嵌入式系统设计师教程

（第 2 版）

崔西宁 主编 张亮 张淑平 副主编

全国计算机专业技术资格考试办公室 组编

清 华 大 学 出 版 社

北京

内 容 简 介

本书是全国计算机专业技术资格考试办公室组织编写的考试指定用书，内容紧扣《嵌入式系统设计师考试大纲》（2019 年审定通过），对嵌入式系统设计师资格所要求的主要知识及应用技术进行了阐述。

全书共 11 章，主要内容包括计算机系统基础知识，嵌入式系统硬件基础知识，嵌入式硬件设计，嵌入式系统软件基础知识，嵌入式系统设计与开发，嵌入式程序设计，嵌入式系统的项目开发与维护知识，嵌入式系统软件测试，嵌入式系统安全性基础知识，标准化、信息化与知识产权基础知识，嵌入式系统设计案例分析。

本书内容涉及知识广泛，结构清晰、合理，既可作为全国计算机技术与软件专业技术资格（水平）考试中的嵌入式系统设计师级别的考试用书，也可作为高等院校嵌入式系统相关课程的教材。

图书在版编目（CIP）数据

嵌入式系统设计师教程 / 崔西宁主编. —2 版. —北京：清华大学出版社，2019
全国计算机技术与软件专业技术资格（水平）考试指定用书
ISBN 978-7-302-53697-0

Ⅰ. ①嵌… Ⅱ. ①崔… Ⅲ. ①微型计算机－系统设计－资格考试－教材 Ⅳ. ①TP360.21

中国版本图书馆 CIP 数据核字（2019）第 178368 号

责任编辑：杨如林
封面设计：何凤霞
责任校对：徐俊伟
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn, http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　邮　　编：100084
社 总 机：010-62770175　　邮　　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：
经　　销：全国新华书店
开　　本：185mm×230mm　　**印　张**：37.75　　**防伪页**：1　　**字　数**：828 千字
版　　次：2006 年 8 月第 1 版　2019 年 12 月第 2 版　　**印　次**：2019 年 12 月第 1 次印刷
定　　价：118.00 元

产品编号：084060-01

前　言

全国计算机技术与软件专业技术资格（水平）考试实施至今已经历了二十多年，在社会上产生了很大的影响，对我国软件产业的形成和发展做出了重要的贡献。为了适应我国计算机信息技术发展的需求，人力资源和社会保障部、工业和信息化部决定将考试的级别拓展到计算机信息技术行业的各个方面，以满足社会上对各种计算机信息技术人才的需要。

编者受全国计算机专业技术资格考试办公室的委托，对《嵌入式系统设计师教程》进行改写，以适应新的考试大纲要求。在考试大纲中，要求考生掌握的知识面很广，每个章节的内容都能构成相关领域的一门甚至多门课程，因此编写的难度很高。考虑参加考试的人员已有一定的基础，所以本书中只对考试大纲中所涉及的知识领域的要点加以阐述，但限于篇幅，不能详细地展开，请读者谅解。

全书共 11 章，各章内容安排如下。

第 1 章 计算机系统基础知识，概要介绍嵌入式系统，对计算机系统常用进位计数制、数据的表示和运算、计算机系统硬件基本组成和体系结构以及可靠性与系统性能评测等基础知识进行了简要介绍。

第 2 章 嵌入式系统硬件基础知识，主要介绍嵌入式系统所涉及的硬件知识，重点介绍嵌入式微处理器、嵌入式存储体系、嵌入式系统的输入输出接口、嵌入式系统通信接口等方面的硬件接口基础知识。

第 3 章 嵌入式硬件设计，主要介绍嵌入式硬件设计过程中所涉及的基础知识，包括嵌入式系统电源分类、电源管理和电子电路设计中的 PCB 设计、电子电路测试基础知识。

