0x388000 起始地址处，但 Setup.c 文件编译后，其机器码对应的目标 FLASH 地址仍就是 Linker.ld 文件中指定的线性地址，所以需要把它转换为相应的内存扩展地址。

4.4 调试模块详细设计

调试模块是集成开发系统中必可不少的一部分。当前国内对调试技术方面的研究很少，一般嵌入式系统的调试工具都是使用国外的产品，作者在深入研究 TBDML 动

态链接库函数源代码基础上，对其进行合理改造，实现对 HCS12 系列微处理器的单步调试，较复杂的调试功能将在后续工作中完成。单步调试的主要设计思路：首先对 LST 或 ELF 目标文件信息格式进行分析，并给出用于提取加工调试信息的相关数据结构；其次，提取源程序中设置断点的行号，由该行号查找到当前的断点地址；获得断点地址后，通过设置断点寄存器并调用 TBDML 动态链接库函数实现单步调试。下面详细介绍单步调试的功能设计和具体实现。

4.4.1 .lst 文件结构

图 4-12 为一个 C 工程的.lst 目标文件结构，该文件中包含了整个 C 工程的调试信息，每条 C 语句包括以下几行信息：第一行为该条 C 语句编译后的内存首地址；第二行为该条 C 语句所在的子函数路径及行号；第三行为该条 C 语句的源代码；其余的行为该条 C 语句编译后生成目标代码的内存地址、目标代码及对应的汇编指令。例如，图 4-12 中的“Light_D|=1<<Light_Pin”这条 C 语句编译后，在 LST 文件中会对应如下几行信息：第一行的“00004043 <.LM4>：”给出了该条 C 语句编译后生成代码的内存首地址 0x00004043；第二行的“D:/1/main.c:26”给出了该条 C 语句所在的子函数路径为 D:/1/main.c 及所在行号为 26 行；第三行的“Light_D|=1<<Light_Pin;”为源代码，其他的三行都是该语句编译后生的目标代码及目标代码的内存地址和相应的汇编指令。

```
…
00004037 <main>:
D:/1/main.c:21
int  main()
{
   4037: 18 01 ae 10  movw 1018
   403b: 18
   403c: 7f 10 18    sts 1018

0000403f <.LM2>:
D:/1/main.c:22
   MCUInit ();
   403f: 16 40 b2    jsr 40b2
…
00004043 <.LM4>:
D:/1/main.c:26
   Light_D|=1<<Light_Pin;
   4043: f6 00 02    ldab  2
   4046: ca 02       orab  #2
   4048: 7b 00 02    stab   2

0000404b <.LM5>:
D:/1/main.c:27
   Light_P|=1<<Light_Pin;
   404b: f6 00 00    ldab  0
   404e: ca 02       orab  #2
      4050: 7b 00 00    stab   0
…
```

图 4-12 LST 文件结构

4.4.2 设置断点的实现

在工程的任意文件中选择一行代码，点击 SdIDE12 的“调试”→“设置断点”，即可完成断点的设置。设置断点模块的功能如下：

首先，提取被设置断点的代码所属文件的文件全名，并确定该行代码在该文件中所处的行号；其次，打开编译后生成的.LST 目标文件，在文件中查找“文件全名：行号”这一行信息，若能够找到，则该行的前一行就是文中选择的代码行所对应的内存首地址，即为断点地址，然后将当前选择的代码行红色高亮显示，若没有找到，则表明选择设置断点的行不是代码行，断点设置无效。断点设置的程序流程图如图 4-13 所示。

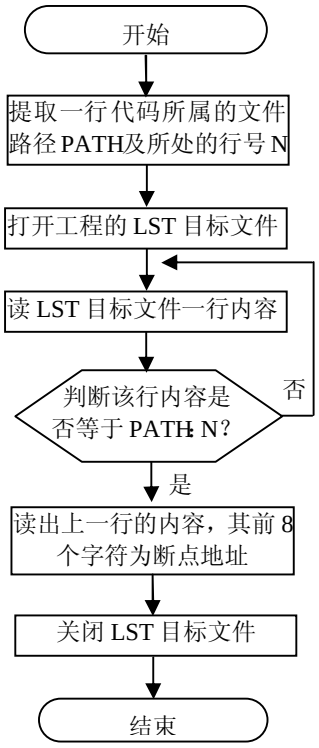


图 4-13 设置断点的流程图

4.4.3 调试环境的初始化

点击“调试”—>“开始调试”，对调试环境进行初始化，主要包括以下几个方面：

① BDM 设备初始化：检测 BDM 设备连接情况；

② 进入 BDM 模式设置：使当前状态为 BDM 模式；

③ 设置 BDM 状态寄存器(BDMSTS)：

④ 设置断点控制寄存器 BKPCT0＝0xB0 和 BKPCT1＝0x00：允许断点、双地址模式、断点地址匹配成功后自动进入 BDM 模式和全地址比较；

⑤ 设置第一个断点寄存器：主要是实现第一个断点地址匹配成功后发生中断。把第一个断点地址的扩展页面值赋给第一个断点扩展寄存器 BKP0X、第一个断点地址的高字节部分赋给第一个断点高字节寄存器 BKP0H、第一个断点地址的低字节部分赋给第一个断点低字节寄存器 BKP0L；

⑥ 设置第二个断点寄存器：由于 CPU12 的断点模块允许同时设置两个断点，因而，对第二个断点寄存器的设置与第一个断点寄存器类似，第二个断点的设置也涉及到第二个断点扩展寄存器 BKP1X、第二个断点高字节寄存器 BKP1H 和第二个断点低字节寄存器 BKP1L。

以上对寄存器的设置均是通过调用 TBDML 动态链接库函数 tbdml_write_byte(寄存器地址,寄存器值)实现的。产生断点中断后，当前设置的断点行将会呈绿色高亮显示，并且目标 MCU 的所有寄存器的值将会发送给 PC 机，并显示在寄存器跟踪窗口中，如图 2-5 所示。若在变量跟踪窗口中输入变量并按回车键，该变量的当前值也会

被显示出来，变量的数据读取和处理借鉴硕士论文《M68HC08 系列 MCU 嵌入式开发平台的设计与实现》的 5.5.6 节[35]，此处不再给出。

4.4.4 单步调试的实现

单步调试的基本功能：由 PC 机确定复位后程序开始执行的入口地址，将其赋给 PC 寄存器，使程序从当前 PC 寄存器中的地址处开始逐条向下执行 C 语句，当到达与所设置的断点地址相同的位置时，程序将会停止向下执行，自动进入 BDM 模式，因此，此时可把当前目标 MCU 的所有寄存器(如 PC、SP、X、Y、A、B、CCR)的值发送给 PC 机，并显示在寄存器跟踪窗口中。在变量跟踪窗口中输入变量名后按回车键，则显示出该变量的当前值。所有这些信息均是程序调试的重要信息，有助于用户快速调试源程序。由于断点匹配成功后，程序就暂停执行，此时 PC 寄存器的值就是断点地址。如果想使程序继续向下执行，则要把 PC 寄存器的值加 1，跳过此断点继续向下执行，直到下一个断点为止，这样一次单步调试完毕。下面以图 4-12 工程例子的.lst 文件为例，阐述设置两个断点的单步调试实现流程。

① 调用设置断点模块：将源代码中的“MCUInit ();”和“Light_D|=1<<Light_Pin;”这两个代码行设为断点，由设置断点模块可以确定第一个断点所在的行号为 22，第二个断点所在的行号为 26，两个断点的文件路径均为“D:/1/main.c”；由这两个信息，在 LST 文件中逐行查找，确定第一断点地址为“0x0000403f”，第二断点地址为“0x00004043”。

② 设置 PC 寄存器(用户程序的复位 地址)：调用动态链接库函数 tbdml_write_reg_pc(0x4000)，设置程序执行的起始地址为 0x4000。

③ 目标机开始执行用户程序：调用动态链接库函数 tbdml_target_go()，此时目标机从 BDM 模式进入普通模式，开始从当前 PC 所指的内存地址开始向下执行。

④ 判断断点地址匹配是否成功：调用动态链接库函数读出 BDMSTS 寄存器的值，根据寄存器第 5 位的值判断是否进入 BDM 模式。若为 1，则进入 BDM 模式，说明断点匹配成功，此时程序已暂停执行；若为 0，则没有进入 BDM 模式，断点设置无效。

⑤ 读取 PC、SP、...CCR 寄存器的值：若进入 BDM 模式，则此时可读取当前的 MCU 的所有寄存器值。

⑥ PC 寄存器值加 1：当前断点操作完成后，则调用动态链接库函数 tbdml_target_step()和 tbdml_target_go()，实现从当前 PC 寄存器地址的下一地址处执行用户程序，此时目标机从 BDM 模式进入普通模式。

⑦ 重复④⑤⑥的操作，实现对第二个断点的调试。直到最后程序运行完毕。

4.5 测试体会

无论是硬件开发平台的设计还是软件开发平台的设计，其前期的设计构思和系统规划都是至关重要的，这一点笔者深有体会。只有前期的分析充分了，考虑全面了，后期才会少走弯路，才不会出现严重的设计错误。特别是对一个通用的集成开发系统来说，其前期的设计就显得尤其重要。一套功能完善，可适用于多平台的集成开发系统包括很多的功能模块，如何将这些功能模块有机的融合在一起，是设计者在开发前期的主要工作。解决平台间、模块间的耦合与内聚是设计过程中的难点，也是确保开发环境具有稳定性和通用性的关键环节。在分析和借鉴了市场上著名的集成开发环境的设计优点之后，根据自己对开发环境的剖析，设计出了具有相对通用的 SdIDE12，在后期的实验测试中表明，这种设计方法和思想为自己的开发赢得了很多时间，在开发和调试过程中，获得了宝贵的经验和教训。以开发和调试内存扩展模块为例，由于前期对这一模块设计的准备不够充分，而且有关这方面的参考资料也比较少，因此作者花费了较长的时间进行摸索，获得了很深刻的开发体会。

① 在实现扩展内存之前，写入模块主要是适用于没有扩展内存的情况，有许多部分要经过修改后才能实现扩展内存的写入。关键的是要修改写入程序，在向 FLASH 目标地址(扩展内存地址)中写入数据之前，要将 S2 文件中的线性地址转换为内存扩展地址，然后才能将线性地址中存放的数据写入到相对应的内存扩展地址中。

② 在线性地址和扩展内存地址的映射部分，走了不少弯路。主要原因在于起初没有进行这个模块的大量调试，缺少直观的实验现象，对于 CPU12 用户手册的内存扩展部分知识也没有做透彻的理解，经过后期的测试，发现编译器在编译调用扩展页跳转指令时，将相应的线性地址自动转换为内存扩展地址。但起初编译时，并没有注意这一点，所以直接把程序代码写入到线性地址处，运行机器码时，就造成跳转错误。

③ 调试时发现不能向 0x0e0000~0x0e3fff 和 0x0f0000~0x0f3fff 这两段线性地址所对应的内存扩展地址写入数据。逐步调试写入模块 PC 方软件发现了不能写入的原因：

当这两段的 3B 线性地址的后两个字节小于 0x1000 时，在PC 方重组一页用户数据包时会出现错误，数据包中地址不再占有三个字节，而在 MCU 方写入程序从 RAM 区取该页用户数据时，是按照三个字节取的，从而造成之后数据取出时全部错位。通过特定的方法，使用户数据包中线性地址始终占 3 个字节，问题得以解决。

④ 在整体擦除时，只能擦除固定页面(即 0x4000~0x7ffff 和 0xc000~0xffff 这两页)，不能擦除扩展页。经过多次调试及查阅相应的 FLASH 手册，发现 FTSTMOD 寄存器的设置与此有关，最后经调试发现，在整体擦除时，FTSTMOD 寄存器的 WRALL 位置“1”时，可擦除包括扩展页在内的所有 FLASH 空间，而 FTSTMOD 寄存器的 WRALL 位置“0”时只擦除 FLASH 的固定页。

