

ARM Cortex M4 嵌入式系统开发实践

-基于飞思卡尔 K60 系列微控制器

王宜怀 王林 编著

内容简介

本书以飞思卡尔半导体公司（原摩托罗拉半导体部）的 32 位 K60 系列微控制器中 MK60N512VMD100 为蓝本阐述嵌入式系统的软件与硬件设计。全书共 17 章，其中第 1 章为概述，阐述嵌入式系统的知识体系、学习误区、学习建议及基于硬件构件的嵌入式系统开发方法。第 2 章给出 MK60N512VMD100 硬件最小系统。第 3 章给出第一个样例程序及 CodeWarrior、IAR 工程组织，完成第一个 MK60N512VMD100 工程的入门。第 4 章阐述串行通信接口 UART，并给出第一个带中断的实例。1-4 章完成了学习一个新 MCU 完整要素（知识点）的入门。6-16 章分别给出 GPIO 的应用（键盘、LED 及 LCD）、定时器、A/D 转换、SPI、I2C、I2S、Flash、CAN、USB、SDHC、TSI、以太网及 MK60N512VMD100 其他模块等。第 17 章讲述了嵌入式操作系统有关的知识。

本书提供了所有实例源程序、辅助资料、相关芯片资料及常用软件工具。

本书可供大学有关专业的高年级学生和研究生用作教材或参考读物，也可供嵌入式系统开发与研究人员用作参考和进修资料。

前言

嵌入式计算机系统简称为嵌入式系统，其概念最初源于传统测控系统对计算机的需求。随着以微处理器（MPU）为内核的微控制器（MCU）制造技术的不断进步，计算机领域在通用计算机系统与嵌入式计算机系统这两大分支分别得以发展。通用计算机已经在科学计算、事物管理、通信、日常生活等各个领域产生重要的影响。在后 PC 时代，嵌入式系统的广泛应用将是计算机发展的重要特征。一般来说，嵌入式系统的应用范围可以粗略分为两大类：一类是电子系统的智能化（如工业控制、现代农业、家用电器、汽车电子、测控系统、数据采集、传感网应用等）；另一类是计算机应用的延伸（如手机、电子书、通信、网络、计算机外围设备等）。不论如何分类，嵌入式系统的技术基础是不变的，即要完成一个以 MCU 为核心的嵌入式系统应用产品设计，需要有硬件、软件及行业领域相关知识。但是，随着嵌入式系统中软件规模日益增大，对嵌入式底层驱动软件的封装提出了更高的要求，可复用性与可移植性受到特别的关注，嵌入式软硬件构件化开发方法逐步被业界所重视。

本书基本思想

本书以嵌入式硬件构件与底层软件构件设计为主线，按照嵌入式软件工程的要求，以飞思卡尔半导体公司（原摩托罗拉半导体部）的 32 位 K60 系列中 MK60N512VMD100 微控制器为蓝本阐述嵌入式系统的软件与硬件设计。并阐述嵌入式操作系统相关知识。

我从事单片机与嵌入式系统科研与教学工作是从 1991 年开始的。1991-1999 年间，使用 MCS-51 系列 MCU。2000 年至现在，一直使用飞思卡尔（2004 年以前是摩托若拉半导体部）的 MCU。十多年来，陆续以飞思卡尔的 HC08/S08（8 位）、S12/S12X（16 位）、ColdFire（32 位）、M*Core（32 位，该内核转给中国后称为 C*Core）进行科研开发与教学工作，并以这些 MCU 为蓝本先后写了一些嵌入式应用技术入门方面的书，得到了大多数读者的肯定，深受感动。2010-2011 年，苏州大学嵌入式团队的工作重点是进行 ARM Cortex-M4 核 Kinetis 系列 MCU（K60）、新型 Zigbee 芯片 MC1323x、DSC 芯片 MC56F825x 等方面的工作，这些工作成果也将会逐步与读者分享。在写书方面，多年来一直在探索如何能够使读者不误入歧途，如何能够快速入门，如何能够规范编程，如何能够由浅入深、循序渐进，如何能够使读者打好嵌入式硬件与软件基础。为此从以下几点把握写作：（1）把与芯片无关的通用知识分离出来，从涉及底层编程角度对基本原理进行简明扼要的阐述，分别放入相应章节的前面或网上光盘中。这些知识主要包括通用 I/O、串行通信、键盘编码原理、LED 扫描原理、SPI、PWM、USB、I²C、CAN、A/D、D/A、嵌入式以太网等。并在各书中基本保持不变。这一点是接受了飞思卡尔全球大学计划负责人 Andy Mastronardi 先生的建议，经过几年不断修改完善，可把通用部分斟酌得更好一些。也使得 8 位、16 位、32 位的书风格保持一致。新的芯片出来后，书的修改只要更新与芯片的相关部分。（2）硬件相关的部分，采用了硬件构件思想，制定了一些基本规范，对底层驱动进行构件化封装，提高了可复用性与可移植性。使程序结构更加清晰，初学者可以“先使用、后理解”。（3）不论是 8 位、16 位、32 位，也不论是哪个芯片，从编程角度，把与硬件相关的共性和与硬件无关的共性分别抽象出来，力求做到，硬件相关部分风格一致，硬件无关部分程序一致。这样便于融会贯通，不再纠结芯片位数、操作系统等问题。

关于飞思卡尔微控制器

飞思卡尔半导体是全球最大半导体公司之一,在微控制器领域长期居全球市场领先地位,以高可靠性获得业界的一致赞誉。该公司的微控制器产品系列齐全,由不同位数(如8位、16位、32位等)、不同封装形式(如DIP、SOIC、QFP、LQFP、BGA等)、不同温度范围(0~70℃、-40~85℃、-40~105℃、-40~125℃等)、所含模块不同等构成了庞大的产品系列。飞思卡尔的S08(8位)、S12/S12X(16位)、ColdFire(32位)、ARM Cortex(32位)等系列MCU,广泛地应用于汽车电子、消费电子、工业控制、网络和无线市场等嵌入式系统各个领域。系列齐全的微控制器产品,为嵌入式系统各种应用提供了选择与解决方案,使得用户可以各取所需。不论是电子系统智能化还是计算机应用延伸的嵌入式应用设计,无论需要怎样的系统功能和集成度,总能从飞思卡尔庞大产品系列中选取一款合适的芯片进行应用开发。这正是嵌入式系统产品设计者所期望的,也节省了嵌入式学习者的时间,可以加快开发进度,提高开发质量。

本书特点

2009年,我撰写了《基于32位ColdFire构建嵌入式系统》一书,2010年又撰写了《嵌入式技术基础与实践(第2版)》。两书中系统阐述和应用了嵌入式构件开发思想,本书秉承这些工作,按照“通用知识—芯片编程结构概要—基本编程方法—底层驱动构件封装—应用方法与举例”的线条,逐步阐述电子系统智能化嵌入式应用的软件与硬件设计。

(1) 把握通用知识与芯片相关知识之间的平衡。书中对于嵌入式“通用知识”的基本原理,以应用为立足点,进行语言简洁、逻辑清晰的阐述,同时注意与芯片相关知识之间的衔接,使读者在更好地理解基本原理的基础上,理解芯片应用的设计,同时反过来,加深对通用知识的理解。

(2) 把握硬件与软件的关系。嵌入式系统是软件与硬件的综合体,嵌入式系统设计是一个软件、硬件协同设计的工程,不能像通用计算机那样,软件、硬件完全分开来看。特别是对电子系统智能化嵌入式应用来说,没有对硬件的理解就不可能写好嵌入式软件,同样没有对软件的理解也不可能设计好嵌入式硬件。因此,本书注重把握硬件知识与软件知识之间的关系。

(3) 对底层驱动进行构件化封装。书中对每个模块均给出根据嵌入式软件工程基本原则并按照构件化封装要求编制底层驱动程序,同时给出详细、规范的注释及对外接口,为实际应用提供底层构件,方便移植与复用,可以为读者进行实际项目开发节省大量时间。

(4) 设计合理的测试用例。书中所有源程序均经测试通过,并保留测试用例在本书的随书光盘中,避免了因例程的书写或固有错误给读者带来烦恼。这些测试用例,也为读者验证与理解带来方便。

(5) 随书光盘提供了所有模块完整的底层驱动构件化封装程序、文档与测试用例,同时随书光盘中还包含芯片参考手册、写入器安装与使用方法、工具软件(如开发环境、程序写入与读出软件、串口调试工具等)、有关硬件原理图及其他技术资料。

(6) 提供硬件评估版、写入调试器,并给出单独进行程序写入与读出的软件工具,方便读者进行实践与应用。

本书主要内容

本书以飞思卡尔半导体的32位K60系列微控制器中MK60N512VMD100为蓝本阐述嵌入式系统的软件与硬件设计。全书共17章,其中第1章为概述,阐述嵌入式系统的知识体系、学习误区、学习建议及基于硬件构件的嵌入式系统开发方法。第2章给出MK60N512VMD100硬件最小系统。第3章给出第一个样例程序及CodeWarrior、IAR工程

组织，完成第一个 MK60N512VMD100 工程的入门。第 4 章阐述串行通信接口 UART，并给出第一个带中断的实例。1-4 章完成了学习一个新 MCU 完整要素（知识点）的入门。6-16 章分别给出 GPIO 的应用（键盘、LED 及 LCD）、定时器、A/D 转换、SPI、I2C、I2S、Flash、CAN、USB、SDHC、TSI、以太网及 MK60N512VMD100 其他模块等。第 17 章讲述了嵌入式操作系统有关的知识。

致谢

本书除封面署名作者外，还有苏州大学计算机科学与技术学院嵌入式应用方向研究生姚丹丹、李翠霞、朱乐乐、冯上栋、石晶、苏勇等协助书稿整理及程序调试工作，他们卓有成效的工作，使本书更加实用。飞思卡尔半导体有限公司的 Andy Mastronardi 先生、马莉女士一直关心支持苏州大学飞思卡尔嵌入式系统研发中心的建设，为本书的撰写提供了硬件及软件资料，并提出了许多宝贵建议。飞思卡尔半导体有限公司的许多技术人员提供了技术支持。北京航空航天大学出版社的董立娟编辑为本书的出版付出了大量细致的工作。在此一并表示诚挚的谢意。

鉴于作者水平有限，书中难免存在不足和错误之处，恳望读者提出宝贵意见和建议，以便再版时改进。

王宜怀

2011 年 4 月于苏州大学

网上光盘资料目录结构

- [-] 文件夹 SD-WYH-MK60N512VMD100Book-CD (VO. 1-2011)
 - [-] 文件夹 01-整体资料
 - 文件夹 0101-开发环境CodelWarrior10.1
 - 文件夹 0102-开发环境IAR for ARM 6.10
 - 文件夹 0103-(ARM Cortex M4)(写入器)安装与使用
 - 文件夹 0104-仪器用户手册
 - 文件夹 0105-ARM Cortex M4及K60参考手册
 - 文件夹 0106-SD-WYH-MK60N512VMD100BOOK(课件VO. 1)
 - 文件夹 0107-简介、前言及目录
 - [-] 文件夹 02-分章MCU源程序
 - [-] 文件夹 SD-WYH-MK60N512VMD100BOOK-Program (VO. 1)-2011
 - 文件夹 Ch03-GPIO (Light)_C
 - 文件夹 Ch04-SCI_C
 - 文件夹 Ch05-KBI-LED-LCD_C
 - 文件夹 Ch06-Timer_C
 - 文件夹 Ch07-AD_C
 - 文件夹 Ch08-SPI_C
 - 文件夹 Ch09-I2C-I2S_C
 - 文件夹 Ch10-Flash_C
 - 文件夹 Ch11-CAN_C
 - 文件夹 Ch12-USB_C
 - 文件夹 Ch13-SD_C
 - 文件夹 Ch14-TSI_C
 - 文件夹 Ch15-ENET_C
 - 文件夹 Ch16-Other_C
 - 文件夹 Ch17-OS_C
 - [-] 文件夹 03-PC机源程序
 - 文件夹 ADC-C#
 - 文件夹 SCI-C#
 - 文件夹 Timer-C#
 - [-] 文件夹 04-PC机免费工具
 - 文件夹 pcb_hanzi
 - 文件夹 USB辅助工具
 - 文件夹 USB转串口驱动
 - 文件夹 USB-转串口驱动CH341SER
 - [-] 文件夹 05-分章阅读材料
 - 文件夹 第01章 (概述)阅读资料
 - [+] 文件夹 第02章 MCU阅读资料
 - 文件夹 第03章 (第一个程序)阅读资料
 - 文件夹 第04章 (构件开发方法)阅读资料
 - 文件夹 第05章 (SCI)阅读资料
 - 文件夹 第06章 (键盘、LED与LCD)阅读资料
 - 文件夹 第08章 AD转换模块阅读资料
 - 文件夹 第11章 其他阅读资料
 - [+] 文件夹 07-其他材料

ARM CORTEX M4嵌入式系统开发实践.....	I
-基于飞思卡尔K60系列微控制器	I
第1章 概述.....	1
1.1 嵌入式系统定义、由来及特点	1
1.1.1 嵌入式系统的定义	1
1.1.2 嵌入式系统的由来及其与微控制器的关系	1
1.1.3 嵌入式系统的特点	3
1.2 嵌入式系统开发所遇到的若干问题	4
1.3 嵌入式硬件构件的基本思想与应用方法	4
1.4 基于硬件构件的嵌入式系统硬件电路设计	5
1.4.1 设计时需要考虑的基本问题	5
1.4.2 硬件构件化电路原理图绘制的简明规则	7
1.4.3 实验 PCB 板设计的简明规则	9
1.5 基于硬件构件的嵌入式底层软件构件的编程方法	13
1.5.1 嵌入式硬件构件和软件构件的层次模型	13
1.5.2 底层构件的实现方法与编程思想	14
1.5.3 硬件构件及底层软件构件的重用与移植方法	14
第2章 KINETIS概述与MK60N512VMD100硬件最小系统.....	17
2.1 ARM 公司的发展史及 ARM 架构的发展	17
2.2 Kinetis 系列微处理器概述	24
2.3 Kinetis 系列微控制器存储器映像与编程结构	26
2.3.1 K60 系列 MCU 性能概述与内部结构简图	27
2.3.2 K60 系列存储器映像	29
2.4 K60 的引脚功能与硬件最小系统	31
2.4.1 K60 的引脚功能	31
2.4.2 K60 硬件最小系统	39
2.4.3 硬件最小系统测试方法	42
第3章 第一个样例程序及工程组织.....	43
3.1 通用 I/O 接口基本概念及连接方法	43
3.2 MK60N512VMD100 的 GPIO	44
3.3 开发环境与 JTAG 写入器	45
3.3.1 IAR 开发环境简介与基本使用方法	45
3.3.2 CW 开发环境简介与基本使用方法	46
3.3.3 JTAG 写入器	47
3.3.4 MK60N512VMD100 硬件核心板	47
3.4 IAR 工程文件组织	48
3.4.1 工程文件的组织	49
3.4.2 初始化相关文件	51
3.5 CW 工程文件组织	55
3.5.1 工程文件的组织	55
3.5.2 初始化相关文件	55

3.6 第一个应用实例：控制小灯闪烁.....	56
3.6.1 GPIO 构件	57
3.6.2 Light 构件	59
3.6.3 Light 测试工程主程序	61
3.7 理解第一个 C 工程的执行过程	62
第4章 异步串行通信.....	63
4.1 异步串行通信的基础知识.....	63
4.1.1 基本概念.....	63
4.1.2 RS-232C 总线标准.....	65
4.1.3 电平转换电路原理.....	66
4.2 MK60N512VMD100 的 UART 模块功能描述.....	67
4.3 MK60N512VMD100 的 UART 模块的编程结构.....	69
4.4 基于构件方法的 UART 编程	74
4.4.1 UART 构件的函数原型设计	74
4.4.2 UART 构件的头文件	75
4.4.3 UART 构件的源程序文件	77
4.4.4 UART 构件的测试工程	82
4.5 K60 第一个带有中断功能的实例.....	84
4.6 进一步讨论.....	87
4.6.1 流控制与 Break 信号	87
4.6.2 延长串口通信的距离.....	87
4.6.3 串口的扩展.....	87
第5章 GPIO的应用实例——键盘、LED与LCD.....	88
5.1 键盘技术概述.....	88
5.1.1 键盘模型及接口.....	88
5.1.2 键盘编程的基本问题.....	88
5.1.3 键盘构件设计与测试实例.....	88
5.2 LED 技术概述.....	88
5.2.1 扫描法 LED 显示编程原理.....	88
5.2.2 LED 构件设计与测试实例.....	88
5.3 LCD 技术概述.....	88
5.3.1 LCD 的特点和分类.....	88
5.3.2 点阵字符型液晶显示模块.....	88
5.3.3 HD44780.....	88
5.3.4 LCD 构件设计与测试实例.....	88
第6章 定时器相关模块.....	89
6.1 计数器/定时器的基本工作原理.....	89
6.2 可编程延时模块 PDB.....	89
6.2.1 PDB 工作原理	89
6.2.2 PDB 相关寄存器.....	89
6.2.3 PDB 构件设计及测试实例.....	89
6.3 Flex 定时器 FTM	89

6.3.1 FTM 工作原理.....	89
6.3.2 FTM 相关寄存器.....	89
6.3.3 FTM 中断	89
6.3.4 FTM 构件设计及测试实例.....	89
6.4 周期中断定时器 PIT.....	89
6.4.1 PIT 工作原理.....	89
6.4.2 PIT 相关寄存器.....	89
6.4.3 PIT 构件设计及测试实例.....	89
6.5 低功耗定时器 LPTMR	90
6.5.1 LPTMR 工作原理	90
6.5.2 LPTMR 相关寄存器	90
6.5.3 LPTMR 构件设计及测试实例.....	90
6.6 载波调制发射器 CMT	90
6.6.1 CMT 工作原理	90
6.6.2 CMT 相关寄存器	90
6.6.3 CMT 中断.....	90
6.6.4 CMT 构件设计及测试实例	90
6.7 实时时钟 RTC	90
6.7.1 RTC 工作原理	90
6.7.2 RTC 相关寄存器	90
6.7.3 RTC 构件设计及测试实例	90
第7章 A/D与D/A.....	91
7.1 A/D 和 D/A 转换的基本问题.....	91
7.2 A/D 转换模块.....	91
7.2.1 A/D 转换模块寄存器	91
7.2.2 A/D 转换构件设计及测试实例	91
7.3 D/A 转换模块.....	91
7.3.1 D/A 转换模块寄存器	91
7.3.2 D/A 转换构件设计及测试实例	91
7.4 比较器 CMP 概述	91
第8章 SPI.....	92
8.1 SPI 的基本工作原理.....	92
8.1.1 SPI 概述.....	92
8.1.2 SPI 的数据传输	92
8.1.3 SPI 模块的时序.....	92
8.2 SPI 模块的编程基础.....	92
8.2.1 SPI 模块的引脚.....	92
8.2.2 SPI 模块的寄存器	92
8.2.3 SPI 编程基本方法.....	92
8.3 SPI 构件设计及测试实例	92
第9章 I2C与I2S	93
9.1 I2C 总线概述.....	93

9.1.1 I2C 总线特点.....	93
9.1.2 I2C 总线标准的发展历史.....	93
9.1.3 I2C 总线的相关术语.....	93
9.2 I2C 总线工作原理.....	93
9.2.1 总线上数据的有效性.....	93
9.2.2 总线上的信号.....	93
9.2.3 总线上数据的传输格式.....	93
9.2.4 总线寻址约定.....	93
9.3 I2C 模块的编程基础.....	93
9.3.1 I2C 模块寄存器.....	93
9.3.2 I2C 模块编程基本方法.....	93
9.4 I2C 构件设计及测试实例.....	93
9.5 I2S 概述.....	93
9.5.1 I2S 的特点.....	93
9.5.2 I2S 操作模式.....	93
9.5.3 I2S 的相关寄存器定义.....	93
9.6 I2S 的构件化设计与测试实例.....	94
第10章 FLASH.....	95
10.1 Flash 内存控制寄存器 FMC.....	95
10.1.1 FMC 特点.....	95
10.1.2 FMC 相关寄存器.....	95
10.1.3 FMC 的功能.....	95
10.2 Flash 存储器概述与编程模式.....	95
10.2.1 Flash 存储器编程的基本概念.....	95
10.2.2 Flash 存储器的编程寄存器.....	95
10.2.3 Flash 存储器的编程过程.....	95
10.2.4 Flash 存储器测试实例.....	95
10.3 FlexBUS 概述.....	95
10.3.1 FlexBUS 的特点.....	95
10.3.2 FlexBUS 相关的寄存器.....	95
10.3.3 FlexBUS 的功能.....	95
10.4 EzPort 概述.....	95
10.4.1 EzPort 的特点.....	95
10.4.2 EzPort 信号描述.....	95
10.4.3 EzPort 命令.....	95
10.5 Flash 存储器的保护特性和安全性.....	96
10.5.1 周期性冗余检测 CRC.....	96
10.5.2 存储器映像密码加速单元 MMCAU.....	96
10.5.3 随机数操作 RNGB.....	96
第11章 CAN模块FLEXCAN.....	97
11.1 CAN 总线通用知识.....	97
11.1.1 CAN 总线协议的历史概况.....	97
11.1.2 CAN 硬件系统的典型电路.....	97

11.1.3 CAN 总线的有关基本概念.....	97
11.1.4 帧结构.....	97
11.1.5 位时间.....	97
11.2 K60 的 CAN 模块概述与编程结构.....	97
11.2.1 CAN 特性	97
11.2.2 操作模式.....	97
11.2.3 CAN 模块的内存映像及寄存器定义.....	97
11.2.4 CAN 报文缓冲区	97
11.3 K60 的 CAN 模块报文发送与接收函数设计.....	97
11.3.1 数据帧发送/接收.....	97
11.3.2 远程帧发送与接收.....	97
11.3.3 仲裁处理、匹配处理及报文缓冲区管理	97
11.4 K60 的 CAN 模块编程实例.....	97
11.4.1 初始化函数设计	97
11.4.2 K60 的 CAN 模块构件化设计及测试实例.....	98
第12章 USB 2.0 编程.....	99
12.1 USB 基本概念及硬件特性	99
12.1.1 USB 特性	99
12.1.2 USB 相关基本概念	99
12.1.3 USB 的物理特性.....	99
12.2 USB 的通信协议.....	99
12.2.1 USB 基本通信单元：包	99
12.2.2 USB 通信中的事务处理	99
12.2.3 从设备的枚举看 USB 数据传输	99
12.3 K60 的 USB 模块功能简介	99
12.3.1 K60 的 USB 模块功能简介	99
12.3.2 K60 的 USB 模块主要寄存器介绍	99
12.4 K60 作为 USB 从机的开发方法	99
12.4.1 PC 端 USB 设备驱动程序的选择及基本原理	99
12.4.2 PC 作为 USB 主机的程序设计	99
12.4.3 K60 作为 USB 从机的程序设计	99
12.5 K60 作为 USB 主机的开发方法	99
12.5.1 K60 作为 USB 主机的基本功能	99
12.5.2 USB 主机与 USB 设备通信	100
12.6 K60 的 USB 设备电量检测模块 USBDCD	100
12.6.1 USBDCD 概述.....	100
12.6.2 USBDCD 的内存映射与寄存器定义.....	100
12.6.3 USBDCD 构件化设计与测试实例.....	100
12.7 K60 的 UAB 电压调节器.....	100
12.7.1 电压调节器特征.....	100
12.7.2 电压调节器操作模式.....	100
第13章 大容量SD存储卡SDHC	101
13.1 SDHC 基本概念及硬件特性	101

13.1.1 SD 概述.....	101
13.1.2 SD 相关基本概念.....	101
13.1.3 SD 的物理特性.....	101
13.2 SD 的通信协议.....	101
13.2.1 SD 基本通信单元.....	101
13.2.2 SD 通信中的事务处理.....	101
13.2.3 SD 数据传输.....	101
13.3 K60 的 SD 模块基本编程方法.....	101
13.3.1 K60 的 SD 模块功能简介.....	101
13.3.2 K60 的 SD 模块存储器映像与寄存器定义.....	101
13.3.3 K60 的 SD 模块构件化设计与测试实例.....	101
第14章 TSI.....	102
14.1 TSI 概述.....	102
14.1.1 TSI 特点.....	102
14.1.2 TSI 的操作模式.....	102
14.1.3 TSI 信号描述.....	102
14.2 TSI 编程.....	102
14.2.1 TSI 的相关寄存器定义.....	102
14.2.2 TSI 的功能描述.....	102
14.2.3 TSI 的构件化设计与测试实例.....	102
第15章 基于K60的嵌入式以太网.....	103
15.1 嵌入式以太网相关基础知识.....	103
15.1.1 以太网的由来与协议模型.....	103
15.1.2 以太网中主要物理设备.....	103
15.1.3 IEEE 1588 概述.....	103
15.1.4 相关名词解释.....	103
15.2 K60 以太网概述.....	103
15.2.1 K60 以太网特性.....	103
15.2.2 K60 以太网外部引脚说明.....	103
15.2.3 K60 以太网存储映像与寄存器带那个一.....	103
15.3 链路层编程.....	103
15.3.1 MAC 帧格式.....	103
15.3.2 MAC 帧的接收与发送.....	103
15.3.3 MAC 帧收发测试实例.....	103
15.4 网络层及更高层编程.....	103
15.4.1 Ipv4 与 Ipv6 简介.....	103
15.4.2 ICMP 简介.....	103
15.4.3 UDP 简介.....	103
15.4.4 TCP 简介.....	104
15.4.5 测试实例.....	104
15.5 FIFO.....	104
15.5.1 FIFO 概述.....	104
15.5.2 FIFO 的接收与发送.....	104

15.5.3 FIFO 的保护机制	104
15.5.4 FIFO 测试实例	104
15.6 PHY 管理接口与以太网接口	104
15.6.1 MDIO 简介	104
15.6.2 以太网接口的发送与接收	104
15.7 K60 以太网模块的其他功能	104
15.7.1 全双工流控制操作	104
15.7.2 魔术包检测	104
15.7.3 IP 加速器控制	104
15.7.4 复位与停止控制	104
15.7.5 遗留缓冲区描述符	104
15.7.6 增强缓冲区描述符	104
第16章 系统时钟与其他功能模块	105
16.1 时钟模块	105
16.2 芯片配置模块	105
16.2.1 芯片配置模块简介	105
16.2.2 芯片配置模块寄存器定义	105
16.3 电源管理模块	105
16.3.1 电源模式	105
16.3.2 低功耗模式	105
16.4 端口控制与中断模块	105
16.5 复位与启动模块	105
16.6 杂项控制模块	105
16.7 交叉开关模块	105
16.8 看门狗	106
第17章 操作系统的移植	106

第1章 概述

作为全书导引，本章主要知识点有：①简要给出嵌入式系统定义、由来及特点；②讨论嵌入式开发中的硬件、软件的可复用与可移植性问题；③提出嵌入式硬件构件的基本思想，阐述基于硬件构件的嵌入式系统开发方法；④给出基于硬件构件的嵌入式系统硬件电路设计方法与嵌入式底层软件构件的编程方法。

1.1 嵌入式系统定义、由来及特点

1.1.1 嵌入式系统的定义

嵌入式系统（Embedded system）有多种多样的定义，但本质是相同的。本书关于嵌入式系统的定义取自美国 CMP Books 出版的 Jack Ganssle 和 Michael Barr 著作《Embedded System Dictionary》¹。

嵌入式系统的定义：一种计算机硬件和软件的组合，也许还有机械装置，用于实现一个特定功能。在某些特定情况下，嵌入式系统是一个大系统或产品的一部分。世界上第一个嵌入式系统是 1971 年 Busicom 公司用 Intel 单芯片 4004 微处理器完成的商用计算器系列。该词典还给出了嵌入式系统的一些示例：微波炉、手持电话、计算器、数字手表、录像机、巡航导弹、GPS 接收机、数码相机、传真机、跑步机、遥控器和谷物分析仪等，难以尽数。通过与通用计算机的对比可以更形象地理解嵌入式系统的定义。该词典给出的通用计算机定义是：计算机硬件和软件的组合，用作通用计算平台。PC、MAC 和 Unix 工作站是最流行的现代计算机。

我国《国家标准 GB/T 5271 信息技术词汇—嵌入式系统与单片机》部分，给出的嵌入式系统定义是：置入应用对象内部起操作控制作用的专用计算机系统。

国内对嵌入式系统定义曾进行过广泛讨论，有许多不同说法。其中嵌入式系统定义的涵盖面问题是主要争论焦点之一。例如，有的学者认为不能把手持电话叫嵌入式系统，而只能把其中起控制作用的部分叫嵌入式系统，而手持电话可以称为嵌入式系统的应用产品。其实，这些并不妨碍人们对嵌入式系统的理解，所以不必对定义感到困惑。有些国内学者特别指出，在理解嵌入式系统定义时，不要把嵌入式系统与嵌入式系统产品相混淆。实际上，从口语或书面语言角度，不区分“嵌入式系统”与“嵌入式系统产品”，只要不妨碍对嵌入式系统的理解就没有关系。

为了更清楚阐述嵌入式系统特点，首先介绍大多数嵌入式系统的核心部件—MCU（微控制器）的基本概念。

1.1.2 嵌入式系统的由来及其与微控制器的关系

1. MCU（微控制器）的基本含义

MCU 是单片微型计算机（单片机）的简称，早期的英文名是 Single-chip Microcomputer，后来大多数称之为微控制器（Microcontroller）或嵌入式计算机（Embedded computer）。现

¹ 中译本：Jack Ganssle 等著，马广云等译，《英汉双解嵌入式系统词典》，北京航空航天大学出版社，2006 年。

在 Microcontroller 已经是计算机中一个常用术语，但在 1990 年代之前，大部分英文词典并没有这个词。我国学者一般使用中文“单片机”一词，而缩写使用“MCU”²。所以本书后面的简写一律以 MCU 为准。**MCU 的基本含义是：在一块芯片上集成了中央处理单元(CPU)、存储器 (RAM/ROM 等)、定时器/计数器及多种输入输出 (I/O) 接口的比较完整的数字处理系统。**图 1-1 给出了典型的 MCU 组成框图。

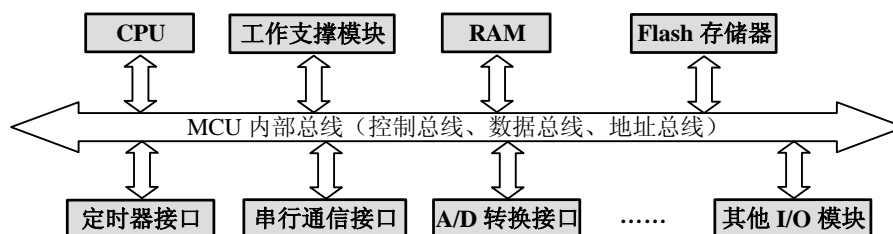


图 1-1 一个典型的 MCU 内部框图

MCU 是在计算机制造技术发展一定阶段的背景下出现的，它使计算机技术从科学计算领域进入到智能化控制领域。从此，计算机技术在两个重要领域——通用计算机领域和嵌入式 (Embedded) 计算机领域都获得了极其重要的发展，为计算机的应用开辟了更广阔的空间。

就 MCU 组成而言，虽然它只是一块芯片，但包含了计算机的基本组成单元，仍由运算器、控制器、存储器、输入设备、输出设备五部分组成，只不过这些都集成在一块芯片上，这种结构使得 MCU 成为具有独特功能的计算机。

2. 嵌入式系统的由来

通俗地说，计算机是因科学家需要一个高速的计算工具而产生的。直到二十世纪七十年代，电子计算机在数字计算、逻辑推理及信息处理等方面表现出非凡的能力。在通信、测控与数据传输等领域，人们对计算机技术给予了更大的期待。这些领域的应用与单纯的高速计算要求不同，主要表现在：直接面向控制对象；嵌入到具体的应用体中，而非计算机的面貌出现；能在现场连续可靠地运行；体积小，应用灵活；突出控制功能，特别是对外部信息的捕捉与丰富的输入输出功能等。由此可以看出，满足这些要求的计算机与满足高速数值计算的计算机是不同的。因此，一种称之为 MCU 或微控制器的技术得以产生并发展。为了区分这两种计算机类型，通常把满足海量高速数值计算的计算机称为通用计算机系统，而把嵌入到实际应用系统中，实现嵌入式应用的计算机称之为嵌入式计算机系统，简称嵌入式系统。

3. 嵌入式系统与 MCU 的关系

何立民先生说：“有些人搞了十多年的 MCU 应用，不知道 MCU 就是一个最典型的嵌入式系统”³。实际上，MCU 是在通用 CPU 基础上发展起来的，MCU 具有体积小、价格低、稳定可靠等优点，它的出现和迅猛发展，是控制系统领域的一场技术革命。MCU 以其较高的性能价格比、灵活性等特点，在现代控制系统中具有十分重要的地位。大部分嵌入式系统以 MCU 为核心进行设计。MCU 从体系结构到指令系统都是按照嵌入式系统的应用特点专门设计的，它能很好地满足应用系统的嵌入、面向测控对象、现场可靠运行等方面的要求。因此以 MCU 为核心的系统是应用最广的嵌入式系统。在实际应用时，开发者可以根据具体要求与应用场合，选用最佳型号的 MCU 嵌入到实际应用系统中。

在 MCU 出现之前，人们必须用模拟电路、数字电路实现大部分计算与控制功能，这样

² MCU 的英文全称是 Microcontroller Unit。

³ 详见《单片机与嵌入式系统应用》，2004 年第 1 期。

使得控制系统体积庞大，易出故障。MCU 出现以后，情况发生了变化，系统中的大部分计算与控制功能由 MCU 的软件实现。其它电子线路成为 MCU 的外围接口电路，承担着输入、输出与执行动作等功能，而计算、比较与判断等原来必须用电路实现的功能，可以用软件取代，大大地提高了系统的性能与稳定性，这种控制技术称之为嵌入式控制技术。在嵌入式控制技术中，核心是 MCU，其它部分依此而展开。