第 4 章 嵌入式系统软件基础知识，主要介绍嵌入式系统软件相关基础知识，包括嵌入式软件基础知识、嵌入式操作系统、嵌入式文件系统、嵌入式数据库等。

第 5 章 嵌入式系统设计与开发，主要介绍嵌入式软件开发基础知识、嵌入式软件开发环境、嵌入式软件开发过程、嵌入式软件移植等。

第 6 章 嵌入式程序设计，主要介绍程序语言及其翻译基础知识以及汇编语言、C 和 C++编程基础知识。

第 7 章 嵌入式系统的项目开发与维护知识，主要介绍嵌入式系统开发与维护的相关基础知识，主要包括系统开发过程与过程模型、项目管理、系统质量、开发工具与开

发环境、系统分析、系统设计、系统实施、系统运行与维护等相关知识。

第 8 章 嵌入式系统软件测试，主要介绍嵌入式软件测试的相关内容，包括软件测试概述、测试过程、测试方法、测试类型、测试工具、测试环境、软件测试实践等。

第 9 章 嵌入式系统安全性基础知识，主要介绍安全性基础知识，包括计算机信息系统安全概述、信息安全基础、安全威胁防范、嵌入式系统安全方案等内容。

第 10 章 标准化、信息化与知识产权基础知识，主要介绍标准化基础知识、信息化基础知识和知识产权基础知识。

第 11 章 嵌入式系统设计案例分析，主要通过案例分析介绍嵌入式系统的整体设计方法和典型嵌入式硬件设计中所涉及的软硬件协同设计、程序设计等内容。

本书第 1 章由张淑平和朱光明编写，第 2 章、第 3 章由张亮和朱光明编写，第 4 章、第 5 章由崔西宁、谌卫军和韩炜编写，第 6 章由张淑平和刘伟编写，第 7 章由霍秋艳编写，第 8 章由周敏刚编写，第 9 章由严体华编写，第 10 章由刘强和王亚平编写，第 11 章由崔西宁、张亮和戴小氐编写，最后由崔西宁、张淑平、张亮统稿。

在本书的编写过程中，参考了许多相关的书籍和资料，编者在此对这些参考文献的作者表示感谢。同时感谢清华大学出版社在本书出版过程中所给予的支持和帮助。

因水平有限，书中难免存在错漏和不妥之处，望读者指正，以利改进和提高。

编 者

2019 年 10 月

目　录

第 3 章　嵌入式硬件设计

本章主要介绍嵌入式硬件设计过程中所涉及的基础知识，包括嵌入式系统电源分类、电源管理和电子电路设计中的 PCB 设计、电子电路测试基础知识。

3.1　嵌入式系统电源管理

嵌入式电源系统是集成在嵌入式系统中，为嵌入式设备提供直流基础电能的电源设备，是一种安全、可靠、高性能的供电系统。一般来说，嵌入式电源的输入都为交流市电，输出是常见直流 12V、5V、3.3V，是一类二次电源设备。

交流电源是嵌入式系统较为重要的电能来源之一。嵌入式系统的电能由该类电源直接或者间接提供。通常使用市电作为输入，通过一系列变化、转化操作将交流高压电转变为低压直流电。

电池是许多嵌入式系统直接供电的电源，诸如手机、传感器，都会使用电池供电。电池的供电设备往往是功耗相对较小，而连续工作时间较长的设备，因此嵌入式系统的功耗有着较为严格的要求，在不同的应用场景需求下可能会增加电池的容量。

稳压器则是常见配合交流电源与电池使用的一种元器件。由于嵌入式系统中往往需要多种电压，因此在嵌入式系统中会使用稳压器将电压降至所需范围。

1．电源管理

嵌入式系统的一个典型的硬性需求是降低功耗，许多嵌入式设备往往使用电池供电，并且常年无人看管，因此功耗问题非常重要。而在电池容量有限或者设备数量较大的时候，系统的功耗就变得至关重要。

首先绝大多数嵌入式系统都会包含基础电源管理功能以降低功耗。

（1）系统上电行为。嵌入式系统的组件往往在系统正常启动之后才能进入低功耗模式，因此在上电的时候通常会以较高的功率来运行。而上电期间很多设备并不需要工作，因此在上电启动的时候需要有效管理这些设备以减小功耗。

（2）空闲模式。CMOS 电路有效的功耗是在电路时钟工作的时候产生的，因此可以通过关闭不需要的时钟来降低功耗。而现代嵌入式系统所使用的元器件往往都提供了通过外部事件唤醒的功能，因此在不使用某些模块的期间内，可以通过主处理器向相关元器件发送“睡眠”指令，以指示其进入低功耗状态。当需要重新触发器件进入工作时，通过特定的触发事件进行元器件唤醒。

（3）断电。由于逆向偏压泄露，电路元器件在低功耗模式下依然会损耗电能，因此

对于低功耗模式消耗电能较大或者长期不使用的元器件，可以做断电处理以减少功耗。

（4）电压与频率缩放。有效功率与切换频率成线性比例，但与电源电压平方成正比。经常以较低的频率运行于全时钟频率，然后转入闲置，并不能节约很多功率。在此种情况下，可以通过降低电压来节约功率。

例如，某嵌入式系统数字电路部分需要支流电源供电，输入电压为 220V 交流电，电源管理模块首先采用的开关电源将 220V 的交流电转换为直流电压，再利用低压线性稳压器为各个子模块供电，对应的实现框图如图 3-1 所示。