4.6 本章小结

本章详细介绍了通用嵌入式集成开发系统 SdIDE12 的软件设计与实现。通用集成开发环境 IDE 模块主要实现了对各个系列 MCU 源文件的编辑和编译，HCS12 系列 MCU 写入模块主要是实现把编译生成的 S 格式机器码下载到目标 MCU 的 Flash 区，调试模块目前只实现了初步调试。本章从 PC 方软件到 MCU 方软件均对系统通用性进行了深入研究和设计，使其尽可能的拥有独立性和可扩展性。写入模块不仅实现了对基本内存的写入，也实现了对扩展内存的写入，解决了内存空间小的问题。在每个模块开发完成后，都编写了相应的测试用例对其进行测试，在多次的测试过程中不断完善系统的功能，提高了系统运行的稳定性。

第五章 基于 CPU12 微处理器的 μC/OS-Ⅱ移植

μC/OS-Ⅱ是 Jean J.Labrosse 在 1990 年前后写的一个实时操作系统内核[36]。μC/OS-Ⅱ的目标是实现一个基于优先级调度的抢占式的实时内核，并在这个内核之上提供最基本的系统服务，例如信号量、邮箱、消息队列、内存管理、中断管理等。μC/OS-Ⅱ的 90%以上的代码都是用 C 语言写成，将 μC/OS-Ⅱ移植到 CPU12 处理器上，只需要修改 OS_CPU.H、OS_CPU_C.C 和 OS_CPU_A.S 三个文件，而基于 μC/OS-Ⅱ编写应用程序时，需要改写 OS_CFG.H 和 INCLUDES.H 两个文件，修改的代码总量大约是 500 行。移植工作的绝大部分都集中在多任务切换的实现上，因为这部分代码主要是用来保存和恢复处理器现场(即相关寄存器)，因此不能用 C 语言，只能使用特定的处理器汇编语言完成。μC/OS-Ⅱ 的全部源代码量大约是 6000－7000 行，一共有 15 个文件，μC/OS-Ⅱ的文件列表如表 5-1 所示。μC/OS-Ⅱ可以管理和调度约 64 个任务，由于 μC/OS-Ⅱ是可剥夺型的，在任务调度过程中要处理可能发生的竞争，每个任务都要有自己的栈空间，故任务越多，RAM 的空间需求就越大。而大部分以 CPU12 为核心的单片机的 RAM 空间范围在 8KB~12KB 之间，这类单片机主要应用于工业控制，有较好的抗干扰特性，所以考虑到 RAM 空间和使用环境，这类单片机特别需要像 μC/OS-Ⅱ这样的 RTOS(实时操作系统)，也特别适合运行这样的 RTOS。

表 5-1 μC/OS-Ⅱ**的文件列表**

文件名		功能描述
μC/OS-Ⅱ与 CPU 类型无关的文件	OS_CORE.C	实时内核
	OS_TASK.C	实时内核任务管理
	OS_TIME.C	实时内核时间管理
	OS_SEM.C	实时内核信号量管理
	OS_MBOX.C	实时内核消息邮箱管理
	OS_Q.C	实时内核消息队列管理
	OS_MEM.C	实时内核内存管理
	OS_MUTEX.C	实时内核互斥型信号量管理
	OS_FLAG.C	实时内核事件标志组管理
	μCOS-II.C	包含以上内核文件
μC/OS-Ⅱ配置文件	OS_CFG.H	需要根据应用修改的文件，用于配置内核的属性
	INCLUDES.H	公共头文件，被所有的 C 源程序引用
μC/OS-Ⅱ移植文件(与 CPU 类型有关的文件)	OS_CPU.H	定义用于特定 CPU 的数据类型、定义相关的宏
	OS_CPU_C.C	用 C 语言编写的与硬件有关的代码。
	OS_CPU_A.S	用汇编语言编写的与硬件有关的代码

有关 μC/OS-Ⅱ内核的执行机制可参见相关的资料[37]，本文不再给出。下面详细阐述基于 CPU12 微处理器的 μC/OS-Ⅱ移植。

5.1 μC/OS-Ⅱ移植过程

最初的 μC/OS 是为 68HC11 写的，由于 CPU12 单片机的 RAM 空间一般都在 8KB 以上，ROM 空间一般在 32KB 以上，拿出几 KB 来运行实时内核没有问题。当然，μC/OS-Ⅱ 也要占用一部分片内 RAM 空间，对于超过 8KB 片内 RAM 的 CPU12 单片机，运行 μC/OS-Ⅱ 就更没有问题了。

要实现 μC/OS-Ⅱ向 CPU12 移植，主要需修改 OS_CFG..H、OS_CPU.H、OS_CPU_C.C 和 OS_CPU_A.S 四个文件。μC/OS-Ⅱ与 CPU 类型无关的文件有很多，它们是 μC/OS-Ⅱ的内核和很多功能函数，表 5-1 中前 3 个文件是必须的。后面六个功能函数用于任务间通信，应用程序中不一定会全部用到，用不到的可以不包含，以节省空间。这部分代码与 CPU 类型无关，不需要改动。

5.1.1 修改 OS_CPU.H 文件

(1) 数据类型定义

由于不同编译器有可能使用不同的字节长度来表示同一数据类型，因而采用宏定义的方法来预先定义编译器用到的数据类型。CPU12 单片机的字长均为 16 位，在移植过程中，需要重新定义数据类型，具体的定义此处不再给出，可参见硕士论文光盘的 uCOS-II 文件夹中 OS_CPU.H 文件的代码。

(2) 堆栈单位及增长方向定义

CPU12 单片机中的堆栈是按字进行操作的，任务堆栈的数据类型需要设置成 INT 型。对于不同的处理器，数据入栈时堆栈指针的增长方向是不同的，有的是由低地址向高地址增长，有的是从高地址向低地址增长。因而，需要预先设定堆栈的增长方向。

```
//定义了 HCS12 CPU 中堆栈入口的数据长度
typedef unsigned int   OS_STK;
//定义了 HCS12 CPU 中堆栈增长方向：从高向低
#define  OS_STK_GROWTH    1;
//定义了 HCS12 CPU 中 CCR 的长度
typedef unsigned char  OS_CPU_SR;
```

(3) 开关中断和任务切换的宏定义

与所有的实时内核一样，μC/OS-Ⅱ也需要在访问代码的临界区前先禁止中断，在访问完毕后重新放开中断，从而保护临界区代码，使其免受多任务或中断服务程序的破坏。μC/OS-Ⅱ定义了两个宏来保护临界区：OS_ENTER_CRITICAL()和 OS_EXIT_CRITICAL()。该方法通过定义 OS_CPU_SR 数据类型的临时变量 cpu_sr 来保存进入临界代码之前的中断开关状态。

```
//将 CCR 寄存器的值保存到 cpu_sr 变量中，保存之前关中断
#define  OS_ENTER_CRITICAL()   {asm("tpa");asm("sei");asm("staa cpu_sr")}
//脱离临界段代码时，恢复 CCR 寄存器的值
#define  OS_EXIT_CRITICAL()    {asm("ldaa cpu_sr");asm("tap")}
//定义任务切换指令
#define  OS_TASK_SW()          {asm("swi")}
```

OS_TASK_SW()是一个宏，它是在 μC/OS-Ⅱ从低优先级任务切换到高优先级任务时被调用。该宏就是模拟一次中断过程，在中断返回时进行任务切换。CPU12 都支持软中断，软中断指令为 SWI。

5.1.2 修改 OS_CPU_C.C 文件

在该文件中主要是有关钩子函数和任务堆栈初始化的代码设计。在系统中钩子函数需要声明，但内容一般设为空，以便用户扩展 μC/OS-Ⅱ的功能。在 μC/OS-Ⅱ中的任务是以函数的形式实现的，每一个任务都需要有自己的堆栈空间，以便进行任务切换的时候能够将当时的处理器现场保存到任务的堆栈空间中，在下次执行时能够再恢复出来。在每次进行处理器现场保存的时候，需要按照一定的顺序进行堆栈操作，这个顺序就是任务堆栈设计。

任务堆栈的设计是在任务初始化时进行的，主要就是在堆栈增长方向上如何定义每个需要保存的寄存器位置。在CPU12 体系结构下，处理器的现场通常是指{PC、Y、X、A、B、CCR、PPAGE}这些寄存器。在目前实现的系统中，任务堆栈空间由高至低依次保存 PC、Y、X、A、B、CCR、PPAGE。修改后的任务堆栈初始化代码参见硕士论文光盘的 μC/OS-Ⅱ文件夹中 OS_CPU_C.C 文件的 OSTaskStkInit()函数。

5.1.3 修改 OS_CPU_A.S 文件

该文件中包含 OSStartHighRdy()、OSCtxSw()、OSIntCtxSw()和 OSTickISR()四个函数。其中，最高优先级就绪任务启动函数 OSStartHighRdy()的功能是运行优先级最高的就绪任务；任务级任务切换函数 OSCtxSw()通过一次中断来实现任务切换；中断级任务切换函数 OSIntCtxSw()是在中断服务程序中完成任务的切换；OSTickISR()为时钟节拍中断服务函数，任务延迟和超时控制等功能都是通过时钟节拍来实现的。改写这四个函数，是移植工作的重点，下面介绍这几个函数的改写过程。

(1) 改写 OSStartHighRdy()

在函数 OSStart()中通过调用 OSStartHighRdy()实现运行优先级最高的就绪任务。若要运行最高优先级的任务，则要将所有寄存器值从由函数 OSTaskStkInit()创建的任务堆栈中恢复出来，并且执行中断返回。所以，该函数要完成的操作如下：

① 调用 OSTaskSwHook()函数；

② OSRunning 赋值为 TURE(或 1)；

③ 取得任务堆栈指针；

④ 从新任务的堆栈中恢复 PPAGE;

⑤ 中断返回；

修改后的程序代码如下：

```
/*********************************************************************
函数：OSStartHighRdy(void);
作用：让优先级最高的任务运行
参数：无　；返回：无
*********************************************************************/
void OSStartHighRdy(void)
{
   OSTaskSwHook();
   OSRunning = 1;
      asm{
        ldx OSTCBCur; lds 0,x;   //          将 TCB 的地址给 SP
        pula; staa $30; nop; rti    //          恢复页面寄存器
   }
}
```

(2) 改写 OSCtxSw()

为了实现任务级调度，函数 OSSched()使 OSStartHighRdy 指向最高优先级任务的任务控制块 OS_TCB，然后调用宏 OS_TASK_SW()，人为模仿一次中断的过程。软

中断的中断向量指向 OSCtxSw()，这样一旦调用了 OS_TASK_SW()，程序就转而执行函数 OSCtxSw()，由它来完成任务的切换。函数 OSCtxSw()的主要功能如下：

① 保存页面寄存器 PPAGE;

② 将当前任务的堆栈指针保存到任务控制块中；

③ 调用 OSTaskSwHook()函数；

④ 将优先级最高的就绪态任务的任务控制块指针 OSTCBHighRdy 赋给当前运行的任务控制块中的 OSTCBCur；

⑤ 将新任务的优先级 OSPrioHighRdy 赋给当前运行任务的优先级 OSPrioCur；

⑥ 使当前指针指向新任务控制块；

⑦ 从新任务的堆栈中恢复 PPAGE;

修改后的程序代码如下：

```
/*********************************************************************
函数：OSCtxSw(void);
作用：当前运行的任务切换成比它优先级高的任务
参数：无 ；返回：无
*********************************************************************/
void OSCtxSw(void)
{
 asm{ldaa $30; psha;       //        保存页面寄存器
    ldx OSTCBCur; sts 0,x;  //       将当前任务堆栈指针保存到任务控制块中
    }
 OSTaskSwHook();
 OSTCBCur = OSTCBHighRdy;
 OSPrioCur = OSPrioHighRdy;
 asm{ ldx OSTCBCur; lds 0,x; pula; staa $30; rti }
}
```

(3) 改写 OSIntCtxSw()

中断级任务调度，是在中断服务程序 OSTickISR 中调用函数 OSIntExt()来实现的。如果函数 OSIntExt()发现中断激活了更高优先级的任务，则调用中断级的任务切换函数 OSIntCtxSw()。由于中断时所有 CPU 寄存器都保存过了，包括 PPAGE 寄存器，故不需要再保存。以下是中断级任务切换函数 OSCtxSw()的主要功能：