1.1.3 嵌入式系统的特点

要谈嵌入式系统特点，不同学者也许有不同说法。这里从与通用计算机对比的角度谈嵌入式系统的特点。

1. 嵌入式系统属于计算机系统，但不单独以通用计算机的面目出现

嵌入式系统的本名叫嵌入式计算机系统 (Embedded computer system)，它不仅具有通用计算机的主要特点，又具有自身特点。嵌入式系统也必须要有软件才能运行，但其隐含在种类众多的具体产品中。同时，通用计算机种类屈指可数，而嵌入式系统不仅芯片种类繁多，而且由于应用对象大小各异，嵌入式系统作为控制核心，已经融入到各个行业的产品之中。

2. 嵌入式系统开发需要专用工具和特殊方法

嵌入式系统不像通用计算机那样有了计算机系统就可以进行应用开发。一般情况下，MCU 芯片本身不具备开发功能，必须要有一套与相应芯片配套的开发工具和开发环境。这些工具和环境一般基于通用计算机上的软硬件设备以及各种逻辑分析仪、混合信号示波器等。开发时往往有主机和目标机的概念，主机用于程序的开发，目标机作为程序的执行机，开发时需要交替结合进行。

3. 使用 MCU 设计嵌入式系统，数据与程序空间采用不同存储介质

在通用计算机系统中，程序存储在硬盘上。实际运行时，通过操作系统将要运行的程序从硬盘调入内存 (RAM)，运行中的程序、常数、变量均在 RAM 中。而以 MCU 为核心的嵌入式系统，其程序被固化到非易失性存储器中⁴。变量及堆栈使用 RAM 存储器。

4. 开发嵌入式系统涉及软件、硬件及应用领域的知识

嵌入式系统与硬件紧密相关，嵌入式系统的开发需要硬件、软件协同设计、协同测试。同时，由于嵌入式系统专用性很强，通常是用在特定应用领域，如嵌入在手机、冰箱、空调、各种机械设备、智能仪器仪表中起核心控制作用，功能专用。因此，进行嵌入式系统的开发，还需要对领域知识有一定的理解。当然，一个团队协作开发一个嵌入式产品，其中各个成员可以扮演不同角色，但对系统的整体理解与把握并相互协作，有助于一个稳定可靠嵌入式产品的诞生。

⁴ 目前，非易失性存储器通常为 Flash 存储器，特点见有关“Flash 存储器在线编程”章节。

5. 嵌入式系统的其他特点

除了以上特点之外，嵌入式系统还具有其他方面的特点。

在资源方面：嵌入式系统通常专用于某一特定应用领域，其硬件资源不会像通用计算机那样丰富；**在可靠性方面：**嵌入式系统一般要求更高可靠性和稳定性；**在实时性方面：**相当多嵌入式系统有实时性要求；**在成本方面：**嵌入式系统通常极其关注成本；**在功耗要求方面：**一些嵌入式系统要求低功耗；**在生命周期方面：**嵌入式系统通常比通用计算机系统生命周期长，升级换代比通用计算机慢。**在知识综合方面：**嵌入式系统是将先进的计算机技术、半导体技术及电子技术与各个行业的具体应用相结合的产物，是一个技术密集、资金密集、高度分散、不断创新的知识集成系统。它的构成既有硬件又有软件，不仅包括应用软件，也可能包括系统软件。它既有数字电路又有模拟电路。其产品技术含量高，涉及多种学科，不容易开发，因此也不容易形成技术垄断。

这些特点决定了嵌入式系统的开发方法、开发难度、开发手段等，均不同于通用计算机，也不同于常规的电子产品。

1.2 嵌入式系统开发所遇到的若干问题

自从1974年第一款微处理器芯片问世以来，嵌入式系统应用已深入到军事、航空航天、通信、家电等各个领域。近年来，随着微控制器（MCU）内部Flash存储器可靠性提高及擦写方式的变化，内部RAM及Flash存储器容量的增大，以及外部模块内置化程度的提高，设计复杂性、设计规模及开发手段已经发生了根本变化。

在嵌入式系统发展的最初阶段，嵌入式系统开发（包括硬件和软件设计）通常是由一个工程师来承担，软件在整个工作中的比例很小。随着时间的推移，硬件设计变得越来越复杂，软件的份量也急剧增长，嵌入式开发人员也由一人发展为由若干人组成的开发团队。

目前，嵌入式系统开发主要存在以下两大问题：

问题 1：硬件设计缺乏重用支持

导致硬件设计缺乏重用支持的主要原因是：目前缺少可供硬件设计工程师们共同遵守的设计规范。设计人员往往是凭借个人工作经验和习惯的积累进行系统硬件电路的设计。在开发完一个嵌入式应用系统再进行下一个应用开发时，硬件电路原理图往往需要从零开始，重新绘制；或者在一个类似的原理图上修改，但往往又很麻烦，容易出错。

问题 2：驱动程序可移植性差

驱动程序的开发在嵌入式系统的开发中具有举足轻重的地位。驱动程序的好坏直接关系到整个嵌入式系统的稳定性和可靠性。然而，开发出完备、稳定的驱动程序并非易事。长期以来，开发人员在编写驱动程序时缺少软件工程思想的支撑，软、硬件设计过程孤立，造成与硬件密切相关的底层软件缺乏通用性，可移植性和可复用性较差，开发过程中缺少标准化、文档化的管理，给开发人员之间的交流以及日后系统的维护带来很大的困难。

上述两个问题导致的结果是系统开发周期长，效率低。下面提出的基于硬件构件的软硬件设计思想可在一定程度上解决这些问题。

1.3 嵌入式硬件构件的基本思想与应用方法

什么是嵌入式硬件构件？它与我们常说的硬件模块有什么不同？

众所周知，嵌入式硬件是任何嵌入式产品不可分割的重要组成部分，是整个嵌入式系统

的构建基础，嵌入式应用程序和操作系统都运行在特定的硬件体系上。一个以 MCU 为核心的嵌入式系统通常包括以下硬件模块：电源、写入器接口电路、硬件支撑电路、UART、USB、Flash、A/D、D/A、LCD、键盘、传感器输入电路、通信电路、信号放大电路、驱动电路等模块。其中有些模块集成在 MCU 内部，有的位于 MCU 之外。

与硬件模块的概念不同，嵌入式硬件构件是指将一个或多个硬件功能模块、支撑电路及其功能描述封装成一个可重用的硬件实体，并提供一系列规范的输入/输出接口。由定义可知，传统概念中的硬件模块是硬件构件的组成部分，一个硬件构件可能包含一个硬件功能模块，也有可能包含多个。

根据接口之间的生产消费关系，接口可分为提供接口和需求接口两类。根据所拥有接口类型的不同，硬件构件分为核心构件、中间构件和终端构件三种类型。核心构件只有提供接口，没有需求接口。也就是说，它只为其它硬件构件提供服务，而不接受服务。在以单 MCU 为核心的嵌入式系统中，MCU 的最小系统就是典型的核心构件。中间构件既有需求接口又有提供接口，即它不仅能够接受其它构件提供的服务，而且也能够为其它构件提供服务。而终端构件只有需求接口，它只接受其它构件提供的服务。这三种类型构件的区别如表 4-1 所示。

表 1-1 核心构件、中间构件和终端构件的区别

类型	需求接口	提供接口	举例
核心构件	无	有	芯片的硬件最小系统
中间构件	有	有	电源控制构件、232电平转换构件
终端构件	有	无	LCD构件、LED构件、键盘构件

利用硬件构件进行嵌入式系统硬件设计之前，应该进行硬件构件的合理划分，按照一定规则，设计与系统目标功能无关的构件个体，然后进行“组装”，完成具体系统的硬件设计。这样，这些构件个体也可以被组装到其他嵌入式系统中。在硬件构件被应用到具体系统中后，设计人员需要做的仅仅是为需求接口添加接口网标。

1.4 基于硬件构件的嵌入式系统硬件电路设计

1.4.1 设计时需要考虑的基本问题

设计以 MCU 为核心的嵌入式系统硬件电路根据需求分析进行综合考虑，需要考虑的问题较多，这里给出几个特别要注意的问题。

1. MCU 的选择

选择 MCU 时要考虑 MCU 所能够完成的功能、MCU 的价格、功耗、供电电压、I/O 口电平、管脚数目以及 MCU 的封装等因素。MCU 的功耗可以从其电气性能参数中查到。供电电压有 5V、3.3V 以及 1.8V 超低电压供电模式。为了能合理分配 MCU 的 I/O 资源，在 MCU 选型时可绘制一张引脚分配表，供以后的设计使用。

2. 电源

(1) 考虑系统对电源的需求，例如系统需要几种电源，如 24V、12V、5V 或者 3.3V 等，估计各需要多少功率或最大电流 (mA)。在计算电源总功率时要考虑一定的余量，可按公式“电源总功率=2×器件总功率”来计算。

(2) 考虑芯片与器件对电源波动性的需求。一般允许电源波动幅度在±5%以内。对于 A/D 转换芯片的参考电压一般要求±1%以内。

(3) 考虑工作电源是使用电源模块还是使用外接电源。

3. 普通 I/O 口

(1) 上拉、下拉电阻：考虑用内部或者外部上/下拉电阻，内部上/下拉阻值一般在 $700\ \Omega$ 左右，低功耗模式不宜使用。外部上/下拉电阻根据需要可选 $10\text{K}\ \Omega \sim 1\text{M}\ \Omega$ 之间。

(2) 开关量输入：一定要保证高低电压分明。理想情况下高电平就是电源电压，低电平就是地的电平。如果外部电路无法正确区分高低电平，但高低仍有较大压差，可考虑用 A/D 采集的方式设计处理。对分压方式中的采样点，要考虑分压电阻的选择，使该点通过采样端口的电流不小于采样最小输入电流，否则无法进行采样。

(3) 开关量输出：基本原则是保证输出高电平接近电源电压，低电平接近地电平。I/O 口的吸纳电流一般大于放出电流。对小功率元器件控制最好是采用低电平控制的方式。一般情况下，若负载要求小于 10mA ，则可用芯片引脚直接控制；电流在 $10\sim 100\text{mA}$ 时可用三极管控制，在 $100\text{mA}\sim 1\text{A}$ 时用 IC 控制；更大的电流则适合用继电器控制，同时建议使用光电隔离芯片。

4. A/D 电路与 D/A 电路

(1) A/D 电路：要清楚前端采样基本原理，对电阻型、电流型和电压型传感器采用不同的采集电路。如果采集的信号微弱，还要考虑如何进行信号放大。

(2) D/A 电路：考虑 MCU 的引脚通过何种输出电路控制实际对象。

5. 控制电路

对外控制电路要注意设计的冗余与反测，要有合适的信号隔离措施等。在评估设计的布板时，一定要在构件的输入输出端引出检测孔，以方便排查错误时测量。

6. 考虑低功耗

低功耗设计并不仅仅是为了省电，更多的好处在于降低了电源模块及散热系统的成本。由于电流的减小也减少了电磁辐射和热噪声的干扰。随着设备温度的降低，器件寿命则相应延长，要做到低功耗一般需要注意以下几点：

(1) 并不是所有的总线信号都要上拉。上下拉电阻也有功耗问题需要考虑。上下拉电阻拉一个单纯的输入信号，电流也就几十微安以下。但拉一个被驱动了的信号，其电流将达毫安级。所以需要考虑上下拉电阻对系统总功耗的影响。

(2) 不用的 I/O 口不要悬空，如果悬空的话，受外界的一点点干扰就可能成为反复振荡的输入信号，而 MOS 器件的功耗基本取决于门电路的翻转次数。

(3) 对一些外围小芯片的功耗也需要考虑。对于内部不太复杂的芯片功耗是很难确定的，它主要由引脚上的电流确定。例如有的芯片引脚在没有负载时，耗电大概不到 1mA ，但负载增大以后，可能功耗很大。

7. 考虑低成本

(1) 正确选择电阻值与电容值。比如一个上拉电阻，可以使用 $4.5\text{K}\sim 5.3\text{K}$ 的电阻，你觉得就选个整数 5K ，事实上市场上不存在 5K 的阻值，最接近的是 4.99K （精度 1% ），其次是 5.1K （精度 5% ），其成本分别比精度为 20% 的 4.7K 高 4 倍和 2 倍。 20% 精度的电阻阻值

只有 1、1.5、2.2、3.3、4.7、6.8 几个类别（含 10 的整数倍）；类似地，20%精度的电容也只有以上几种值，如果选了其它的值就必须使用更高的精度，成本就翻了几倍，却不能带来任何好处。

（2）指示灯的选择。面板上的指示灯选什么颜色呢？有些人按颜色选，比如自己喜欢蓝色就选蓝色。但是其它红绿黄橙等颜色的不管大小（5mm 以下）封装如何，都已成熟了几十年，价格一般都在 5 毛钱以下，而蓝色却是近三四年才发明的，技术成熟度和供货稳定度都较差，价格却要贵四五倍。

（3）不要什么都选最好的。在一个高速系统中并不是每一部分都工作在高速状态，而器件速度每提高一个等级，价格差不多要翻倍，另外还给信号完整性问题带来极大的负面影响。

1.4.2 硬件构件化电路原理图绘制的简明规则

1. 硬件构件设计的通用规则

在设计硬件构件的电路原理图时，需遵循以下基本原则：

（1）元器件命名格式：对于核心构件，其元器件直接编号命名，同种类型的元件命名时冠以相同的字母前缀。如电阻名称为 R1、R2 等，电容名称为 C1、C2 等，电感名称为 L1、L2 等，指示灯名称为 E1、E2 等，二极管名称为 D1、D2 等，三极管名称为 Q1、Q2 等，开关名称为 K1、K2 等。对于中间构件和终端构件，其元器件命名格式采用“构件名-标志字符？”。例如，LCD 构件中所有的电阻名称统一为“LCD-R？”，电容名称统一为“LCD-C？”。当构件原理图应用到具体系统中时，可借助原理图编辑软件为其自动编号。

（2）为硬件构件添加详细的文字描述，包括中文名称、英文名称、功能描述、接口描述、注意事项等，以增强原理图的可读性。中英文名称应简洁明了。

（3）将前两步产生的内容封装在一个虚线框内，组成硬件构件的内部实体。

（4）为该硬件构件添加与其它构件交互的输入/输出接口标识。接口标识有两种：接口注释和接口网标。它们的区别是：接口注释标于虚线框以内，是为构件接口所作的解释性文字，目的是帮助设计人员在使用该构件时理解该接口的含义和功能；而接口网标位于虚线框之外，且具有电气特性。为使原理图阅读者便于区分，接口注释采用斜体字。

在进行核心构件、中间构件和终端构件的设计时，除了要遵循上述的通用规则外，还要兼顾各自的接口特性、地位和作用。

2. 核心构件设计规则

设计核心构件时，需考虑的问题是：“核心构件能为其他构件提供哪些信号？”。核心构件其实就是某型号 MCU 的最小系统。核心构件设计的目标是：凡使用该 MCU 进行硬件系统设计时，核心构件可以直接“组装”到系统中，无须任何改动。为了实现这一目标，在设计核心构件的实体时必须考虑细致、周全，包括稳定性、扩展性等，封装要完整。核心构件的接口都是为其它构件提供服务的，因此接口标识均为接口网标。在进行接口设计时，需将所有可能使用到的引脚都标注上接口网标（不要考虑：核心构件将会用到怎样的系统中去）。若同一引脚具有不同功能，则接口网标依据第一功能选项命名。遵循上述规则设计核心构件的好处是：当使用核心构件和其它构件一起组装系统时，只要考虑其它构件将要连接到核心构件的哪个接口（不是考虑：核心构件将要连接到其它构件的哪个接口），这也符合设计人员的思维习惯。

3. 中间构件设计规则

设计中间构件时,需考虑的问题是:“中间构件需要接受哪些信号,以及提供哪些信号? ”。中间构件是核心构件与终端构件之间通信的桥梁。在进行中间构件的实体封装时,实体的涉及范围应从构件功能和编程接口两方面考虑。一个中间构件应具有明确的且相对独立的功能,它既要有接受其它构件提供的服务的接口,即需求接口,又要有为其他构件提供服务的接口,即提供接口。**描述需求接口采用接口注释,描述提供接口采用接口网标。**当中间构件被作为一个“零件”组装到具体系统中时,设计人员只要考虑为构件提供服务的来源,为接口注释添加对应的应用网标即可,其它内容无须关心或改动。

中间构件的接口数目没有核心构件那样丰富。为直观起见,设计中间构件时,将构件的需求接口放置在构件实体的左侧,提供接口放置在右侧。接口网标的命名规则是:构件名称-引脚信号/功能名称。而接口注释名称前的构件名称可有可无,它的命名隐含了相应的引脚功能。

电源控制构件(如图 1-2 所示)、可变频率产生构件(如图 1-3 所示)是常用的中间构件。图 1-2 中的 *Power-IN* 和图 1-3 中的 *SDI*、*SCK* 和 *SEN* 均为接口注释, *Power-OUT* 和 *LTC6903-OUT* 为接口网标。

4. 终端构件设计规则

设计终端构件时,需考虑的问题是:“终端构件需要什么信号才能工作? ”。终端构件是嵌入式系统中最常见的构件。终端构件没有提供接口,它仅有与上一级构件交互的需求接口,因而接口标识均为斜体标注的接口注释。*LCD* (*YM1602C*) 构件(如图 1-4 所示)、*LED* 构件、指示灯构件以及键盘构件(如图 1-5)等都是典型的终端构件。

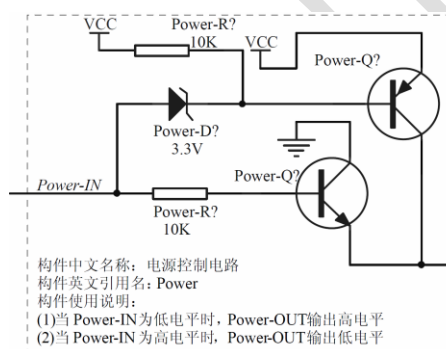


图 1-2 电源控制构件

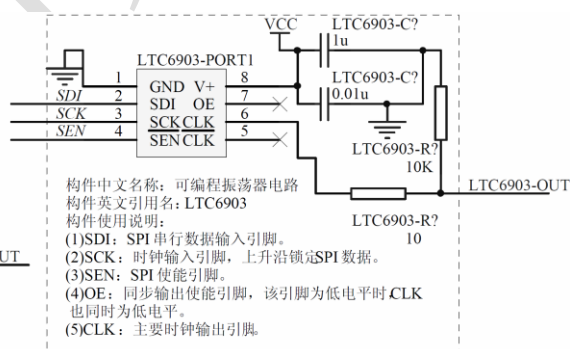


图 1-3 可变频率产生构件

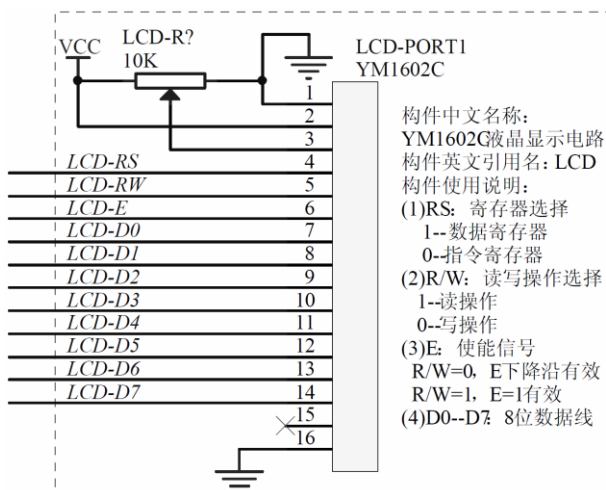


图 1-4 LCD 构件

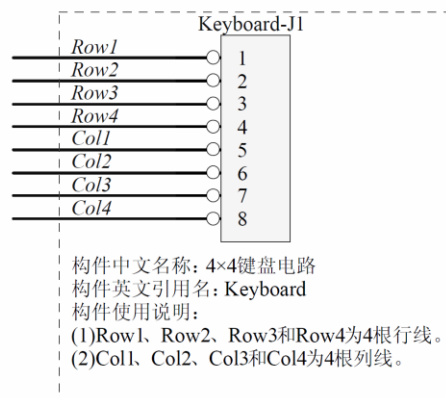


图 1-5 键盘构件

5. 使用硬件构件组装系统的方法

对于核心构件，在应用到具体的系统中时，不必作任何改动。具有相同 MCU 的应用系统，其核心构件完全相同。对于中间构件和终端构件，在应用到具体的系统中时，仅需为需求接口添加接口网标，在不同的系统中，接口网标名称不同，但构件实体内部完全相同。

使用硬件构件化思想设计嵌入式硬件系统的过程与步骤是：

- (1) 根据系统的功能划分出若干个硬件构件。
- (2) 将所有硬件构件原理图“组装”在一起。
- (3) 为中间构件和终端构件添加接口网标。

1.4.3 实验 PCB 板设计的简明规则

印刷电路板（Printed-Circuit Board, PCB）是由环氧树脂粘合而成的镀了铜的玻璃纖維板。蚀刻掉部分镀铜，只留下以构成电路互连通路的铜层走线。PCB 可以是单层、双层或者四层、六层、八层、十二层甚至更多。层数越多，连接的布线就越容易，但成本却越高，调试也越难。如果 PCB 有专用供电层和接地层，系统将会有更强的噪声抗扰度。

在使用电子设计自动化软件（如 DXP）设计 PCB 时需注意以下几个方面的问题。

1. PCB 板布局

在正式走线之前要对 PCB 的大体格局进行规划。布局规划应遵循下列基本原则。

- (1) 在 PCB 布板之前首先要打印出相应的原理图，然后根据原理图确定整个 PCB 板的大体布局，即各个硬件构件的位置安排。
- (2) PCB 板的形状如无其他要求，一般为矩形，长宽比为 4:3 或 3:2。
- (3) 考虑面板上元件的放置要求。
- (4) 考虑边缘接口。

2. 元件放置

- (1) 元件放置要求整齐，尽可能正放，属于同一硬件构件内的元件尽可能排放在一起。排列方位尽可能与原理图一致，布线方向最好与电路图走线方向一致。元器件在 PCB 上的

排列可采用不规则、规则和网格等三种排列方式中的一种，也可同时采用多种。

不规则排列：元件轴线方向彼此不一致，这对印制导线布设是方便的，且平面利用率高，分布参数小，特别对高频电路有利。

规则排列：元器件轴线方向排列一致，布局美观整齐，但走线较长且复杂，适于低频电路。

网格排列：网格排列中的每一个安装孔均设计在正方形网格交点上。

布局的元器件应有利于发热元器件散热，高频时要考虑元器件之间的分布参数，高、低压之间要隔离，隔离距离与承受的耐压有关。

(2) 电容的位置要特别注意，其中电源模块的滤波电容要求靠近电源，而 IC 的滤波电容要靠近 IC 的引脚。

(3) 考虑元件间的距离，防止元件之间出现重叠。还要考虑元器件的引脚间距，元器件不同，其引脚间距也不相同，在 PCB 设计中必须弄清楚元器件的引脚间距，因为它决定着焊盘放置间距。对于非标准器件的引脚间距的确定最直接的方法就是：使用游标卡尺进行测量。

(4) PCB 四周留有 5-10mm 空隙不布器件。

(5) 先放置占用面积较大的元器件，先集成后分立，先主后次，多块集成电路时先放置主电路。

(6) 可调元件应放置在便于调节的地方，质量超过 15g 的元器件应当用支架，热敏元件应远离发热元件。晶体应平放，而不要竖直放置。

(7) PLL 滤波电路应尽量靠近 MCU。

3. 有关设定

在正式布线之前，需要对软件环境的以下参数进行设置：

(1) 线宽。导线的最小宽度由导线与绝缘基板间的粘附强度和流过它们的电流值决定。走线时导线尽可能宽，最好不要小于 0.5mm，手工制板应不小于 0.8mm，这样既可以减小阻抗，又可以防止由于制造工艺的原因导致导线断路。电源线和接地线因电流较大，宽度要大于普通信号线，一般不要小于 1mm。三者关系为：地线宽度>电源线宽度>信号线宽度。

(2) 间距。导线间的间距由它们之间的安全工作电压决定，相邻导线之间的峰值电压、基板的质量、表面涂层、电容耦合参数等都影响导线的安全工作电压，为满足电气安全要求，导线间距离以及导线与元件间距离要尽可能地大，一般不要小于 1mm，这样可以有效解决焊接时短路的问题。

(3) 过孔大小。过孔大小设定要适中。

4. 布线

布线时，应该首先对时钟和高速信号进行布线，以确保它们的走线尽可能直接。石英晶振和对噪声特别敏感的器件下面不要走线。对总线进行布线时，尽可能地保持信号线平行走线，而时钟信号线则要避免平行，且在上述导线之间最好加接地线。布板完成后一定要进行自动与人工检查。过孔数尽可能少。最小系统中未使用的 I/O 口，可通过电阻接地。走线尽量少拐弯，线宽不要突变，导线拐角应 $\geq 90^\circ$ ，力求线条简单明了。信号线的拐角应设计成钝角走向，为圆形或圆弧形，切忌画成 90° 或更小角度形状。

5. 测量点

考虑到硬件测试的方便，在 PCB 布板时要留下一些测量点，以便调试之用。测量点要

根据原理图确定。以下几处需要留测量点：

- (1) 原理图中模块的输入输出引脚。
- (2) 最小系统模块中 MCU 的引脚。
- (3) 各硬件功能模块单元的输入、输出；

画测量点的步骤是：引出、打孔、标字。孔的大小以万用表头方便测量为原则，一般不要在线上直接打孔。

6. 模块标示

由于在整体布局时，已经将各个硬件构件的组成元件放在一起，因而可在 PCB 板上用矩形框将各个硬件构件区分开，并用汉字标出构件名（与原理图一致），并注意字体字号。

7. 铺地

在布板的最后都要铺地，目的是减小干扰，提高 PCB 板的稳定性。铺地需注意：

- (1) 在铺地前，要设定地与导线、地与引脚之间的距离，并要求该距离尽可能大。
- (2) 铺地本应该双面铺，作为实验用板，为了方便检查，可只铺反面地。
- (3) 如果电路板中有数字地和模拟地，应将它们隔离开，两者间使用磁珠相连。

8. 空余位置的利用并标注相关信息

PCB 板的空余位置可适度作如下用途：

- (1) 电源、地。空白处多留几排电源和地。
- (2) 双排孔。留出几排两孔相连的排孔，以用来扩展或试验时焊接其他元件。
- (3) 固定孔。在 PCB 上画固定板的固定孔，一般在板的四个角落。

在完成 PCB 板的铺地之后，要在板的正面适当位置标出以下信息：单位、日期、责任人、PCB 板的名称、编号等。

9. 抗干扰问题的特别考虑

PCB 设计中除了考虑以上问题外，还要考虑一些隐藏的问题，这些问题设计时不起眼，但是解决的时候，却非常麻烦，这就是电路的干扰问题，为此，在 PCB 设计时还应解决如下问题：

1) 热干扰及抑制

元器件在工作中都有一定程度的发热，尤其是功率较大的器件所发出的热量会对周边比较敏感的器件产生干扰，若热干扰得不到很好的抑制，整个电路的电性就会发生变化，最后造成短路。为了对热干扰进行抑制，可采取以下措施：

(1) 发热元件的放置

不要贴板放置，也可以单独设计为一个功能单元，放在靠近边缘容易散热的地方。另外，发热量大的器件与小热量的器件应分开放置。

(2) 大功率器件的放置

应尽量靠近边缘布置，在垂直方向时应尽量布置在板上。

(3) 温度敏感器件的放置

对温度比较敏感的器件应安置在温度最低的区域，千万不要将它放在发热器件的正上方。

(4) 器件的排列与气流

非特定要求，一般设备内部均以空气自由对流进行散热，故元器件应以纵式排列；若强制散热，元器件可横式排列。另外，为了改善散热效果，可添加与电路原理无关的零部件以引导热量对流。

2) 共阻抗及抑制

共阻干扰是由 PCB 上大量的地线造成，当两个或两个以上的回路共用一段地线时，不同的回路电流在共用地线上产生一定压降，此压降经放大就会影响电路性能，当电流频率很高时，会产生很大的感抗而使电路受到干扰。为了抑制共阻抗，可采取以下措施：

(1) 一点接地

使同级单元电路的几个接地点尽量集中，适用于信号的工作频率小于 1MHZ 的低频电路，如果工作频率在 1 — 10MHz 而采用一点接地时，其地线长度应不超过波长的 $1/20$ 。

(2) 就近多点接地

PCB 上有大量公共地线分布在板的边缘，且呈现半封闭回路(防磁场干扰)，各级电路采取就近接地，以防地线太长。适用于信号的工作频率大于 10MHz 的高频电路。

(3) 大面积接地

在高频电路中将 PCB 上所有不用面积均布设为地线，以减少地线中的感抗，从而削弱在地线上产生的高频信号，并对电场干扰起到屏蔽作用。

(4) 加粗接地线

若接地线很细，接地电位则随电流的变化而变化，致使电子设备的定时信号电平不稳，抗噪声性能变坏，其宽度至少应大于 3mm。

(5) D / A(数 / 模)电路的地线分开

两种电路的地线各自独立，然后分别与电源端地线相连，以抑制它们相互干扰。

3) 电磁干扰及抑制

电磁干扰是由电磁效应而造成的干扰，由于 PCB 上的元器件及布线越来越密集，如果设计不当就会产生电磁干扰。针对由电源布线、信号布线产生的电磁干扰，可采取不同的措施。

(1) 电源布线引起的电磁干扰

电源布线可采用以下预防措施：

布线要宽。

加去耦电容。这种电容起到旁路滤波的作用。要在电源的输入端并联较大的和较小的滤波电容。

地线环绕。地线要靠近供电电源母线和信号线，因电流沿路径传输会产生回路电感，地线靠近，回路面积减小，电感量减小，回路阻抗减小，从而减小电磁干扰耦合。

(2) 信号布线引起的电磁干扰

信号布线可采用以下预防措施：

不同功能的单元电路(如数字电路与模拟电路，高频与低频)分开设置，布线图形应易于信号流通且使信号流向尽可能保持一致。

合理使用屏蔽和滤波技术，注意高低压之间的隔离。

尽量不选用比实际需要的速度更快的元件，在元件的位置安排上，易受电磁干扰的元器件不能相距太近，应大于信号波长的四分之一，输入器件与输出器件尽量远离。

做到安全接地。

1.5 基于硬件构件的嵌入式底层软件构件的编程方法

嵌入式系统是软件与硬件的综合体，硬件设计和软件设计相辅相成。嵌入式系统中的驱动程序是直接工作在各种硬件设备上的软件，是硬件和高层软件之间的桥梁。正是通过驱动程序，各种硬件设备才能正常运行，达到既定的工作效果。

1.5.1 嵌入式硬件构件和软件构件的层次模型

嵌入式软件构件（Embedded Software Component, ESC）是实现一定嵌入式系统功能的一组封装的、规范的、可重用的、具有嵌入特性的软件单元，是组织嵌入式系统的功能单位。

嵌入式软件构件分为高层软件构件和底层软件构件（以下简称高层构件和底层构件）。高层构件与硬件无关。而底层构件与硬件密不可分，是硬件驱动程序的封装。前面提到，在硬件构件中，核心构件为 MCU 的最小系统。通常，MCU 内部包含有 GPIO（即通用 IO）口和一些内置功能模块，可将通用 I/O 口的驱动程序封装为 GPIO 构件，各内置功能模块的驱动程序封装为功能构件，如芯片内含模块的功能构件有串行通信构件、Flash 构件、定时器构件等。

在硬件构件层中，相对于核心构件而言，中间构件和终端构件是核心构件的“外设”。由这些“外设”的驱动程序封装而成的软件构件称为底层外设构件。注意，并不是所有的中间构件和终端构件都可以作为编程对象。例如：键盘、LED、LCD 等硬件构件与编程有关，而电平转换硬件构件就与编程无关，因而不存在相应的底层驱动程序，当然也就没有相应的软件构件。嵌入式硬件构件与软件构件的层次模型如图 4-5 所示。

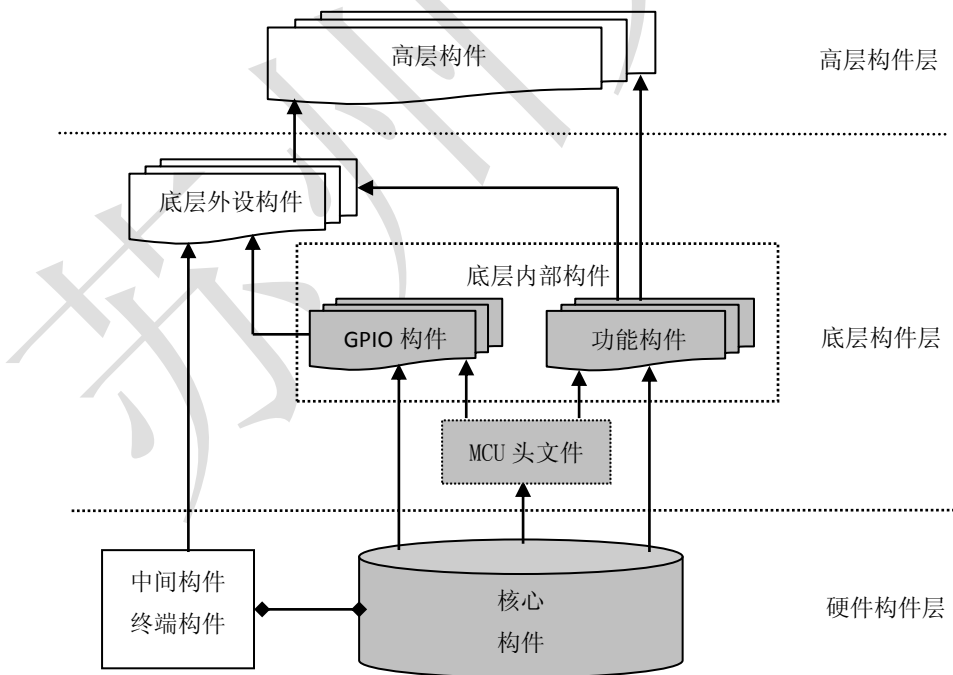


图 1-6 嵌入式硬件构件与软件构件的层次模型

由图 1-6 可看出，底层外设构件可以调用底层内部构件，如 LCD 构件可以调用 GPIO 构件、PCF8563 构件（时钟构件）可以调用 I2C 构件等。而高层构件可以调用底层外设构件和底层内部构件中的功能构件，而不能直接调用 GPIO 构件。另外，考虑到几乎所有的底层内部构件都涉及到 MCU 各种寄存器的使用，因此将 MCU 的所有寄存器定义组织在一起，形成 MCU 头文件，以便其它构件头文件中包含该头文件。