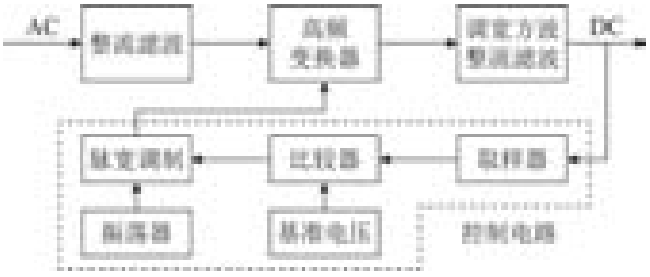

图 3-1　某电路电源模块框图

在电源产生电路中，为了避免模拟信号与数字信号地之间的相互干扰，将输入的 220V 交流电压转换为两个独立的直流电源，再分别为模拟电路和数字电路的电源供电。例如该项目设计中需要 12V、24V、5V、8V、−8V、3.3V 等不同电压，对应的电源管理系统拓扑结构如图 3-2 所示。

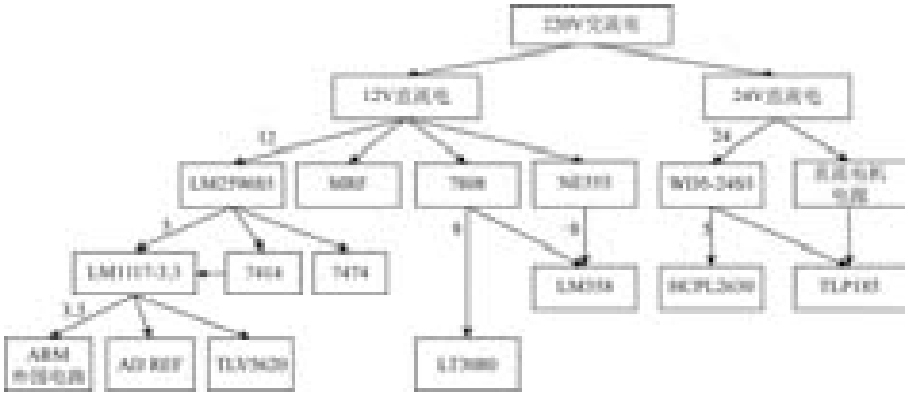

图 3-2　某电源管理系统拓扑图

具体实现如下：

① +12V 转+8V 采用的是 LM7808，这是一款三端集成的稳压电路，能够准确的降压到+8V，输入要保证为 12V 直流电源，保证输入比输出稳压值 8V 高出一定压差，即可实现 8V 稳压，设计时需要注意电流不要超载。在具体设计时，电路两端的电容作用都为

滤波，用来平滑电压与提高抗干扰能力。其中输出端可并联 220μF/25V 的电解电容，其自谐频率小，能够起到储能滤波的功能，消除低频干扰。但是由于大电容的电解电容自身存在一定的电感，对于高频信号以及脉冲干扰信号无法有效滤除，因此，设计中一般会并联一个或几个容值比较小的陶瓷电容，以达到滤除高频干扰信号的作用，对应的设计如图 3-3 所示。

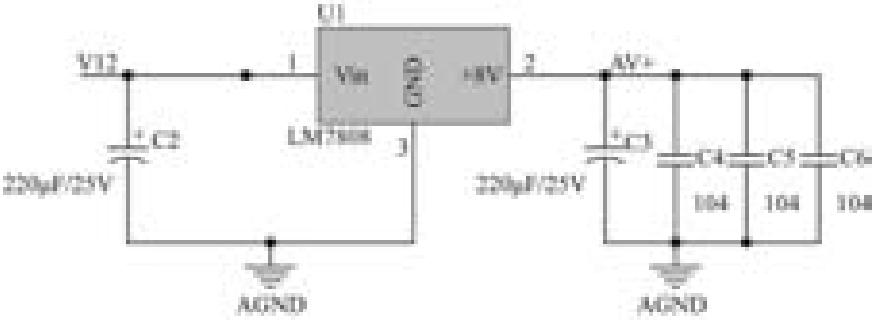

图 3-3　12V 转 8V 电路示意图

② +12V 转−8V 采用 NE555 芯片，这是一款将模拟功能和逻辑功能很好地结合在一起的芯片，该款芯片为 8 脚集成电路，大约在 1971 年由 Signetics 公司发布，在当时是唯一非常快速且商业化的芯片，在之后的 40 余年中被普遍使用，且延伸出许多的应用电路。后来则是基于 CMOS 技术版本的芯片（如 Motorola 的 MC1455）被大量使用，但原规格的 NE555 依然正常供应，尽管新版 IC 在功能上有部分改善，但其脚位功能并没变化，所以到目前都可直接的代用应用的范围十分广泛，其实现的典型电源转换电路如图 3-4 所示。