① 调用 OSTaskSwHook()函数；

② 将优先级最高的就绪态任务的任务控制块指针 OSTCBHighRdy 赋给当前运行的任务控制块中的 OSTCBCur；

③ 将新任务的优先级 OSPrioHighRdy 赋给当前运行任务的优先级 OSPrioCur；

④使当前指针指向新任务控制块；

⑤从新任务的堆栈中恢复 PPAGE

程序代码实际上是任务级任务切换函数的后半部分，此处不再给出。

(4) 改写 OSTickISR()

时钟节拍中断发生时，CPU12 会自动把 CPU 寄存器压入堆栈，但是并不包括存储页面寄存器 PPAGE。故 PPAGE 寄存器也要推入堆栈，然后才清中断标志。由于时钟节拍中断服务子程序可能激活一个优先级高于当前被中断任务的优先级的任务。其实现过程就是由时钟节拍中断服务子程序连续调用 OSIntEnter()、OSTimeTick()、OSIntExit()三个函数。这三个函数是用 C 语言编写的，OSIntEnter()通知 μC/OS-Ⅱ进入中断服务子程序了。OSTimeTick()将要求延迟若干时钟节拍的任务延迟计数器减 1，若减 1 后为 0 则该任务进入就绪态。OSIntExit()函数告诉 μC/OS-Ⅱ时钟节拍中断服务子程序结束。修改后的程序代码如下：

```
/*****************************************************************
函数：OSTickISR(void);
作用：定时器中断函数，用来产生时钟节拍
参数：无 ；返回：无
*****************************************************************/
void OSTickISR(void)
{
  asm{ ldaa $30; psha;}      //        把存储页面寄存器入栈
  OSIntEnter();
  OS_SAVE_SP();
  //  用户代码开始
  OSTimeTick();
  TFLG2 = 0x80;      //                清定时中断
  //  用户代码结束
  OSIntExit();            //            退出中断,切换任务
  asm{pula; taa $30; nop; rti; } //     恢复存储页面寄存器
}
```

5.2 μC/OS-Ⅱ移植的测试

当用户为自己的处理器完成 μC/OS-Ⅱ的移植后，紧接着的工作就是验证移植的 μC/OS-Ⅱ是否正常工作，这是移植中最复杂的一步。应该首先不加任何应用代码来测试内核自身运行状况。这样做可以在测试中定位没有正常工作的原因在于移植本身的问题，而不是应用代码产生的问题。如果 μC/OS-Ⅱ中 2 个基本的任务和节拍中断运行起来，那么下一步再添加应用任务。

在移植测试过程中，可使用不同的测试技术，这取决于用户在嵌入式系统方面的经验和对处理器理解的程度。作者以 MC9S12DG128 微处理器的 μC/OS-Ⅱ移植为例，来阐述测试移植代码的主要步骤。

(1) 最小测试

作为测试的第一步，主函数中不需要加入复杂的代码。这一步只是想验证是否可以编译出正确的代码，需要用户解决各种编译器和链接器的错误，包括出现的一些警告。在主函数中只需要调用 OSInit()和 OSStart()这两个函数进行初步测试。

(2) 验证 OSTaskStkInit()和 OSStartHighRdy()函数

上一步测试通过后，则可以编译生成移植好的 μC/OS-Ⅱ代码，真正的测试工作从该步开始，首先要验证 OSTaskStkInit()和 OSStartHighRdy()函数。这两个函数的测试可以采用判断目标系统上的发光二极管(LED 小灯)是否运行的方法。即在主函数中的开始时置小灯灭，如果 OSTaskStkInit()和 OSStartHighRdy()函数正常工作，则由 OS_TaskIdle()函数置小灯亮。为实现以上测试，需要修改 OS_CFG.H 文件中的 OS_TASK_STAT_EN为 0，以禁止统计任务；主函数中添加关闭 LED 的代码；修改 OS_CPU_C.C 文件中的钩子函数 OSTaskIdleHook()，添加实现 LED 的开关的语句。修改好后，就可以编译生成代码，并写入到 MC9S12DG128T 目标芯片中进行测试。如果 LED 闪烁不正常，则说明这两个函数没有正常工作，需要用户认真的去检查程序代码，反复调试，直到修改成功。

(3) 验证 OSCtxSw()函数

只有上一步测试成功后，才能保证堆栈初始化函数是正确的，因而，这一步测试可以在主程序中添加一个应用程序，以实现不断切换到空闲任务。该步测试是在上一步测试的基础上开始的，也就是在主程序的开始时置小灯灭，然后创建一个任务，该任务循环调用 OSTimeDly()函数，以强行进入等待状态，调度高优先级任务。由于没有开中断，也没有让时钟节拍开始，所以 OSTimeDly()不会返回到任务开始处执行。当调用 OSTimeDly()时，会发生任务切换。如果 OSCtxSw()函数正确，LED 小灯会快速闪烁。如果小灯一直是关着的，则说明 OSCtxSw()函数有问题，需要仔细检查代码。

(4) 验证 OSIntCtxSw()和 OSTickISR()函数

OSIntCtxSw()与 OSCxtSw()函数很类似，只是前者是中断级任务切换，在测试时，要初始化时钟节拍并放开中断，同时也要使时钟中断向量指向时钟节拍中断服务子程

序。在运行其他代码之前，先关闭 LED，设置时钟节拍中断向量，使时钟节拍中断指向 OSTickISR()；然后建立一个更高优先级的测试任务，在该任务函数中，先初始化定时器，再开中断，允许时钟中断调用 OSTickISR()，然后就循环调用 OSTimeDly()函数和使 LED 开关状态转换的代码。调用 OSTimeDly()，使 OSCxtSw()将任务切换到空闲任务。空闲任务一直运行，直到接收到时钟节拍中断。时钟节拍中断调用 OSTickISR()，继而调用 OSTimeTick()。OSTimeTick()将测试任务的.OSTCBDly 计数器递减到 0，使该任务处于就绪态。当 OSTickISR()完成并调用 OSIntExit()函数时，OSIntExit()会检测有没有更高优先级的任务进入了就绪态。如果没有，则中断服务子程序接着执行 OSIntExit()下面的语句；如果有，就调用中断级的任务切换函数 OSIntCtxSw()做任务切换，让更高优先级的任务运行。如果 OSIntCtxSw()正常工作，则 LED 小灯会闪烁，如果不闪烁，则可先单独运行 OSTickISR()函数，而不调用 OSIntExit()。此时小灯能闪烁的话，则说明问题现在 OSIntCtxSw()中，然后逐步调试该函数代码，发现问题所在。

5.3 本章小结

由于前四章已经设计开发出了功能完善的 SdIDE12，为本章研究和实现基于 CPU12 单片机的 μC/OS-Ⅱ移植提供了软硬件平台，同时移植工作也可验证软硬件开发平台的稳定性和强大功能。基于 CPU12 单片机的 μC/OS-Ⅱ移植的主要工作在于理解并修改与微处理器体系结构相关的几个文件。移植好后，还需要掌握测试移植代码的方法和步骤。移植过程的主要内容包括两部分：修改与微处理器相关的代码和测试移植代码的步骤及方法。

第六章 基础实验例程

SdIDE12 可作为 Freescale HCS12 系列 MCU 教学和开发的实验平台，也可以作为技术人员开发嵌入式产品的工具。为了广大用户和开发人员快速入门，本章主要阐述在 SdIDE12 中如何编写 Freescale HCS12系列 MCU 各模块的实验程序[38]。MC9S12NE64、MC9S12UF32、MC9S12XDP512 和 MC9S12DG128 芯片的基本模块实验例程源代码放在硕士研究生光盘的实验例程文件夹中。限于篇幅，本章只给出编程规范、MC9S12DG128 芯片的模板程序(即该芯片的通用程序及框架程序)及各个实验例程列表。以普通 I/O 口实验程序的编写说明其他 MCU 的模板程序和模块程序的编写步骤及规范。

6.1 编程规范

高质量的程序不仅要保证能正确、高效地运行，而且赏心悦目的程序版式及清晰易懂的程序结构也是必不可少的。只有这样才便于程序的阅读和维护。子程序的长短、文件的大小、行的缩进对齐、层次的嵌套、标识符的命名及注释的格式等都是在程序编写过程中应该注意的地方。在进行程序代码编写时，有效的注释是非常必要的，注释清晰的程序便于阅读，而只有代码的程序会让人很难理解,且后期的调试和维护异常困难。一般在注释子程序时，应在程序头说明程序名、功能、入口、出口、编程者、调用举例等内容。这样，用户可以不必了解程序的具体实现细节，按说明调用即可。

一个高质量的软件系统肯定会合理地进行软件功能模块划分。为避免以后重复开发，则需要在设计的初期对整个系统的模块架构设计清楚，各部分功能进行合理的切割并尽量保证每个功能模块独立，明确各入口和出口，不能多个模块交织在一起。当软件有错误时，能迅速定位到模块，甚至到某个函数。只有符合这些要求的软件系统才有生命力。以下是作者编写实验例程时制定的一些规范，附录 C 也给出了一个规范的程序实例。

(1) 标识符命名规范

① 命名要清晰，有明确含义，使用完整单词或约定俗成的缩写，要有注释说明；

② 命名风格要自始至终保持一致；

③ 同一软件产品内模块之间接口部分的标识符名称之前加上模块标识；

④ 宏和常量全部用大写字母来命名，词与词之间用下划线分隔。变量名用小写

字母命名，每个词的第一个字母大写，全局变量另加前缀 g_，局部变量应简明扼要；

⑤ 函数名用小写字母命名，每个词的第一个字母大写，并将模块标识加在前面；

⑥ 一个文件包含一类功能或一个模块的所有函数，文件名称应清楚表明其功能或性质。每个.c 文件应该有一个同名的.h 文件作为头文件；

(2) 注释规范

有效的注释有助于对程序的阅读理解，说明程序在"做什么"，解释代码的目的、功能和采用的方法。一般源程序有效注释量在 30%左右。注释语言必须准确、易懂、简洁。边写代码边注释，修改代码同时修改相应的注释。汇编和 C 中都用"//"注释

① 文件注释必须说明文件名、函数功能、创建人、创建日期、版本信息等相关信息。修改文件代码时，应在文件注释中记录修改日期、修改人员，并简要说明此次修改的目的。所有修改记录必须保持完整。文件注释放在文件顶端，用"/*……*/"格式包含。

② 函数注释。函数头部注释应包括函数名称、函数功能、入口参数、出口参数等内容。如有必要还可增加作者、创建日期、修改记录(备注)等相关项目。函数头部注释放在每个函数的顶端，用"/*……*/"的格式包含。函数代码注释应与被注释的代码紧邻，放在其上方或右方，不可放在下方。函数代码注释用"//…"的格式。

③ 变量、常量、宏的注释。同一类型的标识符应集中定义，并在定义之前一行对其共性加以统一注释。对单个标识符的注释加在定义语句的行尾。全局变量一定要有详细的注释，包括其功能、取值范围、哪些函数或过程存取它以及存取时的注意事项等。注释也用"//…"的格式。

(3) 函数编写规范

函数编写要满足以下基本要求：

① 正确性：程序要实现设计要求的功能。

② 稳定性和安全性：程序运行稳定、可靠、安全。

③ 可测试性：程序便于测试和评价。

④ 规范 / 可读性：程序书写风格、命名规则等符合规范。

⑤ 扩展性：代码为下一次升级扩展留有空间和接口。

编写函数的基本原则：

① 单个函数的规模尽量限制在 200 行以内(不包括注释和空行)。一个函数只完成一个功能。

② 函数局部变量的数目一般不超过 5～10 个。

③ 函数名应准确描述函数的功能。

④ 函数的返回值要清楚明了，尤其是出错返回值的意义要准确无误。

⑤ 减少函数本身或函数间的递归调用。

⑥ 函数内部尽量少用或不用全局变量，要输出的数据一般可通过指针将其存储在指定内存单元中。如果是单值输出，也可以通过函数返回。

⑦ 若是汇编函数，则输出一个数据时，用函数名返回。输出两个数据时，用寄存器 X 或 D。超过二个数据时，用 X 为首地址的 n 个内存(即带地址传输)。

6.2 MC9S12DG128 芯片的模板程序

在 SdIDE12 中，对每一系列 MCU 芯片都编写了相应的模板程序，并打包到系统的安装目录下。在新建相应芯片工程时，系统自动生成这些模板中的通用程序和框架程序，用户只要在此基础上添加模块功能子程序并在框架程序中添加相应实现代码即可完成模块编程。本节主要阐述这些通用程序和框架程序的功能和代码编写。