1.5.2 底层构件的实现方法与编程思想

底层构件是与硬件直接打交道的软件，由头文件和源程序文件两部分组成。

头文件中的内容主要有：包含下层构件头文件的`#include`语句、用以描述构件属性的宏定义语句以及对外接口函数原型说明。在头文件中使用函数原型，对于建立代码模块和外部接口的规范，便于他人使用，都是很有帮助的。使用这些函数的用户，不需要查找源代码去了解参数的具体类型，直接查看函数原型即可。

源程序文件中存放构件的内部函数和外部函数的定义，即函数的实现代码，以完成函数所要实现的功能。

在对底层构件进行设计时，最关键的工作是要对构件的共性和个性进行分析，抽取出构件的属性和对外接口函数。尽量做到：当一个底层构件应用到不同系统中时，仅需修改构件的头文件，对于构件的源程序文件则不必修改或改动很小。

例如，串行通信模块 SCI 是大多数 MCU 都具有的内部模块。仔细分析各种 MCU 串行通信程序发现：在查询方式下，各种 MCU 都是根据状态寄存器中的两个标志位来判断是否接收到数据和数据是否发送完毕，这就是 SCI 模块的共性。对于不同的 MCU，该状态寄存器的名称可能不同，这两个标志位的位号也有可能不同。此外，用以设置波特率、通信格式、是否校验、是否允许中断等参数的寄存器也不同，这就是 SCI 模块的个性。分析出了共性和个性之后，就可以抽取出 SCI 构件的属性和操作，编制构件头文件和程序文件了。有关内容参见第五章。

在编写构件时，一般应注意以下几方面的内容：

- (1) 构件的头文件和源程序文件的主文件名一致，且为构件名。
- (2) 属性和操作的命名统一以构件名开头。这样做的好处是：当使用底层构件组装软件系统时，避免构件之间出现同名现象。同时，名称要使人有“顾名思义”的效果。
- (3) 对 MCU 内的模块寄存器名和端口名进行重定义，在其它的代码里面都将使用宏名对模块寄存器和端口进行操作。这样，当底层驱动程序移植到其它 MCU 时，只要修改重定义语句就可以了。
- (4) 内部函数与外部函数要设计合理，函数参数个数及类型要考虑全面。内部函数仅提供给同一构件中的其它内部函数或外部函数调用，作用域仅限于定义该函数的文件。外部函数是对外接口函数，供上层应用程序调用。在定义外部函数时，应该对函数名、函数功能、入口参数、函数返回值、使用说明、函数适用范围等进行详细描述，以增强程序的可读性。上层应用程序不能直接对构件的属性进行读取或设置，必须借助于该构件提供的接口操作函数来实现。
- (5) 应用程序在使用底层构件时，严格禁止通过全局变量来传递参数，所有的数据传递都要通过函数的形式参数来接收。这样做不但使得接口简洁，更加避免了全局变量可能引发的安全隐患。

1.5.3 硬件构件及底层软件构件的重用与移植方法

重用是指在一个系统中，同一构件可被重复使用多次。移植是指将一个系统中使用到的构件应用到另外一个系统中。

1. 硬件构件的重用与移植

对于以单 MCU 为核心的嵌入式应用系统而言，当用硬件构件“组装”硬件系统时，核心构件（即最小系统）有且只有一个，而中间构件和终端构件可有多，并且相同类型的构

件可出现多次。下面以终端构件 LCD 为例，介绍硬件构件的移植方法。

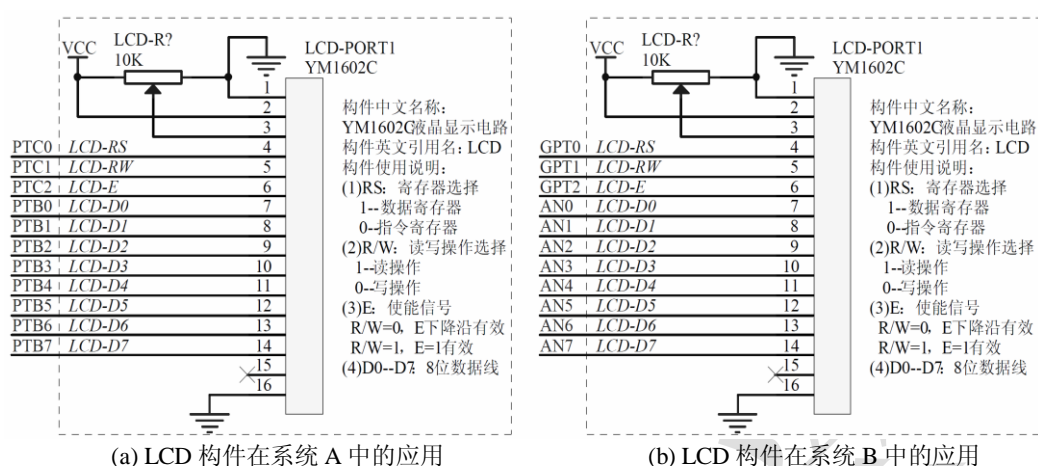


图 1-7 LCD 构件在实际系统中的应用

在应用系统 A 中，若 LCD 的数据线（LCD-D0~LCD-D7）与芯片 AW60（8 位 MCU）芯片的通用 IO 口的 B 口相连，C 口作为 LCD 的控制信号传送口，其中，LCD 寄存器选择信号 LCD-RS 与 C 口第 0 引脚连接，读写信号 LCD-RW 与 C 口第 1 引脚连接，使能信号 LCD_E 与 C 口第 2 引脚连接，则 LCD 硬件构件实例如图 1-7(a)所示。虚线框左边的文字（如 PTC0、PTC1 等）为接口网标，虚线框右边的文字（如 LCD-RS、LCD-RW 等）为接口注释。

在应用系统 B 中，若 LCD 的数据线（LCD-D0~LCD-D7）与 MCF52233（32 位 MCU）芯片的通用 IO 口的 AN 口相连，TA 口的第 0、1、2 引脚分别作为寄存器选择信号 LCD-RS、读写信号 LCD-RW、使能信号 LCD_E，则 LCD 硬件构件实例如图 1-7(b)所示。

2. 底层构件的移植

当一个已设计好的底层构件移植到另外一个嵌入式系统中时，其头文件和程序文件是否需要改动呢？这要视具体情况而定。例如：系统的核心构件发生改变（即 MCU 型号改变）时，底层内部构件头文件和某些对外接口函数也要随之改变，例如模块初始化函数。

而对于外接硬件构件，希望不改动程序文件，而只改动头文件，那么，头文件就必须充分设计。以 LCD 构件为例，与图 1-7(a)相对应的底层构件头文件 LCD.h 可如下编写。

```
#include "GPIO.h"           //包含 GPIO 构件头文件
#define LCD_Data             GPIO_PORTB    //显示数据传送口
#define LCD_Ctrl             GPIO_PORTC    //控制信号传送口
#define LCD_RS               0             //寄存器选择信号
#define LCD_RW               1             //读写信号
#define LCD_E                2             //使能信号
void LCD_Init(void);         //液晶显示初始化
void LCD_Fill(INT8U a);      //填充 LCD，可实现清屏
void LCD_PutDot(INT8U x,INT8U y); //画点
void LCD_PutLine(INT8U x1,INT8U y1,INT8U x2,INT8U y2); //画线
void LCD_PutChar(INT8U a);   //显示一个字符
void LCD_PutString(INT8U str[]); //显示一串字符
```

当 LCD 硬件构件发生如图 1-7(b)所示的移植时，显示数据传送口和控制信号传送口发生了改变，此时，只要将上面的第 1 和第 2 条宏定义语句修改成：

```
#define LCD_Data             GPIO_PORTAN    //显示数据传送口
```

```
#define LCD_Ctrl    GPIO_PORTTA    //控制信号传送口
```

必须申明的是，本书提出构件化设计方法的目的是，在进行软硬件移植时，设计人员所做的改动要尽量小，而不是不作任何改动。希望改动在头文件中进行，而不希望改动程序文件，事实上，不作任何改动是不现实的。

苏南大学

第2章 Kinetis概述与MK60N512VMD100硬件最小系统

本章简要概述 Kinetis 系列微处理器，给出 K60 系列微控制器存储器映像与编程结构、寻址方式、异常向量表；重点指出学习一个新 MCU 芯片比较快速的学习过程；给出 K60N512 的引脚功能与硬件最小系统电路。本章的重点是：存储空间地址分配、中断结构、硬件最小系统电路。

2.1 ARM公司的发展史及ARM架构的发展

1.ARM 公司的历史

ARM 公司是世界领先的半导体知识产权供应商，其提供的产品是数字电子产品的核心。今天 ARM 为世界上四分之一的电子产品提供技术基础，被半导体及电子业界评为过去 30 年全球最有影响力的 10 家公司之一。ARM 公司于 1990 年 11 月成立于英国，原名 Advanced RISC Machine 有限公司，是苹果电脑、Acorn 电脑集团和 VLSI Technology 的合资企业。Acorn 曾推出世界首个商用单芯片 RISC 处理器，而苹果电脑当时希望将 RISC 技术应用于自身系统，ARM 的微处理器新标准因此应运而生。ARM 成功地研制了首个低成本 RISC 架构，迅速在市场上崭露头角，与此同时 RISC 结构的竞争对手都着眼于提高性能发展高端工作站处理器的 RISC 结构。

1991 年 ARM 推出首个嵌入式 RISC 核心——ARM6™ 系列处理器，不久 VLSI 率先获得授权，一年后夏普和 GEC Plessey 也成为授权用户，1993 年德州仪器和 Cirrus Logic 亦签署了授权协议。从此 ARM 的知识产权产品和授权用户急剧扩大，1993 年 Nippon Investment and Finance (NIF) 成为 ARM 股东后，ARM 开始向全球拓展，分别在亚洲、美国和欧洲设立了办事处。1998 年 4 月 ARM 在伦敦证券交易所纳斯达克交易所上市。

ARM Holdings 在半导体革新过程中初露峥嵘，被 Dataquest 誉为世界第一的知识产权供应商。20 世纪 90 年代初 ARM 率先推出 32 位 RISC 微处理器芯片系统 (SoC) 知识产权公开授权概念。

现在，ARM 芯片的出货量每年都比上一年多 20 亿片以上。不像很多其他的半导体公司，ARM 从不制造和销售具体的处理器芯片，而是把处理器的设计授权给相关的商务合作伙伴，让他们根据自己的强项设计具体的芯片。更重要的是 ARM 开创了电子新纪元：采用 ARM 技术的微处理器遍及各类电子产品，在汽车、消费、娱乐、成像、工业控制、网络、存储、安保和无线等市场中，ARM 技术无处不在。

2.ARM 架构发展史

ARM 的设计历史起源是 Acorn 电脑公司 (Acorn Computers Ltd) 于 1983 年开始的开发计划。这个团队由 Roger Wilson 和 Steve Furber 带领，着手开发一种新架构，类似进阶的 MOS Technology 6502 处理器。设计团队在 1985 年时开发出 ARM1 Sample 版，而首颗“真

正”的产能型 ARM2 于次年量产。ARM2 具有 32 位的数据总线、26 位的寻址空间，并提供 64MB 的寻址范围与 16 个 32 位的寄存器。这些寄存器其中有一个作为程序计数器，其前面 6 位和后面 2 位用来保存处理器状态标记。ARM2 可能是全世界最简单实用的 32 位微处理器，其仅容纳了 30,000 个晶体管。之所以精简的原因在于它不含微码，而与现今大多数的 CPU 不同，它没有包含任何高速缓存。这个精简的特色使它只需消耗很少的电能，却能发挥比 Intel 80286 更好的效能。后继的处理器 ARM3 更备有 4KB 的告诉缓存，使它能发挥更佳的效能。

1991 年，ARM 技术首次发布，然后苹果电脑使用 ARM6 架构的 ARM610 当作其 Apple Newton PDA 的基础。1994 年，Acorn 使用 ARM610 作为其 RISC PC 电脑内的 CPU。

随着 ARM2 到 ARM6 的技术进阶发展，内核部分却大多维持一样的大小。ARM2 有 30,000 个晶体管，但 ARM6 只增长到 35,000。主要概念是以 ODM 的方式，使 ARM 核心能搭配一些选配的零件而制成一颗完整的 CPU，而且可在现有的晶圆制造厂里制作并以低成本的方式达到很大的效能。

随着技术的不断发展，ARM 公司不断推陈出新，发展至今最新的是 ARMv7 架构，要说明的是，架构版本号和名字中的数字并不是一码事。比如，ARM7TDMI 是基于 ARMv4T 架构的（T 表示支持 Thumb 指令）。ARMv7 架构采用 Thumb-2 技术，是在 ARM 的 Thumb 代码压缩技术的基础上发展起来的，并且保持了对现存 ARM 解决方案的完整代码兼容性。Thumb-2 技术比纯 32 位代码少使用 31% 的内存，减小了系统开销。同时能够提供比已有的基于 Thumb 技术的解决方案高出 38% 的性能。ARMv7 架构还采用了 NEON 技术，将 DSP 和媒体处理能力提高了近 4 倍，并支持改良的浮点运算，满足下一代 3D 图形、游戏物理应用以及传统嵌入式控制应用的需求。此外，在这个版本中，内核架构首次从单一款式变成三种款式。

- 款式 A：设计用于高性能的“开方应用平台”——越来越接近电脑。
- 款式 R：用于高端的嵌入式系统，尤其是那些带有实时要求的——既要快又要实时。
- 款式 M：用于深度嵌入的、单片机风格的系统中——本书的主角。

让我们再近距离的考察这三种模式：

- ◆ 款式 A（ARMv7-A）：需要运行复杂应用程序的“应用处理器”。支持大型嵌入式操作系统，比如 Symbian（诺基亚智能手机用）、Linux，以及微软的 Windows CE 和智能手机操作系统 Windows Mobile。
- ◆ 款式 R（ARMv7-R）：硬实时且高性能的处理器。目标是高端实时市场。像高档轿车的组件、大型发电机控制器、机器人手臂控制器等，它们使用的处理器不但要很好很强大，还要极其可靠，对事件的反应也要极其敏捷。
- ◆ 款式 M（ARMv7-M）：认准了旧时代单片机应用而量身定制。在这些应用中，尤其是对于实时控制系统，低成本、低功耗、极速中断反应以及高处理效率都是至关重要的。Cortex 系列是 v7 架构的第一次亮相。其中 Cortex-M4 就是按款式 M 设计的。

ARM 架构发展版本如表 2-1 所示。

表 2-0-1 ARM 架构发展版本

ARM 版本系列	架构	内核名	特色	高速缓存 (I/D)/MMU	常规 MIPS 与 MHz	代表芯片与应用
ARM1	ARM v1	ARM1		无		
ARM2	ARM v2	ARM2	Architecture2	无	4MIPS@8 MHZ	Acorn Architmedes, Chessmachine

ARM 版本系列	架构	内核名	特色	高速缓存 (I/D)/MMU	常规 MIPS 与 MHz	代表芯片与应用
			加入了 MUL (乘法)指令			
	ARM v2a	ARM250	集成 MEMC(MMU),图像与 I/O 处理器。Architecture2a 加入了 SWP 和 SWPB (置换) 指令	无, MEMC1a	7MIPS@12MHz	Acorn Archimedes
ARM3	ARM v2a	ARM2a	首次在 ARM 架构上使用处理器高速缓存	均为 4KB	12MIPS@25MHz	Acorn Archimedes
ARM6	ARM v3	ARM610	V3 架构首创支援寻址 32 位的内存(针对 26 位)	均为 4KB	28MIPS@33MHz	Acorn Risc PC 600, Apple Newton
ARM7	ARM v3					
ARM7 TDMI	ARM v4T	ARM7TDMI(-S)	三级流水线	无	15MIPS@16.8MHz	Game Boy Advance, Nintendo DS,iPod
		ARM710T		均为 8KB,MMU	36MIPS@40MHz	Acorn Risc PC 700,Psion5 series, Apple eMate 300
		ARM720T		均为 8KB,MMU	60MIPS@59.8MHz	Zipit
		ARM740T		MPU		
	ARM v5TEJ	ARM7EJ-S	Jazelle DBX	无		
Strong ARM	ARM v4					
ARM8	ARM v4					
ARM9 TDMI	ARM v4T	ARM9TDMI	五级流水线	无		
		ARM920T		16KB/16KB, MMU	200MIPS @180MHz	Armadillo,GP32,GP2X(第一颗内核), TapwareZodiac(Motorola/Free scale i.MX1)
		ARM922T		8KB/8KB,M		

ARM 版本系列	架构	内核名	特色	高速缓存 (I/D)/MMU	常规 MIPS 与 MHz	代表芯片与应用
				MU		
		ARM940T		4KB/4KB, MMU		GP2X(第二颗内核)
ARM9E	ARM v5TE	ARM946E-S		可变动, tightly coupled memories, MPU		Nintendo DS, Nokia N-Gage Conexant 802.11 chips
		ARM966E-S		无高速缓存, TCMS		ST Micro STR91xF
		ARM968E-S		无高速缓存, TCMS		
	ARM v5TEJ	ARM926EJ-S	Jazelle DBX	可变动, TCMS, MMU	200MIPS @ 200MHz	移动电话: Sony Ericsson(K、W 系列), Siemens 和 Benq(x65 系列)
	ARM v5TE	ARM996HS	无振荡器处理器	无高速缓存, TCMS, MMU		
ARM10E	ARM v5TE	ARM1020E	(VFP), 六级流水线	32KB/32KB, MMU		
		ARM1022E	(VFP)	16KB/16KB, MMU		
	ARM v5TEJ	ARM1026EJ-S	Jazelle DBX	可变动, MMU, 可包含 MPU		
Xscale	ARM v5TE	80200/IOP310/IOP315	I/O 处理器			
		80219			400/600MHz	Thecus N2100
		IOP321			600 BogoMips @ 600MHz	Iyonix
		IOP33x				
		IOP34x	1~2 核, RAID 加速器	32K/32KL1,5 12KL2, MMU		
		PXA210/ PXA250	应用处理器, 七级流水线			Zaurus SL-5600
		PXA255		32KB/32KB, MMU	400 BogoMips @ 400MHz	Gumstix, Palm TungstenE2

ARM 版本系列	架构	内核名	特色	高速缓存 (I/D) /MMU	常规 MIPS 与 MHz	代表芯片与应用
		PXA26x			可达 400MHz	Palm Tungsten T3
		PXA27x			800MIPS @624MHz	HTC Universal,Zaurus SL-C1000,3000,3100,3200,Del 1 Axim x30,x50 和 x51 系列
		PXA800(E) F				
		Monahans			1000MIPS @1.25GHz	Mavell PXA300/PXA310/PXA320,Max frequency:PXA300@624MHz, PXA310/PXA320@806MHz
		PXA900				Blackberry 8700 Blackberry Pearl(8100)
ARM11	ARM v6	ARM1136J (F)-S	SIMD,Jazelle DBX,(VFP), 八级流水线	可变动, MMU	532-665MHz(i.MX31 SoC)	Nokia N93,Microsoft Zune(Freescale i.MX31),Nokia N800
	ARM v6T2	ARM1156 T2(F)-S	SIMD,Thumb-2,(VFP),九级流水线	可变动, MPU		
	ARM v6KZ	ARM1176JZ(F)-S	SIMD,Jazelle,(VFP)	可变动, MMU+TrustZone		Freescale i.MX37
	ARM v6K	ARM11 MPCore	1~4 核对称多处理器, SIMD,Jazelle,(VFP)	可变动, MMU		
Cortex	ARM v7-A	Cortex-A8	Application profile,VFP, NEON,Jazelle RCT,Thumb-2,13-stage pipeline	可变动 (L1+L2),MMU+TrustZone	2.0DMIPS/MHz 从 600MHz 到超过 1GHz 的速度	Freescale i.MX51,Texas Instruments OMAP3
		Cortex-A9				
		Cortex-A9 MPCore				
	ARM v7-R	Cortex-R4(F)	Embedded profile,(FPU)	可变动高速缓存, MMU 可选配	600 DMIPS	
	ARM	Cortex-M3	Microcontrol	无高速缓存,	120	Luminary Micro 微控制器家

ARM 版本系列	架构	内核名	特色	高速缓存 (I/D)/MMU	常规 MIPS 与 MHz	代表芯片与应用
	v7-M		ler profile	(MPU)	DMIPS @ 100MHz	族
	ARM	Cortex-M0				
	v6-M	Cortex-M1				
	ARM v7-M E	Cortex-M4		Optional 8region MPU with sub regions and background region	1.25DMIPS/MHz	

3.ARM 处理器命名方法

以前, ARM 使用一种基于数字的命名法。在早期(1990s), 还在数字后面添加字母后缀, 用来进一步明细该处理器支持的特性。就拿 ARM7TDMI 来说, T 代表 Thumb 指令集, D 是说支持 JTAG 调试(Debugging), M 意指快速乘法器, I 则对应一个嵌入式 ICE 模块。后来, 这 4 项基本功能成了任何新产品的标配, 于是就不再使用这 4 个后缀——相当于默许了。但是新的后缀不断加入, 包括定义存储器接口的, 定义高速缓存的, 以及定义“紧耦合存储器(TCM)”的, 于是形成了新一套命名法, 这套命名法也是一直在使用的。

到了架构 7 时代, ARM 改革了一度使用的, 冗长的、需要“解码”的数字命名法, 转到另一种看起来比较整齐的命名法。比如, ARMv7 的三个款式都以 Cortex 作为主名。这不仅更加澄清并且“精装”了所使用的 ARM 架构, 也避免了新手对架构号和系列号的混淆。例如, ARM7TDMI 并不是一款 ARMv7 的产品, 而是辉煌起点——v4T 架构的产品。

4.ARM 专有技术

(1) Thumb

ARM 处理器的一种 16 位指令模式, 称为 Thumb。Thumb 指令集可以看作是 ARM 指令压缩形式的子集, 它是为减小代码量而提出, 具有 16bit 的代码密度。Thumb 指令体系并不完整, 只支持通用功能, 必要时仍需要使用 ARM 指令, 如进入异常时。其指令的格式与使用方式与 ARM 指令集类似, 而且使用并不频繁

(2) Jazelle

Jazelle 是 ARM 体系结构的一种相关技术, 用于在处理器指令层次对 JAVA 加速。ARM 还开发出一项技术, Jazelle DBX (Direct Bytecode eXecution), 允许它们在某些架构的硬件上加速执行 Java bytecode, 就如其他执行模式般, 当呼叫一些无法支援 bytecodes 的特殊软件时, 能提供某些 bytecodes 的加速执行。它能在现存的 ARM 与 Thumb 模式之间互相执行。

首颗具备 Jazelle 技术的处理器是 ARM926EJ-S: Jazelle 以一个英文字母 J 标示于 CPU

名称中。它用来让手机制造商能够加速执行 Java ME 的游戏和应用程序，也因此促使了这项技术不断地发展。

(3) Thumb-2

Thumb-2 技术首先见于 ARM1156 核心，并于 2003 年发表。Thumb-2 扩充了受限的 16 位 Thumb 指令集，以额外的 32 位指令让指令集的使用更广泛。因此 Thumb-2 的预期目标是要达到近乎 Thumb 的编码密度，但能表现出近乎 ARM 指令集在 32 位内存下的效能。

Thumb-2 至今也从 ARM 和 Thumb 指令集中派生出多种指令，包含位段操作、分支跳转和条件执行等功能。

(4) ThumbEE

ThumbEE，也就是所谓的 Thumb-2EE，业界称为 Jazelle RCT 技术，于 2005 年发表，首见于 Cortex-A8 处理器。ThumbEE 提供从 Thumb-2 而来的一些扩充性，在所处的执行环境（Execution Environment）下，使得指令集能特别适用于执行阶段（Runtime）的编码产生（例如即时编译）。Thumb-2EE 是专为一些语言如 Limbo、Java、C#、Perl 和 Python，并能让即时编译器能够输出更小的编译码却不会影响到效能。

ThumbEE 所提供的新功能，包括在每次存取指令时自动检查是否无效指标，以及一种可以执行阵列范围检查的指令，并能够分支到分类器（handlers），其包含一小部份经常呼叫的编码，通常用于高阶语言功能的实作，例如对一个新物件做内存配置。

(5) 高级 SIMD (NEON)

高级 SIMD 延伸集，业界称为 NEON 技术，它是一个结合 64 和 128 bit 的 SIMD（Single Instruction Multiple Data 单指令多重数据）指令集，其针对多媒体和讯号处理程式具备标准化加速的能力。NEON 可以在 10 MHz 的 CPU 上执行 MP3 音效解码，且可以执行 13 MHz 频率以下的 GSM AMR (Adaptive Multi-Rate) 语音编码。NEON 具有一组广泛的指令集、各自的寄存器阵列，以及独立执行的硬件。NEON 支援 8-、16-、32- 和 64-bit 的整数及单精度浮点数据，并以 SIMD 的方式运算，执行图形和游戏处理中关于语音 / 视讯的部分。SIMD 在向量超级处理机中是个决定性的要素，它具备同时多项处理功能。在 NEON 技术中，SIMD 最高可支援到同时 16 个运算。

(6) VFP

VFP 是在协同处理器针对 ARM 架构的衍生技术。它提供低成本、单精度和倍精度浮点运算能力，并完全相容于 ANSI/IEEE Std 754-1985 二进制浮点算数标准。VFP 提供大多数适用于浮点运算的应用，例如 PDA、智慧手机、语音压缩与解压、3D 图像以及数位音效、打印机、机上盒，和汽车应用等。VFP 架构也支援 SIMD（单指令多重数据）平行化的短向量指令执行。这在图像和讯号处理等应用上，非常有助于降低编码大小并增加输出效率。在 ARM-based 处理器中，其他可见的浮点、或 SIMD 的协同处理器还包括了 FPA, FPE, iwMMXt。他们提供类似 VFP 的功能但在 opcode 层面上来说并不具有相容性。

(7) 安全性扩充 (TrustZone)



















TrustZone(TM) 技术出现在 ARMv6KZ 以及较晚期的应用核心架构中。它提供了一种低成本的方案，针对系统单芯片 (SoC) 内加入专属的安全核心，由硬件建构的存取控制方式支援两颗虚拟的处理器。这个方式可使得应用程序核心能够在两个状态之间切换（通常改称为领域 (worlds) 以避免和其他功能领域的名称混淆），在此架构下可以避免资讯从较可信的核心领域泄漏至较不安全的领域。这种内核领域之间的切换通常是与处理器其他功能完全无关联性 (orthogonal)，因此各个领域可以各自独立运作但却仍能使用同一颗内核。内存和周边装置也可因此得知目前内核运作的领域为何，并能针对这个方式来提供对装置的机密和编码进行存取控制。典型的 TrustZone 技术应用是要能在一个缺乏安全性的环境下完整地执行操作系统，并在可信的环境下能有更少的安全性的编码。


Cortex-M4 处理器是由 ARM 专门开发的最新嵌入式处理器,它完美融合了高效的信号处理能力以及 Cortex-M 系列处理器诸多无可比拟的优势,包括低功耗、低成本和易于使用,旨在满足那些新兴的、灵活多变的解决方案的需求。飞思卡尔 Kinetis 系列使用 ARM Cortex-M4 处理器,下一节将阐述 Kinetis 系列。


2.2 Kinetis系列微处理器概述

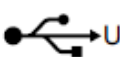
Kinetis系列微控制器是飞思卡尔公司于2010年下半年推出的基于ARM Cortex-M4内核的微控制器,是业内首款Cortex-M4内核芯片。Kinetis系列微控制器采用了飞思卡尔 90纳米薄膜存储器 (TFS) 闪存技术和 Flex存储器功能(可配置的内嵌EEPROM),支持超过1000万次的擦写。Kinetis 微控制器系列融合了最新的低功耗革新技术,具有高性能、高精度的混合信号能力,宽广的互连性,人机接口和安全外设。


第一阶段产品由五个微控制器系列组成,见图2-1所示,包含超过两百种器件,在引脚、外设和软件上可兼容。每个系列提供了不同的性能,存储器和外设特性。通过通用外设、存储器映射和封装的一致性来实现系列内和各系列间的便捷移植。


系列	程序闪存	封装	关键特性
K60 系列	256KB-1MB	100-256引脚	     
K40 系列	64-512KB	64-144引脚	   
K30 系列	64-512KB	64-144引脚	  
K20 系列	32KB-1MB	32-144引脚	  
K10 系列	32KB-1MB	32-144引脚	 


 低功耗

 混合信号

 USB

 段式LCD

 以太网

 加密和防篡改检测


 DDR

图2-1 Kenetis 微控制器产品组合

由图2-1可见,所有的 Kinetis 系列都包含丰富的模拟、通信和定时控制外设,提供多种闪存容量和输入输出引脚数量。所有Kinetis 系列都具有以下特性:

- 内核:
 - ARM Cortex-M4 内核带DSP指令,性能可达1.25 DMIPS/MHz (部分Kinetis 系列提供浮点单元)
 - 多达32通道的DMA可用于外设和存储器数据传输并减少CPU干预
 - 提供不同级别的CPU频率,有50 MHz、72 MHz 和100 MHz (部分Kinetis系列提供120 MHz 和150 MHz)
- 极低的功耗:
 - 10 种低功耗操作模式用于优化外设活动和唤醒时间以延长电池的寿命
 - 低漏唤醒单元、低功耗定时器和低功耗RTC可以更加灵活地实现低功耗
 - 行业领先的快速唤醒时间

-
- **存储器:**
 - 内存空间可扩展, 从32 KB闪存/ 8 KB RAM 到 1 MB 闪存 / 128 KB RAM。多个独立的闪存模块使同时进行代码执行和固件升级成为可能
 - 可选的16 KB 缓存用于优化总线带宽和闪存执行性能
 - Flex 存储器具有高达512 KB的FlexNVM 和高达16 KB的FlexRAM。FlexNVM 能够被分区以支持额外的程序闪存 (例如引导加载程序)、数据闪存 (例如存储大表) 或者EEPROM 备份。FlexRAM 支持EEPROM 字节写/ 字节擦除操作, 并且指示最大 EEPROM 空间
 - EEPROM 最高超过一千万次的使用寿命
 - EEPROM 擦除/ 写速度远高于传统的EEPROM
 - **模拟混合信号:**
 - 快速、高精度的16位ADC、12位DAC、可编程增益放大器、高速比较器和内部电压参考。提供强大的信号调节、转换和分析性能的同时降低了系统成本
 - **人机接口 (HMI):**
 - 低功耗感应触摸传感接口在所有低功耗模式均可工作
 - **连接性和通信:**
 - UART支持ISO7816 和 IrDA, I2S、CAN、I2C 和 SPI
 - **可靠性和安全性:**
 - 硬件循环冗余校验引擎用于验证存储器内容、通信数据和增加的系统可靠性
 - 独立时钟工作的COP 用于防止代码跑飞
 - 外部看门狗监控
 - **定时和控制:**
 - 强大的FlexTimers 支持通用、PWM 和电机控制功能
 - 载波调制器发射器用于产生红外波形
 - 可编程中断定时器用于RTOS 任务调度或者为ADC 转换和可编程延迟模块提供触发源
 - **外部接口:**
 - 多功能外部总线接口提供和外部存储器、门阵列逻辑或LCD 的接口
 - **系统:**
 - 5 V 容限的GPIO 带引脚中断功能
 - 从 1.71 V 到 3.6 V 的宽操作电压范围, 闪存编程电压低至 1.71 V , 并且此时闪存和模拟外设所有功能正常
 - 运行温度 -40 °C 到105 °C

除了以上共性, 图 2-2列出了各Kinetis 系列所特有的性能。

	USB OTG (FS & HS)	段式 LCD	NAND 闪存控制器	片上单元	以太网 (IEEE 1588)	加密 (CAU/PNG)	双CAN	硬件防篡改检测	DSP 控制器				
K60系列 256KB-1MB 100-256引脚	●		●	●	●	●	●	●	●	共有的系统 IP	共有的模拟 IP	共有的数字 IP	开发工具
K40系列 64-512KB 64-144引脚	●	●					●			32位ARM Cortex-M4 内核带 DSP 指令	16位 ADC	CRC	带 Processor Expert的 IDE
K30系列 64-512KB 64-144引脚		●					●			下一代闪存, 高可靠性, 快速访问	可编程增益放大器	PC	OS USB、TCP/IP、 安全库
K20系列 32KB-1MB 32-144引脚	●		●	●			●			Flex存储器 w/ EEPROM 性能	12位 DAC	I²S	模块化嵌入式 硬件开发系统
K10系列 32KB-1MB 32-144引脚			●	●			●			SRAM	可编程延迟块	UART/SPI	应用软件栈、外设 驱动器和应用库 (电机控制、 HMI、USB)
										存储器保护单元	高速比较器	外部总线接口	应用软件栈、外设 驱动器和应用库 (电机控制、 HMI、USB)
										低电压低功耗 多操作模式, 时钟门控 (1.71-3.6V 5V 容限 I/O)	低功耗 感应触摸传感	电机控制定时器	应用软件栈、外设 驱动器和应用库 (电机控制、 HMI、USB)
										DMA		SDHC	强大的第三方 生态系统
												RTC	

图 2-2 Kinetis 系列微控制器特性

2.3 Kinetis系列微控制器存储器映像与编程结构

2010年11月，Freescale开始提供K60的样片，2011年上半年K60批量上市。MK60N512VMD100是K60系列 MCU，是Kinetis系列的代表芯片。本书以该芯片为主要蓝本阐述嵌入式开发所必备的硬件设计、软件设计及相关技术。一般来说，学习一个新的MCU芯片，若用C语言进行编程，比较快速的学习过程是：