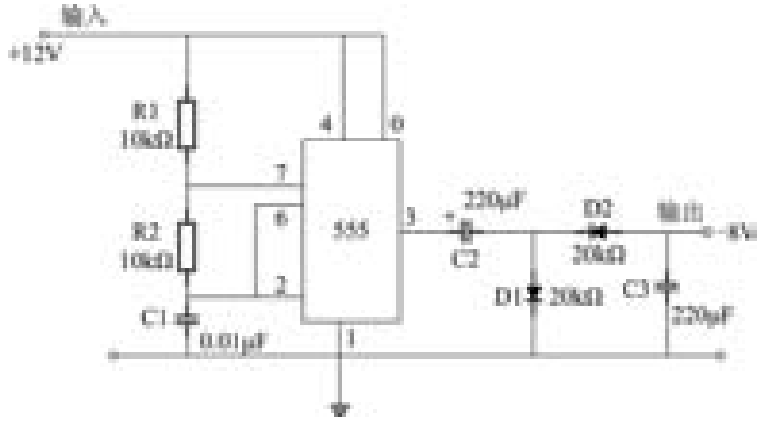

图 3-4　12V 转−8V 电路示意图

在其设计中，当 NE555 的第三脚输出高电平，通过 D1 向 C1 充电，电压可达 11V。当 NE555 输出为低电平时，D1 被 C2 反偏截止。C2 向 C3 转移电荷，重复多次后 C3 电压达 8V，相对地线则输出视为−8V。

③ +12V 转+5V 采用的是开关型集成稳压芯片 LM2596，它内含固定频率振荡器以

及基准稳压器，并具备完善的保护电路、热关断电路、电流限制等。LM2596 是降压型电源管理单片集成电路的开关电压调节器，能够输出 3A 的驱动电流，同时具有很好的线性和负载调节特性。固定输出版本有 3.3V、5V、12V，可调版本可以输出小于 37V 的各种电压。使用 LM2596 进行+12V 转+5V 的典型电路图如图 3-5 所示。

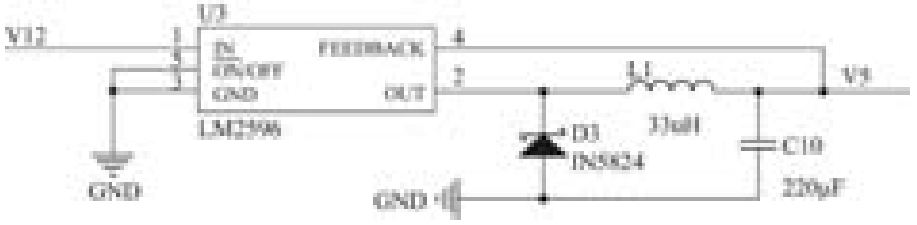

图 3-5　12V 转 5V 电路示意图

④ +5V 转+3.3V 采用 LM1117-3.3，这也是一款低压差线性稳压器，输入电压只要在允许范围内，它的输出电压都可以稳定在一个电压，使用 LM1117-3.3 来进行+5V 转+3.3V 的电路如图 3-6 所示。

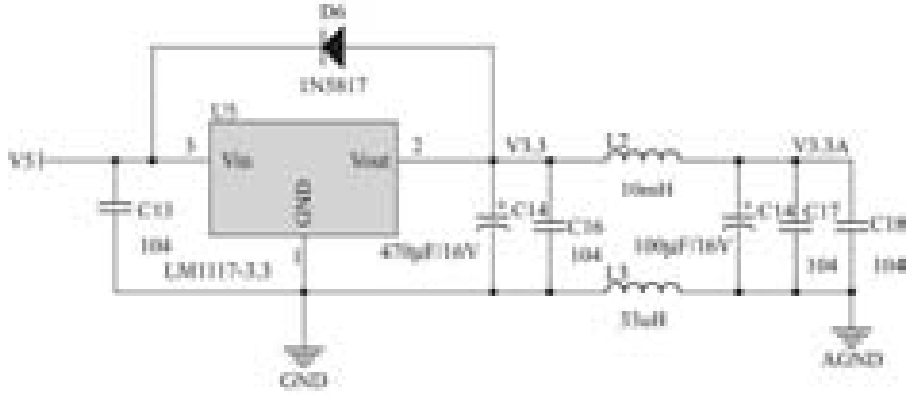

图 3-6　5V 转 3.3V 电路示意图

⑤ +24V 转+5V 直接采用 WD5-24S5，DC-DC 电源模块 WD5 系列具有 5W 输出功率、宽电压输入、输入/输出隔离、小型化封装等特性。

3.2　电子电路设计

3.2.1　电子电路设计基础知识

1．电子电路设计原理

电路设计主要分三个步骤：设计电路原理图、生成网络表、设计印制电路板。在典型的电子电路设计中，其基本步骤如下。

（1）充分了解设计任务的具体要求，如性能指标、内容及要求，明确设计任务。

（2）方案选择：根据掌握的知识和资料，针对设计提出的任务、要求和条件，设计合理、可靠、经济、可行的设计框架，对其优缺点进行分析。

（3）根据设计框架进行电路单元设计，具体设计时可以模仿成熟的电路进行改进和创新，需要特别注意信号之间的关系和限制。

（4）根据电路工作原理和分析方法，进行参数的估计与计算。

（5）元器件选择时，元器件的工作、电压、频率和功耗等参数应满足电路指标要求，元器件的极限参数必须留有足够的裕量，一般应大于额定值的 1.5 倍，电阻和电容的参数应选择计算值附近的标称值。