(1) 编写程序启动文件 crt0.s

该文件是程序启动代码文件，主要是设置复位后程序执行的起始地址、堆栈初始化、看门狗初始化等。针对 HCS12 系列 MCU，其内容基本相同，主要代码如下：

```
.section .text
/*定义复位后，程序执行的入口*/
.globl  _start
_start:
/*堆栈的设置*/
lds #_stack
/*清看门狗*/
clr  0x3C
/*转主程序*/
jsr main
…
```

(2) 编写链接文件 linker.ld

该文件是用于设定编译链接后产生的二进制代码和相关变量在目标 FLASH 中的存放区域，其内容针对不同的芯片会有较大的差别，特别是 FLASH 区的分配差异，该文件编写要参照该芯片的数据手册。该文件的内容编写要了解基本命令 MEMORY 和 SECTIONS 的功能，这两个命令的功能可参见 4.3.3 节中有关分页窗口的链接文件 Linker.ld 的编写。下面给出 MC9S12DG128 芯片不进行分页的链接文件代码：

```
ENTRY(_start)   /*   指定程序入口*/
MEMORY      /*      内存区域的划分*/
{
 text (rx)   : ORIGIN = 0x4000, LENGTH = 0x3fff
 text1 (rx)  : ORIGIN = 0xC000, LENGTH = 0x3fff
 rom (r)     : ORIGIN = 0xff80, LENGTH = 0x0080
 data (rwx)  : ORIGIN = 0x1000, LENGTH = 0x0400
}
PROVIDE (_stack = 0x1800-1);  /* 堆栈的设置*/
/*代码和变量存放区域*/
SECTIONS {
 .bss 0x1000: {        /*变量存放区域*/
   *(.bss)
 } > data
  stext 0x4000: {    /*代码存放区域 1*/
  *(.stext)
 } > text
 text1 0xC000: {     /*代码存放区域 2*/
  *(.init)
  *(.text1)
 } > text1
  vectors 0xff8C: {    /*中断向量代码存放区域*/
  ./OBJ/vectors.o(*.rodata)
 } > rom
}
```

(3) 编写系统初始化文件 setup.c

该文件主要是选择内部总线的时钟源，可对系统总线时钟进行倍频和分频，也可以开关 IRQ 中断和看门狗中断。如果用户在项目开发中想提升或降低系统时钟工作频率，则需要对该文件中的相应代码进行修改，但修改时，一定要参考芯片的 PLL 模块，理解如何进行倍、分频操作。Setup.c 文件的代码如下：

```
/*********************************************************************
函数：MCUInit()
作用：系统初始化函数，通过设置 CLKSEL 寄存器，确定内部总线的时钟源;
通过 PLL 编程，实现系统倍频或减频;开关 IRQ 中断和看门狗设置
参数：无      返回：无
*********************************************************************/
//头文件
#include "S12_c.h"   //   芯片头文件
#include "Setup.h"   //   系统初始化头文件
void MCUInit(void)
 {
  //  禁止总中断
  asm(  “ sei ”);
  //  设置内部总线时钟来源，内部总线频率=OSCCLK/2
  CLKSEL&=0x7f;
  //  倍、分频计算公式:PLLCLK=2*OSCCLK*((SYNR+1)/(REFDV+1))
  PLLCTL&=0xbf;  //     先关闭 PLL
```

```
  SYNR=0x00;    //       设置 PLLCLK 增频的因子
  REFDV=0x00;   //       设置 PLLCLK 分频的因子
  PLLCTL|=(1<<6); //    开 PLL
  //  等待时钟频率稳定后允许锁相环时钟作为系统时钟
  while(1)
    if((CRGFLG&0x08)!=0x00)  break;
  //  转换系统时钟源为 PLL 方式
  CLKSEL|=(1<<7); //    本句执行后:BusClock=PLLCLK/2
  //  其他初始化设置
  INTCR&=0xbf;   //IRQCR.6(IRQEN) =0       禁止 IRQ 中断(默认开)
  COPCTL=0x00;   //COPCTL.2-0(cr2:cr0) =000      禁止看门狗
 }
```

(4) 编写中断向量表文件 vectors.c

该文件是中断向量表文件，其中列出该芯片的所有中断源，当程序中使用到哪几种中断，则在中断源的相应位置，填写上处理该中断的中断服务函数名。若没有使用到该中断，一般在其中断源的相应位置填上“ISR_Empty”，表示空中断。每一个型号的 MCU 中断源都会有差异，所以编写该文件时，要参考手册中的中断源部分。例如，要实现 MC9S12DG128 芯片的 SCI 串行接收中断，其中断向量表文件代码如下：

```
//[头文件]
#include "isr.h"
//[函数声明]
extern void  ISR_Empty(void);
extern void  _start(void);
//中断矢量表，需定义中断函数，可修改下表中的相应项目
void  ( * const vector[])()  =  {
  ISR_Empty,            // ff80 (reserved)
…
   ISR_Empty,           // ff8c (PWM           中断允许)
…
   ISR_Receive,         // ffd6 (SCI0          中断)
…
   ISR_Empty,           // fffa (COP Failure)
   ISR_Empty,           // fffc (COP Clock monitor)
   _start               // fffe (reset)
}
```

(5) 编写中断服务程序框架文件 isr.c

该文件主要是编写所用到的中断服务子函数，用于中断处理。文件中的中断服务子函数名必须和中断向量表文件 vectors.c 中填写的中断函数名完全相同，只有这样，当中断发生时，才可转到该中断服务子函数处执行。isr.c 文件的框架代码如下：

```
//[头文件]
#include "isr.h"        //       中断子函数的头文件
#include "SCI.h"        //SCI       收发子函数头文件
//ISR_Empty:空中断函数
```

```
void  __attribute__ ((interrupt)) ISR_Empty(void)
{
}
//其他中断子函数在以下区域定义
/*********************************************************************
函数：void  __attribute__ ((interrupt)) ISR_Receive(void);
作用：若收到一个数据，则将该数据发送出去
参数：无
返回：无
*********************************************************************/
…
```

(6) 编写头文件 S12_c.h、Type.h

S12_c.h 是 MC9S12DG128 芯片的头文件，主要是关于各个模块寄存器和 I/O 口定义。每一型号 MCU 的模块寄存器和 I/O 口地址，在参考手册都有详细说明，要严格参考这一部分内容编写头文件。由于寄存器和 I/O 口较多，代码量较大，具体代码不再给出。Type.h 是有关数据类型的别名定义，如 typedef unsigned char INT8U 等。

(7) 编写主程序框架文件 main.c

该文件中主要给出主程序的编写框架，用户在新建工程时，会自动生成该文件，但文件中主要是有关以上通用文件的声明及主函数中通用代码的调用，其他与模块有关的代码，用户可以在主程序的指定区域添加。main.c 文件的框架代码如下：

```
/*********************************************************************
工 程 名:*.prj
硬件连接:
程序描述:
说      明:
*********************************************************************/
/*[头文件]*/
#include "S12_c.h"        //        芯片头文件
#include "Setup.h"        //        系统初始化头文件
/*主函数*/
int  main()
{
 MCUInit();           //           调芯片初始化子函数
 // 以下为用户添加代码区域
}
```

6.3 MC9S12DG128 各模块的实验例程

当嵌入式集成开发环境 IDE 及程序写入模块开发完成后，可通过编写各模块实验程序[38]来测试相应的硬件模块。在编写各个模块的实验程序之前，要先看懂各个模块的用户手册，熟悉硬件对象及相应寄存器的功能。SCI 和指示灯闪烁的实验例程要

先于其他模块编写，这两个程序调试通过后，可作为编写和调试其他模块例程的工具，在程序中设置相应的指示灯或通过串行口输出相应变量值，可检测出程序出错的具体位置，从而加快程序的开发速度。本开发系统的实验例程主要是针对 MC9S12DG128 芯片的各个模块而编写的，表 **6-1** 给出了 **MC9S12DG128** 芯片的基础实验例程清单，具体代码可见硕士毕业生论文光盘的实验例程文件夹中的各个模块。

表 6-1 基础实验例程清单

例程编号	实验例程名	程序文件名	包含的子函数及其功能描述
各模块都有的文件	通用文件	**Crt0.s**	进行有关堆栈、看门狗和跳转到主程序的设置
		Linker.ld	指定编译后产生的二进制代码和相关变量在目标 **FLASH** 中存放区域
		Setup.c	包含芯片初始化函数 **MCUInit()**:选择内部总线的时钟源,可对系统总线时钟进行倍频和分频,也可以开关 **IRQ** 中断和看门狗中断
		Setup.h	芯片初始化声明
		Vectors.c	中断向量表文件,其中列出该芯片的所有中断源
		S12_c.h	**MC9S12DG128** 芯片头文件,主要是关于各个模块寄存器和 **I/O** 口定义
		Type.h	类型别名定义
		Includes.h	包含所有 **.h** 文件
	框架文件	**Main.c**	给出主程序的编写框架
		Isr.c	编写所用到的中断服务子函数,用于中断处理
		Isr.h	中断处理函数的声明
C01	普通 **I/O** 口	**LED.c**	包含小灯的初始化子函数 **LEDInit(void)**,驱动小灯亮灭子函数 **LED_L_A(INT8U flag)**
		LED.h	小灯控制引脚定义及相关子函数声明
C02	**SCI** 串行查询	**SCI.c**	包含串行口初始化 **SCIInit(void)**、串行发送 **1** 个字节 **SCISend1(INT8U o)**、串行发送 **n** 字节 **SCISendN(INT8U n,INT8U ch[])**、串行接收 **1** 字节 **SCIRe1(INT8U *p)**、串行接收 **n** 字节 **SCIReN(INT8U n,INT8U ch[])**等串行通用子函数
		SCI.h	串行通信子函数声明
C03	**SCI** 串行中断	**Isr.c**	添加了 **SCI** 串行接收中断服务子函数 **ISR_Receive(void)**,主要实现接收一个数据,则发送一个数据
		SCI.c、**SCI.h**	这两个文件和 **SCI** 串行查询中的完全相同
C04	**4×4** 键盘	**KB.c**	包含键盘初始化 **KB_Init(void)**、扫描读取键值 **KB_Scan(void)**、键值转为定义值子函数 **KB_Def(INT8U KB_valve)**
		KB.h	键盘相关子函数声明及所用到的寄存器别名定义
C05	**LCD** 液晶	**LCD.c**	包含液晶显示初始化 **LCDInit(void)**、在 **HD44780** 显示屏上显示数据 **LCDShow(INT8U str[])**、执行给定的 **cmd** 命令子函数 **LCD_Command(unsigned char cmd)**
		LCD.h	液晶相关子函数声明及所用到的寄存器别名定义
C06	定时器溢出中断	**Timer.c**	本文件只包含定时器初始化子函数 **SetTimer1Ch0I(void)**
		Timer.h	相关定时器子函数声明