- (1) 了解性能及内部主要功能模块与存储空间的地址分配。
- (2) 了解基本的编程结构、编程模式及寻址方式。
- (3) 了解中断结构。
- (4) 了解芯片的引脚的总体布局情况，掌握硬件最小系统电路。
- (5) 理解第一个工程的结构，理解工程中各个文件的基本功能。一般来说，第一个工程为一个简单的小程序，如利用通用 I/O 模块编程控制几个发光二极管，主要目的是给出程序框架和工作过程。
- (6) 进行实际环境的编译（compile）、链接（link）生成可以下载到芯片内部 Flash 存储器中的程序（可以运行的机器码），基本理解列表文件、机器码文件。
- (7) 一定要有硬件评估环境，这是学习新 MCU 的必需品。这样就可将程序利用写入调试器下载到目标 MCU 中，在目标板上，观察运行情况。随后，可进一步利用嵌入式软件的打桩调试技术，即在被测程序代码中插入一些函数或语句，利用这些函数或语句产生可在硬件板上显示物理现象，供观察程序运行情况之用。
- (8) 从整个工程组成、各个文件、写入 Flash 存储器的机器码等角度，透彻理解第一工程的执行过程。
- (9) 理解第一个带有中断过程的 C 语言工程结构，理解主循环与中断两条程序执行路线各自的作用。

至此，以上学习过程已经覆盖学习一个新 MCU 硬件设计与软件编程的基本要素。完成了以上学习步骤，就完成了“基本入门”过程。随后，可以在此框架下，结合嵌入式构件方

法，逐个模块学习就方便了。

本章给出上述过程的（1）—（4）步，下一章给出上述过程的（5）—（8）步。这两章完成了“基本入门”的前8步。“基本入门”的第9步（第一个中断例程）将在第4章阐述。为了规范编程，符合嵌入式软件工程的基本要求，提高硬件及底层驱动软件的可复用与可移植性。

2.3.1 K60 系列 MCU 性能概述与内部结构简图

K60 微控制器系列具有IEEE 1588 以太网，全速和高速 USB 2.0 On-The-Go 带设备充电探测，硬件加密和防篡改探测能力，具有丰富的模拟、通信、定时和控制外设，从100 LQFP 封装 256 KB 闪存开始可扩展到256 MAPBGA 1MB 闪存。大闪存的 K60 系列器件还可提供可选的单精度浮点单元、NAND 闪存控制器和DRAM 控制器。

图 2-3 给出了 K60 系列的模块结构框图。

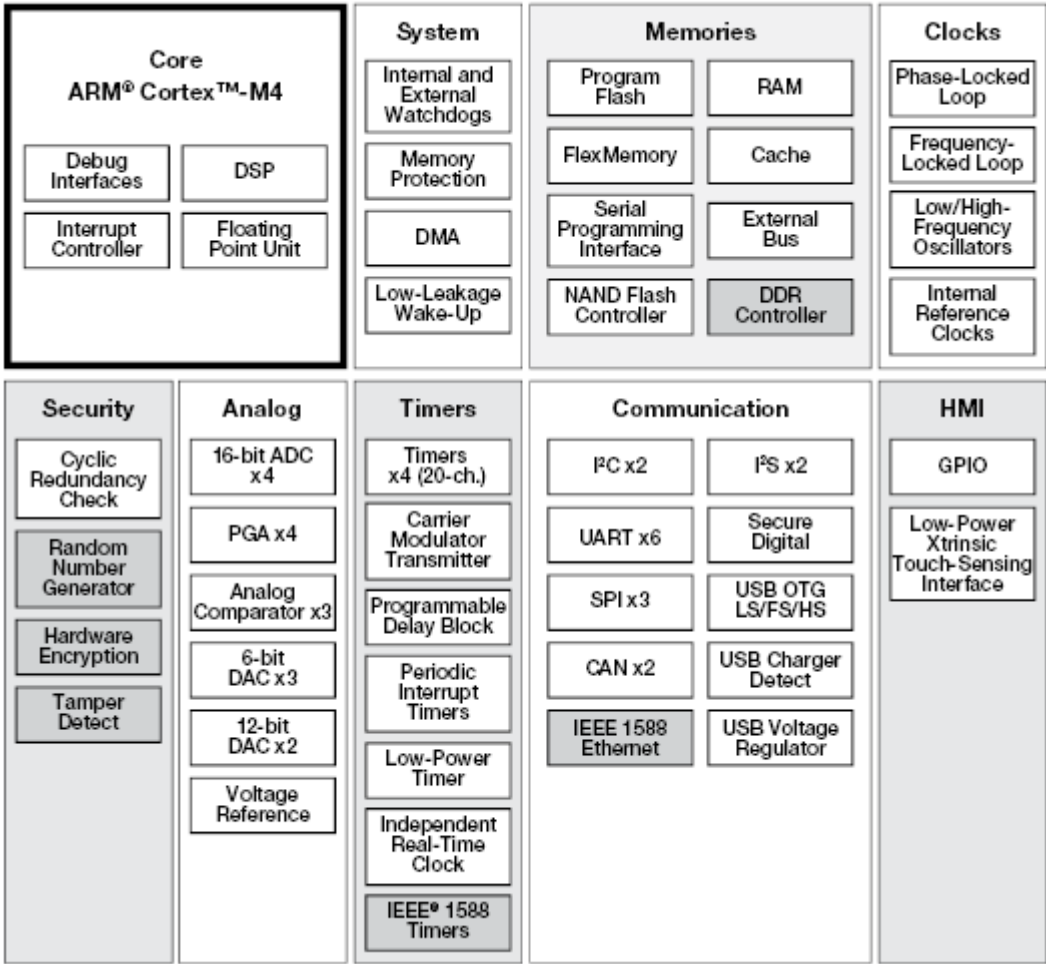


图 2-3 K60 模块结构图

可选的芯片类型有 MK60N256VLQ100、MK60X256VLQ100、MK60N512VLQ100、MK60N256VMD100、MK60X256VMD100 和 MK60N512VMD100。这些芯片的 CPU 频率与引脚数都一样，但是在封装、Flash 容量、程序空间等方面还是有差别的。表 2-2 给出了可选芯片的 MCU 简要描述，供选型参考之用。

表 2-2 K60 可选择的芯片类型

芯片类型	CPU 频率	引脚数	封装	Flash 容量	程序空间	EEPROM	SRAM	GPIO
MK60N256VLQ100	100 MHz	144	LQFP	256 KB	256 KB	—	64 KB	100
MK60X256VLQ100	100 MHz	144	LQFP	512 KB	256 KB	4 KB	64 KB	100
MK60N512VLQ100	100 MHz	144	LQFP	512 KB	512 KB	—	128 KB	100
MK60N256VMD100	100 MHz	144	MAP BGA	256 KB	256 KB	—	64 KB	100
MK60X256VMD100	100 MHz	144	MAP BGA	512 KB	256 KB	4 KB	64 KB	100
MK60N512VMD100	100 MHz	144	MAP BGA	512 KB	512 KB	—	128 KB	100

表 2-3 总结了 K60 系列 MCU 的共性

表 2-3. K60 系列器件的共性

工作特性	<ul style="list-style-type: none"> • 电压范围 1.71V - 3.6V • 闪存编程电压最低至1.71V • 温度范围(TA) -40 to 105°C • 灵活的工作模式
内核特性	<ul style="list-style-type: none"> • 32 位 ARM Cortex-M4 内核 • 支持DSP 指令 • 嵌套向量中断控制器(NVIC) • 异步唤醒中断控制器(AWIC) • 调试和跟踪 <ul style="list-style-type: none"> • 2 引脚串口调试 (SWD) • IEEE 1149.1 JTAG 调试 (JTAG) • IEEE 1149.7 简洁 JTAG (cJTAG) • 端口跟踪接口单元(TPIU) • 闪存片和断点单元 (FPB) • 数据检测和跟踪单元(DWT) • 指令跟踪宏单元 (ITM)
系统和功耗管理	<ul style="list-style-type: none"> • 带外部监控引脚的软件和硬件看门狗 • 带16 个通道的DMA 控制器 • 低漏唤醒单元 (LLWU) • 带10 种功耗模式的功耗管理控制器 • 不可屏蔽中断(NMI) • 每个芯片 128 位唯一标识(ID) 数
时钟	<ul style="list-style-type: none"> • 多用途时钟发生器 • PLL 和FLL • 内部参考时钟(32kHz 或 2MHz) • 4MHz 到 32MHz 晶振 • 32kHz 到 40kHz 晶振

	<ul style="list-style-type: none"> • 内部 1kHz 低功耗振荡器 • DC 到 50MHz 外部方波输入时钟
存储器和存储器接口	<ul style="list-style-type: none"> • Flex 存储器有FlexNVM (非易失闪存用于执行程序代码、存储数据或者备份EEPROM 数据) 或者FlexRAM (RAM 存储器被用作传统的RAM 或者高耐擦写EEPROM 存储和加快闪存程序运行) • 闪存安全性和保护特性 • 串行闪存编程接口(EzPort)
安全性和集成性	<ul style="list-style-type: none"> • 循环冗余校验(CRC)
模拟	<ul style="list-style-type: none"> • 16 位 SAR ADC • 可编程的电压参考(VREF) • 12 位 DAC • 带 6 位 DAC 的高速模拟比较器 (CMP)
定时器	<ul style="list-style-type: none"> • 1x8ch 电机控制/ 通用/PWM 定时器(FTM) • 2x2ch 正交解码器/ 通用/PWM 定时器 (FTM) • 载波调制定时器(CMT) • 可编程延迟模块 (PDB) • 1x4ch 可编程中断定时器(PIT) • 低功耗定时器(LPT)
通信	<ul style="list-style-type: none"> • 支持IEEE 1588 的以太网接口 • USB 全速/ 低速 OTG/ 主机/ 从设备接口 • CAN • SPI • I2C, 支持SMBUS • UART (带 ISO7816、IrDA 和硬件流控)
人机接口	<ul style="list-style-type: none"> • GPIO 支持引脚中断、DMA 请求、数字滤波和其他引脚控制选项 • 最大允许5V 输入 • 电容式触摸传感输入

2.3.2 K60 系列存储器映像

表 2-4 列出了不同频率、不同封装的 K60 系列微控制器的存储器大小。

表2-4 K60 系列MCU 概述

	存储器				封装					
CPU 频率 (MHz)	闪存 (KB)	FlexNVM (KB)	SRAM (KB)	FlexRAM (KB)	100 LQFP (14x14)	104 BGA (8x8)	144 LQFP (20x20)	144 BGA (13x13)	196 BGA (15x15)	256 BGA (17x17)
100	256	—	64	—	+	+	+	+	—	—
100	512	—	128	—	+	+	+	+	—	—
100	256	256	64	4	+	+	+	+	—	—
120	512	512	128	16	—	—	+	+	+	+
150	512	512	128	16	—	—	+	+	+	+
120	1024	—	128	—	—	—	+	+	+	+

150	1024	—	128	—	—	—	+	+	+	+
-----	------	---	-----	---	---	---	---	---	---	---

Flex存储器是飞思卡尔的新一代Flex存储器技术，为需要片上EEPROM、额外程序或数据的开发者提供非常多样化和强大的解决方案。Flex 存储器和SRAM 一样简单快速，当用作高耐久性擦写EEPROM 时，在完成程序运行和擦除功能时不需要用户或者系统干预。Flex 存储器同时能提供平行于主程序闪存的额外闪存 (FlexNVM) 用于数据或者程序存储。Flex 存储器使您能完全配置 FlexNVM 和 FlexRAM 模块，从而为应用提供最均衡的存储器资源。用户可配置参数包括：EEPROM 大小、擦写次数、写大小和额外程序/ 数据闪存的大小。

FlexNVM能被用作EEPROM 配置的一部分、额外的程序或者数据闪存。也可以一部分用作闪存同时另一部分被用作增强型EEPROM 备份

FlexRAM能被用作EEPROM 配置的一部分或者额外的系统RAM

K60 系列的 MCU 为 32 位微控制器，可寻址 4GB 的地址空间，地址范围为 0x0000_0000~0xFFFF_FFFF。K60 系列的存储器空间地址映像见表 2-5 所示。

表 2-5 K60 系列的存储器空间地址映像

地址范围	空间大小	实际的物理对象
0x0000_0000~0x0FFF_FFFF	256MB	可编程 flash 和只读数据(包括一开始 1024 字节的异常中断向量)
0x1000_0000~0x13FF_FFFF	64MB	对 MK60N256VLQ100 芯片：未使用 对 MK60X256VLQ100 芯片：FlexNVM 对 MK60N512VLQ100 芯片：未使用 对 MK60N256VMD100 芯片：未使用 对 MK60X256VLQ100 芯片：FlexNVM 对 MK60N512VMD100 芯片：未使用
0x1400_0000~0x17FF_FFFF	64MB	对有 FlexNVM 的设备：FlexRAM 对只有可编程 Flash 的设备：可编程加速 RAM
0x1800_0000~0x1FFF_FFFF	128MB	SRAM_L
0x2000_0000~0x200F_FFFF	1MB	SRAM_U
0x2010_0000~0x21FF_FFFF	31MB	未使用
0x2200_0000~0x23FF_FFFF	32MB	SRAM_U 的混合位宽区
0x2400_0000~0x3FFF_FFFF	448MB	未使用
0x4000_0000~0x4007_FFFF	512KB	外设桥 0 (AIPS-Lite0)
0x4008_0000~0x400F_FFFF	508KB	外设桥 1 (AIPS-Lite1)
0x400F_F000~0x400F_FFFF	4KB	通用输入输出
0x4010_0000~0x41FF_FFFF	25MB	未使用
0x4200_0000~0x43FF_FFFF	32MB	外设桥 (AIPS-Lite) 与通用输入输出的混合位宽区
0x4400_0000~0x5FFF_FFFF	448MB	未使用
0x6000_0000~0xDFFF_FFFF	2GB	Flexbus
0xE000_0000~0xE00F_FFFF	1MB	私有外设
0xE010_0000~0xFFFF_FFFF	511MB	未使用

片上 RAM 分为 SRAM_L 和 SRAM_U，从表 2-5 可见它们是连续的存储映射。对于 SRAM_L 和 SRAM_U 如果处于片外时，在请求主机访问总线时将会产生一个错误。外设存储映射在 0x4000_0000 到 0x400F_FFFF 区有两个从机端口，实现了 2 个外设桥(AIPS-Lite0 和 AIPS-Lite 1)： AIPS-Lite0 占 512KB ， AIPS-Lite1 占 508KB, 4KB 的 GPIO。 AIPS-Lite0

连接到从机端口 2，可访问区间是 0x4000_0000 到 0x4007_FFFF。AIPS-Lite1 和 GPIO 共享从机端口 3，AIPS-Lite1 的可访问区间是 0x4008_0000 到 0x400F_EFFF，GPIO 的 4KB 可访问区间是 0x400F_F000 到 0x400F_FFFF。它可直接连接到门开关提供的主机不需要等待 AIPS-Lite 控制器的状态。本模块的时钟是由 SIM 寄存器的 AIPS 控制位控制的。访问任何未实现或未使能的外设桥信号将产生错误。对于可编程模块访问外设桥，只有 4KB 的实现空间。

2.4 K60的引脚功能与硬件最小系统

本书以 144 引脚 LQFP 封装的 MK60N512VMD100 芯片为例介绍 K60 的编程和应用。

2.4.1K60 的引脚功能

图 2-4 是 144 引脚 LQFP 封装的 MK60N512VMD100 的引脚图。每个引脚都可能有多多个复用功能，有的引脚有两个复用功能，有的有四个复用功能，还有的有六个复用功能，系统设计时必须注意只能使用其中的一个功能。一般情况下，需要按引脚功能分类了解 MCU 引脚功能情况。

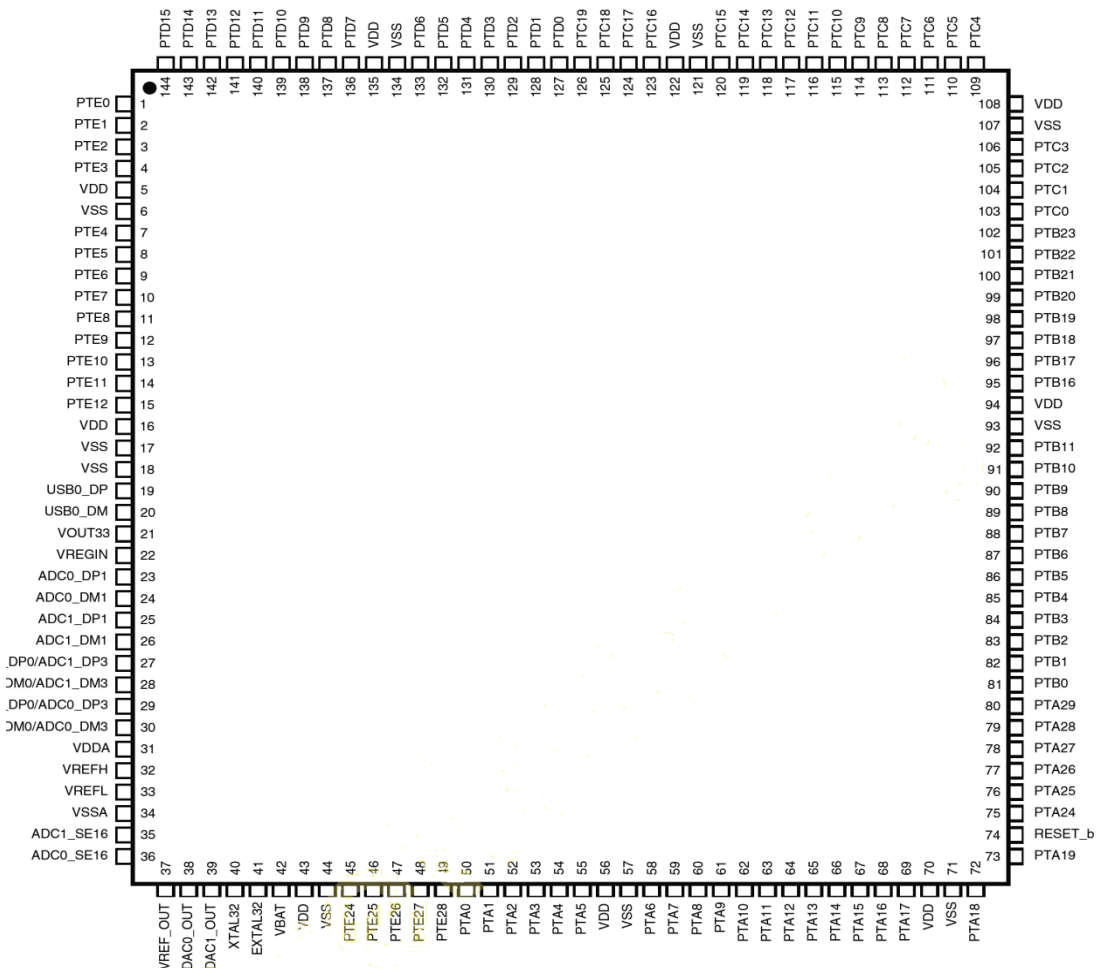


图 2-4 是 144 引脚 LQFP 封装的 MK60N512VMD100 的引脚图

表 2-6 按引脚功能分类列出了 MK60N512VMD100 的工作支撑引脚。

表 2-6 MK60N512VMD100 工作支撑引脚表

分类	引脚名	引脚号	典型值	功能描述	备注
电源类	VDD	5、16、43、 56、70、94、 108、122、 135	3.3V	为 I/O 引脚提供电源	范围 1.71~3.6V
	VSS	6、17、18、 44、57、71、 93、107、 121、134、	0V	为 I/O 引脚提供参考地	
	VDDA	31	3.3V	为 AD 转换模块供电电源	范围 1.71~3.6V
	VSSA	34	0V	为 AD 转换模块供参考地	
	VREFH	32		AD 模块参考高电平	
	VREFL	33		AD 模块参考高低平	
	VREF_OUT	37		内部产生的参考输出电压	
	VREGIN	22		USB 调节器输入电源	2.7~5.5
	VOUT33	21		USB 调节器输出电源	
	VBAT	42		32KHz 晶振电源	1.71~3.6
复位	$\overline{\text{RESET}}$	74			低电平时芯片复位
晶振	EXTAL32	41		RTC 晶振输入引脚 (32KHz)	
	XTAL32	40		RTC 晶振输出引脚 (32KHz)	
	EXTAL	72		外部晶振输入引脚	
	XTAL	73		外部晶振输出引脚	
写入器	JTAG_TDI	51		JTAG 测试数据输入引脚	TCK 线上升沿时, 从线上取数
	JTAG_TDO	52		JTAG 测试数据输出引脚	TCK 线下降沿时, 数据上线
	JTAG_TCLK	50		JTAG 测试时钟线	
	JTAG_TMS	53		JTAG 测试模式选择线	用于设置 JTAG 控制器的状态
	JTAG_TRST	55		JTAG 复位	

除去需要服务的引脚外, 其它引脚可以为实际系统提供 I/O 服务。芯片提供服务的引脚也可称为 I/O 端口资源类引脚。MK60N512VMD100 (144 引脚 LQFP 封装) 的有多达 100 个 I/O 引脚。其中 A 口 26 个, B 口 20 个, C 口 20 个, D 口 16 个, E 口 18 个, 详细情况见表 2-7 所示。

表 2-7 I/O 端口资源表

口名	引脚数	引脚名	引脚号	功能描述					
				第一	第二	第三	第四	第五	第六
A	26	PTA0	50	TSIO_C H1	GPIO	UART0_C TS_b	FTM0_ CH5	JTAG_TCLK/ SWD_CLK	EZP_CLK
		PTA1	51	TSIO_C H2	GPIO	UART0_R X	FTM0_ CH6	JTAG_TDI	EZP_DI
		PTA2	52	TSIO_C H3	GPIO	UART0_T X	FTM0_ CH7	JTAG_TDO/ TRACE_SW O	EZP_DO
		PTA3	53	TSIO_C H4	GPIO	UART0_R TS_b	FTM0_ CH0	JTAG_TMS/ SWD_DIO	
		PTA4	54	TSIO_C H5	GPIO	FTM0_CH 1	NMI_b	EZP_CS_b	
		PTA5	55	GPIO	FTM0_ _CH2	RMII0_R XER/MII0 _RXER	CMP2_ _OUT	I2S0_RX_BC LK	JTAG_T RST
		PTA6	58	GPIO	FTM0_ CH3	TRACE_C LKOUT			
		PTA7	59	ADC0_ SE10	GPIO	FTM0_CH 4	TRACE_ D3		
		PTA8	60	ADC0_ SE11	GPIO	FTM1_CH 0	FTM1_ QD_P HA	TRACE_D2	
		PTA9	61	GPIO	FTM1_ _CH1	MII0_RX D3	FTM1_ _QD_P HB	TRACE_D1	
		PTA1 0	62	GPIO	FTM2_ CH0	MII0_RXD 2	FTM2_ QD_P HA	TRACE_D0	
		PTA1 1	63	GPIO	FTM2_ CH1	MII0_RXC LK	FTM2_ QD_P HB		
		PTA1 2	64	CMP2_ IN0	GPIO	CAN0_TX	FTM1_ CH0	RMII0_RXD1/ MII0_RXD1	I2S0_TX D
		PTA1 3	65	CMP2_ IN1	GPIO	CAN0_RX	FTM1_ CH1	RMII0_RXD0/ MII0_RXD0	I2S0_TX_ FS
		PTA1 4	66	GPIO	SPI0_P CS0	UART0_T X	RMII0_ CRS_ DV/ MII0_R XDV	I2S0_TX_BC LK	

		PTA1 5	67	GPIO	SPI0_S Ck	UART0_R X	RMII0_ TXEN/ MII0_T XEN	I2S0_RXD	
		PTA1 6	68	GPIO	SPI0_S OUT	UART0_C TS_b	RMII0_ TXD0/ MII0_T XD0	I2S0_RX_FS	
		PTA1 7	69	ADC1_ SE17	GPIO	SPI0_SIN	UART0_ RTS _b	RMII0_TXD1/ MII0_TXD1	I2S0_MC LK
		PTA1 8	72	EXTAL	GPIO	FTM0_FLT 2	FTM_C LKIN0		
		PTA1 9	73	XTAL	GPIO	FTM1_FLT 0	FTM_C LKIN1	LPT0_ALT1	
		PTA2 4	75	GPIO	MII0_T XD2	FB_A29			
		PTA2 5	76	GPIO	MII0_T XCLK	FB_A28			
		PTA2 6	77	GPIO	MII0_T XD3	FB_A27			
		PTA2 7	78	GPIO	MII0_C RS	FB_A26			
		PTA2 8	79	GPIO	MII0_T XER	FB_A25			
		PTA2 9	80	GPIO	MII0_C OL	FB_A24			
B	20	PTB0	81	ADC0_ SESE8/ ADC1_ SE8/ TSI0_C H0	GPIO	I2C0_SCL	FTM1_ CH0	RMII0_MDIO/ MII0_MDIO	FTM1_Q D_PHA
		PTB1	82	ADC0_ SE9/ ADC1_ SE9/ TSI0_C H6	GPIO	I2C0_SDA	FTM1_ CH1	RMII0_MDC/ MII0_MDC	FTM1_Q D_P HB
		PTB2	83	ADC0_ SE12/ TSI0_C H7	GPIO	I2C0_SCL	UART0_ RTS _b	ENET0_1588 _TMR0	FTM0_FL T3

		PTB3	84	ADC0_SE13/ TSI0_C H8	GPIO	I2C0_SDA	UART0 _CTS _b	ENET0_1588 _TMR1	FTM0_FL T0
		PTB4	85	ADC1_SE10	GPIO	ENET0_15 88_TMR2	FTM1_ FLT0		
		PTB5	86	ADC1_SE11	GPIO	ENET0_15 88TMR3	FTM2_ FLT0		
		PTB6	87	ADC1_SE12	GPIO	FB_AD23			
		PTB7	88	ADC1_SE13	GPIO	FB_AD22			
		PTB8	89	GPIO	UART3 _RTS	FB_AD21			
		PTB9	90	GPIO	SPI1_P CS1	UART3_C TS	FB_AD 20		
		PTB1 0	91	ADC1_SE14	GPIO	SPI1_PCS 0	UART3 _RX	FB_AD19	FTM0_FL T1
		PTB1 1	92	ADC1_SE15	GPIO	SPI1_SCK	UART3 _TX	FB_AD18	FTM0_FL T2
		PTB1 6	95	TSI0_C H9	GPIO	SPI1_SO UT	UART0 _RX	FB_AD17	EWM_IN
		PTB1 7	96	TSI0_C H10	GPIO	SPI1_SIN	UART0 _TX	FB_AD16	EWM_OU T_b
		PTB1 8	97	TSI0_C H11	GPIO	CAN0_TX	FTM2_ CH0	I2S0_TX_BC LK	FB_AD15 HA
		PTB1 9	98	TSI0_C H12	GPIO	CAN)_RX	FTM2_ CH1	I2S0_TX_FS	FB_OE_b
		PTB2 0	99	GPIO	SPI2_P CS0	FB_AD31	CMP0_ OUT		
		PTB2 1	100	GPIO	SPI2_S CK	FB_AD30	CMP1_ OUT		
		PTB2 2	101	GPIO	SPI2_S OUT	FB_AD29	CMP2_ OUT		
		PTB2 3	102	GPIO	SPI2_S IN	FB_AD28			
C	20	PTC0	103	ADC0_SE14/ TSI0_C H13	GPIO	SPI0_PCS 4	PDB0_ EXTR G	I2S0_TXD	FB_AD14

		PTC1	104	ADC0_ SE15/ TSI0_C H14	GPIO	SPI0_PCS 3	UART1 _RTS _b	FTM0_CH0	FB_AD13
		PTC2	105	ADC0_ SE4b/ CMP1_ IN0/ TSI0_C H15	GPIO	SPI0_PCS 2	UART1 _CTS _b	FTM0_CH1	FB_AD12
		PTC3	106	CMP1_ IN1	GPIO	SPI0_PCS 1	UART1 _RX	FTM0_CH2	FB_CLK OUT
		PTC4	109	GPIO	SPI0_P CS0	UART1_T X	FTM0_ CH3	FB_AD11	CMP1_O UT
		PTC5	110	GPIO	SPI0_S CK	LPT0_ALT 2	FB_AD 10	CMP0_OUT	
		PTC6	111	CMP0_ IN0	GPIO	SPI0_SOU T	PDB0_ EXTRG	FB_AD9	
		PTC7	112	CMP0_ IN1	GPIO	SPI0_SIN	FB_AD 8		
		PTC8	113	ADC1_ SE4b/ CMP0_ IN2	GPIO	I2S0_MCL K	I2S0_C LKIN	FB_AD7	
		PTC9	114	ADC1_ SE5b/ CMP0_ IN3	GPIO	I2S0_RX_ BC LK	FB_AD 6	FTM2_FLT0	
		PTC1 0	115	ADC1_ SE6b/ CMP0_ IN4	GPIO	I2C1_SCL	I2S0_R X_FS	FB_AD5	
		PTC1 1	116	ADC1_ SE7b	GPIO	I2C1_SDA	I2S0_R XD	FB_RW_b	
		PTC1 2	117	GPIO	UART4 _RTSb	FB_AD27			
		PTC1 3	118	GPIO	UART4 _CTS_ b	FB_AD26			
		PTC1 4	119	GPIO	UART4 _RX	FB_AD25			

D	16	PTC1 5	120	GPIO	UART4 _TX	FB_AD24			
		PTC1 6	123	GPIO	CAN1_ RX	UART3_R X	ENET0 _1588 _TMR0	FB_CS5_b/ FB_TSIZ1/ FB_BE23_16 _BLS15_8_b	
		PTC1 7	124	GPIO	CAN1_ TX	UART3_T X	ENET0 _1588 _TMR1	FB_CS4_b/ FB_TSIZ0/ FB_BE31_24 _BLS7_0_b	
		PTC1 8	125	GPIO	UART3 _RTS_ b	ENET0_15 88 _TMR2	FB_TB ST_b/ FB_CS 2_b/ FB_BE 15_8_ BLS23 _16_b		
		PTC1 9	126	GPIO	UART3 _CTS _b	ENET0_15 88 _TMR3	FB_CS 3_b/ FB_BE 7_0_B LS31_2 4_b	FB_TA_b	
	16	PTD0	127	GPIO	SPI0_P CS0	UART2_R TS_b	FB_AL E/FB_C S1_b/ FB_TS _b		
		PTD1	128	ADC0_ SE5b	GPIO	SPI0_SCK	UART2 _CTS _b	FB_CS0_b	
		PTD2	129	GPIO	SPI0_S OUT	UART2_R X	FB_AD 4		
		PTD3	130	GPIO	SPI0_S IN	UART2_T X	FB_AD 3		
		PTD4	131	GPIO	SPI0_P CS1	UART0_R TS _b	FTM0_ CH4	FB_AD2	EWM_IN
		PTD5	132	ADC0_ SE6b	GPIO	SPI0_PCS 2	UART0 _CTS _b	FTM0_CH5	FB_AD1

E	18	PTD6	133	ADC0_SE7b	GPIO	SPI0_PCS3	UART0_RX	FTM0_CH6	FB_AD0
		PTD7	136	GPIO	CMT_IRO	UART0_TX	FTM0_CH7	FTM0_FLT1	
		PTD8	137	GPIO	I2C0_SCL	UART5_RX	FB_A16		
		PTD9	138	GPIO	I2C0_SDA	UART5_TX	FB_A17		
		PTD10	139	GPIO	UART5_RTSLb	FB_A18			
		PTD11	140	GPIO	SPI2_PCS0	UART5_CTSb	SDHC0_CLKIN	FB_A19	
		PTD12	141	GPIO	SPI2_SCK	SDHC0_D4	FB_A20		
		PTD13	142	GPIO	SPI2_SOUT	SDHC0_D5	FB_A21		
		PTD14	143	GPIO	SPI2_SIN	SDHC0_D6	FB_A22		
		PTD15	144	GPIO	SPI2_PCS1	SDHC0_D7	FB_A22		
	18	PTE0	1	ADC1_SE4a	GPIO	SPI1_PCS1	UART1_TX	SDHC0_D1	I2C1_SDA
		PTE1	2	ADC1_SE5a	GPIO	SPI1_SOUT	UART1_RX	SDHC0_D0	I2C1_SCL
		PTE2	3	ADC1_SE6a	GPIO	SPI1_SCK	UART1_CTSb	SDHC0_DCLK	
		PTE3	4	ADC1_SE7a	GPIO	SPI1_SIN	UART1_RTSLb	SDHC0_CMD	
		PTE4	7	GPIO	SPI_PCS0	UART3_TX	SDHC0_D3		
		PTE5	8	GPIO	SPI_PCS2	UART3_RX	SDHC0_D2		

		PTE6	9	GPIO	SPI_P CS3	UART3_C TS_b	I2S0_M CLK	I2S0_CLKIN	
		PTE7	10	GPIO	UART3 _RTS _b	I2S0_RXD			
		PTE8	11	GPIO	UART5 _TX	I2S0_RX_ FS			
		PTE9	12	GPIO	UART5 _RX	I2S0_RX_ BCLK			
		PTE1 0	13	GPIO	UART5 _CTS _b	I2S0_TXD			
		PTE1 1	14	GPIO	UART5 _RTS _b	I2S0_TX_F S			
		PTE1 2	15	GPIO	I2S0_T X_BC LK				
		PTE2 4	45	ADC0_ SE17	GPIO	CAN1_TX	UART4 _TX	EWM_OUT_b	
		PTE2 5	46	ADC0_ SE18	GPIO	CAN1_RX	UART4 _RX	EWM_IN	
		PTE2 6	47	GPIO	UART4 _CTSb	ENET_158 8_ CLKIN	RTC_C LKOU T	USB_CLKIN	
		PTE2 7	48	GPIO	UART4 _RTSb				
		PTE2 8	49	GPIO					
总	100								

2.4.2K60 硬件最小系统

MCU的硬件最小系统是指可以使内部程序运行的所必须的外围电路，也可以包括写入器接口电路。使用一个芯片，必须完全理解其硬件最小系统。当MCU工作不正常时，首先查找最小系统中可能出错的元件。一般情况下，MCU的硬件最小系统由电源、晶振及复位等电路组成。芯片要能工作，必须有电源与工作时钟，至于复位电路则提供不掉电情况下MCU重新启动的手段。由于Flash存储器制造技术的发展，大部分芯片提供了在板或在系统（On System）写入程序功能，即把空白芯片焊接到电路板上后，再通过写入器把程序下载到芯片中。这样，硬件最小系统应该把写入器的接口电路也包含在其中。基于这个思路，MK60N512VMD100芯片的硬件最小系统包括电源电路、复位电路、晶振电路及JTAG接口电路。下面分别对这些电路给出简明分析。