（6）电路原理图的绘制，电路原理图是组装、焊接、调试和检修的依据，绘制电路图时布局必须合理、排列均匀、清晰、便于看图、有利于读图：信号的流向一般从输入端或信号源画起，由左至右或由上至下按信号的流向依次画出单元电路，反馈通路的信号流向则与此相反；图形符号和标准，并加适当的标注；连线应为直线，并且交叉和折弯应最少，互相连通的交叉处用圆点表示，地线用接地符号表示。

2．电子电路设计方法及步骤

电子电路设计的第一步是电路原理图设计，设计电子电路是后续步骤的基石。电子电路设计的过程如图 3-7 所示。

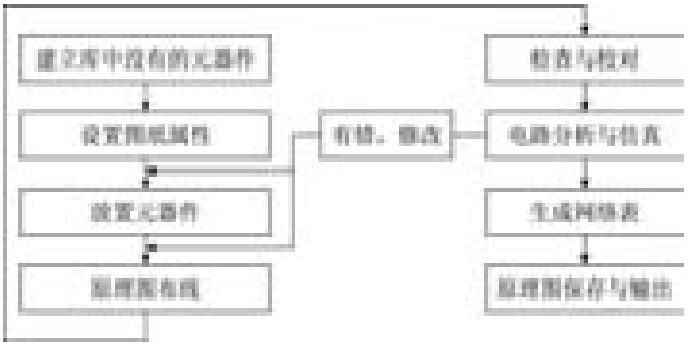

图 3-7　原理图设计流程图

在原理图设计过程中，首先是建立元器件库，其次是元器件布局和布线连接，然后需要进行电路分析与仿真，进而生成网表，最终得到设计完整的原理图。在整个设计过程中，需要不断的检查与校对，以保证各个环节的正确性。

1）建立元器件库中没有的元器件

一般使用的 CAD 软件都会预置一些常用的电路元器件。但是这些元器件并不一定会满足电路原理图的设计需求，而元器件厂家也不会提供元器件库。因此要想设计原理图，第一步是使用 CAD 软件，对有关元器件建立元器件库，同时对元器件库中已有但是不满足要求的元器件进行修改。

一般来说，采用片上系统的设计与传统方式下采用逻辑关系的设计方法并不相同。

建立元器件的原理图时需要基于实际的元器件，参考元器件的数据手册建立。原理图中的标识要简明清晰，同时保证逻辑上的电气特性与实际数据手册所描述的元器件相符，在进行元器件建库时需要注意以下标准：

- 元器件引脚序号与封装库相应元器件应交序号应当保持一一对应；
- 分立元器件要注意元器件的标号与引脚的对应关系，例如多组绕组电感要注意主次绕阻和同名端的标示及引脚序号对应关系；
- 二脚有极性，如二极管，默认以 1 表示正极，2 表示负极；
- 多脚元器件如晶体管、芯片等，引脚序号应该与封装的引脚序号保持对应关系，且芯片引脚的序号为逆时针逻辑；
- 通常元器件的引线引脚长度为 5 个单位。

2）CAD 设置

根据实际的电路及其复杂程度选择对应的元器件库，设置 CAD 图示相关属性，配置 CAD 中设计规则，并建立有关工程。

3）放置元器件

根据设计电路图的需求，将所使用的元器件有选择地放置在合适的位置，并进行修改。同时利用 CAD 自动编号功能为元器件编号，并选择实际印刷电路所使用的封装。

原理图的视图要整体清晰，元器件的放置会影响到原理图整体的美观性和可读性。设计合理的原理图会使后续工作的难度与复杂度降低，同时可维护性与可读性增强。

元器件的放置需要依照主信号流向的方向和规律安排，功能类似或者接近的电路元器件应当摆放在一起，并且符合原理图设计规范。原理图中使用的元器件、表示等要采用国际标准的符号。对于一些特殊情况可以使用非国际标准的符号、标示，但是需要在恰当的位置标注其中含义。

摆放元器件的时候，如果元器件无法在一张原理图中放置，则需要酌情将原理图分割成多张原理图或者采用“子-母”原理图的方式将原理图进行分割。在这个过程中，所有操作都需要确保原理图的逻辑正确性。

如果将完成功能相近的元器件摆放在一起时，元器件的方向要一致，字符位号要保持与对应元器件最近距离，整齐划一，方向一致，以达到读图时美观、拓扑结构清晰、电气逻辑规范的效果。