		Isr.c	本文件包含定时器溢出中断处理子函数 isrTimOver(void),主要功能是实现每过一秒,则更新当前的时分秒变量值
C07	定时器输入捕捉	TimeIC.c	只包含输入捕捉系统配置初始化子函数 ICInit(void)
		TimeIC.h	输入捕捉寄存器及标志位定义及相关子函数声明
		Isr.c	包含输入捕捉中断处理子函数 ISR_TimerChan0(),主要是当相应输入捕捉通道发生沿跳变时，相应口小灯的状态随之变化
C08	脉冲脉宽调制(PWM)	PWM.c	本文件包含 PWM 初始化函数 PWMInit(INT8U channel,INT8U polarity,INT8U align) 、设置 PWM 周期和占空比子函数 PWMSetting(INT8U channel,INT8U period,INT8U duty)
		PWM.h	相关子函数声明
C09	脉冲累加器	PA.c	本文件只包含脉冲累加器初始化子函数 PAInit(void)
		PA.h	子函数声明
		Isr.c	包含有效沿跳变捕捉中断处理函数 ISR_PA(void), 脉冲累加器 A 计数器溢出中断处理函数 ISR_PAOV(void);
C10	A/D 转换	ADSub.c	本文件包含中值滤波子函数 admid(unsigned char channel)、n 次均值滤波子函数 adave(unsigned int n, unsigned char channel)、A/D 转换初始化函数 ATD_init(void)、获取 1 路 A/D 转换结果子函数 advalue(unsigned char channel)
		ADSub.h	中值滤波、n 次均值滤波、AD 转换初始化及 1 路 AD 转换子函数声明
C11	FLASH 擦写	Flash.c	包含擦除一个指定的扇区子函数 Flash_Erase_Sector(INT8U page,INT16U addr)、向一个指定的区域写入若干个字子函数 Flash_Write_Nword(INT8U page, INT16U ddress_destination, INT16U address_source,INT8U len)
		Flash.h	相关子函数的声明
C12	μC/OS-Ⅱ移植的小灯闪应用程序	表 5-1 列出的所有文件	功能如表 5-1 所述
		Main.c	在添加代码区用 OSTaskCreate 函数新建两个任务,一个是开始任务 StartTask(void *pdata):时钟节拍工作,创建其他的任务;另一个是小灯闪的任务 ShineLED(void *pdata): 小灯不停的闪烁
C13	智能寻迹小车	ADSub.c	同 C10
		M_sub.c	功能:用于驱动 2 个电机的子程序，将前向司服电机定义为 Motor1，前进电机为 Motor2。Motor1 和 Motor2 的初始化函数 Motor_init，Motor1 转向子程序 Motor1_turn， Motor2 转动子程序 Motor2_run，Motor2 开始运转 Motor2_start，Motor2 停止运转 Motor2_stop
		PWM.c	用于定义 PWM 的基本程序设置,PWMInit:PWM 初始化函数,PWMSetting:PWM 通道的设置函数,PWMEnable:PWM 通道有效函数,PWMDisEnable:PWM 通道无效函数
		Sensor.c	当前传感器所处状态函数 Sensor_State Sensor_State(unsigned char ch),功能:当前传感器是在黑线上、白纸或进入黑线的临界,参数:ch:AD 的通道值(0~6),返回:0:在白纸上:1:在黑线上;2:在黑线的临界状态
		Trace.c	包含了 智能车寻迹的子函数 Trace(unsigned char *AD_past,unsigned char *AD_now,unsigned char *AD)和获取当前状态子函数 update_state(void)，并将当前状态放入 AD_now[9]中

下面以普通 I/O 口实验程序的编写为例，说明程序的编写步骤及相关注意事项。

(1) 硬件连接

在编写程序之前，要进行硬件连接，以便程序编译后下载到芯片中查看实验现象。普通 I/O 口实验程序的硬件连接：把芯片的一个 I/O 口与发光二极管 LED 的一端相连，LED 的另一端过电阻接+5V 电源。

(2) 新建工程

用户在 SdIDE12 系统中单击“新建”→“新建工程”后，弹出新建工程对话框，如图 4-4 所示。在该对话框中先选择 MC9S12DG128 芯片平台，设定工程类型为“C 工程”，输入工程名和工程路径后，单击“确定”。此时，会在指定的工程路径下生成一个新的工程，在该工程中会自动生成表 6-1 中所示的通用文件和框架文件，这些文件的编写可参见 6.2 节。由此可见，对于所有的模块程序的编写，只需用户添加相应模块的子程序文件，然后在框架文件中添加相应代码即可完成整个工程的代码编写。

(3) 编写模块子程序文件

工程新建成功后，框架程序和通用程序自动生成，接下来需要添加相应模块所需的子程序文件，并编写模块子程序代码(每个模块要添加的文件在表 6-1 中已给出)。若工程中用到相应中断，则要在 isr.c 文件中添加相应中断处理子函数，并在 vectors.c 文件中的中断表相应位置填写上中断处理子函数名，若没有用到，不需要修改这两个文件。最后，要在主框架程序 main.c 中添加相应的变量、函数声明及子函数调用等代码，从而完成整个工程的代码编写。

针对普通 I/O 口实验程序，添加了 LED.h 和 LED.c 这两个子程序文件，LED.h 子文件主要是有关 I/O 口的定义和相关子函数的声明。该文件中的主要代码如下：

```
//小灯控制需要用到的头文件
#include "DG128C.h"        //MCU         映像寄存器名
#include "Type.h"          //          类型别名定义
//小灯控制引脚定义
#define  Light_P    PTA    //         灯(Light)接在 PTA 口
#define  Light_D    DDRA   //        相应的方向寄存器
#define  Light_Pin  1      //          灯所在的引脚
//小灯控制相关函数声明
void LED_Init(void);       //        定义控制小灯的 MCU 引脚为输出
void LED_L_A(INT8U flag);  //  驱动小灯"亮"、"暗"
```

为了使程序通用，编程时把 I/O 口和寄存器定义在芯片头文件中，使用时，把芯片头文件包含进来。这样，在每个使用到这些 I/O 口和寄存器的文件中，可在开头的

宏定义区对它们进行重新定义，如 LED.h 文件中的小灯控制引脚定义。这样设计代码的优点是显而易见的，若想更换 PTB2 口来控制 LED，只需要把指示灯接引脚定义处的 PTA 改为 PTB，DDRA 改为 DDRB，Light_Pin 的值改为“2”，程序中其他的地方可以不改动，这样就增加了程序的通用性。其他模块实验程序的设计，也是坚持这种思路，把与底层硬件有关的子程序通过宏定义 I/O 口和寄存器的方式封装，而与底层硬件无关的子程序完全可以通过参数传递的方式进行封装。

LED.c 文件中主要有两个子函数：定义控制小灯的 MCU I/O 引脚为输出的子函数 LEDInit()和驱动小灯亮灭的子函数 LED_L_A(INT8U flag) 。这两个函数的代码分别如下：

```
void LEDInit(void)
{
   Light_D |= 1<<Light_Pin;                // 令小灯引脚为输出
   Light_P |= 1<<Light_Pin;                // 初始时，小灯"暗"
}

void LED_L_A(INT8U flag)
{
   if (flag == 'A')
      Light_P |= 1<<Light_Pin;    //          小灯"暗"
   else if (flag == 'L')
      Light_P &= ~(1<<Light_Pin); //        小灯"亮"
}
```

(4) 修改框架程序文件

模块子程序编写完成后，则只需向框架程序 main.c 中添加相应代码和调用相子程序即可，主框架程序中的主函数代码如下：

```
int main()
{
   MCUInit();                          // 系统初始化
   //   以下为用户添加代码区
   LEDInit();                          //小灯控制引脚初始化
   //   程序总循环入口
   while (1)
   {
      LED_L_A('L');                    //调用使小灯亮子程序
      Delay(500);                      //调用延时子程序
      LED_L_A('A');                    //调用使小灯暗子程序
      Delay(500);                      //调用延时子程序
   }
}
```

(5) 工程编辑编译

以上把整个工程的程序代码编写完毕，但程序中可能存在语法错误、不规范的语

句、程序编译设置错误等。所以需要对工程进行编译，逐步排除所出现的错误。单击“编译”→“编译”，在编译调试输出窗口中会列出所有的错误信息，包括错误出现的位置及原因。用户双击错误信息行，即可快速定位到相应的代码行。

(6) 代码写入

当编译成功后，可生成相应的 S 格式文件，数据以十六进制代码形式表示，然后把生成的 S 格式文件代码下载到芯片，使程序运行，观看实验现象。单击“编译”→“写入”，弹出如图 4-6 所示的写入界面，首先，设置芯片型号和目标板晶振频率，对所选的芯片进行初始化；其次，在写入代码之前，要单击“擦除”，对 Flash 区进行擦除，以保证写入区域没有数据；最后，单击“写入”，将 S 格式文件代码写入到指定的 Flash 区。用户可单击“运行程序模式”，即运行实验程序，观察实验现象。

6.4 本章小结

利用 SdIDE12 进行 MC9S12DG128 芯片的基本模块编程，首要的工作是理解该芯片的各个基本模块的特性和相关寄存器的功能及设置方法。只有完全掌握相关芯片的所有模块的基本知识，如芯片的 I/O 口、寄存器、中断向量、FLASH 存储空间分配及系统配置等，才能着手编写该芯片的模板程序。用户新建工程时，系统自动从工程模板中把模板程序复制到当前工程中，但要完成整个模块的编程，还需要用户编写相对独立的模块功能子程序，并添加到工程中。然后在框架文件 main.c 或 isr.c 中添加代码或调用模块功能子函数。至此，该模块的实验例程代码编写完毕，只需要编译通过后写入芯片，就可通过实验现象，验证模块程序编写的正确性。

第七章 总结与展望

7.1 总结

本文针对目前大多数嵌入式集成开发环境的开放性和可扩展性较弱的缺点，从硬件和软件上考虑，设计了基于 HCS12 系列 MCU 的通用嵌入式集成开发系统。本文重点工作包括以下几个部分：

(1) 嵌入式集成开发系统通用框架设计。通用框架包括软件和硬件两部分。在软件上将集成开发系统的功能模块划分为公有模块和私有模块。公有模块主要是 IDE 模块，为了使其对各个系列 MCU 通用，重点从芯片编译参数设置、通用 makefile 文件编写和工程模板文件编写这三个方面进行设计。私有模块主要是程序下载和代码调试模块，它们具有相对独立性，只能针对每一个系列编写一个通用的下载和调试软件，以供系统调用。硬件平台的通用性更难做到对所有系列通用，一般一个系列 MCU 可制作一套硬件评估板，采用核心板和某个系列通用扩展板的设计方法实现通用。

(2) 通用 IDE 模块设计。该模块实现了对 GCC 编译器所支持的每一系列 MCU 的工程管理、源文件编辑和源文件编译，从而为程序下载和代码调试模块提供了用户程序的 S 格式机器码。该模块的设计难点体现在通用 makefile 文件编写、系统自动获取相应的编译器和工程管理模块在整个系统中核心作用的实现这三个方面。

(3) HCS12 系列 MCU 的程序写入和代码调试功能。程序写入模块包括 MCU 方和 PC 方程序。MCU 方程序主要是编写每款 MCU 的擦除和写入子程序；PC 方程序主要是实现写入界面设计、与动态链接库文件 Tbdml.dll 的连接和擦写操作流程的控制。在该部分的设计、调试、升级与完善过程中解决了以下几个技术难点：

① 通过设置参数数据库的方式，获取不同 MCU 的 Flash 擦写参数及擦写子程序代码的路径，实现了写入模块对 HCS12 系列 MCU 的通用。

② 设置相应编译参数选项，生成 S2 格式文件，通过将线性地址转换为内存扩展地址，实现对扩展内存的写入，解决了用户程序代码大小受限制的问题。

③ 程序写入模块和代码调试模块，主要使用 Tbdml 动态链接库函数实现 PC 机与目标 MCU 的通信，因而在这两个模块的初始化部分要多次检测并连接 Tbdml 设备，确保在擦写之前设备连接正常，再调用 tbdml_target_reset(0)使系统进入 BDM 模式，

然后才执行擦写和调试操作。这样设计，解决了程序写入和代码调试不稳定的问题

(4) 由于本文设计的通用嵌入式集成开发系统采用“V”型软件开发模式，旨在实现软硬件协同设计、相互测试。所以作者在 MC9S12DG128 芯片上展开移植 μC/OS-Ⅱ和编写 MC9S12DG128 芯片各个模块实验例程的工作，以验证通用集成开发系统的功能。