等到当前总线周期结束，这是为了保护数据的完整性。在该总线周期结束后，下一个系统时钟的上升沿时，复位才有效。同步复位有看门狗定时器、软件等。

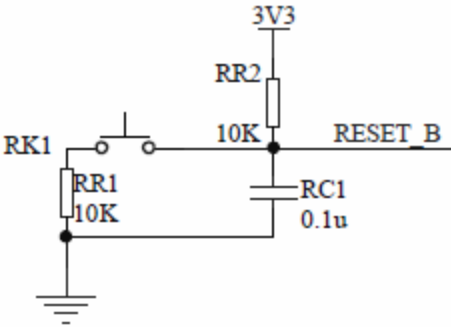


图 2-6 MK60N512VMD100 复位电路

3. 晶振电路

晶振电路为芯片提供准确的工作时钟。
晶体振荡器分为无源晶振和有源晶振两种类型。需要外接电源的晶振称为有源晶振。无源晶振与有源晶振的英文名称不同，无源晶振为crystal(晶体)，而有源晶振则叫做oscillator(振荡器)。无源晶振是有两个引脚的无极性元件，需要借助于时钟电路才能产生振荡信号，

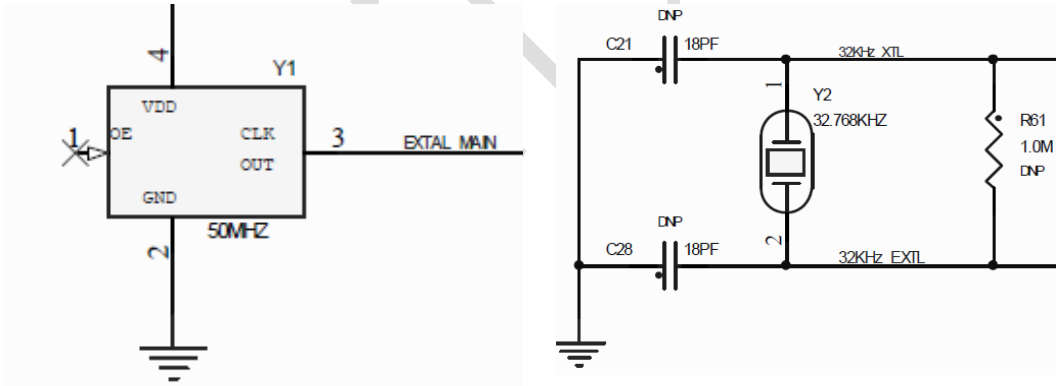


图 2-7 MK60N512VMD100 晶振电路

自身无法振荡起来。MK60N512VMD100内部集成多用途时钟产生器（Multipurpose Clock Generator，MCG）模块，用于将晶振输入时钟倍频至系统所需时钟。MK60N512VMD100共需要两个晶振，一个是芯片的主晶振，用于产生芯片和外设的工作时钟，另一个是实时定时器的晶振（RTC）。苏州大学飞思卡尔嵌入式实验室开发的核心板主芯片时钟使用50MHZ有源晶振，RTC时钟使用32.768KHZ无源晶振，图2-7为系统晶振电路。在硬件布线时需要注意晶振附近不能走高频信号，晶振应该尽量靠近晶振输入引脚。晶振一旦不能正常工作，芯片将无法工作。晶振实际上负责给芯片提供心跳。

4. JTAG 电路

Kinetis 芯片使用的是 ARM Cortex-M4 内核，该内核内部集成了 JTAG（Joint Test Action Group）接口，通过 JTAG 接口可以实现程序下载和调试功能。图 2-8 为 JTAG 接口电路。

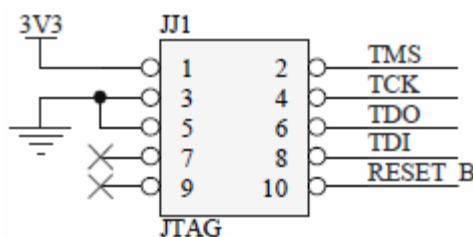


图 2-8 MK60N512VMD100 JTAG 电路

2.4.3 硬件最小系统测试方法

前面介绍了硬件最小系统的设计，给出了硬件最小系统元件的参考值。硬件最小系统电路原理图见光盘。

根据原理图制作了印刷电路板后，就开始硬件电路板的焊接和测试了。具体过程如下：

（1）焊接电源、复位电路、晶振电路、以及 JTAG 接口电路。注意：电源的滤波电容不可漏焊，否则芯片所受干扰较大，影响调试。

（2）在确保电源和地未短路的情况下接通电源，测量电压是否正常，检查按下复位按钮是否能够复位（观察复位指示灯）。

（3）将写入器与电路板连接，启动开发环境 IAR Embedded Workbench for RAM 6.10，对目标 MCU 进行擦除，如果成功则说明最小系统工作正常。

（4）将第一个样例程序下载到 Flash 中，观察小灯闪烁情况。

（5）硬件最小系统测试通过以后就可以进行其他模块焊接。正确的做法是，焊完一个模块后，应紧接着测试该模块工作是否正常，切忌焊接多个模块后再进行测试，因为一旦出现问题，就很难定位具体是哪个模块的问题。

第3章 第一个样例程序及工程组织

本章阐述“入门”过程的（5）—（8）步，通过这个过程，完成第一个 CodeWarrior 工程、IAR 工程的入门。利用 GPIO 模块编程控制发光二极管作为入门例子，给出 CodeWarrior、IAR 工程组织、框架，阐述各个文件的功能，主要目的是使读者理解程序框架和工作过程。重点是透彻理解第一工程的执行过程。

3.1 通用I/O接口基本概念及连接方法

1. I/O 接口的概念

I/O 接口，即输入输出接口，是微控制器同外界进行交互的重要通道。这里的接口英文是 port，也可以翻译为“端口”，另一个英文单词是 interface，也翻译为接口。从中文文字面看，接口与端口似乎有点区别，但在嵌入式系统中它们的含义是相同的。有时 I/O 引脚称为接口（interface），而把用于对 I/O 引脚进行编程的寄存器称为端口（port），实际上它们是紧密相连的。因此，不必深究它们之间的区别。有些书中甚至直接称 I/O 接口（端口）为 I/O 口。在嵌入式系统中，接口千变万化，种类繁多，有显而易见的人机交互接口，如操纵杆、键盘、显示器；也有无人介入的接口，如网络接口、机器设备接口。

2. 通用 I/O

所谓通用 I/O，也记为 GPIO（General Purpose I/O），即基本的输入/输出，有时也称并行 I/O，或普通 I/O，它是 I/O 的最基本形式。本书中使用正逻辑，电源（Vcc）代表高电平，对应数字信号“1”；地（GND）代表低电平，对应数字信号“0”。作为通用输入引脚，MCU 内部程序可以通过端口寄存器读取该引脚，知道该引脚是“1”（高电平）或“0”（低电平），即开关量输入。作为通用输出引脚，MCU 内部程序通过端口寄存器向该引脚输出“1”（高电平）或“0”（低电平），即开关量输出。大多数通用 I/O 引脚可以通过编程来设定工作方式输入或输出，称之为双向通用 I/O。

3. 上拉下拉电阻与输入引脚的基本接法

芯片输入引脚的外部有三种不同的连接方式：带上拉电阻的连接、带下拉电阻的连接和“悬空”连接。通俗地说，若 MCU 的某个引脚通过一个电阻接到电源（Vcc）上，这个电阻被称为“上拉电阻”。与之相对应，若 MCU 的某个引脚通过一个电阻接到地（GND）上，则相应的电阻被称为“下拉电阻”。这种做法使得，悬空的芯片引脚被上拉电阻或下拉电阻初始化为高电平或低电平。根据实际情况，上拉电阻与下拉电阻可以取值在 $1\text{K}\Omega \sim 10\text{K}\Omega$ 之间，其阻值大小与静态电流及系统功耗相关。

图 3-1 给出了一个 MCU 的输入引脚的三种外部连接方式，假设 MCU 内部没有上拉或下拉电阻，图中的引脚 I3 上的开关 K3 采用悬空方式连接就不合适，因为 K3 断开时，引脚 I3 的电平不确定。在图 3-1 中， $R1 \gg R2$ ， $R3 \ll R4$ ，各电阻的典型取值为： $R1=20\text{K}$ ， $R2=1\text{K}$ ， $R3=10\text{K}$ ， $R4=200\text{K}$ 。

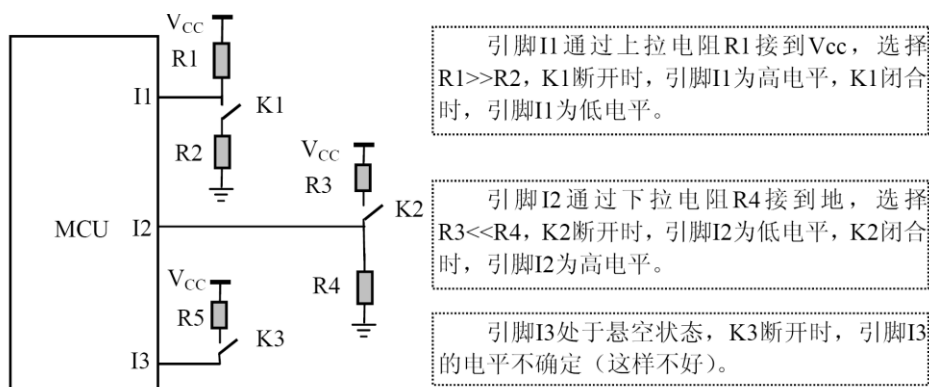


图 3-1 I/O 口输入电路

4. 输出引脚的基本接法

作为通用输出引脚，MCU 内部程序向该引脚输出高电平或低电平来驱动器件工作，即开关量输出。如图 3-2 所示。

一种接法是 O1 引脚直接驱动发光二极管 LED，当 O1 引脚输出高电平时，LED 不亮；当 O1 引脚输出低电平时，LED 点亮。这种接法的驱动电流一般在 2mA~10mA。

另一种接法是 O2 引脚通过一个 NPN 三极管驱动蜂鸣器，当 O2 脚输出高电平时，蜂鸣器响；O2 脚输出低电平时，蜂鸣器不响。这种接法的驱动电流可达 100mA 左右，而 O2 引脚控制电流可以在几个 mA 左右。

若负载需要更大的驱动电流，就必须另外的驱动电路，但对 MCU 编程来说，没有任何影响。

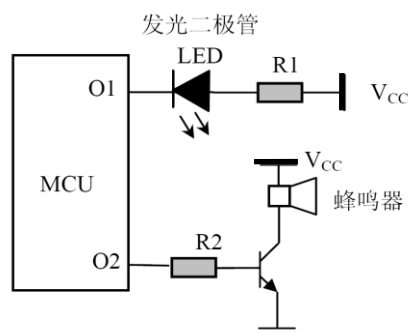


图 3-2 I/O 口输出电路

3.2 MK60N512VMD100的GPIO

MK60N512VMD100 的大部分引脚具有多重功能，可以通过编程设定使用其中一种功能。GPIO 作为基本功能，MK60N512VMD100 有 5 个 GPIO 口，共 100 个引脚。分别为 A 口、B 口、C 口、D 口、E 口。A 口有 26 个引脚，分别为 PTA0~PTA19、PTA24~PTA29，B 口有 20 个引脚，分别为 PTB0~PTB11、PTB16~PTB23，C 口有 20 引脚，分别为 PTC0~PTC19，D 口有 16 个引脚，分别为 PTD0~PTD15，E 口有 18 个引脚，分别为 PTE0~PTE12、PTE24~PTE28。

每个口均包含 6 个寄存器，下面分别介绍一下。

1. 数据输出寄存器（GPIOx_PDOR），可读写，32 位，复位时为 0。无论引脚上的逻辑电平为 0 还是为 1，相应的引脚都被配置为输出。

2. 数据输入寄存器（GPIOx_PDIR），只读，32 位，复位时为 0。读为 0 时，说明相应引脚上为低电平或配置为数字功能，读为 1 时，说明相应引脚上为高电平。

3. 数据方向寄存器（GPIOx_PDDR），可读写，32 位，复位时为 0。0——定义为输入，1——定义为输出。

4. 输出设置寄存器（GPIOx_PSOR），可写，32 位，复位时为 0。对该寄存器进行写操作将该表 PDOR 寄存器的值。写 0 时，不改变 PDOR 上的相应位，写 1 时，将 PDOR 上的

相应位置 1。

5. 输出清除寄存器 (GPIOx_PCOR), 可写, 32 位, 复位时为 0。对该寄存器进行写操作将该表 PDOR 寄存器的值。写 0 时, 不改变 PDOR 上的相应位, 写 1 时, 将 PDOR 上的相应位清 0。

6. 输出触发寄存器 (GPIOx_PTOR), 可写, 32 位, 复位时为 0。对该寄存器进行写操作将该表 PDOR 寄存器的值。写 0 时, 不改变 PDOR 上的相应位, 写 1 时, 将 PDOR 上的相应位反转。

GPIO 的基本编程方法:

- (1) 通过“数据方向寄存器”设置相应引脚为输入或输出;
- (2) 若是输出引脚, 则通过“数据输出寄存器”设置引脚输出高电平或低电平;
- (3) 若是输入引脚, 则通过“数据输入寄存器”获得引脚的状态。
- (4) 若是输出引脚, 可通过“输出设置寄存器”、“输出清除寄存器”、“输出触发寄存器”来改变引脚的状态。

3.3 开发环境与JTAG写入器

嵌入式软件开发有别于桌面软件开发的一个显著的特点,是它一般需要一个交叉编译和调试环境,即编辑和编译软件在通常的 PC 机上进行,而编译好的软件需要通过写入工具下载到目标机上执行,如 MK60N512VMD100 的目标机上。由于主机和目标机处理器的体系结构彼此不同,从而增加了嵌入式软件开发的难度。所以选择一些好的开发套件有助于对目标机的学习与开发。本书将介绍 IAR Systems 公司的 IAR Embedded Workbench for ARM 6.10 集成开发环境(简称 IAR 环境)、Freescall 公司的 CodeWarrior10.1 集成开发环境(简称 CW 环境)、苏州大学的 MK60N512VMD100 硬件评估板以及 JTAG 写入器。

3.3.1 IAR 开发环境简介与基本使用方法

1. IAR 环境功能和特点

Embedded Workbench for ARM 6.10 是 IAR Systems 公司为 ARM 微处理器开发的一个集成开发环境。比较其他的 ARM 开发环境, IAR 具有入门容易、使用方便和代码紧凑等特点。

IAR 中包含一个全软件的模拟程序(simulator)。用户不需要任何硬件支持就可以模拟各种 ARM 内核、外部设备甚至中断的软件运行环境。从中可以了解和评估 IAR 的功能和使用方法。

IAR 的主要特点如下:

- 1、高度优化的 IAR ARM C/C++ Compiler
- 2、IAR ARM Assembler
- 3、一个通用的 IAR XLINK Linker
- 4、IAR XAR 和 XLIB 建库程序和 IAR DLIB C/C++运行库
- 5、功能强大的编辑器
- 6、项目管理器
- 7、命令行实用程序
- 8、IAR C-SPY 调试器(先进的高级语言调试器)

2. IAR 环境安装与设置

IAR 环境安装并不复杂,按提示步骤一步步来即可,光盘中给出了详细的安装步骤。关键要注意 IAR 环境的设置,这部分光盘中也给出了操作步骤。IAR 环境的运行界面如图 3-3 所示。

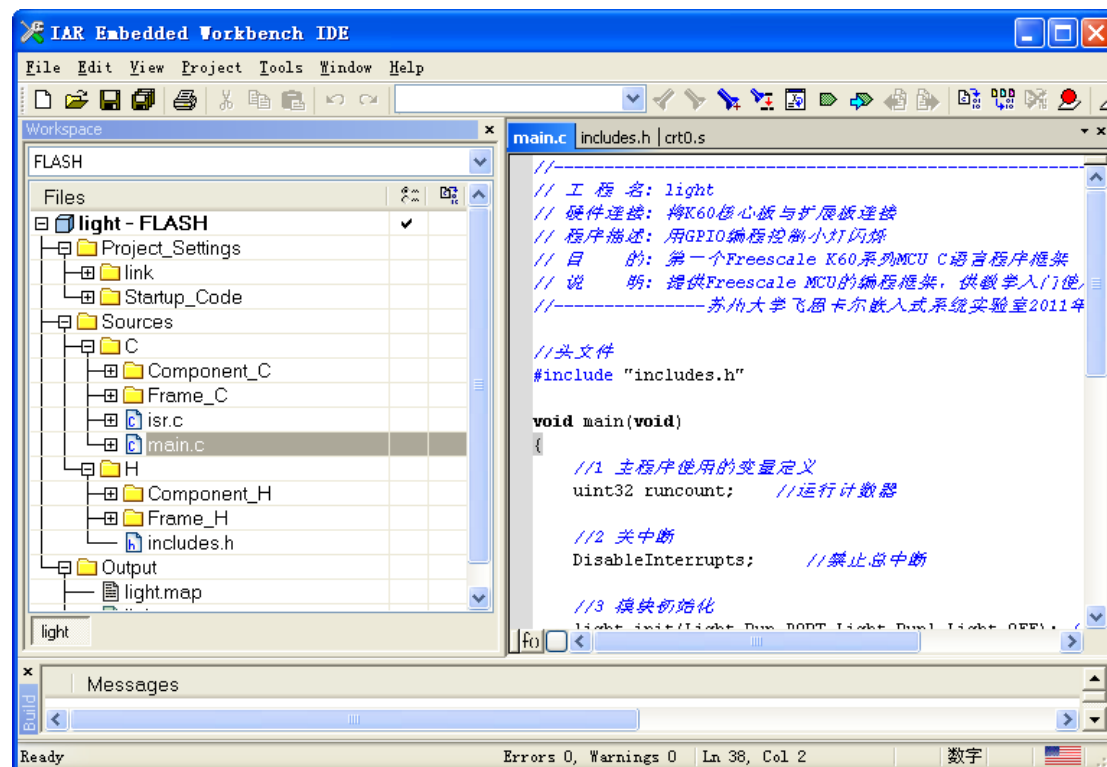


图 3-3 IAR 环境的运行界面

3.3.2 CW 开发环境简介与基本使用方法

1. CW 环境功能和特点

CodeWarrior 开发环境(简称 CW 环境)是 Freescale 公司研发的面向 Freescale MCU 与 DSP 嵌入式应用开发的商业软件工具,其功能强大,是 Freescale 向用户推荐的产品。

CodeWarrior 分为 3 个版本:特别版(Special Edition)、标准版和专业版。特别版是免费的,用于教学目的,对生成的代码量有一定限制,C 语言代码不得超过 12KB,对工程包含的文件数目也限制在 30 个以内。标准版和专业版没有这种限制。3 个版本的区别在于用户所获取的授权文件(license)不同,特别版的授权文件随安装软件附带,不需要特殊申请,标准版和专业版的授权文件需要付费。CodeWarrior 特别版、标准版和专业版的定义随所支持的微处理器的不同而不同,如 CodeWarrior for HC08 V6.0、CodeWarrior for HC12 V4.6、CodeWarrior for ColdFire V6.3 等,本书使用 CodeWarrior V10.1,这个版本支持所有系列的芯片。

CW 环境包括以下几个功能模块:编辑器、源码浏览器、搜索引擎、构造系统、调试器、工程管理器。编辑器、编译器、连接器和调试器对应开发过程的四个主要阶段,其它模块用以支持代码浏览和构造控制,工程管理器控制整个过程。该集成环境是一个多线程应用,能在内存中保存状态信息、符号表和对象代码,从而提高操作速度;能跟踪源码变化,进行自

动编译和连接。

2. CW 环境安装与设置

CW 环境安装没有什么特别之处，在 Windows 操作系统上，只要按照安装向导单击鼠标就可以自动完成。

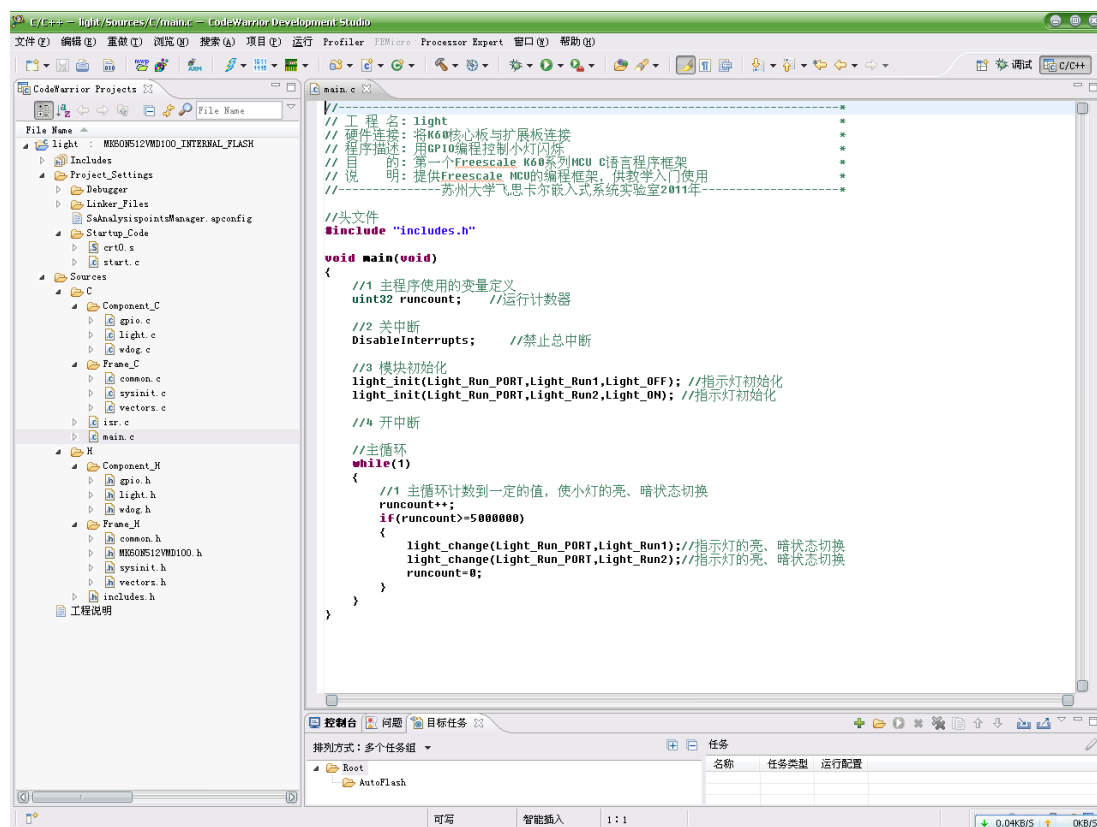


图 3-4 CW10.1 环境运行界面

需要说明的是，安装完毕以后要上网注册以申请使用许可（license key）。无论是下载的软件还是申请到的免费光盘，安装后都要通过因特网注册，以申请使用许可（licenseKey）。这里可通过登录其网站，单击“Request a Key”实现。由于这一注册过程是在网上自动实现的，故只要网络通畅，这个往返过程在数分钟之内即可可完成。申请后会通过 E-MAIL 得到一个 License.dat 文件。将该文件复制到相应目录下即可，例如：“C:\Program Files\Freescale\CW MCU v10.1\”。对于免费的特别版本，安装好后用 License.dat 覆盖安装目录下的 License.dat。CW 环境的运行界面如图 3-4 所示。

3.3.3 JTAG 写入器

开发人员可以通过 JTAG 写入器对目标板中的 Flash 进行擦除、写入等操作。将机器码下载到 Flash 后，可以进行程序的运行、调试。图 3-5 给出了写入器的实物图。使用该写入器时，一端连接 PC 的 USB 口，一端连接目标板的 BDM 口。详细的使用说明请见光盘。

3.3.4 MK60N512VMD100 硬件核心板

MK60N512VMD100 硬件核心板如图 3-6 所示，该硬件核心板使用 144 引脚的 MAPBGA

封装，为 4 层电路板。通过扩展板为其供电，扩展板上实现了 K 系列芯片的大部分模块，其中包括 LED，CAN，UART，SD 卡，USB OTG，以太网和 LCD 等模块。扩展板使用两层电路板，提供 12V 电源输入插头，板上使用 LM2576 芯片将 12V 电压转换至 3.3V 电压。LM2576 产生的 3.3V 电源提供 MK60N512VMD100 核心板和扩展板上的所有元件，LM2576 可以提供 3A 的输出电流，足够核心板和外设使用。扩展板实物图见图 3-7 所示。



图 3-5 写入器的实物图

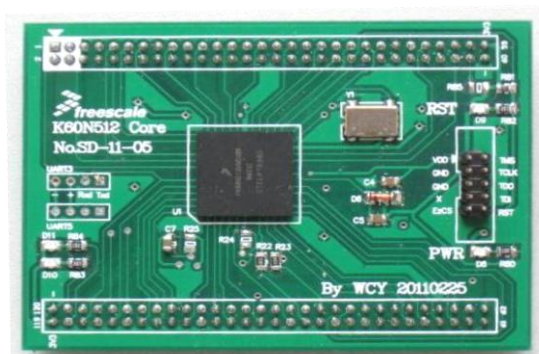


图 3-6 MK60N512VMD100 硬件核心板

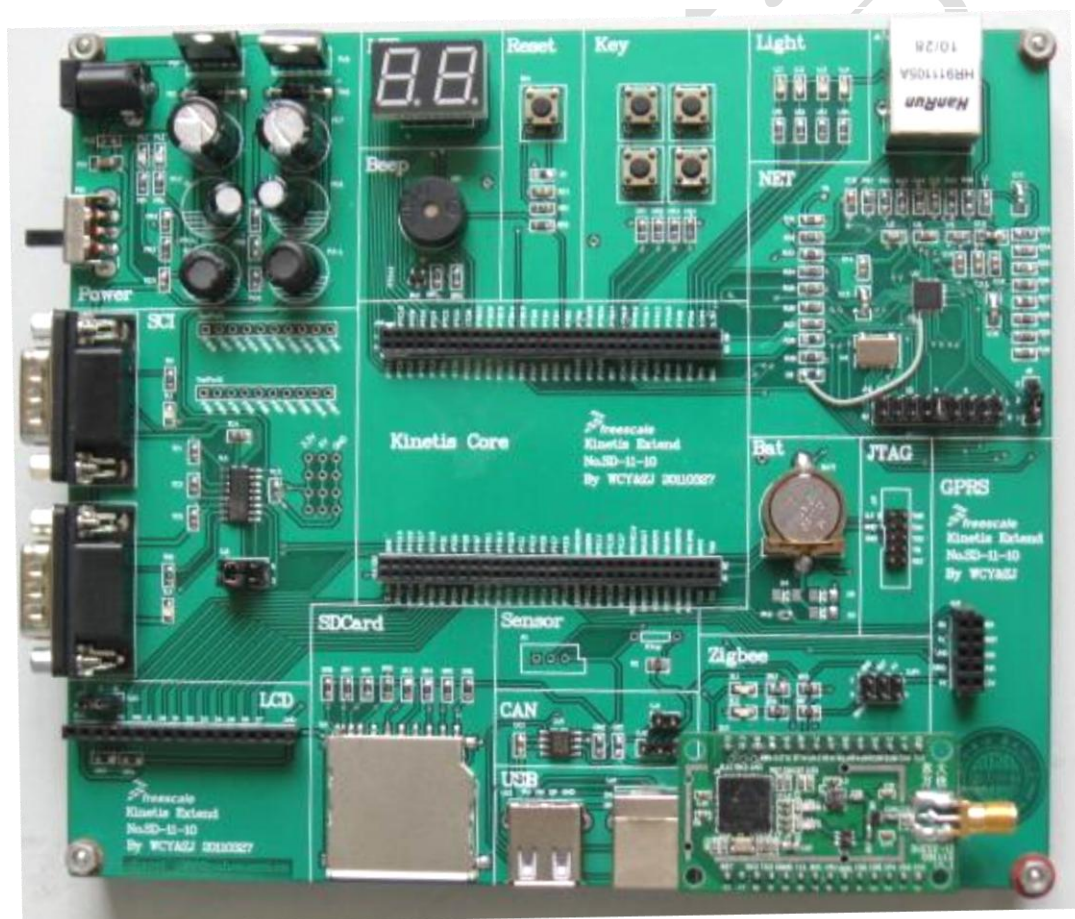


图 3-7 K 系列扩展板

3.4 IAR工程文件组织

嵌入式系统工程往往包含很多文件，如：程序文件、头文件、与编译调试相关的信息文

件、工程说明文件以及工程目标代码文件等。工程文件的合理组织对一个嵌入式系统工程尤为重要，它不但会提高项目的开发效率，同时也降低项目的维护难度。

嵌入式系统工程的文件组织方法以硬件对象为核心来展开，系统中每个对象应包含相关的头文件、程序文件及说明文件等。以硬件对象的方式来组织文件，会使得工程结构清晰，调试定位方便，后期维护容易，这也是嵌入式系统软件工程的基本思想。

3.4.1 工程文件的组织

图 3-8 给出了用 I/O 口控制小灯闪烁工程的树形结构模板，该模板是苏州大学飞思卡尔嵌入式系统实验室专门为 MK60N512VMD100 开发板设计的工程模板。读者在新建工程时可以选择该模板。该模板方便易懂，与 IAR 提供的 DEMO 的工程模板相比文件少，去掉了一些初学者不易理解且不是必须的文件，同时应用底层软件构件的概念改进了程序结构，目的是一开始就引导读者进行规范的文件组织与编程。

新建工程有两种方法，一种是使用工程模板，另一种是使用已存在的工程来建立另外一个工程。

第一种方法的操作步骤如下：

选择 file->WorkSpace，建立工作区，然后选择 Project->Create New Project，弹出 Create New Project 对话框，选择“ARM”，选择编程语言，选中 main，然后点确定，输入文件名，点保存，这样就创建了一个工程。创建工程后需要按照具体的要求来进行配置，这点详见“创建新工程的步骤.docx”。

第二种方法是使用已存的工程来建立另一个工程。当在已有工程的基础上，做另一个项目时，比如在 light 工程的基础上编写 LCD 程序，需要进行如下设置：

- (1) 更改工程文件夹名为 LCD
- (2) 更改 light.dep 为 LCD.dep
- (3) 更改 light.ewd 为 LCD.ewd
- (4) 更改 light.ewp 为 LCD.Ewp
- (5) 更改 light.eww 为 LCD.eww
- (6) 用记事本方式打开 LCD.Eww，代码如下

```
<?xml version="1.0" encoding="iso-8859-1"?>
```

```
<workspace>
  <project>
    <path>$WS_DIR$\light..ewp</path>
  </project>
  <batchBuild/>
</workspace>
```

将其中的 light.ewp 改为 LCD..ewp

(7) 打开该工程，你会看到机器码文件还是 light.map 与 light.out，这时进行编译，编译结束后，你会看到 light.map 与 light.out 变为 LCD.map 与 LCD.out。

新建工程时，我们建议采用第二种方式，这种方式比较简单，无需配置，不容易出错。

下面以控制小灯闪烁工程为例，介绍基于 IAR 环境的嵌入式工程文件组织方法。图 3-7 给出了该工程相关源文件的树型结构，可分为“工程配置文件”、“源程序文件”、“机器码文件”三个部分。

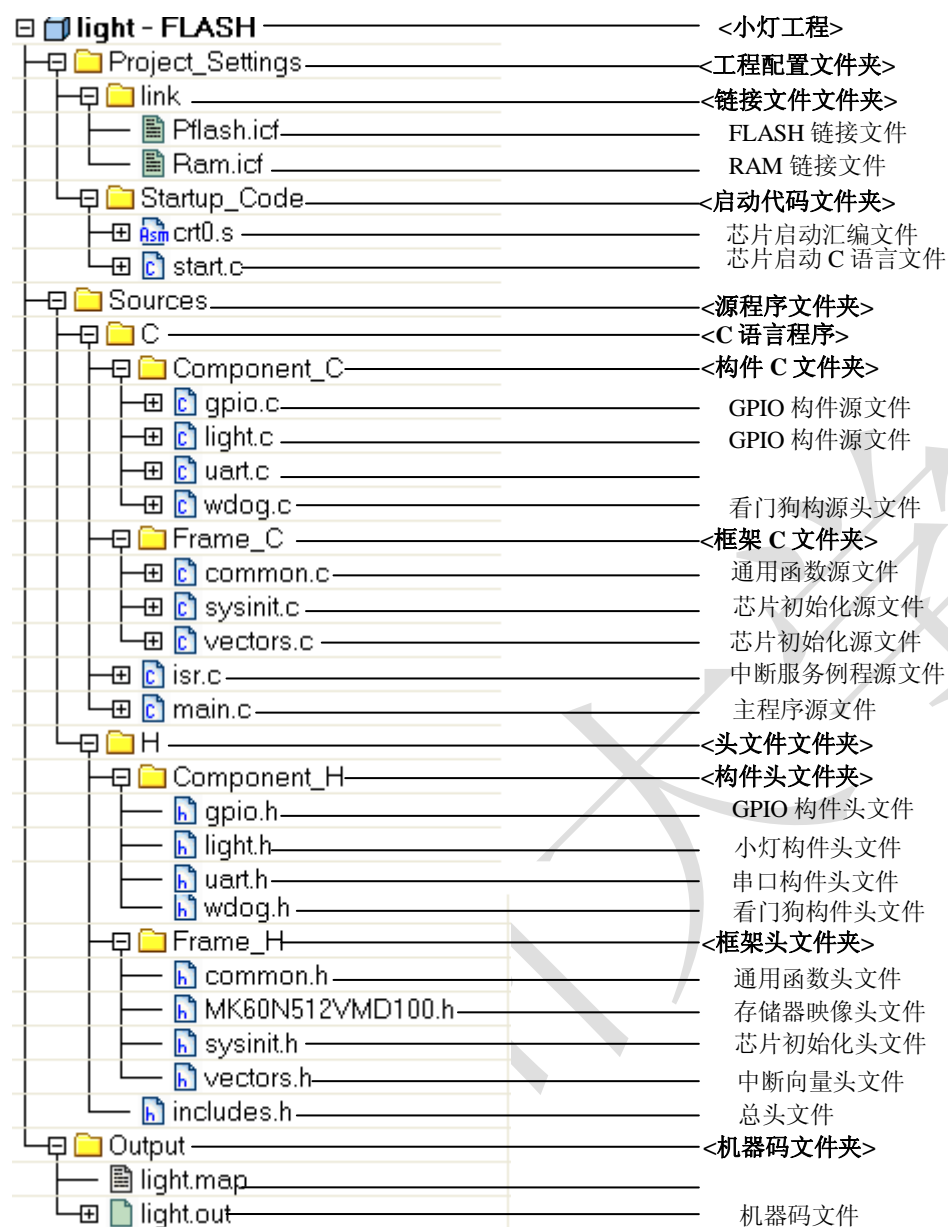


图 3-8 工程相关源文件的树型结构

“工程配置文件”中包含的文件与芯片及工程初始化相关，包括链接文件与启动代码文件。启动代码文件有“crt0.s”与“start.c”，而且前面都有一个“+”，展开后会看到好多文件，这些都是与其相关联的文件。“链接文件”定义了芯片存储器的分配和可执行代码地址空间的分配，包括 Pflash.icf 与 Ram.icf 两个文件，通过修改它们可以将可执行代码链接到芯片 RAM 中或 Flash 中。