此外，摆放的元器件要放置在原理图的标准模板框中。统一元器件不能使用不同的符号表示出现在原理图上。有极性的元器件应标识正确、清楚、易识别电感的同名端要标识正确、清楚，同一原理图上的电感的同名端标识要统一。

4）原理图连接

根据原理图需要，将原理图上的各个元器件按照需求设计将对应引脚通过合适的方式进行连接，即可形成完整的原理图。

通常主功率路线及大电流连线需要加粗表示，其他的信号线则使用细线表示。

对于原理图中有电气连接的导线是不可以弯曲的，应当以垂直或者平行的线进行表示，尽量减小大幅度的跨接。没有逻辑连接的不可以有电气结点。

5）检查校对

根据系统需求与电路功能对所设计的电路图进行校验，保证原理图符合电器规则，同时布局较为清晰、简明、美观。对元器件、导线位置、连接等进行检查修改。

6）电路分析与仿真

利用 CAD 软件提供的分析仿真功能或者使用专用行业软件对检录进行分析，分析之后对电路进行仿真，检查电路是否符合需求设计与相关设计指标。

7）生成网络表

使用 CAD 软件生成原理图的网络表。电路会以结点、元器件和连线组成的网络表示。PCB 设计中，布线或者自动布线会依赖这些数据。对于 CAD 软件，网络表是原理图与印刷电路中间的接口。

当人工创建网络表时，应根据原理图设计工具的特性，结合原理图设计一同排除错误，保证网络表的正确性和完整性。

8）保存与输出

将设计的电路所在工程存储并提交至版本控制系统中，等待下一步操作或者审阅人员审阅，审阅完成后才可进行输出。

此外对于电路的原理图来说，还需要及时填写标准模板框中的相关信息，包括：所适用的产品型号、版本号、修改记录、绘制者、修改者、审核者批准者、日期等关键信息。

3．电子电路可靠性设计

电子设备的可靠性设计可以保证在绝大部分情况下电子设备能够稳定可靠地工作，同时在发生故障时可以将损失降到最低。在电子电路可靠性设计中，涉及可靠性定义、故障衡量、可靠性成本、可靠性设计和设计故障等概念。

1）可靠性定义

可靠性的严格定义如下：“在规定的时间和环境条件下系统无故障运行的概率”。这个概率受到三个控制量的影响：

（1）故障的规定：许多系统在运行中可能出现不同级别的故障，有些故障可能导致整个嵌入式系统物理上的损毁，有些则可能对系统根本没有影响。

（2）工作寿命：嵌入式系统不可能永远运行，不同使用年限的电子器件、设备发生故障的概率也不同。

（3）实际环境：温度、适度、腐蚀性气液体、灰尘、震动、冲击、电源、磁场、各类辐射射线等对于设备的正常工作都会有一定的影响。实际环境的限制最终对于可靠性的评价有着实际意义的限制。

2）故障衡量

对于大多数电子设备来说，故障率是一个常数。一般来说，嵌入式设备的故障率会在设备运行初期较高，然后随着易损元器件被非易损替换，故障率会逐步下降。随着设备运行，并逐渐接近使用寿命，元器件会开始损耗，同时腐蚀率会升高，从而导致故障率会再次升高。所以通常会用某一方式衡量可靠性。

在确定时间内的故障率的倒数就是通常所指的平均故障间隔时间（Mean Time Between Failures，MTBF）。一般用小时表示，而故障率使用每个小时故障的次数进行表示。MTBF 通常与运行周期无关，可以方便地表示可靠性。

MTBF 通常描述可以修复的设备的可靠性，而对不可以修复的设备，则无法使用该指标衡量。因此对于不可修复设备，一般使用平均失效时间（Mean Time To Failure，MTTF）来表示可靠性。一般的工厂生产该类设备时，会使用抽样调查的方式进行寿命测试，以此来估算 MTTF。

对于可靠性，还有一个评价指标是可用性，表示系统工作的总时长中，正常可用的时间所占的比例，即一个设备正常服务的时间与正常和故障总时间的比值。通常可表示为 U/(U+D)，其中 U 表示正常运行的时间，D 表示故障的时间。

3）可靠性成本

嵌入式系统可靠性的提高需要一个团队人力、物力的大量投入。总的来说，投入的金钱与人月会随着可靠性的提升而先降低再提高，而维护成本则是先提升再降低。通过建立数学模型可以确定的是，将大量的资源投入提高很少的可靠性是不值当的。