设计与实现一个通用集成开发系统，其工作量很大。本开发系统中 MC9S12NE64、MC9S12UF32、MC9S12XDP512 芯片的核心板设计及相应芯片的实验例程编写均是在师弟师妹们的协助下完成。最后在该系统上也进行了对 M*Core 系列、C*Core 系列、ColdFire 系列和 ARM 系列 MCU 开发的实验，并通过测试，但由于时间原因，这些系列 MCU 的硬件平台均使用 Freescal 公司提供的评估板及下载工具。作者在论文撰写期间，对第一版进行多次改进与完善之后，又升级为当前的第二版。为了方便用户的开发和使用，在第二版中给出了系统的硬件和软件模块划分及详细的使用说明。第二版的软硬件平台已在我校的嵌入式单片机教学中投入使用，使用结果证明，系统的基本功能均已实现，且运行较稳定。

7.2 展望

由于本文选用 Freescale HCS12 系列 MCU 作为实现 SdIDE12 系统开发的主要目标芯片，虽然和 8 位单片机有类似之处，但是作者是第一次开发 16 位的单片机系统，而且中文资料很少，必须通过研读 Freescale 公司提供的英文手册来理解和掌握 HCS12 系列 MCU 的特性。因此，产品虽然研发成功，但是还存在许多不足，如写入模块的稳定性还需改进，调试器目前只实现初步调试。还有许多后继工作需要进一步研究：

(1) 实现功能强大的 C 语言断点调试；

(2) 尽可能的将所有 HCS12 系列 MCU 加入到 SdIDE12 中；

(3) 将 GCC 所支持的其他系列 MCU 加入该开发系统中，进一步扩大其适用范围；

当今有关嵌入式集成开发平台和集成开发环境的研究很多，也开发出了许多优秀的 IDE，但这些优秀的 IDE 也只适用于某一系列 MCU。因而，今后作者会借鉴陆续出品的优秀 IDE，进一步对 SdIDE12 进行完善和升级。相信通过不断的交流、相互帮助，一定会使 SdIDE12 成为用户满意的优秀 IDE 产品。

参考文献

[1] 赵艳秋.MCU“位”置格局有变[J].中国电子报,2005.10.

[2] 2006 Product Selection Guide,产品及开发工具选型手册[DB/OL].弘忆国际股份有限公司，2006.

[3] 吕京建,肖海桥.面向二十一世纪的嵌入式系统综述[J].电子质量,2001,(8): 10~13.

[4] 何立民 . 嵌入式系统的定义与发展历史[J].单片机与嵌入式系统应用,2004,(1): 6~8.

[5] GNU 软件的研究和发展成果[DB/OL].http://www.gnu.org.

[6] 张跃,代少升.嵌入式系统硬件协同设计与实践指南[M].2004.8

[7] Thomas D.E,Adamas J.K,Schemit H.A model and methodoloyt for hardware-software codesign.IEEE design and test of computers.1993,10(3) 6~15.

[8] MC68HC908JB8 Technical Data.Motorola Inc.2002

[9] Freescal MC9S12DG128 DataSheet[DB/OL]. 2004.

[10] Freescal MC9S12NE64 DataSheet[DB/OL]. 2004.

[11] Freescal MC9S12UF32 DataSheet[DB/OL]. 2004.

[12] Freescal MC9S12XDP512 DataSheet[DB/OL]. 2004.

[13] 清源科技编著.Protel DXP 电路设计及应用 教程[M].北京-机械工业出版社,2003

[14] Richard M.Stallman,Roland McGrath.GNU Make Version3.79[DB/OL].2000.4

[15] 陈渝,李明,杨晔等.源码开放的嵌入式系统软件分析与实践---基于 SkyEye 和 ARM 开发平台[M].北京航空航天大学出版社,2005.2

[16] 田泽.嵌入式系统开发与应用实验教程[M].北京航空航天大学出版社,2004.6

[17] Win-Bin See,Pao-Ann Hsiung,Trong-Yen Lee,Software Platform for Embedded Software Development,Volume 2968 of LNCS,2004.

[18] 姜兆义.星载嵌入式软件集成开发环境设计与关键技术研究[D].国防科学技术大学,2004.

[19] [美] George Shepherd,Scot Wingo 著,赵剑云,卿瑾译.深入解析 MFC (MFC Internals)[M].中国电力出版社,2003.10

[20] [美] Stanley B.lippman,Josee Lajoie 著,潘爱民，张丽译.C++ Primer (第

三版)中文版[M].中国电力出版社,2002.5

[21] L.P. Maguire,T.M. McGinnity,L.J.McDaid.Issues in the Development of an Integrated Environment for Embedded System Dsign.Microprocessors and Microsystems,2000.23(4): 191~198

[22] 冯钢.基于 GCC 的嵌入式系统编译器研究与开发[D].浙江大学,2004.

[23] 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.

[24] 赵民栋.嵌入式软件集成开发环境中调试器的设计与实现[D].西北工业大学, 2004.

[25] HD44780 Data Sheet[DB/OL]. http://data.eeworld.com.cn/pdf/63464_HITACHI_HD44780.html.

[26] MAX232N Data Sheet Rev 3a.Texas Instruments Incorporated[DB/OL].2000.

[27] 杨辉前,王耀南,袁小芳等.电动汽车 CAN 总线数据采集显示系统开发[J].计算机工程与应用,2006,(20): 228~232.

[28] MC33388 Data Sheet.Motorola,IncNational Semiconductor[DB/OL].

[29] 李伯成.嵌入式系统可靠性设计[M].电子工业出版社,2006.1

[30] BCGControlBar Library Professional Edition v7.3.1[DB/OL].

[31] [美]Todd D.Morton 著,严隽永译.嵌入式微控制器(Embedded Microcontrollers)[M].机械工业出版社,2005.9

[32] Turbo BDM light interface .Daniel Malik rev 1.3[DB/OL]. 2005.

[33] Motorola Inc.M68HC12 & HCS12 Microcontrollers CPU12 Reference Manual [DB/OL].2002.

[34] 邵贝贝.单片机嵌入式应用的在线开发方法[M].清华大学出版社,2004.10

[35] 刘雪兰.M68HC08 系列 MCU 嵌入式开发平台的设计与实现[D].苏州大学,2006.

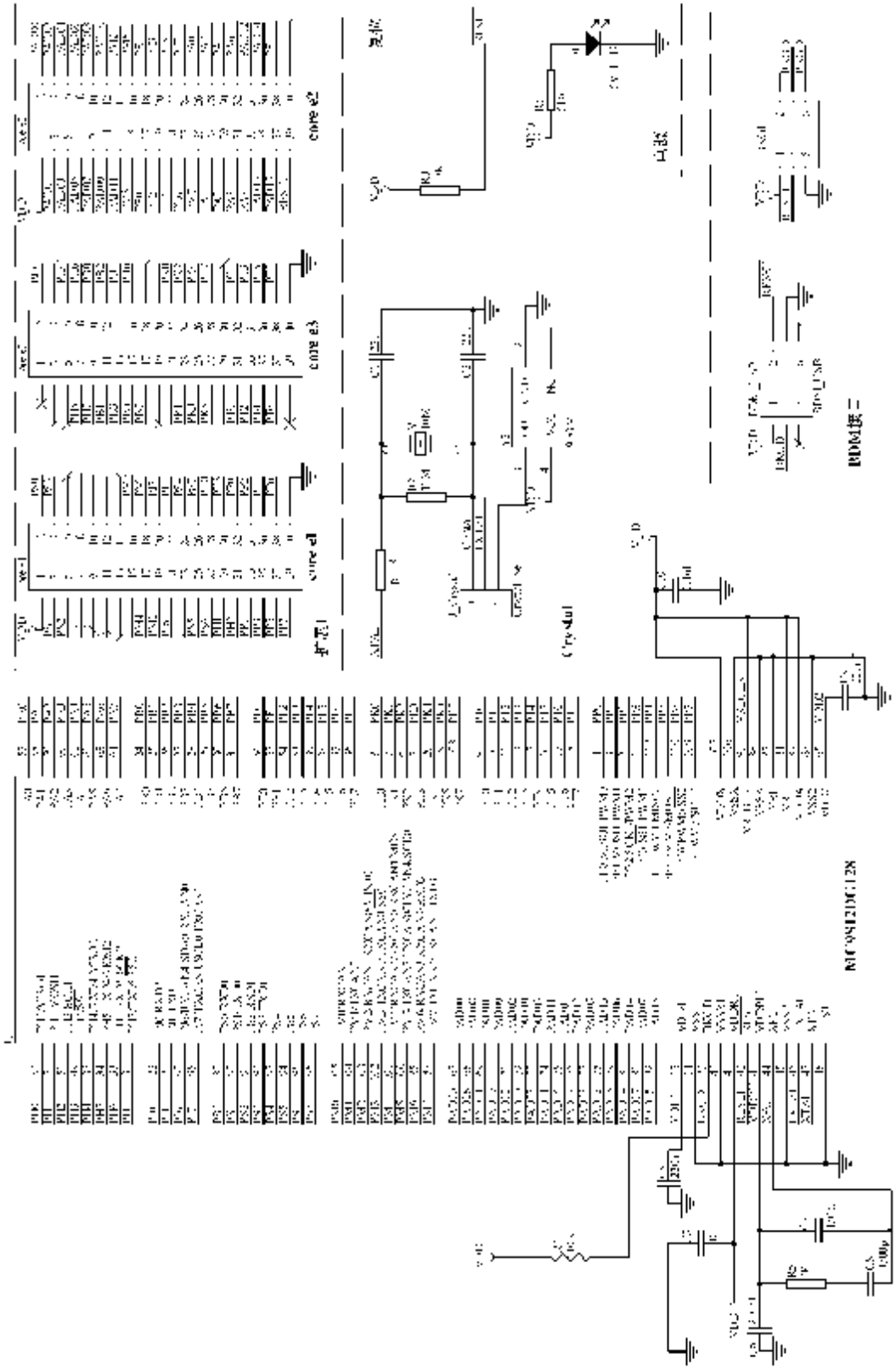
[36] [美]Jean J.labrosse 著,邵贝贝等译. 嵌入式实时操作系统 μC/OS-II(第 2 版) [M].北京航空航天大学出版社,2003.9

[37] 倪敏,周怡廷页,杨继堂.μC/OS-II 的任务切换机理及中断调度优化[J].单片机与嵌入式系统应用, 2003,(10).