“源程序文件夹”包括 C 语言程序、头文件。其中 C 语言程序包括构件 C 文件夹“Component_C”、“框架 C 文件夹”Frame_C、“中断服务例程源文件 isr 与主程序源文件 main，头文件包括构件头文件”Component_H“、“框架头文件夹”Frame_H“与总头文件 includes.h。系统启动并初始化后，程序根据 main.c 中定义的主循环顺序执行，当遇到中断请求时，转而执行 isr.c 中定义的相应中断处理程序；中断处理结束，则返回中断处继续顺序执行。由于 main.c 和 isr.c 文件反映了软件系统的整体执行流程，故而在工程文件组织时，将它与其余 C 语言程序文件分开管理。” Component_C “与” Component_H “包含构件代码，每个构件都对应一个.c 文件与.h 文件，例如 GPIO.c 与 GPIO.h 文件。以后的章节还会出现“串行

通信”、“键盘”、“LED”、“液晶”等构件。与总体框架程序相关的头文件和源文件分别放在了 Frame_H 和 Frame_C 文件夹中，以归类管理。Frame_H 里包含了 common.h、MK60N512VMD100.h、sysinit.h 与 vectors.h 四个头文件。MK60N512VMD100.h 是芯片寄存器及相关位定义头文件，它被视为芯片的接口文件，没有这个文件，就不可能对该芯片进行任何操作。sysinit.h 与 Frame_C 文件夹中的 sysinit.c 对应，它定义了系统初始化时的基本参数，如系统时钟等，而 sysinit.c 文件则包含实际初始化代码。common.h 与 common.c 对应，它提供常用且基本的软件功能性子函数。

“机器码文件”包括.out 文件与.map 文件，写到 Flash 中的文件为.out 文件，该文件在 IAR 下打不开，不过我们可以打开.srec 文件，它相当于 CodeWarrior 下的.s19 文件。

3.4.2 初始化相关文件

1. 启动文件 crt0.s 与 start.c

由于芯片启动代码直接面对内核和硬件控制器进行编程，一般都是用汇编语言实现。在芯片上电复位后，初始化 CPU 各寄存器，关闭中断等，需要用 ARM 的汇编语言编写启动代码（crt0.s 文件），然后跳转到用户 C 程序（start.c），在这里复制中断向量与代码到 RAM 中，初始化芯片时钟，打开中断，然后跳转到 main 函数继续执行。在 ARM 设计开发中，启动代码的编写是一个极重要的过程。启动代码随具体的目标系统和开发系统有所区别，MK60N512VMD100 芯片的启动流程见图 3-9 所示。

1) 芯片上电

MK60N512VMD100 允许将中断向量放置在 Flash 或者 RAM 中，但是上电时刻中断向量只能在地址 0x0000_0000 处。上电后，K60 首先从地址 0x0000_0000 处取栈地址，从地址 0x0000_0004 处取复位向量地址，然后跳转至复位向量地址处执行，代码在 vectors.h 处，如下：

```
#define VECTOR_000      (pointer*)__BOOT_STACK_ADDRESS    // 初始化 SP
#define VECTOR_001      __startup // 0x0000_0004 1 -        初始化 PC
```

2) crt0.s 文件

芯片上电后，转至 crt0.s 文件执行代码如下：

```
_startup
MOV    r0,#0          ; 初始化寄存器
MOV    r1,#0
MOV    r2,#0
MOV    r3,#0
MOV    r4,#0
MOV    r5,#0
MOV    r6,#0
MOV    r7,#0
MOV    r8,#0
MOV    r9,#0
MOV    r10,#0
MOV    r11,#0
MOV    r12,#0
CPSIE  i              ; 屏蔽中断
import start
BL      start          ; 调用 start
```


复位向量中首先清零所有 CPU 通用寄存器，关闭总中断，然后调用 start，依次关闭看门狗、复制中断向量表到 RAM 中、初始化芯片时钟、转到 main 函数继续执行。

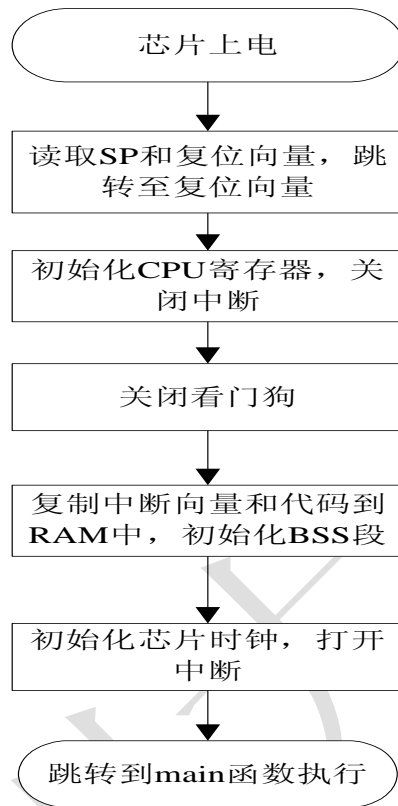


图 3-9 MK60N512VMD100 芯片启动流程

3) start.c 文件

该文件的部分代码如下

```
void start(void)
{
    //关闭看门狗
    wdog_disable();
    //复制中断向量表到 RAM 中
    common_startup();
    //系统设置
    sysinit();
    //进入主函数
    main();
}
```

下面详细讲述下这部分代码。

(1) 关闭看门狗

看门狗在嵌入式设计中特别重要，它可以在芯片代码跑飞或死机的情况下复位芯片。嵌入式产品往往 24 小时全天候运行，为了保证程序一直运行正常必须在产品正式发布时使能看门狗。而在程序调试阶段，为了保证程序执行流程，往往先关闭看门狗中断。MK60N512VMD100 的看门狗控制寄存器是只写一次寄存器，即上电后只能对其进行一次写

入，如果想进行多次写入必须首先解锁看门狗。看门狗解锁是向看门狗的解锁寄存器连续写入 0xC520 和 0xD928，两次写入不能超过 20 个时钟周期，否则解锁失败并产生看门狗中断。看门狗解锁后，通过配置看门狗控制寄存器的 WDOGEN 位来关闭看门狗。

如果程序在完成启动之后要使用看门狗，首先要解锁看门狗然后设置 WDOGEN 位来使能看门狗。看门狗使能以后，会在溢出时间超时后产生看门狗复位。程序中必须在溢出超时前“喂狗”。“喂狗”后看门狗模块重新计时，MK60N512VMD100 的“喂狗”是向看门狗刷新寄存器连续写入 0xB480 和 0xA602。两次写入间隔不能超过 20 个时钟周期。源代码请见样例程序中的 wdog.c 文件。

(2) 复制中断向量表到 RAM 中

代码在 RAM 中执行的效率比在 Flash 中执行高。通常 ARM 芯片都会将中断向量表复制到 RAM 中。未初始化的数据段（BSS）应该清零。其中涉及到几个 Link 文件中定义的变量，包括：Flash 中断向量表地址 VECTOR_ROM，RAM 中断向量表地址 VECTOR_RAM。这些变量定义在 Link 文件两个 .icf 中。源代码请见样例程序中的 startup.c 文件、Pflash.icf 文件与 Ram.icf 文件。

(3) 系统设置

芯片主时钟是利用 MCG 模块中的 PLL 模块，通过倍频板上 50MHZ 的有源晶振得到的。在 Kinetis 芯片内部存在 3 种不同时钟：内核时钟，总线时钟和 Flash 时钟。内核时钟是基本时钟，其他时钟均从内核时钟分频得到。Kinetis 手册推荐了 3 种时钟配置方式，分别将内核时钟配置为 50MHZ，96MHZ 和 100MHZ。源代码请见样例程序中的 sysinit.c 文件。

d) 跳转至 main 函数执行

上述三步执行结束后，芯片已经进入了正常运行状态，可以执行 main 函数中用户定义的代码了。代码中直接调用 main 函数即可。

到此为止，芯片开始执行用户功能代码，芯片启动阶段结束。

2. 芯片相关文件

1) 映像寄存器定义文件 MK60N512VMD100.h

MK60N512VMD100.h 中定义了编程时需要访问的外设寄存器，该文件不修改。

2) 系统初始化文件 sysinit.c 与 sysinit.h

系统初始化操作是由 sysinit.c 与 sysinit.h 来实现的，具体内容参见样例程序。

3. 主程序、中断程序及其它文件

1) 总头文件 includes.h 与主程序文件 main.c

includes.h 文件包含主函数（main）文件用到的头文件、外部函数或变量引用、有关常量和全局变量定义以及内部函数声明。而 main.c 文件是工程任务的核心文件，里面包含了一个主循环，对具体事务过程的操作几乎都是添加在该主循环中。

2) 中断文件 isr.h 与 isr.c

isr.h 文件包含 isr.c 文件用到的头文件，外部函数以及内部函数声明等，isr.c 文件执行具体的操作，具体内容可参见带有中断的样例程序。

4. 链接文件 Pflash.icf 文件与 Ram.icf 文件

链接文件定义了芯片存储器的分配和可执行代码地址空间的分配。可以选择将可执行代码链接到芯片 RAM 中或 Flash 中。具体内容可参见样例程序。

5. 机器码文件

在编译链接过程中，IAR 会产生机器码文件 .out 文件，这是写入到 Flash 中的文件，在 IAR 环境中打不开。如果在项目配置时选择在编译链接过程产生 .srec 文件，那么在控制小灯闪烁的工程中我们可以得到 GPIO.srec 文件，这个文件可以打开，内容以 S 记录格式表示。S 记录格式是 Freescale 公司的十六进制目标代码文件，它将目标程序和数据以 ASCII 码格

式表示，可直接显示和打印。目标文件由若干行 S 记录构成，每行 S 记录可以用 CR/LF/NUL 结尾。一行 S 记录由下列五部分组成，分别说明如下。

(1) 类型

表示 S 记录的类型。有 8 种记录类型 S0、S1、S2、S3、S5、S7、S8、S9。这是为了满足不同的编码、解码及传送方式的需求，表 3-1 给出了 S 记录的格式。

表 3-1 S 记录格式

类型	记录长度	地址	编码/数据	校验和
2 字节	2 字节	2、3 或 4 字节	0~n 字节	1 字节

S0—该记录包含 S19 文件的文件名信息。

S1—该记录包含代码/数据以及两个字节存储其代码/数据的存储器首地址。

S2—该记录包含要写到 Flash 的扩展地址处的代码/数据以及三个字节存储其代码/数据的存储器首地址。

S3—该记录包含要写到 Flash 的扩展地址处的代码/数据以及四个字节存储其代码/数据的存储器首地址。

S7—S3 记录的结束记录。

S8—S2 记录的结束记录。

S9—S1 记录的结束记录。

每个 S 记录块都使用唯一的终止记录。

(2) 记录长度

表示该记录行中字符对的数目，不包括类型和记录长度。

(3) 地址

它可以是 2 个字节、3 个字节或 4 个字节，取决于记录类型。S1 记录、S9 记录均是 2 个字节，S2 记录、S8 记录是 3 个字节，S3 记录、S7 记录是 4 个字节。它表示其后的代码/数据部分将要装入的存储器起始地址。

(4) 代码/数据

就是实际的目标代码或数据，这一部分将被下载到目标芯片的存储器并运行。其字节数是由“记录长度”域的实际数值减去地址长度和校验码长度的值而得到的。

(5) 校验和

为 1 个字节，它是“记录长度”、“地址”、“代码/数据”三个部分所有字节之和的反码的低 8 位，用于校验。

下面是 01_GPIO 工程中的 GPIO.srec 的部分内容。

```
S00C00006770696F2E7372656369...
```

```
S3151FFF0000F8FF00201104FF1FD127FF1FD127FF1F56
```

```
...
```

```
S7051FFF291F94
```

第一行为 S0 记录，表示文件名信息。S0 之后的 0C 是十六进制数（十进制数 12），表示后面有 12 个字节的数据；随后的“0000”是 2 字节地址，“0000”表示本行信息不是程序/数据，不需要装入存储空间；最后的 69 是本记录的校验和。其中 S3151FFF0000F8FF00201104FF1FD127FF1FD127FF1F56 的前两个符号 S3 表示这一行是 S3 记录，其后的“15”是十六进制数（十进制数的 21），表示在此行其后有 21 个字节的数据，包括 4 个字节的地址 1FFF0000、16 个字节的代码/数据，最后字节 56 为校验和。该行记录所表示的实际代码/数据 F8FF00201104FF1FD127FF1FD127FF1F 将被装入起始地址为 1FFF0000 的 MCU 存储器中。

最后一行是 S7 记录，S7 之后的 05 是十六进制 0x05，表示其后有 5 个字节的/数据。1FFF291F 为 4 个字节的地址，94 是校验和。

3.5 CW工程文件组织

CW 工程文件组织与 IAR 工程文件组织基本一样，但略有不同，下面简单阐述一下。

3.5.1 工程文件的组织

图 3-10 给出了用 I/O 口控制小灯闪烁工程的树形结构模板，该模板是苏州大学飞思卡尔嵌入式系统实验室专门为 MK60N512VMD100 开发板设计的工程模板。读者在编程时可以选择该模板。该模板方便易懂，与 CW 提供的 DEMO 工程模板相比，去掉了一些初学者不易理解且不是必须的文件，同时应用底层软件构件的概念改进了程序结构，目的是一开始就引导读者进行规范的文件组织与编程。

新建工程有两种方法，一种是使用工程模板，另一种是使用已存在的工程来建立另外一个工程。

第一种方法的操作步骤如下：

选择 file->new->Bareboard Project，弹出 New Bareboard Project 对话框，然后根据芯片型号来选择配置就创建了一个工程，详见“创建新工程的步骤.docx”。

第二种方法是使用已存的工程来建立另一个工程。当在已有工程的基础上，做另一个项目时，只需更改工程名即可。

新建工程时，我们建议采用第二种方式，这种方式比较简单，无需配置，不容易出错。

下面以控制小灯闪烁工程为例，介绍基于 CW 环境的嵌入式工程文件组织方法。图 3-9 给出了该工程相关源文件的树型结构，可分为“头文件路径”、“工程配置文件”、“输出文件”、“应用程序文件”4 大部分。

“头文件路径”为工程配置后自动生成的文件，不用改动。

“输出文件”，包含 .afx 和 S19 等格式的目标机器码。

“工程配置文件”包含与调试相关的配置文件，链接文件，启动代码文件。

“应用程序文件”包含通用函数，构件文件，主程序文件，中断服务例程文件等。

3.5.2 初始化相关文件

这部分与 IAR 环境基本一致，只是机器码文件稍有不同而已。在编译链接过程中，CW 会产生机器码文件 .afx 文件，这是写入到 Flash 中的文件，在 CW 环境中打不开。如果在项目配置时选择在编译链接过程产生 S19 文件，那么在控制小灯闪烁的工程中我们可以得到 Light.afx.S19 文件，这个文件可以打开，内容以 S 记录格式表示。



图 3-10 工程相关源文件的树型结构

3.6 第一个应用实例：控制小灯闪烁

本书用 MK60N512VMD100 控制发光二极管指示灯的例子开始我们的程序之旅，程序中使用到了 GPIO 构件来编写指示灯程序。指示灯是最简单不过的硬件对象了，当灯两端引脚上有足够高的正向压降时，它就会发光。在本书的工程实例中，灯的正端引脚接 MK60N512VMD100 的普通 I/O 口，负端引脚过电阻接地。当在 I/O 引脚上输出高或低电平时，指示灯就会亮或暗。

3.6.1 GPIO 构件

GPIO 引脚可以被定义成输入、输出两种情况。若是输入，程序需要获得引脚的状态（逻辑 1 或 0）。若是输出，程序可以设置引脚状态（逻辑 1 或 0）。MCU 的 GPIO 引脚分为许多端口（Port），每个口有若干引脚。为了实现对所有 GPIO 引脚统一编程，设计了 GPIO 构件（由 GPIO.h、GPIO.c 两个文件组成）。这样，要使用 GPIO 构件，只需要将这两个文件加入到所建工程中，方便了对 GPIO 的编程操作。实际上，若只是使用构件，只需看头文件中的相关函数说明。

1. GPIO 构件的头文件 gpio.h

```
//-----*
// 文件名: gpio.h                                *
// 说 明: gpio驱动头文件                        *
//-----*

#ifndef __GPIO_H
#define __GPIO_H

//1 头文件
#include "common.h"

//2 宏定义
//2.1 端口宏定义
#define PORTA PTA_BASE_PTR
#define PORTB PTB_BASE_PTR
#define PORTC PTC_BASE_PTR
#define PORTD PTD_BASE_PTR
#define PORTE PTE_BASE_PTR

//3 函数声明
//-----*
//函数名: gpio_init                             *
//功 能: 初始化gpio                             *
//参 数: port:端口名                             *
//      index:指定端口引脚                       *
//      dir:引脚方向, 0=输入, 1=输出             *
//      data:初始状态, 0=低电平, 1=高电平        *
//返 回: 无                                     *
//说 明: 无                                     *
//-----*
void gpio_init (GPIO_MemMapPtr port, int index, int dir, int data);

//-----*
//函数名: gpio_ctrl                             *
//功 能: 设置引脚状态                             *
//参 数: port:端口名                             *
//      index:指定端口引脚                       *
//      data: 状态, 0=低电平, 1=高电平          *
//返 回: 无                                     *
//说 明: 无                                     *
//-----*
void gpio_ctrl (GPIO_MemMapPtr port, int index, int data);
```

```

//-----*
//函数名: gpio_reverse*
//功 能: 改变引脚状态*
//参 数: port:端口名;*
//      index:指定端口引脚*
//返 回: 无*
//说 明: 无*
//-----*
void gpio_reverse (GPIO_MemMapPtr port, int index);

```

```
#endif
```

2. GPIO 构件的程序文件 gpio.c

```

//-----*
// 文件名: gpio.c*
// 说 明: gpio驱动程序文件*
//-----*

#include "gpio.h" //包含gpio头文件

//-----*
//函数名: gpio_init*
//功 能: 初始化gpio*
//参 数: port:端口名*
//      index:指定端口引脚*
//      dir:引脚方向, 0=输入, 1=输出*
//      data:初始状态, 0=低电平, 1=高电平*
//返 回: 无*
//说 明: 无*
//-----*
void gpio_init (GPIO_MemMapPtr port, int index, int dir, int data)
{
    PORT_MemMapPtr p;
    switch((uint32)port)
    {
        case 0x400FF000u:
            p = PORTA_BASE_PTR;
            break;
        case 0x400FF040u:
            p = PORTB_BASE_PTR;
            break;
        case 0x400FF080u:
            p = PORTC_BASE_PTR;
            break;
        case 0x400FF0C0u:
            p = PORTD_BASE_PTR;
            break;
        case 0x400FF100u:
            p = PORTE_BASE_PTR;
            break;
        default:
            break;
    }
    PORT_PCR_REG(p, index)=(0|PORT_PCR_MUX(1));
}

```

```

        if(dir == 1)//output
        {
            GPIO_PDDR_REG(port) |= (1<<index);
            if(data == 1)//output
            GPIO_PDOR_REG(port) |= (1<<index);
        }
        else
        GPIO_PDOR_REG(port) &= ~(1<<index);
    }

    else
        GPIO_PDDR_REG(port) &= ~(1<<index); }

//-----*
//函数名: gpio_ctrl
//功 能: 设置引脚状态
//参 数: port:端口名
//      index:指定端口引脚
//      data: 状态,0=低电平,1=高电平
//返 回: 无
//说 明: 无
//-----*
void gpio_ctrl (GPIO_MemMapPtr port, int index, int data)
{
    if(data == 1)//output
        GPIO_PDOR_REG(port) |= (1<<index);
    else
        GPIO_PDOR_REG(port) &= ~(1<<index);
}

//-----*
//函数名: gpio_reverse
//功 能: 改变引脚状态
//参 数: port:端口名;
//      index:指定端口引脚
//返 回: 无
//说 明: 无
//-----*
void gpio_reverse (GPIO_MemMapPtr port, int index)
{
    GPIO_PDOR_REG(port) ^= (1<<index);
}

```

3.6.2 Light 构件

控制指示灯的亮或暗，通过调用 GPIO 构件完成。设有两盏灯，分别为运行指示灯 1、运行指示灯 2，分别叫做 Light_Run1、Light_Run2。它们所接在的 MCU 的 GPIO 口的名字叫做 Light_Run_PORT。它们具体接在 MCU 的哪个端口，哪个引脚，只要在 light.h 中给出具体宏定义就可以了。

1.Light 构件的头文件 light.h

```

//-----*
// 文件名: light.h
// 说 明: 指示灯驱动程序头文件
//-----*

```

```

#ifndef LIGHT_H
#define LIGHT_H

//1 头文件
#include "common.h"
#include "gpio.h"

//2 宏定义
//2.1 灯控制引脚定义
#define Light_Run_PORT PORTC //运行指示灯使用的端口
#define Light_Run1 13 //运行指示灯使用的引脚
#define Light_Run2 14 //运行指示灯使用的引脚

//2.2 灯状态宏定义
#define Light_ON 0 //灯亮(对应低电平)
#define Light_OFF 1 //灯暗(对应高电平)

//3 函 数 声 明
//-----*
//函数名: light_init *
//功 能: 初始化指示灯状态 *
//参 数: port:端口名 *
//      name:指定端口引脚号 *
//      state:初始状态,1=高电平,0=低电平 *
//返 回: 无 *
//说 明: 调用GPIO_Init函数 *
//-----*
void light_init(GPIO_MemMapPtr port, int name, int state);
//-----*
//函数名: Light_control *
//功 能: 控制灯的亮和暗 *
//参 数: port:端口名 *
//      name:指定端口引脚号 *
//      state:状态,1=高电平,0=低电平 *
//返 回: 无 *
//说 明: 调用GPIO_Set函数 *
//-----*
void light_control(GPIO_MemMapPtr port, int name, int state);
//-----*
//函数名: Light_change *
//功 能: 状态切换:原来"暗",则变"亮";原来"亮",则变"暗" *
//参 数: port:端口名 *
//      name:指定端口引脚号 *
//返 回: 无 *
//说 明 : 调 用 GPIO_Get 、 GPIO_Set 函 数 *
//-----*
void light_change(GPIO_MemMapPtr port, int name);
#endif

```

2.Light 构件的程序文件 light.c

```

//-----*
// 文件名: light.c *
// 说 明: 小灯驱动函数文件 *
//-----*

```

```
#include "light.h" //指示灯驱动程序头文件
```

```
//-----*
//函数名: light_init *
//功 能: 初始化指示灯状态 *
//参 数: port:端口名 *
//      name:指定端口引脚号 *
//      state:初始状态,1=高电平,0=低电平 *
//返 回: 无 *
//说 明: 调用gpio_init函数 *
//-----*
void light_init(GPIO_MemMapPtr port,int name,int state)
{
    gpio_init(port,name,1,state); //初始化指示灯
}

//-----*
//函数名: light_control *
//功 能: 控制灯的亮和暗 *
//参 数: port:端口名 *
//      name:指定端口引脚号 *
//      state:状态,1=高电平,0=低电平 *
//返 回: 无 *
//说 明: 调用gpio_ctrl函数 *
//-----*
void light_control(GPIO_MemMapPtr port,int name,int state)
{
    gpio_ctrl(port,name,state); //控制引脚状态
}

//-----*
//函数名: light_change *
//功 能: 状态切换:原来"暗",则变"亮";原来"亮",则变"暗" *
//参 数: port:端口名 *
//      name:指定端口引脚号 *
//返 回: 无 *
//说 明: 调用gpio_reverse函数 *
//-----*
void light_change(GPIO_MemMapPtr port,int name)
{
    gpio_reverse(port,name);
}
```

3.6.3 Light 测试工程主程序

在 includes.h 文件中需要包含 GPIO.h, 这样在该工程中就可以调用 GPIO 构件的接口函数。首先调用 gpio_init 函数, 初始化所需的每一盏指示灯。随后, 通过 gpio_reverse 函数将引脚电平取反, 就能够在程序运行时, 较明显的看到指示灯闪烁的现象。代码如下:

```
//-----*
//工 程 名: light *
//硬件连接: 将K60核心板与扩展板连接 *
//程序描述: 用GPIO编程控制小灯闪烁 *
//目 的: 第一个Freescale K60系列MCU C语言程序框架 *
//说 明: 提供Freescale MCU的编程框架, 供教学入门使用 *
//-----苏州大学飞思卡尔嵌入式系统实验室2011年-----*
```

```

//头文件
#include "includes.h"

//全局变量声明

//主函数
void main(void)
{
    //1 主程序使用的变量定义
    uint32 runcount; //运行计数器

    //2 关中断
    DisableInterrupts; //禁止总中断

    //3 模块初始化
    light_init(Light_Run_PORT, Light_Run1, Light_OFF); //指示灯初始化
    light_init(Light_Run_PORT, Light_Run2, Light_ON); //指示灯初始化

    //4 开中断

    //主循环
    while(1)
    {
        //1 主循环计数到一定的值，使小灯的亮、暗状态切换
        runcount++;
        if(runcount>=5000000)
        {
            light_change(Light_Run_PORT, Light_Run1); //指示灯的亮、暗状态切换
            light_change(Light_Run_PORT, Light_Run2); //指示灯的亮、暗状态切换
            runcount=0;
        }
    }
}

```

3.7 理解第一个C工程的执行过程

当 MK60N512VMD100 芯片上电复位后或热复位后，系统程序的执行流程如下：从复位向量处取出程序执行的首地址，跳转并按地址执行；执行 crt0.s 文件中的 `__startup`，复位所有的通用寄存器，关闭总中断，然后调用 start.c 文件，进行系统初始化，最终跳转到 main 主函数入口继续执行；这部分在“3.4.2 初始化相关文件”的第一部分“1.启动文件 crt0.s 与 start.c”已经讲述，这里不再赘述。

第4章 异步串行通信

目前几乎所有的台式电脑都带有 9 芯的异步串行通信口，简称串行口或 COM 口。由于历史的原因，通常所说的串行通信就是指异步串行通信。USB、以太网等也用串行方式通信，但与这里所说的异步串行通信物理机制不同。

有的台式电脑带有两个串行口：COM1、COM2 口，部分笔记本电脑也带有串行口。随着 USB 接口的普及，串行口的地位逐渐降低，但是作为设备间简便的通信方式，在相当长的时间内，串行口还不会消失，在市场上也可很容易的购买到 USB 到串行口的转接器。因为简单且常用的串行通信只需要三根线（发送线、接收线和地线），所以串行通信仍然是 MCU 与外界通信的简便方式之一。

实现异步串行通信功能的模块在一部分 MCU 中被称为通用异步收发器（Universal Asynchronous Receiver/Transmitters, UART），在另一些 MCU 中被称为串行通信接口（Serial Communication Interface, SCI）。串行通信接口可以将终端或个人计算机连接到 MCU，也可将几个分散的 MCU 连接成通信网络。

本章主要介绍 MK60N512VMD100 的 UART 模块的工作原理以及编程实例，这些编程实例都使用了基于构件的编程思想，读者在阅读时可以仔细体会，以求得对编程方法更深刻的理解。下文所出现的 UART 字眼，在没有其他说明的情况下，都是特指 MK60N512VMD100 的 UART 模块。

4.1 异步串行通信的基础知识

本节简要概括了串行通信中常用的基本概念，为学习 MCU 的串行接口编程做准备。对于已经了解这方面知识的读者，可以略读本节。

4.1.1 基本概念

“位”（bit）是单个二进制数字的简称，是可以拥有两种状态的最小二进制值，分别用“0”和“1”表示。在计算机中，通常一个信息单位用 8 位二进制表示，称为一个“字节”（Byte）。串行通信的特点是：数据以字节为单位，按位的顺序（例如最高位优先）从一条传输线上发送出去。这里至少涉及到以下几个问题：第一，每个字节之间是如何区分开的？第二，发送一位的持续时间是多少？第三，怎样知道传输是正确的？第四，可以传输多远？这些问题属于串行通信的基本概念。串行通信分为异步通信与同步通信两种方式，本节主要给出异步串行通信的一些常用概念。正确理解这些概念，对串行通信编程是有益的。

1. 异步串行通信的格式

在 MCU 的英文芯片手册上，通常说的异步串行通信采用的是 NRZ 数据格式，英文全称是：“standard non-return-zero mark/space data format”，可以译为：“标准不归零传号/空号数据格式”。这是一个通信术语，“不归零”的最初含义是：用负电平表示一种二进制值，正电平表示另一种二进制值，不使用零电平。“mark/space”即“传号/空号”分别是表示两种状态的物理名称，逻辑名称记为“1/0”。对学习嵌入式应用的读者而言，只要理解这种格式只有“1”、“0”两种逻辑值就可以了。图 4-1 给出了 8 位数据、无校验情况的传送格式。



图 4-1 串行通信数据格式

这种格式的空闲状态为“1”，发送器通过发送一个“0”表示一个字节传输的开始，随后是数据位（在 MCU 中一般是 8 位或 9 位，可以包含校验位）。最后，发送器发送 1 到 2 位的停止位，表示一个字节传送结束。若继续发送下一字节，则重新发送开始位，开始一个新的字节传送。若不发送新的字节，则维持“1”的状态，使发送数据线处于空闲。从开始位到停止位结束的时间间隔称为一帧（frame）。所以，也称这种格式为帧格式。

每发送一个字节，都要发送“开始位”与“停止位”，这是影响异步串行通信传送速度的因素之一。同时因为每发送一个字节，必须首先发送“开始位”，所以称之为“异步”（Asynchronous）通信。

2. 串行通信的波特率

位长（Bit Length），也称为位的持续时间（Bit Duration）。其倒数就是单位时间内传送的位数。人们把每秒内传送的位数叫做波特率（Baud Rate）。波特率的单位是：位/秒，记为 bps。bps 是英文 bit per second 的缩写，习惯上这个缩写不用大写，而用小写。通常情况下，波特率的单位可以省略。

通常使用的波特率有 600、900、1200、1800、2400、4800、9600、19200、38400、57600、115200 等。在包含开始位与停止位的情况下，发送一个字节需 10 位，很容易计算出，在各波特率下，发送 1K 字节所需的时间。显然，这个速度相对于目前许多通信方式而言是慢的，那么，异步串行通信的速度能否提得很高呢？答案是不能。因为随着波特率的提高，位长变小，以致于很容易受到电磁源的干扰，通信就不可靠了。当然，还有通信距离问题，距离小，可以适当提高波特率，但这样毕竟提高的幅度非常有限，达不到大幅度提高的目的。

3. 奇偶校验

在异步串行通信中，如何知道传输是正确的？最常见的方法是增加一个位（奇偶校验位），供错误检测使用。字符奇偶校验检查（Character Parity Checking）称为垂直冗余检查（Vertical Redundancy Checking, VRC），它是为每个字符增加一个额外位使字符中“1”的个数为奇数或偶数。奇数或偶数依据使用的是“奇校验检查”还是“偶校验检查”而定。当使用“奇校验检查”时，如果字符数据位中“1”的数目是偶数，校验位应为“1”，如果“1”的数目是奇数，校验位应为“0”。当使用“偶校验检查”时，如果字符数据位中“1”的数目是偶数，则校验位应为“0”，如果是奇数则为“1”。

这里列举奇偶校验检查的一个实例，看看 ASCII 字符“R”，其位构成是 1010010。由于字符“R”中有 3 个位为“1”，若使用奇校验检查，则校验位为 0；如果使用偶校验检查，则校验位为 1。

在传输过程中，若有 1 位（或奇数个数据位）发生错误，使用奇偶校验检查，可以知道发生传输错误。若有 2 位（或偶数个数据位）发生错误，使用奇偶校验检查，就不能知道已经发生了传输错误。但是奇偶校验检查方法简单，使用方便，发生 1 位错误的概率远大于 2 位的概率，所以“奇偶校验”这种方法还是最为常用的校验方法。几乎所有 MCU 的串行异步通信接口，都提供这种功能。

4. 串行通信的传输方式

在串行通信中，经常用到“单工”、“双工”、“半双工”等术语。它们是串行通信的不同传输方式。下面简要介绍这些术语的基本含义。

(1) 单工 (Simplex): 数据传送是单向的，一端为发送端，另一端为接收端。这种传输方式中，除了地线之外，只要一根数据线就可以了。有线广播就是单工的。

(2) 全双工 (Full-duplex): 数据传送是双向的，且可以同时接收与发送数据。这种传输方式中，除了地线之外，需要两根数据线，站在任何一端的角度看，一根为发送线，另一根为接收线。一般情况下，MCU 的异步串行通信接口均是全双工的。

(3) 半双工 (Half-duplex): 数据传送也是双向的，但是在这种传输方式中，除地线之外，一般只有一根数据线。任何时刻，只能由一方发送数据，另一方接收数据，不能同时收发。

4.1.2 RS-232C 总线标准

现在回答“可以传输多远”这个问题。MCU 引脚输入/输出一般使用 TTL (Transistor Transistor Logic) 电平，即晶体管-晶体管逻辑电平。而 TTL 电平的“1”和“0”的特征电压分别为 2.4V 和 0.4V (目前使用 3V 供电的 MCU 中，该特征值有所变动)，即大于 2.4V 则识别为“1”，小于 0.4V 则识别为“0”。它适用于板内数据传输。若用 TTL 电平将数据传输到 5m 之外，那么可靠性就很值得考究了。为使信号传输得更远，美国电子工业协会 EIA (Electronic Industry Association) 制订了串行物理接口标准 RS-232C。RS-232C 采用负逻辑，-15V~-3V 为逻辑“1”，+3V~+15V 为逻辑“0”。RS-232C 最大的传输距离是 30m，通信速率一般低于 20Kbps。当然，在实际应用中，也有人用降低通信速率的方法，通过 RS-232 电平，将数据传送到 300m 之外，这是很少见的，且稳定性很不好。