4）可靠性设计

嵌入式系统硬件相关的可靠性设计往往是为“在有限的资源下尽可能提高可靠性”。因此可靠性设计通常需要考虑如下因素，以平衡不同因素对可靠性的影响：

- 有效的散热，降低高温对系统的危害；
- 尽量减少高敏感元器件的使用；
- 更多的使用可靠度高、质量好的元器件；
- 指定采用屏蔽性好或者内嵌的测试方法；
- 使用最少的元器件设计出来简单的电路；
- 在电子元器件级别进行冗余。

温度是影响所有电子元器件的重要因素之一，而所有元器件都会产生热量。过高的温度会对元器件造成不可逆转的损伤，并阻碍电流流动。而且高温也是元器件损害的最主要的原因。同时过低温也会损坏电子设备。一般来说设备需要工作在所设计的环境中，不同级别的设备会对不同环境的耐受级别不同，因此根据不同用途要选择合适的设备，同时使用适当的散热或者保温措施。

重要的是，元器件工作在标称额定值（环境）以内对电子设备的可靠性会有较大的提升。对于电容、电阻等元器件和各类芯片，都会对电压、电阻、功率、频率等有着严

格的规定。保证电子元器件、电路工作在合理的环境中可以有效的保护元器件、降低故障发生的可能，提高可用性。此外选用高可靠的元器件也可以有效地提高可靠性。在选用可靠的元器件并对环境做出保证后，还应当进行筛选和老化实验保证元器件的一部分不合格元器件筛除。

根据概率论的相关知识，若假设所有元器件出错的概率为 p，而 n 个元器件中任意元器件出错都会导致系统崩溃，则整个系统出错的概率为 $1-(1-p)^n$。当 n 增加时，出错的概率会以指数形式增长。因此，降低元器件个数、简化设计可以有效地降低故障发生的概率从而提高可靠性。当一个部件的故障率为 p 而同时有 n 个冗余部件时，其整体故障的概率为 p^n。可以看出，当冗余元器件增多的时候，整体的故障概率会成指数形式降低。因此，有效的冗余设计可以保证系统的可靠性提高。

5）设计故障

正如“设计故障”字面意思所揭示的，许多故障是由设计者人为设计的，一个最极端的例子将电源两端使用电阻连接，但是使用的是 0 欧姆电阻。所以可靠性设计中对于经验的依赖十分重要，由于设计者本身的经验缺乏或者其他问题造成的系统可靠性降低是不容易解决的，但是又不容易避免。因此，设计审查是必不可少的环节，“设计故障”在实际的生产过程中应该极力避免。

3.2.2 PCB 设计基础知识

1．PCB 设计原理

在原理图设计完成并生成网络之后，就可以着手设计印刷电路板了，也就是常说的 PCB（Printed Circuit Board）。现在所有电子设备都离不开 PCB，PCB 承载着形形色色的电子元器件，作为电子系统的基石。PCB 的出现与发展使得电子产品生产可以更加工业化，同时伴随着工业化使得 PCB 的生产更加标准化、规模化、自动化。此外 PCB 技术的发展还使得电子电路与电子产品的体积不断缩小，从而降低成本，同时可靠性与稳定性还能够得以提高，并且使得装配与维修变得十分简单。

PCB 是由印刷电路、基板、元器件组合而成的。下面简要介绍一些 PCB 相关的基础知识。

- PCB 印刷：PCB 印刷是按照设计将电路印刷到基板上，然后重复多次得到多层 PCB，最后添加过孔、阻焊层等；
- PCB 由基板、铜层、阻焊层、字符层等组成；
- 印制线路是指采用诸如刻蚀之类的方法的印制电路，包括导线和焊盘；
- 印制元器件是指通过丝印等手段将元器件符号等文字印刷至电路上的描述；
- PCB 贴片是指使用专用贴片机自动贴片或者使用钢网手工贴片，然后通过各类加热方式或者回流焊接方式将元器件焊接的过程；
- 电镀：通常会使用锡或者金对暴露的焊盘等进行电镀处理。

对于 PCB，有许多分类方式。按照 PCB 的层数，一般可分为单面板、双面板和多层板。按照机械性能来区分，可以分为刚性板和柔性版。按照基板材质可以分为纸基板、玻璃基板、复合材料基板和特征材料基板。目前主流的 PCB 多为树脂刚性基板。

2．PCB 设计方法及步骤

PCB 设计的主要任务是根据电路原理图对 PCB 进行合理的结构与布线布局设计，典型过程如图 3-8 所示，其主要过程是依据网表中的设计进行布局、布线连接，并通过 PCB 仿真来判断设计是否正确，最终得到 PCB 设计输出。