[38] [美] Steven F.Barrett Daniel J.pack 著,郑扣根,唐杰等译.嵌入式系统----使用 68HC12 和 HCS12 的设计 与应用 (Embedded Systems---Design and Applications with the 68HC12 and HCS12)[M]. 电子工业出版社,2006.3

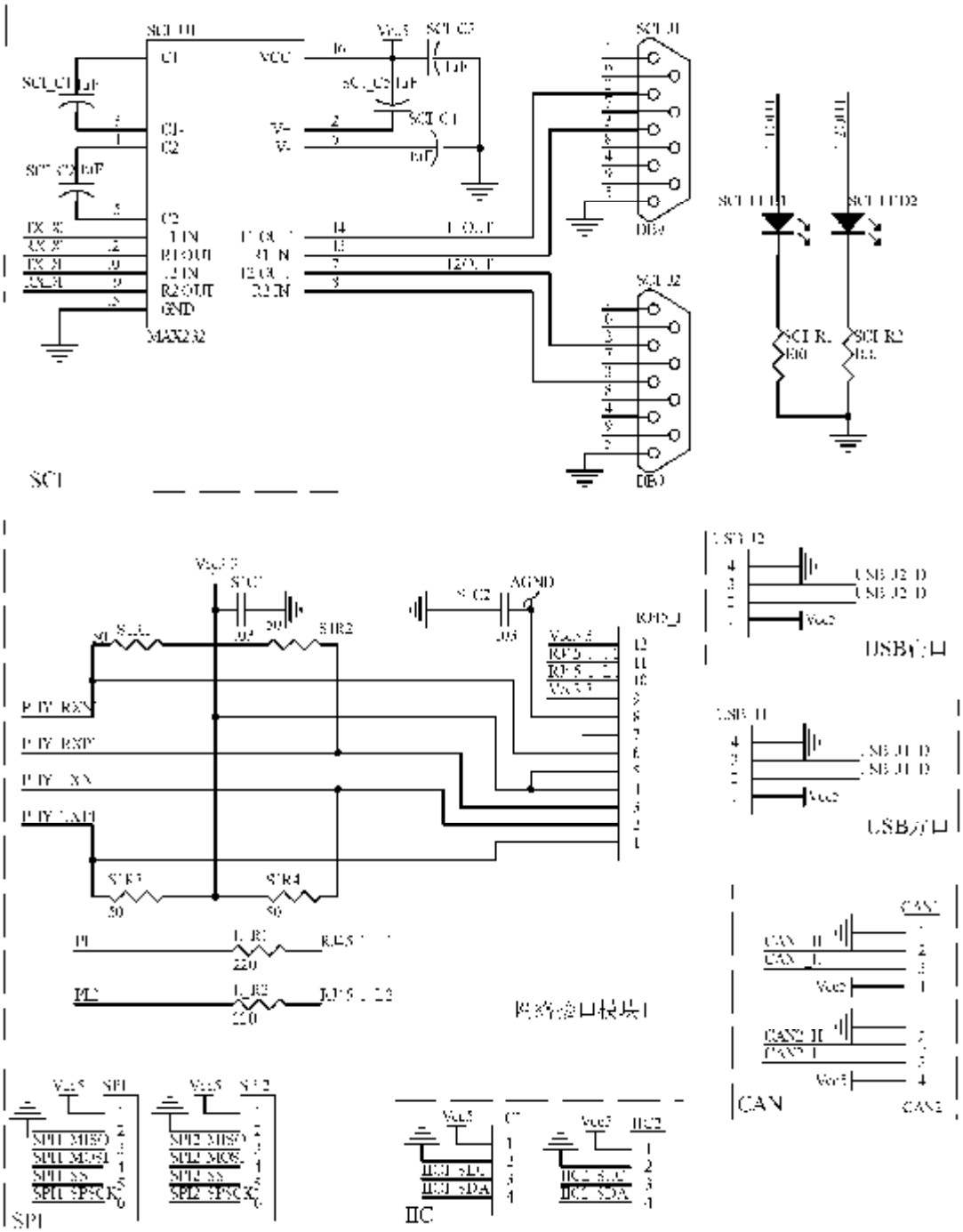
附录 A 硬件评估板原理图

A.1 MC9S12DG128 核心板原理图

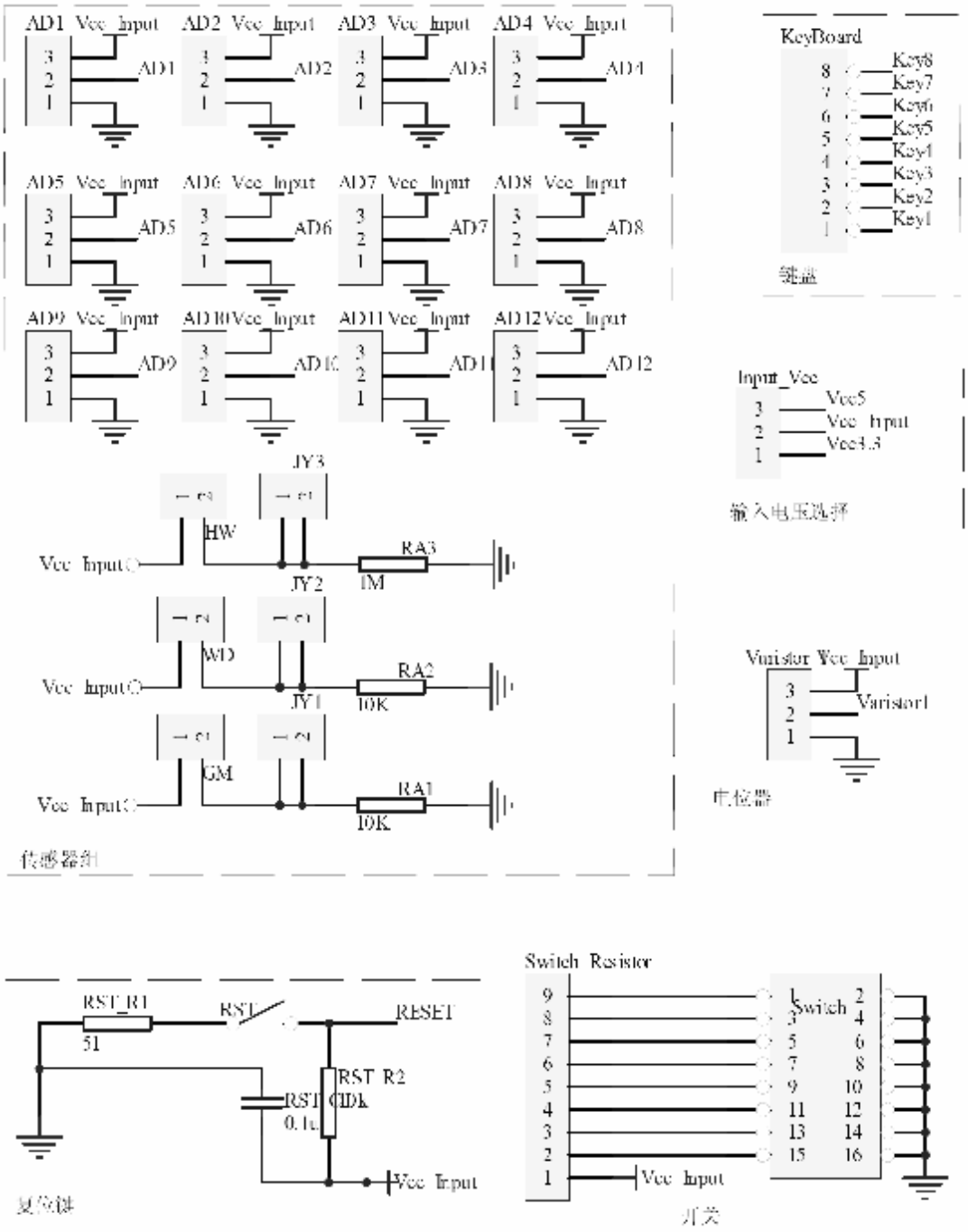


A.2 Freescale HCS12 系列 MCU 通用扩展板的原理图

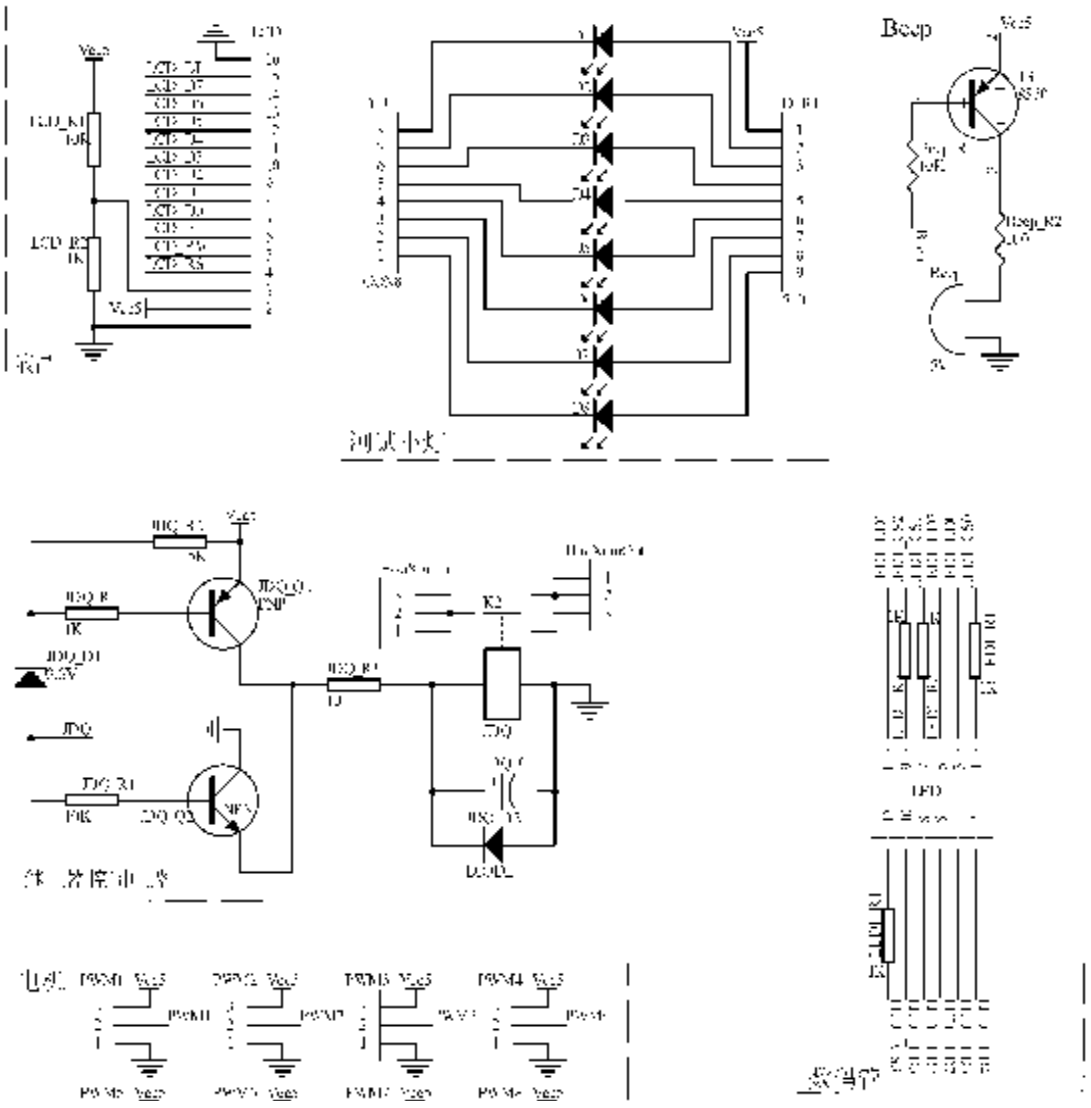
(1) 扩展板外围接口模块原理图



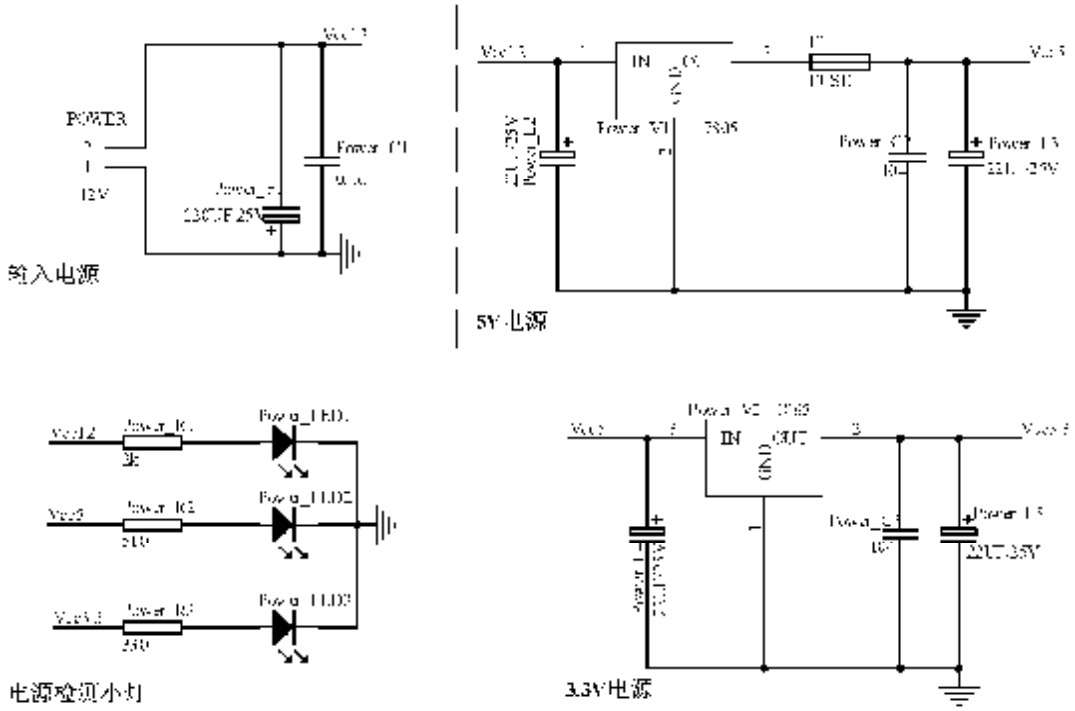
(2) 扩展板输入模块原理图



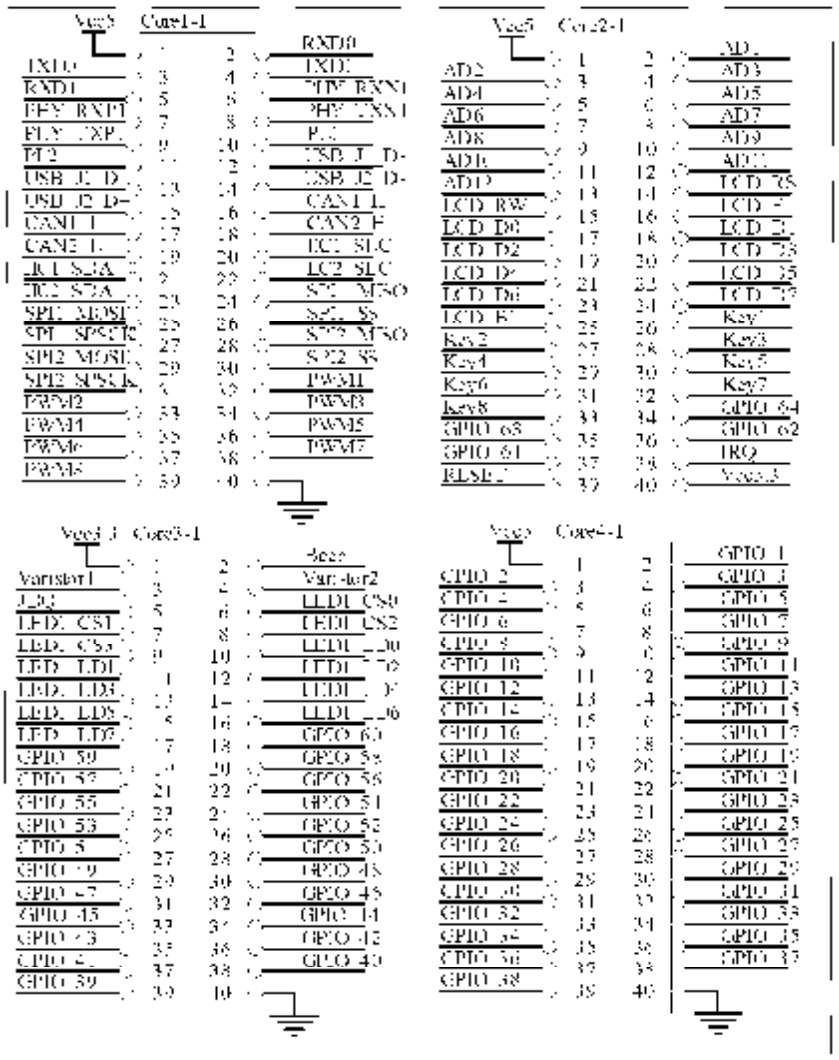
(3) 扩展板输出模块原理图

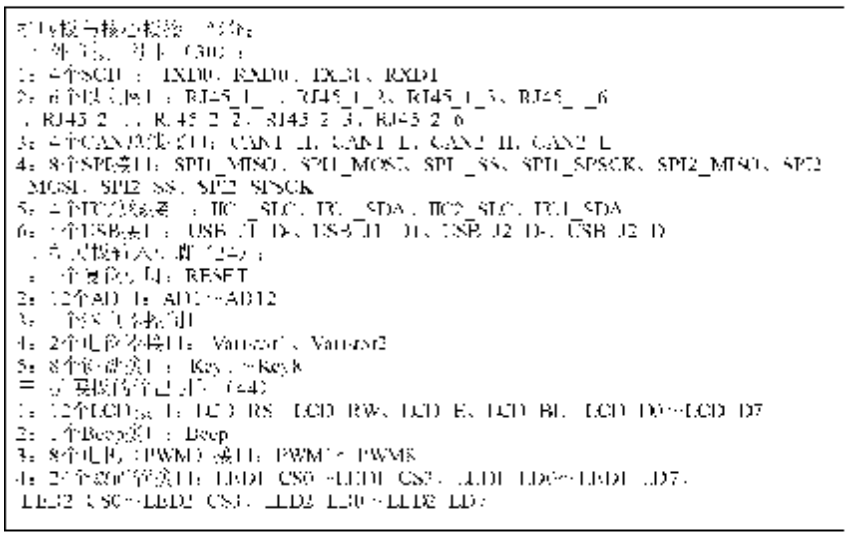


(4) 扩展板电源模块原理图

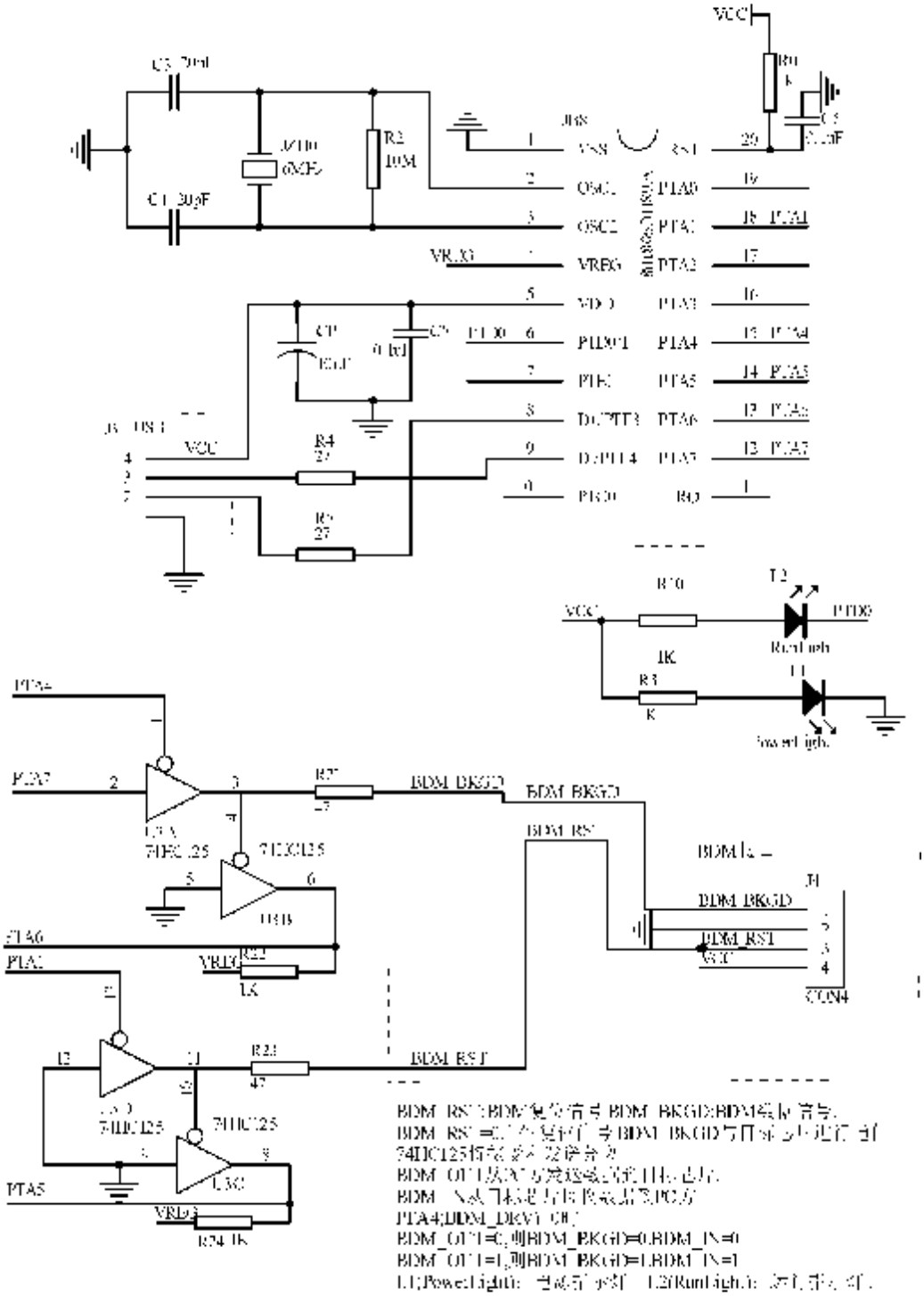


(5) 扩展板与核心板接口模块原理图





A.3 Freescale HCS12 系列 MCU 通用写入模块硬件设计原理图

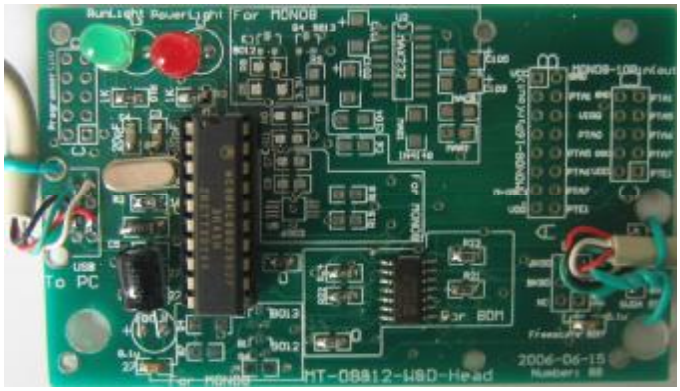


附录 B 硬件评估板实物图

MC9S12DG128 核心板、写入器及通用扩展板的实物图

MC9S12NE64 核心板

MC9S12UF32 核心板



写入模块硬件板

附录 C 规范程序实例

```
/*********************************************************************
工程文件名:PrgFrame.prj                                    *
硬件连接: MCU 的 I/O 口引脚接小灯(见"LED.c"文件说明)          *
程序描述:用 I/O 口控制小灯闪烁                              *
说      明:提供 Motorola MCU 的编程框架,供教学入门使用       *
注      意:如果延时不够长的话,会发觉灯不会闪烁,而是一直亮,这是由于人的*
           视觉的引起的.                                    *
日      期:2006.12.29                                     *