RS-232C 总线标准最初是为远程数据通信制订的，但目前主要用于几米到几十米范围内的近距离通信。有专门的书籍介绍这个标准，但对于一般的读者，不需要掌握 RS-232C 标准的全部内容，只要了解本节介绍的这些基本知识就可以使用 RS-232。目前一般的 PC 机均带有 1 到 2 个串行通信接口，人们也称之为 RS-232 接口，简称“串口”，它主要用于连接具有同样接口的室内设备。早期的标准串行通信接口是 25 芯插头，这是 RS-232C 规定的标准连接器 (其中: 2 条地线，4 条数据线，11 条控制线，3 条定时信号，其余 5 条线备用或未定义)。

后来，人们发现在计算机的串行通信中，25 芯线中的大部分并不使用，逐渐改为使用 9 芯串行接口。一段时间内，市场上还有 25 芯与 9 芯的转接头，方便了两种不同类型之间的转换。后来，使用 25 芯串行插头极少见到，25 芯与 9 芯转接头也极少有售。因此，目前几乎所有计算机上的串行口都是 9 芯接口。图 4-2 给出了 9 芯串行接口的排列位置，相应引脚含义见表 4-1。

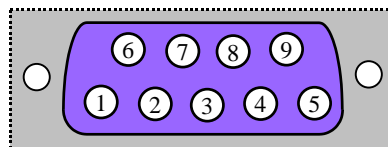


图 4-2 9 芯串行接口排列

在 RS-232 通信中，常常使用精简的 RS-232 通信，通信时仅使用 3 根线: RXD (接收线)、TXD (发送线) 和 GND (地线)。其他为进行远程传输时接调制解调器之用，有的也可作为硬件握手信号，初学时可以忽略这些信号的含义。

表 4-1 9 芯串行接口引脚含义表

引脚号	功 能	引脚号	功 能
1	接收线信号检测(载波检测DCD)	6	数据通信设备准备就绪(DSR)

2	接收数据线(RXD)	7	请求发送(RTS)
3	发送数据线(TXD)	8	允许发送(CTS)
4	数据终端准备就绪(DTR)	9	振铃指示
5	信号地(SG)		

4.1.3 电平转换电路原理

在 MCU 中,若用 RS-232C 总线进行串行通信,则需外接电路实现电平转换。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,需要用接收电路将 RS-232C 电平转换为 TTL 电平。电平转换器不仅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 芯片较多,该芯片使用单一+5V 电源供电实现电平转换。图 4-3 给出了 MAX232 的引脚说明。

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): $VS+=2V_{CC}-1.5V=8.5V$

VS- (6 脚): $VS-=-2V_{CC}-1.5V=-11.5V$

C2+, C2- (4、5 脚): 一般接 $1\mu F$ 的电解电容

C1+, C1- (1、3 脚): 一般接 $1\mu F$ 的电解电容

输入输出引脚分两组,基本含义见表 4-2。在实际使用时,若只需要一路串行通信接口,可以使用其中的任何一组。利用 MAX232 将 TTL 电平转换为 RS-232 电平的电路接法稍后结合串行通信接口的外围硬件电路讨论。

焊接到 PCB 板上的 MAX232 芯片检测方法:正常情况下,(1) $T1IN=5V$,则 $T1OUT=-9V$; $T1IN=0V$,则 $T1OUT=9V$ 。(2) 将 $R1IN$ 与 $T1OUT$ 相连,令 $T1IN=5V$,则 $R1OUT=5V$;令 $T1IN=0V$,则 $R1OUT=0V$ 。

所有型号 MCU 的串行通信接口,都具有发送引脚 TxD、接收引脚 RxD,它们是 TTL 电平引脚。要利用这两个引脚与外界实现异步串行通信,还必须将 TTL 电平转为 RS-232 电平,这可利用上节介绍的 MAX232 来完成。本节从通用角度讨论串行通信接口的电路设计、基本编程结构与编程原理,为实际编程做准备。

表 4-2 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	输入	接MCU的TXD	13	输入	接到9芯接口的2脚RXD
	12	输出	接MCU的RXD	14	输出	接到9芯接口的3脚TXD
2	10	输入	接MCU的TXD	8	输入	接到9芯接口的2脚RXD
	9	输出	接MCU的RXD	7	输出	接到9芯接口的3脚TXD

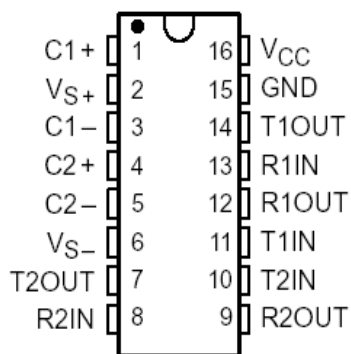


图 4-3 MAX232 引脚

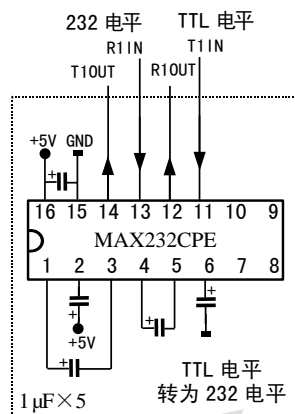


图 4-4 串行通信接口电平转换电路

具有串行通信接口的 MCU，一般具有发送引脚（TXD）与接收引脚（RXD），不同公司或不同系列的 MCU，使用的引脚缩写名可能不一致，但含义相同。串行通信接口的外围硬件电路，主要目的是将 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 4-4 给出了基本串行通信接口的电平转换电路。

基本工作过程是：

发送过程：MCU 的 TXD（TTL 电平）经过 MAX232 的 11 脚（T1IN）送到 MAX232 内部，在内部 TTL 电平被“提升”为 232 电平，通过 14 脚（T1OUT）发送出去。

接收过程：外部 232 电平经过 MAX232 的 13 脚（R1IN）进入到 MAX232 的内部，在内部 232 电平被“降低”为 TTL 电平，经过 12 脚（R1OUT）送到 MCU 的 RXD，进入 MCU 内部。

进行 MCU 的串行通信接口编程时，只针对 MCU 的发送与接收引脚，与 MAX232 无关，MAX232 只是起到电平转换作用。

4.2 MK60N512VMD100的UART模块功能描述

MK60N512VMD100 的通用异步收发器 UART，支持全双工的数据传输，可编程 8 位或者 9 位数据格式，采用标准不归零传号/空号（NRZ）格式，可以选择通过配置波特率采用可编程脉冲宽度的 IrDA 1.4 归零逆转（RZI）格式。基于模块时钟频率的 32 分之一，有 13 位波特率的选择。可以独立地启用发送器和接收器，分别设置发送器与接收器的极性，每一个发送和接收可支持 1、4、8、16、32、64 和 128 数据字的缓冲区，发送与接收有独立的 FIFO 结构。支持 SIM 卡和智能卡接口的 ISO 7816 协议。12 个标志符的中断驱动操作。UART 发送器的硬件可产生并发送奇偶校验位，而接收器的奇偶校验硬件则能据此确保接收数据的完整性。具有接收器帧错误检测功能，带有 DMA 接口。

本节主要阐述 MK60N512VMD100 的 UART 模块的结构及功能，包括波特率的计算方法、发送器和接收器等。

MK60N512VMD100 包括 6 个相同且独立的 UART 模块，每个模块都含有相互独立的发送器和接收器。

1. 外部引脚

UART 的外部引脚有：

(1) 发送数据引脚: UTXD_n

(2) 接收数据引脚: URXD_n

引脚名中的“U”是 UART 的简写,“n”表示模块的编号,取 0~5。通常情况串行通信只使用发送数据引脚 UTXD_n 与接收数据引脚 URXD_n。

2. 波特率发生器

UART0 和 UART1 时钟源为内核时钟, UART2~UART5 的时钟源为外设时钟(总线时钟)。波特率由一个 13 位的模数计数器和一个 5 位的分数微调计数器共同决定。13 位 SBR[SBR]范围 1~8191,它决定了模块的时钟分频。微调计数器给波特率时钟增加一个细微的延时,以便匹配系统波特率。波特率时钟与模块时钟同步并驱动接收器。计算公式如下:

$$\text{UART 波特率} = \text{UART 模块时钟} / (16 * (\text{SBR}[\text{SBR}] + \text{BRFD}))$$

3. 发送器的内部结构

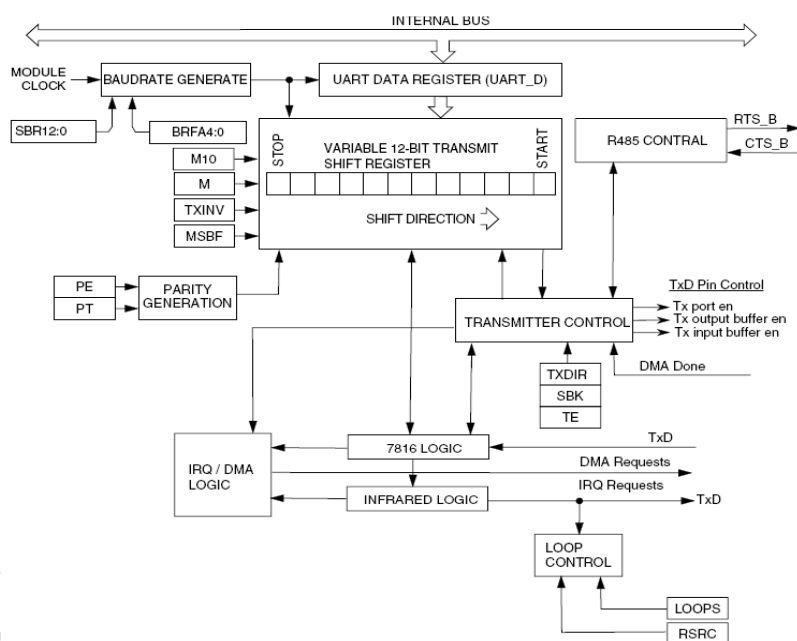


图 4-5 发送器的内部结构

图 4-5 为发送器的内部结构。UART 发送器可以容纳 8、9 或 10 位数据字符。C1[M]和 C1[PE]位和 C4[M10]的状态决定了数据字符的长度。

MCU 通过数据寄存器将数据写入到发送数据缓冲区,然后发送器进行移位发送。当发送结束后,会置位发送缓冲区空标志位(S1[TDRE]),同时也可以根据设置决定是否产生中断。

4. 接收器的内部结构

图 4-6 为接收器的内部结构。UART 接收器可以容纳 8、9 或 10 位数据字符。C1[M]和 C1[PE]位和 C4[M10]的状态决定数据字符的长度。

在 UART 接收期间,接收移位寄存器从异步接收器输入信号移进一个帧。一个完整的帧移进接收移位寄存器后,帧的数据部分发送到 UART 接收缓冲区中。另外,接收进程期间可能的噪音和奇偶校验错误标志也被拷贝到了 UART 接收缓冲区中。接收数据缓冲区通

过数据寄存器和 C3[T8] 寄存器访问。如果接收缓冲区中的数据字数目等于或多于 RWFIFO[RXWATER] 指定的数目，那么 S1[RDRF] 标志位会被置 1。如果 C2[RIE] 也设为 1，则会产生一个接收中断。

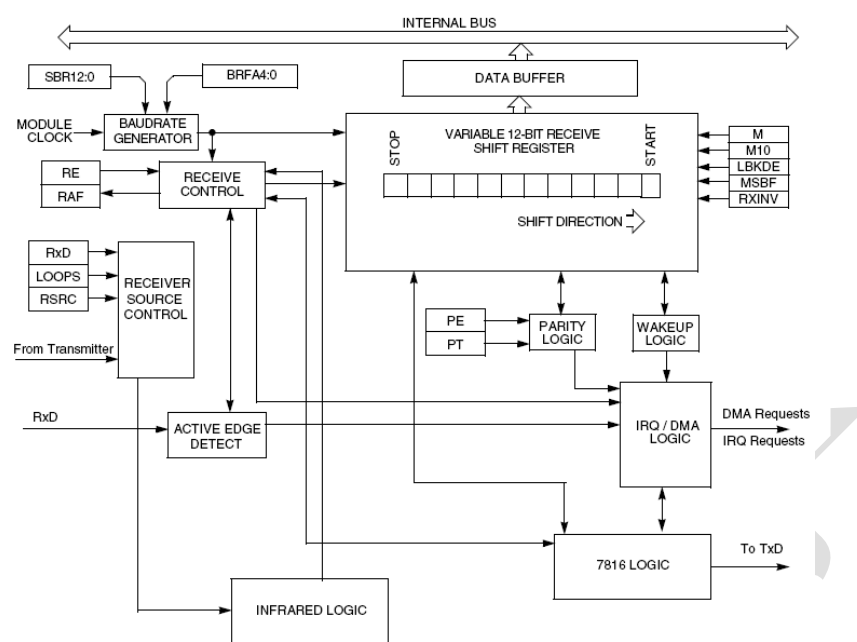


图 4-6 接收器的内部结构

4.3 MK60N512VMD100的UART模块的编程结构

表 4-3 给出了 MK60N512VMD100 的 UART 模块各寄存器相关信息，最左边一列给出了各寄存器的映射地址。最后三列分别给出了相应寄存器的位宽、访问权限和复位时的状态信息。表中给出的都是 UART0 的地址，对于其他 5 个 UART 来说，同名寄存器的地址相对于 UART0 的偏移分别为 0x0000_1000、0x0000_2000、0x0000_3000、0x0008_0000、0x0008_1000。例如，UART0 的波特率高字节寄存器的基址为 0x4006_A000，则 UART1 为 0x4006_B000，UART2 为 0x4006_C000，UART3 为 0x4006_D000，UART4 为 0x400E_A000，UART5 为 0x400E_B000，其他寄存器类似。

表 4-3 UART 模块存储映射

寄存器地址	寄存器	位宽	权限	复位值
UART0				
0x4006_A000	UART波特率高字节寄存器 (UARTx_BDH)	8	读/写	0x00
0x4006_A001	UART波特率低字节寄存器 (UARTx_BDL)	8	读/写	0x04
0x4006_A002	UART控制寄存器1 (UARTx_C1)	8	读/写	0x00
0x4006_A003	UART控制寄存器2 (UARTx_C2)	8	读/写	0x00
0x4006_A004	UART状态寄存器1 (UARTx_S1)	8	只读	0xC0
0x4006_A005	UART状态寄存器2 (UARTx_S2)	8	读/写	0x00
0x4006_A006	UART控制寄存器3 (UARTx_C3)	8	读/写	0x00
0x4006_A007	UART数据寄存器 ((UARTx_D)	8	读/写	0x00
0x4006_A008	UART地址匹配寄存器1 (UARTx_MA1)	8	读/写	0x00
0x4006_A009	UART地址匹配寄存器2 (UARTx_MA2)	8	读/写	0x00
0x4006_A00A	UART控制寄存器4 (UARTx_C4)	8	读/写	0x00
0x4006_A00B	UART控制寄存器5 (UARTx_C5)	8	读/写	0x00
0x4006_A00C	UART扩展数据寄存器 (UARTx_ED)	8	只读	0x00
0x4006_A00D	UART调制解调器寄存器 (UARTx_MODEM)	8	读/写	0x00
0x4006_A00E	UART红外寄存器 (UARTx_IR)	8	读/写	0x00

0x4006_A010	UART FIFO参数寄存器 (UARTx_PFIFO)	8	读/写	0x00
0x4006_A011	UART FIFO控制寄存器 (UARTx_CFIFO)	8	读/写	0x00
0x4006_A012	UART FIFO状态寄存器 (UARTx_SFIFO)	8	读/写	0xC0
0x4006_A013	UART FIFO发送水位寄存器 (UARTx_TWFIFO)	8	读/写	0x00
0x4006_A014	UART FIFO发送计数寄存器 (UARTx_TCFIFO)	8	只读	0x00
0x4006_A015	UART FIFO接收水位寄存器 (UARTx_RWFIFO)	8	读/写	0x01
0x4006_A016	UART FIFO接收计数寄存器 (UARTx_RCFIFO)	8	只读	0x00
0x4006_A018	UART 7816控制数寄存器 (UARTx_C7816)	8	读/写	0x00
0x4006_A019	UART 7816中断使能寄存器 (UARTx_IE7816)	8	读/写	0x00
0x4006_A01A	UART 7816中断状态寄存器 (UARTx_IS7816)	8	读/写	0x00
0x4006_A01B	UART 7816等待参数寄存器 (UARTx_WP7816T0)	8	读/写	0x0A
0x4006_A01B	UART 7816等待参数寄存器 (UARTx_WP7816T1)	8	读/写	0x0A
0x4006_A01C	UART 7816等待N寄存器 (UARTx_WN7816)	8	读/写	0x00
0x4006_A01D	UART 7816等待FD寄存器 (UARTx_WF7816)	8	读/写	0x01
0x4006_A01E	UART 7816错误阈值寄存器 (UARTx_ET7816)	8	读/写	0x00
0x4006_A01F	UART 7816发送长度寄存器 (UARTx_TL7816)	8	读/写	0x00

以上寄存器的用法在 MK60N512VMD100 的芯片手册上有详细的说明，下面按初始化顺序简要阐述基本编程需要使用的寄存器。注意，下面所列寄存器名中的“x”表示 UART 模块编号，取 0~5。

1. UART 控制寄存器 2 (UARTx_C2)

数据位	D7	D6	D5	D4	D3	D2	D1	D0
定义	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
复位	0							

D7—TIE，发送器中断或 DMA 传送使能。TIE 根据 C5[TDMA5]使能 S1[TDRE]标志产生中断请求或者 DMA 传送请求。0：TDRE 中断和 DMA 传送请求禁止。1：TDRE 中断或者 DMA 传送使能。

D6—TCIE，传送结束中断使能。TCIE 使能 S1[TC]传送完成标志产生中断请求。0：TC 中断请求禁止。1：TC 中断请求使能。

D5—RIE，接收器满中断或 DMA 传送使能。RIE 根据 C5[RDMA5]使能 S1[RDRF]标志产生中断请求或 DMA 传送请求。0：RDRF 中断和 DMA 传送请求禁止。1：RDRF 中断或 DMA 传送请求使能。

D4—ILIE，空闲线中断使能。ILIE 根据 C5[ILDMA5]的状态使能 S1[IDLE]空闲线标志产生中断请求。0：IDLE 中断请求禁止。1：IDLE 中断请求使能。

D3—TE，发送器使能。TE 使能 UART 发送器。TE 位可以通过清 0 然后置位 TE 位来排列一个空闲前导。当 7816E 被置位（使能）并且 C7816[TTYPE]=1 时，TE 位在请求块被发送之后会被自动清 0。当 TL7816[TLEN]=0 并且四个附加字符被发送时，TE 位自动清 0 的条件会一直被检测是否达到。0：发送器关闭。1：发送器开启。

D2—RE，接收器使能。RE 使能 UART 接收器。0：接收器关闭。1：接收器开启。

D1—RWU，接收器唤醒控制。RWU 可以被置位来使得 UART 接收器处于备用状态。当 RWU 事件（C1[WAKE]被清 0 的 IDLE 事件或者 C1[WAKE]被置位时的地址匹配）发生时，RWU 自动清 0。当 7816E 被置位时，此位必须被清 0。0：正常操作。1：RWU 使能唤醒功能并禁止接收器中断请求。通常，硬件通过自动清 0RWU 来唤醒接收器。

D0—SBK，发送中止。锁住的 SBK 发送一个中止字符（S2[BRK13]清 0 时，10、11 或 12 个逻辑 0；S2[BRK13]置位时，13 或 14 个逻辑 0）。锁住意味着在中止字符发送结束之前清除 SBK 位。只要 SBK 被置位，发送器会继续发送完整的中止字符。当 7816E 被置位时，此位必须被清 0。0：正常发送器操作。1：发送排列好的中止字符。

2. UART 控制寄存器 1 (UARTx_C1)

数据位	D7	D6	D5	D4	D3	D2	D1	D0
定义	LOOPS	UARTSWAI	RSRC	M	WAKE	ILT	PE	PT
复位	0							

D7—LOOPS，循环模式选择。当 LOOPS 被置位时，RxD 引脚从 UART 分离，发送器输出内部连接到接收器输入。发送器和接收器必须使能来使用循环功能。0：正常操作。1：发送器输出内部连接到接收器输入的循环模式。接收器输入由 RSRC 位决定。

D6—UARTSWAI，UART 在等待模式停止。0：在等待模式 UART 时钟继续运行。1：当 CPU 处于等待模式时，UART 时钟冻结。

D5—RSRC，接收器信号源选择。这个位在 LOOPS 位被置位时才有意义。当 LOOPS 被置位时，RSRC 位决定接收器移位寄存器输入的信号源。0：选择内部回环模式，接收器输入内部连接到发送器输出。1：单线 UART 模式，接收器输入连接到发送器引脚输入信号。

D4—M，9 位或 8 位模式选择。当 7816E 被置位（使能）时，此位必须被置位。0：正常模式-起始位+8 位数据位（由 MSBF 决定 MSB/LSB 优先）+停止位。1：使用模式-起始位+9 位数据位（由 MSBF 决定 MSB/LSB 优先）+停止位。

D3—WAKE，接收器唤醒方法选择。WAKE 决定哪种条件唤醒 UART：接收数据字符最高位的地址标记或者接收引脚输入信号上的空闲情况。0：空闲线唤醒。1：地址标记唤醒。

D2—ILT，空闲线类型选择。ILT 决定接收器何时开始计数当作空闲字符的逻辑 1。在一个有效的起始位或者停止位之后计数开始。如果起始位之后计数开始，那么停止位前的逻辑 1 的字符串可能导致空闲字符的错误识别。停止位后开始计数避免了错误的空闲字符识别，但是需要合适的同步传输。0：起始位后开始计数空闲字符位。1：停止位后开始计数空闲字符位。

D1—PE，奇偶校验使能。使能奇偶校验功能。当奇偶校验使能时，停止位前会被插入一个奇偶校验位。7816E 被置位（使能）时，此位必须被置位。0：奇偶校验功能禁止。1：奇偶校验功能使能。

D0—PT，校验类型。PT 决定了 UART 是否产生并检查奇校验位或者偶校验位。偶校验位中，偶数个 1 会清校验位，奇数个 1 会置位校验位。奇校验位中，奇数个 1 会清校验位，偶数个 1 会置位校验位。当 7816E 被置位（使能）时，此位必须清零。0：偶校验。1：奇校验。

3. UART 波特率高字节寄存器 (UARTx_BDH)

UARTx_BDH 寄存器与 UARTx_BDL 寄存器一起控制 UART 波特率发生器的预分频因子。更新 13 位波特率设置(SBR[12:0])时，首先写 BDH 缓冲新值的高半部分，然后写 BDL。直到 BDL 被写入时 BDH 中的值才会变。

数据位	D7	D6	D5	D4	D3	D2	D1	D0
定义	LBKDIE	RXEDGIE		SBR				
复位	0							

D7—LBKDIE，LIN 中止检测中断使能。LBKDIE 根据 LBKDDMAS 的状态使能 LIN 中止检测标识 LBKDIF 来产生中断请求。0：LBKDIF 中断请求禁止。1：LBKDIF 中断请求使能。

D6—RXEDGIE，RxD 输入有效边沿中断使能。RXEDGIE 使能接收输入有效边沿 RXEDGIF 来产生中断请求。0：RXEDGIF 硬件中断禁止（使用轮询）。1：RXEDGIF 中断请求使能。

D4~D0—SBR，UART 波特率位。UART 的波特率由这 5 位和 UARTx_BDL 共 13 位决定。UARTx_BDL 的复位值为 0x04。

4. UART 控制寄存器 4 (UARTx_C4)

数据位	D7	D6	D5	D4	D3	D2	D1	D0
定义	MAEN1	MAEN2	M10	BRFA				
复位	0							

D7—MAEN1，地址匹配模式使能 1。0：如果 MAEN2 清零，所有接收到的数据被传送到数据缓冲区中。1：所有最高有效位被清的接收数据被丢弃。所有最高有效位被置位的接收数据与 MA1 寄存器中的内容比较。如果没有匹配成功，则数据被丢弃。如果匹配，那么数据传送到数据缓冲区。当 C7816[ISO7816E]被置位（使能）时，此位必须清零。

D6—MAEN2，地址匹配模式使能 2。0：如果 MAEN1 清零，则所有接收的数据被传送到数据缓冲区。1：所有最高有效位清零的接收数据被丢弃。所有最高有效位置位的接收数据，跟 MA2 寄存器中的内容比较。如果没有匹配成功，数据被丢弃。如果匹配，数据被传送到数据缓冲区。当 C7816[ISO7816E]被置位（使能），此位必须清零。

D5—M10，10 位模式选择。M10 位使得第十位非存储器映射位到串行传输中。第十位为奇偶校验位。M10 位并不影响 LIN 发送或者检测中止。如果 M10 被置位，C1[M]和 C1[PE]必须被置位。当 C7816[ISO7816E]被置位（使能），此位必须清零。0：奇偶校验位是串行传输中的第 9 位。1：奇偶校验位是串行传输中的第 10 位。

D4~D0—BRFA，波特率微调。这个位用来对平均波特率以 1/32 的增量增加时间精度。

5. UART 状态寄存器 1 (UARTx_S1)

UARTx_S1 寄存器为 UART 中断或 DMA 请求提供 MCU 的输入。这个寄存器也可以由 MCU 进行轮询来检测。可以通过读状态寄存器之后读或写（取决于中断标志类型）UART 数据寄存器来清除标志。其他的指令只要不影响 I/O 处理也可以插入到上述两步中执行。

数据位	D7	D6	D5	D4	D3	D2	D1	D0
定义	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
复位	1	1	0	0	0	0	0	0

D7—TDRE，发送数据寄存器空标志。当发送缓冲区（D 和 C3[T8]）中的数据字的数目等于或小于 TWFIPO[TXWATER]中的数目时，TDRE 会被置位。正在传输过程中的字符不包含在此计数中。TDRE 置位时读 S1，然后写 UART 数据寄存器（D）会清 TDRE。为了能够获得更有效的中断服务，除了写到缓冲区的最终值之外的所有数据应该都写入 D/C3[T8]。然后在写最终数据前读 S1 可以清 TRDE 标志。0：发送缓冲区中的数据数目比 TWFIPO[TXWATER]中的数目大。1：自从标志清零后，发送缓冲区中的数据数目等于或小于 TWFIPO[TXWATER]中的数目。

D6—TC，发送完成标志。当有效数据传输过程在进行或有前导符或中止符被加载时，TC 被清零。当发送缓冲区为空并且没有数据，也没有前导或者中止符正在传输时，TC 被置位。当 TC 为 1 时，发送数据输出信号变成空闲状态（逻辑 1）。当 7816E 被置位（使能）时，此位在任何一个 NACK 信号接收到了之后但在任何相应的保护期期满之前被置位。清零 TC，需要在 TC 为 1 时读 S1，或者将新的发送数据写入 UART 数据寄存器（D）或者通过清零 C2[TE]位后置位来排队一个前导或者置位 C2 的 SBK 来排队一个中止符。0：发送进行中（发送数据、前导符或者中止字符）。1：发送空闲中（发送完成）。

D5—RDRF，接收数据寄存器满标志。当接收缓冲区中数据字数目等于或多于 TWFIPO[TXWATER]中的数目时，RDRF 被置位。正在接收中的字符不包含在此计数中。当

S2[LBKDE]为 1 时, RDRF 只能为 0。另外, 当 S2[LBKDE]为 1 时, 接收的数据字会被存储在接收缓冲区中但会彼此覆盖。RDRF 为 1 时读 S1, 然后读 UART 数据寄存器 (D) 会清 RDRF。为了获得更有效的中断和 DMA 操作, 除了最终从缓冲区读出的数据之外的所有数据都使用 D/C3[T8]/ED。读 S1 和最终数据值会清 RDRF。0: 接收缓冲区中数据字的数目少于 RXWATER 中的数目。1: 此标志被清之后, 接收缓冲区中数据字的数目等于或多于 RXWATER 中的数目。

D4—IDLE, 空闲线标志。当 10 个连续的逻辑 1 (如果 C1[M]=0)、11 个连续的逻辑 1 (C1[M]=1、C4[M10]=0) 或者 12 个连续的逻辑 1 (C1[M]=1、C4[M10]=1、C1[PE]=1) 出现在接收器输入时, IDLE 为 1。IDLE 标志清零后, 必须接收一个帧 (虽然 C2[RWU]为 1 时没有必要存储在数据缓冲中) 或者一个 LIN 中止字符必须在一个空闲状态可以置位 IDLE 标志之前置位 S2[LBKDIF]标志。在 IDLE 为 1 时读 UART 状态 S1 然后读 D 可以清 IDLE。当 7816E 被置位 (使能) 时, 空闲检测不被支持, 因此这个标志被忽略。0: 自从 IDLE 标志上一次清零后, 接收器输入现在没有数据或者从来都没有数据。1: 自从上一次被置位后, 接收器输入变成空闲的或者一直都没有被清。

D3—OR, 接收器溢出标志。当软件没有成功防止接收数据寄存器数据溢出时, OR 为 1。OR 位会在数据字溢出缓冲区, 并收到停止位后被置位。此时, 所有其他错误标志 (FE、NF 和 PF) 都不能被置位。移位寄存器中的数据丢失, 但是已经在 UART 数据寄存器中的数据没有收到影响。如果 OR 标志被置位, 即使有足够的空间也不会有数据存储在数据缓冲区中。OR 被置位时读 S1 然后读 UART 数据寄存器 (D) 会清 OR。在 7816 模式中, 可以通过编程 C7816[ONACK]位配置返回的 NACK。0: 自从上一次标志被清后没有溢出生。1: 自从上一次溢出发生后, 溢出发生或者溢出标志没被清除。

D2—NF, 噪声标志。UART 在接收器输入检测到噪声时, NF 被置位。在溢出或 LIN 中止检测功能使能 (S2[LBKDE]=1) 时, NF 不会被置位。NF 被置位表明自从它上次被清后接收到带有噪声的数据字。读 S1 然后读 UART 数据寄存器 (D) 可以清 NF。0: 自从上次标志被清后没有检测到噪声。如果接收缓冲区深度大于 1, 那么收器缓冲区的数据可能带有噪声。1: 自从上次标志被清后至少有一个接收到的数据字带有噪声。

D1—FE, 帧错误标志。当逻辑 0 被接收当作停止位时, FE 被置位。在溢出或 LIN 中止检测功能使能 (S2[LBKDE]=1) 时, FE 不会被置位。FE 阻止进一步的数据接收直到它被清零。FE 为 1 时读 S1 然后读 UART 数据寄存器 (D) 可以清 FE。数据缓冲区中最后一个数据代表了帧错误使能的接收数据。然而, 当 7816E 被置位 (使能) 时, 帧错误不被支持。0: 没有检测到帧错误。1: 帧错误。

D0—PF, 奇偶校验错误标志。当 PE 被置位、S2[LBKDE]禁用并且接收到的数据不能和它的奇偶校验位匹配时, PF 被置位。在溢出条件下 PF 不会被置位。PF 位被置位仅表示自从它上次被清零后, 接收到奇偶校验错误的数据字。读 S1 然后读 UART 数据寄存器 (D), 可以清 PF。0: 自从上次这个标志清零后, 没有检测到奇偶校验错误。如果接收缓冲区深度大于 1, 那么接收缓冲区中有可能有数据带有奇偶校验错误。1: 自从上次这个标志被清零后, 至少接收一个带有奇偶校验错误的数据字。

6. UART 数据寄存器 (UARTx_D)

UARTx_D 其实是两个单独的寄存器, 读时会返回只读接收数据寄存器中的内容, 写时会写到只写发送数据寄存器。

4.4 基于构件方法的UART编程

4.4.1 UART 构件的函数原型设计

UART 具有初始化、接收和发送三种基本操作。按照构件的思想,可将它们封装成三个独立的功能函数,初始化函数完成对 UART 模块的工作属性的设定,接收和发送功能函数则完成实际的通信任务。对 UART 模块进行编程,实际上已经涉及到对硬件底层寄存器的直接操作,因此,可将初始化、接收和发送三种基本操作所对应的功能函数共同放置在命名为 `uart.c` 的文件中,并按照相对严格的构件设计原则对其进行封装,同时配以命名为 `uart.h` 的头文件,用来定义模块的基本信息和对外接口。

按照模块所具有的基本操作来确定构件中应该具有哪些功能集合,是很自然也很重要的事情。但是,要实现编程的构件化,对具体的函数原型的设计则是重中之重。函数原型设计的好坏直接体现构件化编程的成败。下面就以 UART 的初始化、接收和发送三种基本操作为例,来说明实现构件化的全过程。

需要说明的是,实现构件化编程的 UART 软件模块应当具有以下几个特点:

(1) UART 模块是最底层的构件,它主要向上提供三种服务,分别是 UART 模块的初始化、接收单个字节和发送单个字节,向下则直接访问模块寄存器,实现对硬件的直接操作。另外,从现实使用角度出发,它还需要封装接收 N 个字节和发送 N 个字节的子功能函数。

(2) UART 模块在软件上对应 1 个 `uart.c` 程序源代码文件和 1 个 `uart.h` 头文件,当需要对它进行移植时,大多数情况下只需简单拷贝这两个文件即可,无需对源代码文件和头文件进行修改,只有当实施不同芯片之间的移植时,才需要修改头文件中与硬件相关的宏定义。

(3) 上层构件或软件在使用该构件时,严格禁止通过全局变量来传递参数,所有的数据传递都直接通过函数的形式参数来接收。这样做不但使得接口简洁,更加避免了全局变量可能引发的安全隐患。

UART 模块是最基本最底层的构件,它在满足功能完整性的情况下,具有不可再分的特性,可以把它叫做“元构件”。下面简要介绍,按照构件化的思想对 UART 模块进行编程的过程。