*********************************************************************/
#include "Includes.h"        //总头文件
//主函数
void main()
{
 MCUInit();          //          调用芯片初始化子函数
 LEDInit();          //          小灯控制引脚初始化
 while(1)
 {
    LED_L_A('L');    //    小灯亮
  Delay(10000);      //         延时
  LED_L_A('A');      //       小灯暗
  Delay(10000);      //         延时
 }
}
/*********************************************************************
函数名:LEDInit                                                    *
功   能:定义控制小灯的 MCU 引脚为输出,并使小灯初始为暗          *
参   数：无                                    *
返   回：无                                    *
*********************************************************************/
void LEDInit(void)
{
  Light_D |= 1<<Light_Pin;   //      令小灯引脚为输出
  Light_P |= 1<<Light_Pin;  //       初始时，小灯"暗"
}
/*********************************************************************
函数名:LED_L_A                                    *
功   能:根据 flag 的值控制小灯的亮和暗                  *
参   数:flag='A', 小灯暗;flag='L', 小灯亮                *
返   回:无                                  *
*********************************************************************/
void LED_L_A(INT8U flag)
{
  if (flag == 'A')
     Light_P |= 1<<Light_Pin;      //            小灯"暗"
  else if (flag == 'L')
     Light_P &= ~(1<<Light_Pin);   //          小灯"亮"
}
```