通过以上分析,可以设计 UART 构件的 7 个基本功能函数。

(1) 初始化

```
void uart_init(UART_MemMapPtr uartch, uint32 sysclk, uint32 baud);
```

(2) 发送单个字节

```
void uart_send1(UART_MemMapPtr uartch, uint8 ch);
```

(3) 接收单个字节

```
uint8 uart_re1(UART_MemMapPtr uartch, uint8 *ch);
```

(4) 发送 N 个字节

```
void uart_sendN(UART_MemMapPtr uartch, uint8* buff, uint16 len);
```

(5) 接收 N 个字节

```
uint8 uart_reN(UART_MemMapPtr uartch, uint8* buff, uint16 len);
```

(6) 使能串口接收中断

```
void enableuartreint(UART_MemMapPtr uartch, uint8 irqno);
```

(7) 禁止串口接收中断

```
void disableuartreint(UART_MemMapPtr uartch, uint8 irqno);
```

4.4.2 UART 构件的头文件

与 UART 通信子函数相关的文件有头文件 `uart.h`，以及包含 UART 初始化和收发子函数的程序文件 `uart.c`。

头文件 `uart.h` 中的内容可分为两个主要的部分，它们分别是7个函数原型的声明和外设模块寄存器相关信息的定义。前者给出了本UART构件对上层构件或软件所提供的接口函数，而后者则指明了本“元构件”与具体硬件相关的信息。串行通信头文件 `UART.h` 含有串行通信寄存器和标志位定义以及串行通信相关函数声明，下面简要给出了它的主要内容。以初始化函数 `uart_init` 为例，通道号、当前的系统时钟、希望实现的通信波特率以及初始化时是否使能中断，都被设计为函数参数。这样的话，应用程序和上层构件在使用（调用）它时，将具有极大的灵活性。文件还给出了必要的硬件相关信息，当要把该构件移植到其他芯片时，就必须检查并修改这些信息。

```
//-----*
// 文件名: uart.h                                     *
// 说 明: uart构件头文件                             *
//-----*
#ifndef __UART_H__
#define __UART_H__
    //1 头文件
    #include "common.h"

    //2 宏定义
    //2.1 串口号宏定义
    #define UART0 UART0_BASE_PTR
    #define UART1 UART1_BASE_PTR
    #define UART2 UART2_BASE_PTR
    #define UART3 UART3_BASE_PTR
    #define UART4 UART4_BASE_PTR
    #define UART5 UART5_BASE_PTR

    //2.2 接收引脚irq号宏定义
    #define UART0irq 45
    #define UART1irq 47
    #define UART2irq 49
    #define UART3irq 51
    #define UART4irq 53
    #define UART5irq 55

    //3 函数声明

//-----*
//函数名: uart_init                                     *
//功 能: 初始化uartx模块。                             *
//参 数: uartch:串口号                                 *
//      sysclk:系统总线时钟,以MHz为单位                 *
//      baud:波特率,如9600, 38400等,一般来说,速度越慢,通信越稳 *
//返 回: 无                                             *
//说 明:                                             *
//-----*
void uart_init (UART_MemMapPtr uartch, uint32 sysclk, uint32 baud);

//-----*
//函数名: uart_rel                                     *
//功 能: 串行接受1个字节                             *
```

```

//参 数: uartch: 串口号 *
//      ch: 接收到的字节 *
//返 回: 成功:1;失败:0 *
//说 明: *

//-----*
uint8 uart_re1 (UART_MemMapPtr uartch,uint8 *ch);

//-----*
//函数名: uart_send1 *
//功 能: 串行发送1个字节 *
//参 数: uartch: 串口号 *
//      ch: 要发送的字节 *
//返 回: 无 *
//说 明: *

//-----*
void uart_send1 (UART_MemMapPtr uartch, uint8 ch);

//-----*
//函数名: uart_reN *
//功 能: 串行 接收n个字节 *
//参 数: uartch: 串口号 *
//      buff: 接收缓冲区 *
//      len:接收长度 *
//返 回: 1:成功;0:失败 *
//说 明: *

//-----*
uint8 uart_reN (UART_MemMapPtr uartch ,uint8* buff,uint16 len);

//-----*
//函数名: uart_sendN *
//功 能: 串行 接收n个字节 *
//参 数: uartch: 串口号 *
//      buff: 发送缓冲区 *
//      len:发送长度 *
//返 回: 无 *
//说 明: *

//-----*
void uart_sendN (UART_MemMapPtr uartch ,uint8* buff,uint16 len);

//-----*
//函数名: enableuartreint *
//功 能: 开串口接收中断 *
//参 数: uartch: 串口号 *
//      irqno: 对应irq号 *
//返 回: 无 *
//说 明: *

//-----*
void enableuartreint (UART_MemMapPtr uartch,uint8 irqno);

```

```

//-----*
//函数名: disableuartreint *
//功 能: 关串口接收中断 *
//参 数: uartch: 串口号 *
//      irqno: 对应irq号 *
//返 回: 无 *
//说 明: *
//-----*

void disableuartreint(UART_MemMapPtr uartch, uint8 irqno);
#endif

```

4.4.3 UART 构件的源程序文件

1. UART 构件的初始化功能函数: uart_init

```

//-----*
//函数名: uart_init *
//功 能: 初始化uartx模块。 *
//参 数: uartch:串口号 *
//      sysclk:系统总线时钟, 以MHz为单位 *
//      baud:波特率, 如9600, 38400等, 一般来说, 速度越慢, 通信越稳 *
//返 回: 无 *
//说 明: *
//-----*

void uart_init (UART_MemMapPtr uartch, uint32 sysclk, uint32 baud)
{
    register uint16 sbr, brfa;
    uint8 temp;

    //使能引脚
    if (uartch == UART0_BASE_PTR)
    {
        //在PTD6上使能UART0_TXD功能
        PORTD_PCR6 = PORT_PCR_MUX(0x3)
        //在PTD7上使能UART0_RXD
        PORTD_PCR7 = PORT_PCR_MUX(0x3);
    }
    else if (uartch == UART1_BASE_PTR)
    {
        //在PTC4上使能UART1_TXD功能
        PORTC_PCR4 = PORT_PCR_MUX(0x3);

        //在PTC3上使能UART1_RXD
        PORTC_PCR3 = PORT_PCR_MUX(0x3);
    }
    else if (uartch == UART2_BASE_PTR)
    {
        //在PTD3上使能UART2_TXD功能
        PORTD_PCR3 = PORT_PCR_MUX(0x3);
        //在PTD2上使能UART2_RXD
        PORTD_PCR2 = PORT_PCR_MUX(0x3);
    }
}

```

```

else if (uartch == UART3_BASE_PTR)
{
//在PTC17上使能UART3_TXD功能

PORTC_PCR17 = PORT_PCR_MUX(0x3);
//在PTC16上使能UART3_RXD
PORTC_PCR16 = PORT_PCR_MUX(0x3);
}
else if (uartch == UART4_BASE_PTR)
{
//在PTE24上使能UART4_TXD功能
PORTE_PCR24 = PORT_PCR_MUX(0x3);
//在PTE25上使能UART4_RXD
PORTE_PCR25 = PORT_PCR_MUX(0x3);
}
else if (uartch == UART5_BASE_PTR)
{
//在PTE8上使能UART5_TXD功能
PORTE_PCR8 = PORT_PCR_MUX(0x3);
//在PTE9上使能UART5_RXD
PORTE_PCR9 = PORT_PCR_MUX(0x3);
}

//使能串口时钟
if(uartch == UART0_BASE_PTR)
SIM_SCGC4 |= SIM_SCGC4_UART0_MASK;
Else if (uartch == UART1_BASE_PTR)
SIM_SCGC4 |= SIM_SCGC4_UART1_MASK;
Else if (uartch == UART2_BASE_PTR)
SIM_SCGC4 |= SIM_SCGC4_UART2_MASK;
Else if(uartch == UART3_BASE_PTR)
SIM_SCGC4 |= SIM_SCGC4_UART3_MASK;
Else if(uartch == UART4_BASE_PTR)
SIM_SCGC1 |= SIM_SCGC1_UART4_MASK;
else
SIM_SCGC1 |= SIM_SCGC1_UART5_MASK;

//禁止发送接受
UART_C2_REG(uartch) &= ~(UART_C2_TE_MASK

UART_C2_RE_MASK );

//配置成8位无校验模式
UART_C1_REG(uartch) = 0;

//计算波特率，串口0、1使用内核时钟，其它串口使用外设时钟，系统时钟为
//外设时钟的2倍
if ((uartch == UART0_BASE_PTR) | (uartch == UART1_BASE_PTR))
sysclk+=sysclk;

sbr = (uint16)((sysclk*1000)/(baud * 16));
temp = UART_BDH_REG(uartch) & ~(UART_BDH_SBR(0x1F));
UART_BDH_REG(uartch) = temp |  UART_BDH_SBR(((sbr & 0x1F00) >> 8));
UART_BDL_REG(uartch) = (uint8)(sbr & UART_BDL_SBR_MASK);
brfa = (((sysclk*32000)/(baud * 16)) - (sbr * 32));
temp = UART_C4_REG(uartch) & ~(UART_C4_BRFA(0x1F));

```

```
UART_C4_REG(uartch) = temp | UART_C4_BRFA(brfa);
```

```
//使能发送接受
```

```
UART_C2_REG(uartch) |= (UART_C2_TE_MASK
```

```
UART_C2_RE_MASK );
```

```
}
```

2. UART 构件单字节发送功能函数：uart_send1

```
//-----*  
//函数名: uart_send1 *  
//功 能: 串行发送1个字节 *  
//参 数: uartch: 串口号 *  
//      ch: 要发送的字节 *  
//返 回: 无 *  
//说 明: *  
//-----*  
void uart_send1 (UART_MemMapPtr uartch, uint8 ch)  
{  
    //等待发送缓冲区空  
    while(!(UART_S1_REG(uartch) & UART_S1_TDRE_MASK));  
    //发送数据  
    UART_D_REG(uartch) = (uint8)ch;  
}
```

3. UART 构件的单字节接收功能函数: uart_re1

```
//-----*
//函数名: uart_re1                                     *
//功 能: 串行接受1个字节                             *
//参 数: uartch: 串口号                               *
//      ch: 接收到的字节                             *
//返 回: 成功:1;失败:0                               *
//说 明:                                              *
//-----*
uint8 uart_re1 (UART_MemMapPtr uartch, uint8 *ch)
{
    uint32 k;

    for (k = 0; k < 0xfbbb; k++)//有时间限制
        if((UART_S1_REG(uartch) & UART_S1_RDRF_MASK) != 0)//判断接收缓冲区是否满
        {
            *ch = UART_D_REG(uartch);
            return 1;                                     //接受成功
        }
    if(k>=0xfbbb)
    {
        return 0;                                     //接受失败
    }
    return 0;
}
```

4. UART 构件的多字节发送功能函数: uart_sendN

```
//-----*
//函数名: uart_sendN                                   *
//功 能: 串行 接收n个字节                             *
//参 数: uartch: 串口号                               *
//      buff: 发送缓冲区                             *
//      len: 发送长度                                 *
//返 回: 无                                           *
//说 明:                                              *
//-----*
void uart_sendN (UART_MemMapPtr uartch , uint8* buff, uint16 len)
{
    int i;
    for(i=0; i<len; i++)
    {
        uart_send1(uartch, buff[i]);
    }
}
```

5. UART 构件的多字节接收功能函数: uart_reN

```
//-----*
//函数名: uart_reN                                     *
//功 能: 串行 接收n个字节                             *
//参 数: uartch: 串口号                               *
//      buff: 接收缓冲区                             *
//      len:接收长度                                 *
//返 回: 1:成功;0:失败                               *
//说 明:                                             *
//-----*
uint8 uart_reN (UART_MemMapPtr uartch ,uint8* buff,uint16 len)
{
    uint16 m=0;
    while (m < len)
    {
        if(0==uart_re1(uartch,&buff[m]))
            return 0; //接收失败
        else m++;
    }

    return 1;      //接收成功
}
```

6. UART 构件的使能串口接收中断功能函数: enableuartreint

```
//-----*
//函数名: enableuartreint                             *
//功 能: 开串口接收中断                             *
//参 数: uartch: 串口号                               *
//      irqno: 对应irq号                             *
//返 回: 无                                           *
//说 明:                                             *
//-----*
void enableuartreint(UART_MemMapPtr uartch,uint8 irqno)
{
    UART_C2_REG(uartch)|=UART_C2_RIE_MASK; //开放UART接收中断
    enable_irq(irqno); //开接收引脚的IRQ中断
}
```

7. UART 构件的禁止串口接收中断功能函数：disableuartreint

```
//-----*
//函数名: disableuartreint                                *
//功 能: 关串口接收中断                                  *
//参 数: uartch: 串口号                                    *
//      irqno: 对应irq号                                  *
//返 回: 无                                                *
//说 明:                                                  *
//-----*

void disableuartreint(UART_MemMapPtr uartch, uint8 irqno)
{
    UART_C2_REG(uartch)&=~UART_C2_RIE_MASK;    //禁止UART接收中断
    disable_irq(irqno); //关接收引脚的IRQ中断
}
```

4.4.4 UART 构件的测试工程

嵌入式软件中，main.c 和 isr.c 这两个文件反映了软件系统的整体执行流程。当系统启动并初始化后，程序根据 main.c 中定义的主循环顺序执行；一旦遇到中断请求，立即转去执行 isr.c 中定义的相应中断处理程序；当中断处理程序运行结束后，再返回中断处继续顺序执行。在 main.c 和 isr.c 中，可通过调用上一小节介绍的 7 个功能接口函数实现 UART 模块收发数据，实现方式有两种：查询方式和中断方式。

查询方式：UART3 模块首先向 PC 机发送字符串“Hello World!”，然后等待接收 PC 机从串口发送来的数据，若成功接收到 1 个数据，则立即将该数据回发给 PC 机，随后继续等待接收 1 个数据并回发，如此循环。主函数文件 main.c 如下编写：

```

//-----*
// 工 程 名: uart_loop *
// 硬件连接: 将K60核心板与扩展板连接 *
// 程序描述: 启动后发送"Hello World!", 之后等待接收一个字节数据, 收到后回发*
// 目 的: 初步掌握利用查询方式进行串行通信的基本知识 *
// 说 明: 波特率为9600, 使用UART3口 *
//-----苏州大学飞思卡尔嵌入式系统实验室2011年-----*
//头文件
#include "includes.h"
//全局变量声明
extern int periph_clk_khz;
//主函数
void main(void)
{
    //1 主程序使用的变量定义
    uint32 runcount; //运行计数器
    uint8 ch;
    //2 关中断
    DisableInterrupts; //禁止总中断
    //3 模块初始化
    light_init(Light_Run_PORT, Light_Run1, Light_OFF); //指示灯初始化
    uart_init (UART3, periph_clk_khz, 9600); //串口初始化
    //4 开中断

    uart_sendN(UART3, (uint8*)"Hello World!", 12);
    //主循环
    while(1)
    {
        //1 主循环计数到一定的值, 使小灯的亮、暗状态切换
        runcount++;
        if(runcount>=10)
        {
            light_change(Light_Run_PORT, Light_Run1); //指示灯的亮、暗状态切换
            runcount=0;
        }
        //2 串口接收一个字节的的数据
        if(uart_reN(UART3, &ch, 1))
        {
            uart_send1 (UART3, ch); //发送回去
        }
    }
}

```

图 4-7 为测试串行通信的高端程序运行界面, 建议读者充分使用串口工具。



图 4-7 串口调试工具软件界面

4.5 K60第一个带有中断功能的实例

采用中断方式收发数据时，需编写中断处理程序。在 CW 环境下使用 K60 芯片中断的步骤是：

- (1) 在 main.c 中，依照“关总中断→开模块中断→开总中断”的顺序打开模块中断；
- (2) 在 isr.c 文件中，编写中断服务程序；
- (3) 在 vectors.h 文件中，修改中断向量表；

K60 开始运行后，系统状态寄存器 SR 中 16、17 和 18 这三位的默认值都是“1”，即关闭所有中断，所以要使用中断，必须更改这三位的值。它就相当于一个总闸，如果总闸不开，所有中断都不可能发生。操作状态寄存器 SR，需要汇编指令来实现：

```
CPSIE i      //关总中断
```

```
CPSID i      //开总中断
```

为了方便代码移植，在 common.h 文件中做了如下定义：

```
#define EnableInterrupts asm(" CPSIE i");//开总中断
```

```
#define DisableInterrupts asm(" CPSID i");//关总中断
```

下面以 UART3 接收中断为例，实现以下功能：UART3 模块首先向 PC 机发送字符串“Hello World!”；同时，串口等待接收从 PC 机发来的数据，一旦接到数据，马上将该数据回发给 PC 机。串口接收程序使用中断来实现，中断处理程序执行完毕后，又回到主程序。

1. 主函数文件 (main.c)

```
//-----*
// 工 程 名: uart_int *
// 硬件连接: 将K60核心板与扩展板连接 *
// 程序描述: 启动后发送"Hello World!", 之后等待接收一个字节数据, 收到后回发*
// 目 的: 初步掌握利用中断方式进行串行通信的基本知识 *
// 说 明: 波特率为9600, 使用UART3口 *
//-----苏州大学飞思卡尔嵌入式系统实验室2011年-----*
//头文件
#include "includes.h"
//全局变量声明
extern int periph_clk_khz;
//主函数
void main(void)
{
    //1 主程序使用的变量定义
    uint32 runcount; //运行计数器
    //2 关中断
    DisableInterrupts; //禁止总中断
    //3 模块初始化
    light_init(Light_Run_PORT, Light_Run1, Light_OFF); //指示灯初始化
    uart_init (UART3, periph_clk_khz, 9600); //串口初
    始化
    //4 开中断
    enableuartreint (UART3, UART3irq); //开串口3接收
    中断
    EnableInterrupts;
    //开总中断

    uart_sendN(UART3, (uint8*)"Hello World!", 12);
    //主循环
    while(1)
    {
        //1 主循环计数到一定的值, 使小灯的亮、暗状态切换
        runcount++;
        if(runcount>=5000000)
        {
            light_change(Light_Run_PORT, Light_Run1); //指示灯的亮、暗状态切换
            runcount=0;
        }
    }
}
```

2. 中断处理函数文件 (isr.c)

以下简要给出了与本工程相关的中断函数定义。

```

//-----*
// 文件名: isr.c                                     *
// 说 明: 中断处理例程                             *
//-----苏州大学飞思卡尔嵌入式系统实验室2011年-----*

#include "includes.h"

//-----*
//函数名: uart3_isr                                 *
//功 能: 串口3数据接收中断例程                     *
//说 明: 无                                           *
//-----*

void uart3_isr(void)
{
    uint8 ch;
    DisableInterrupts;                                //关总中断
    //接收一个字节数据并回发
    if(uart_re1 (UART3,&ch))
        uart_send1(UART3,ch);
    EnableInterrupts;                                //开总中断
}

```

3. 中断向量表文件（vectors.s）

编写好中断处理函数后，接下来需要修改中断向量表。在 `vectors.h` 文件中，找到对应位置，把定义的中断服务程序名写入,然后再声明外部中断处理函数。以下简要给出了与本工程相关的中断向量表内容。

```

//-----*
// 文件名: vectors.h                                 *
// 说 明: 中断向量表宏定义                         *
//-----*

#ifndef __VECTORS_H
#define __VECTORS_H 1
    #include "common.h"
    //中断服务例程
    extern void uart3_isr(void);
    //默认中断服务例程
    void default_isr(void);
    void abort_isr(void);
    void hard_fault_handler_c(unsigned int * hardfault_args);

    typedef void pointer(void);

    extern void __startup(void);
    extern unsigned long __BOOT_STACK_ADDRESS[];
    extern void __iar_program_start(void);
    #define VECTOR_000      (pointer*)__BOOT_STACK_ADDRESS
#define VECTOR_001        __startup
...
#define VECTOR_067        uart3_isr
...
#endif

```

4.6 进一步讨论

本节将讨论串行通信的一些扩展应用。

4.6.1 流控制与 Break 信号

串口通信通常只用到 9 芯串口的 2 (RXD)、3 (TXD) 和 5 (GND) 三个管脚。像这样的简单三线连接,使用的数据线少,通信中可节约通信成本,增加稳定性,能满足绝大多数的需求。但严格来说,当两个系统进行串行通信时,在接收方收完数据之前,应当禁止发送者发送新的数据,这个过程称为流控制 (flow control) 或握手 (shake hands)。流控制常有软件方式和硬件方式。

软件握手又称为 XON/XOFF 方式,用于接受者和发送者之间无法实现硬件握手的场合。它用专门的字符来启动 (XON) 或停止 (XOFF) 数据流。这些字符定义在美国标准信息交换码 (ASCII 码) 中。软件握手用两个字符来实现流控制,一个代表请求对方暂停传输,另一个代表清除暂停的请求,继续数据传送。通常情况下用 ASCII 码为 0x13 和 0x11 二进制数据来表示,这样,在传输的数据中应禁止包含这两个字符,因为它们会被接收者理解为流控制字符而不是数据。假如发送的只是 ASCII 字符,就不会存在这样的问题,但发送二进制数据的话就会遇到这种情况。常见的解决办法是在发送前对二进制数据进行预处理,将它们用 ASCII 字符来代替。

在 RS-232 标准中,硬件握手使用 RTS (Request To Send, 请求发送) 和 CTS (Clear To Send, 允许发送) 两个信号。接收方当准备好接收数据时就将 CTS 信号设置成 0 电平,没有准备好就设置为 1 电平。同样,发送方当准备好发送数据时,就将 RTS 设置成 0 电平,没有准备好就将其设置为 1 电平。硬件流控制使用的是专门的信号设置,而软件要完成同样的任务需要发送和接收额外的数据,所以硬件方式比软件方式速度要快得多。

正常情况下一个数据接收或发送信号保持在高电平,直到一个新数据要传送。如果这个信号跳变到低电平并持续一个较长的时钟周期(通常是 1/4 到 1/2 秒),便说明这是一个 Break (停顿) 状态。它可以用来实现收发双方的同步,有时也用作复位或改变通信设备的操作模式,例如调制解调器。

4.6.2 延长串口通信的距离

串口使用一根发送信号线和一根接收信号线来构成共地的传输形式,这种共地传输容易产生共模干扰,抗噪性能弱。RS-232C 最大的传输距离大约是 30m,通信速率一般低于 20Kbps。当然,可以通过降低传输速率的方法来提升传输的距离,但除此之外,还可通过级联信号维持电路来解决这个问题。例如,MAX232 芯片具有两组 232 电平与 TTL 电平转换通路,在传输通路中,信号强度降低后,可先把 232 电平转化成 TTL 电平,再将 TTL 电平回转为 232 电平,这样获得的 232 电平信号又具有了强信号的特性,可保证传输更远的距离。从理论上来说,这种方法可以多次使用,传输距离想要多远就有多远,但这增加了系统开发的成本。

4.6.3 串口的扩展

通用 PC 的 COM 接口有 1 或 2 个,当 PC 机上需要挂载多个串口设备时,这显然不能满足需求,需要串口扩展。同样在以 MCU 为核心的多主机串口通信网络中,也极有可能出现一对多的串行通信方式。

目前比较通用的串口扩展方案有两种。一是用硬件实现,使用多串口单片机或专用串口

扩展芯片，可选的扩展芯片有 TI 公司等研发的 16C554 系列串口扩展芯片，该系列芯片通过并行口扩展串行口，功能比较强大、通讯速度高，但控制复杂，同时价格较高，主要的应用场合是 PC 机串口扩展产品。而多串口单片机也同样存在价格高的缺点。

另一种串口扩展方案就是用软件实现，当然，这种方案需要在硬件连接上使用总线式连接，软件上用广播式传输控制。具体来说，一个串口挂载多个串口设备，PC 机主动发送数据或控制信号到各个终端，终端则被动响应并适时回复。在终端回复的过程中，线路连接上需要考虑各个终端的信息干扰，可以在每个终端到回发线之间的连接一个单向传输的二极管，以解决此问题。软件模拟串口的缺点：一是采样次数低，分摊到每个终端设备的传输时间有限；二是不能实现高波特率通讯，一般不要使用高于 9600bps 的波特率，以保证传输正确性。

第5章 GPIO的应用实例——键盘、LED与LCD

5.1 键盘技术概述

5.1.1 键盘模型及接口

5.1.2 键盘编程的基本问题

5.1.3 键盘构件设计与测试实例

5.2 LED技术概述

5.2.1 扫描法 LED 显示编程原理

5.2.2 LED 构件设计与测试实例

5.3 LCD技术概述

5.3.1 LCD 的特点和分类

5.3.2 点阵字符型液晶显示模块

5.3.3 HD44780

5.3.4 LCD 构件设计与测试实例

可参见芯片手册第 54 章通用输入输出来进行编程。

第6章 定时器相关模块

6.1 计数器/定时器的基本工作原理

6.2 可编程延时模块PDB

6.2.1 PDB 工作原理

6.2.2 PDB 相关寄存器

6.2.3 PDB 构件设计及测试实例

涉及芯片手册 38 章（可编程延时寄存器）

6.3 Flex定时器FTM

6.3.1 FTM 工作原理

6.3.2 FTM 相关寄存器

6.3.3 FTM 中断

6.3.4 FTM 构件设计及测试实例

涉及芯片手册 39 章（Flex 定时器）

6.4 周期中断定时器PIT

6.4.1 PIT 工作原理

6.4.2 PIT 相关寄存器

6.4.3 PIT 构件设计及测试实例

涉及芯片手册 40 章（周期中断定时器）

6.5低功耗定时器LPTMR

6.5.1 LPTMR 工作原理

6.5.2 LPTMR 相关寄存器

6.5.3 LPTMR 构件设计及测试实例

涉及芯片手册 41 章（低功耗定时器）

6.6载波调制发射器CMT

6.6.1 CMT 工作原理

6.6.2 CMT 相关寄存器

6.6.3 CMT 中断

6.6.4 CMT 构件设计及测试实例

涉及芯片手册 42 章（载波调制发射器）

6.7实时时钟RTC

6.7.1 RTC 工作原理

6.7.2 RTC 相关寄存器

6.7.3 RTC 构件设计及测试实例

涉及芯片手册 26 章（RTC 振荡器）、43 章（实时时钟 RTC）

第7章 A/D与D/A

7.1 A/D和D/A转换的基本问题

7.2 A/D转换模块

7.2.1 A/D 转换模块寄存器

7.2.2 A/D 转换构件设计及测试实例

7.3 D/A转换模块

7.3.1 D/A 转换模块寄存器

7.3.2 D/A 转换构件设计及测试实例

7.4 比较器CMP概述

涉及第 34 章（AD 转换器）、35 章（比较器）、36 章（12 位 D/A 转换器）37 章（参考电压）

第8章 SPI

8.1 SPI的基本工作原理

8.1.1 SPI 概述

8.1.2 SPI 的数据传输

8.1.3 SPI 模块的时序

8.2 SPI模块的编程基础

8.2.1 SPI 模块的引脚

8.2.2 SPI 模块的寄存器

8.2.3 SPI 编程基本方法

8.3 SPI构件设计及测试实例

涉及芯片手册第 49 章

第9章 I2C与I2S

9.1 I2C总线概述

9.1.1 I2C 总线特点

9.1.2 I2C 总线标准的发展历史

9.1.3 I2C 总线的相关术语

9.2 I2C总线工作原理

9.2.1 总线上数据的有效性

9.2.2 总线上的信号

9.2.3 总线上数据的传输格式

9.2.4 总线寻址约定

9.3 I2C模块的编程基础

9.3.1 I2C 模块寄存器

9.3.2 I2C 模块编程基本方法

9.4 I2C构件设计及测试实例

9.5 I2S概述

9.5.1 I2S 的特点

9.5.2 I2S 操作模式

9.5.3 I2S 的相关寄存器定义

9.6 I2S的构件化设计与测试实例

涉及芯片手册第 50 章，53 章



第10章 Flash

10.1 Flash内存控制寄存器FMC

10.1.1 FMC 特点

10.1.2 FMC 相关寄存器

10.1.3 FMC 的功能

10.2 Flash存储器概述与编程模式

10.2.1 Flash 存储器编程的基本概念

10.2.2 Flash 存储器的编程寄存器

10.2.3 Flash 存储器的编程过程

10.2.4 Flash 存储器测试实例

10.3 FlexBUS概述

10.3.1 FlexBUS 的特点

10.3.2 FlexBUS 相关的寄存器

10.3.3 FlexBUS 的功能

10.4 EzPort概述

10.4.1 EzPort 的特点

10.4.2 EzPort 信号描述

10.4.3 EzPort 命令

10.5 Flash存储器的保护特性和安全性

10.5.1 周期性冗余检测 CRC

10.5.2 存储器映像密码加速单元 MMCAU

10.5.3 随机数操作 RNGB

涉及芯片手册第 27 章（FLASH 内存控制器）、28 章（Flash 存储器模块）、29 章（Flex 总线）、30 章（EzPort）、31 章（周期性冗余检测）、32 章（存储器映像密码加速单元 MMCAU）、33 章（随机数操作）

第11章 CAN模块FlexCAN

11.1 CAN总线通用知识

11.1.1 CAN 总线协议的历史概况

11.1.2 CAN 硬件系统的典型电路

11.1.3 CAN 总线的有关基本概念

11.1.4 帧结构

11.1.5 位时间

11.2 K60的CAN模块概述与编程结构

11.2.1 CAN 特性

11.2.2 操作模式

11.2.3 CAN 模块的内存映像及寄存器定义

11.2.4 CAN 报文缓冲区

11.3 K60的CAN模块报文发送与接收函数设计

11.3.1 数据帧发送/接收

11.3.2 远程帧发送与接收

11.3.3 仲裁处理、匹配处理及报文缓冲区管理

11.4 K60的CAN模块编程实例

11.4.1 初始化函数设计

11.4.2 K60 的 CAN 模块构件化设计及测试实例

涉及芯片手册 第 48 章

苏州大学

第12章 USB 2.0 编程

12.1 USB基本概念及硬件特性

12.1.1 USB 特性

12.1.2 USB 相关基本概念

12.1.3 USB 的物理特性

12.2 USB的通信协议

12.2.1 USB 基本通信单元：包

12.2.2 USB 通信中的事务处理

12.2.3 从设备的枚举看 USB 数据传输

12.3 K60 的USB模块功能简介

12.3.1 K60 的 USB 模块功能简介

12.3.2 K60 的 USB 模块主要寄存器介绍

12.4 K60 作为USB从机的开发方法

12.4.1 PC 端 USB 设备驱动程序的选择及基本原理

12.4.2 PC 作为 USB 主机的程序设计

12.4.3 K60 作为 USB 从机的程序设计

12.5 K60 作为USB主机的开发方法

12.5.1 K60 作为 USB 主机的基本功能

12.5.2 USB 主机与 USB 设备通信

12.6 K60的USB设备电量检测模块USBDCD

12.6.1 USBDCD 概述

12.6.2 USBDCD 的内存映射与寄存器定义

12.6.3 USBDCD 构件化设计与测试实例

12.7 K60的UAB电压调节器

12.7.1 电压调节器特征

12.7.2 电压调节器操作模式

涉及芯片手册第 45 章（USB 总线 OTG 控制器）、46 章（USB 设备电量检测模块）、47 章（USB 电压调节器）

第13章 大容量SD存储卡SDHC

13.1 SDHC基本概念及硬件特性

13.1.1 SD 概述

13.1.2 SD 相关基本概念

13.1.3 SD 的物理特性

13.2 SD的通信协议

13.2.1 SD 基本通信单元

13.2.2 SD 通信中的事务处理

13.2.3 SD 数据传输

13.3 K60的SD模块基本编程方法

13.3.1 K60 的 SD 模块功能简介

13.3.2 K60 的 SD 模块存储器映像与寄存器定义

13.3.3 K60 的 SD 模块构件化设计与测试实例

涉及芯片手册第 52 章

第14章 TSI

14.1 TSI概述

14.1.1 TSI 特点

14.1.2 TSI 的操作模式

14.1.3 TSI 信号描述

14.2 TSI编程

14.2.1 TSI 的相关寄存器定义

14.2.2 TSI 的功能描述

14.2.3 TSI 的构件化设计与测试实例

涉及芯片手册第 55 章

第15章 基于K60的嵌入式以太网

15.1 嵌入式以太网相关基础知识

15.1.1 以太网的由来与协议模型

15.1.2 以太网中主要物理设备

15.1.3 IEEE 1588 概述

15.1.4 相关名词解释

15.2 K60以太网概述

15.2.1 K60 以太网特性

15.2.2 K60 以太网外部引脚说明

15.2.3 K60 以太网存储映像与寄存器带那个一

15.3 链路层编程

15.3.1 MAC 帧格式

15.3.2 MAC 帧的接收与发送

15.3.3 MAC 帧收发测试实例

15.4 网络层及更高层编程

15.4.1 Ipv4 与 Ipv6 简介

15.4.2 ICMP 简介

15.4.3 UDP 简介

15.4.4 TCP 简介

15.4.5 测试实例

15.5 FIFO

15.5.1 FIFO 概述

15.5.2 FIFO 的接收与发送

15.5.3 FIFO 的保护机制

15.5.4 FIFO 测试实例

15.6 PHY管理接口与以太网接口

15.6.1 MDIO 简介

15.6.2 以太网接口的发送与接收

15.7 K60以太网模块的其他功能

15.7.1 全双工流控制操作

15.7.2 魔术包检测

15.7.3 IP 加速器控制

15.7.4 复位与停止控制

15.7.5 遗留缓冲区描述符

15.7.6 增强缓冲区描述符

涉及芯片手册第 44 章

第16章 系统时钟与其他功能模块

16.1 时钟模块

涉及第3章3.4节及第5章、第24章、25章的内容。

16.2 芯片配置模块

16.2.1 芯片配置模块简介

16.2.2 芯片配置模块寄存器定义

涉及第12章内容

16.3 电源管理模块

16.3.1 电源模式

涉及第7章的内容

16.3.2 低功耗模式

涉及第14章、15章的内容

16.4 端口控制与中断模块

涉及第11章内容

16.5 复位与启动模块

涉及第6章的内容

16.6 杂项控制模块

涉及第16章的内容

16.7 交叉开关模块

涉及第17章内容

16.8 看门狗

涉及第 22 章、23 章的内容

第17章 操作系统的移植

讲述操作系统的基本知识

苏国文