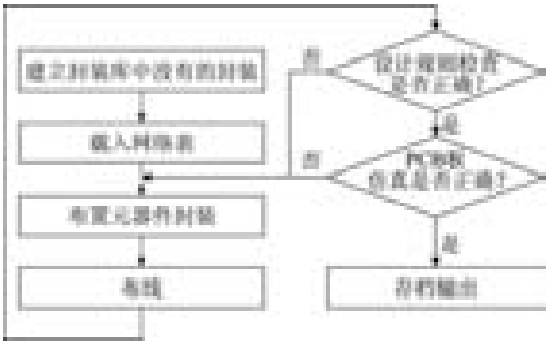

图 3-8　PCB 设计流程图

1）建立封装库中没有的元器件

通常的 CAD 只有一些常见、常用元器件的封装，但是设计 PCB 时，很多元器件并没有对应的封装。因此需要使用 CAD 补全缺失的封装。

2）规划电路板

在封装库准备好之后，设计 PCB 的第一步骤是规划电路板。规划包括如下内容：设置习惯性的环境参数与文档参数，例如选择层面、外形尺标大小等。

首先需要根据 PCB 的结构与设计确定 PCB 的尺寸，同时创建 PCB 的设计文件。然后确定 PCB 设计的坐标原点。PCB 板通常需要将板框的四周进行倒圆角的操作，一般的倒角半径是 5mm。

根据结构图设置板框尺寸，按结构要素布置安装孔、接插件等需要定位的元器件，并给这些元器件赋予不可移动属性。按工艺设计规范的要求进行尺寸标注。根据结构图和生产加工时所需的夹持边设置印制板的禁止布线区、禁止布局区域。根据某些元器件的特殊要求，设置禁止布线区。

3）载入网络和元器件封装

载入之前电路原理设计得到的网络表和有关元器件的封装，并将元器件的摆放到预定位置。

4）布置元器件封装

采用 CAD 自动布置或者手动布置元器件封装的位置。将元器件放置到恰当的方便布

线的位置，同时还能满足整齐美观的效果。

3．PCB 布局要求

通常 PCB 元器件的布局遵照“先大后小，先难后易”的布置原则，即重要的单元电路、核心元器件应当优先布局。布局中应参考原理框图，根据单板的主信号流向规律安排主要元器件。

布局应尽量满足以下要求：总的连线尽可能短，关键信号线最短；高电压、大电流信号与小电流、低电压的弱信号完全分开；模拟信号与数字信号分开；高频信号与低频信号分开；高频元器件的间隔要充分。

相同结构电路部分，尽可能采用“对称式”标准布局，按照均匀分布、重心平衡、版面美观的标准优化布局。器件布局栅格的设置，一般 IC 元器件布局时，栅格应为 50～100 mil，小型表面安装元器件，如表面贴装元器件布局时，栅格设置应不少于 25mil。

PCB 的整体布局应按照信号流程安排各个功能电路单元的位置，使整体布局便于信号流通，而且使信号保持一致的方向，各功能单元电路的布局应以主要元器件为中心，在实际布局中应围绕这个中心进行布局。通常来说元器件布局有如下要求：

- 元器件的摆放不重叠；
- 元器件的摆放不影响其他元器件的插拔和贴焊；
- 元器件的摆放符合限高要求，不会影响其他元器件、外壳的贴焊及安装，如电解电容由立放改为卧放，从而满足高度要求；
- 元器件离板边的距离符合工艺要求，距离不够时加工艺附边，附边上没定位孔时的宽度为 3mm，有定位孔时的宽度为 5mm；
- 有极性元器件的摆放方向要尽可能一致，同一板上最多允许两种朝向；
- 安装孔的禁布区内无元器件和走线（不包括安装孔自身的走线和铜箔）。

1）对于采用通孔回流焊的元器件布局要求

- 对于非传送边尺寸大于 300mm 的 PCB，较重的元器件尽量不要布置在 PCB 的中间，以减轻由于插装元器件的重量在焊接过程对 PCB 变形的影响，以及插装过程对板上已经贴放的元器件的影响；
- 为方便插装，推荐将元器件布置在靠近插装操作侧的位置；
- 对于尺寸较长的元器件（如内存条插座等），其长度方向推荐与传送方向一致；
- 通孔回流焊元器件的焊盘边缘与连接器及所有的 BGA 的丝印之间的距离大于 10mm，与其他表面贴装元器件间距离大于 2mm；
- 通孔回流焊元器件本体间距离大于 10mm，有夹具扶持的插针焊接不做要求。

2）对于插件元器件的布局要求

对于插件元器件的布局，通常要求端子的尺寸、位置要符合结构设计的要求，并达到最佳结构安装。此外过波峰焊的插件元器件焊盘间距大于 1.0mm，为保证过波峰焊时不连锡，过波峰焊的插件元器件焊盘边缘间距应大于 1.0mm（包括元器件本身引脚的焊

盘边缘间距)；优选插件元器件引脚间距大于 2.0mm，焊盘边缘间距大于 1.0mm；在元器件本体不相互干涉的前提下，相邻元器件焊盘边缘间距满足如图 3-9 所示。

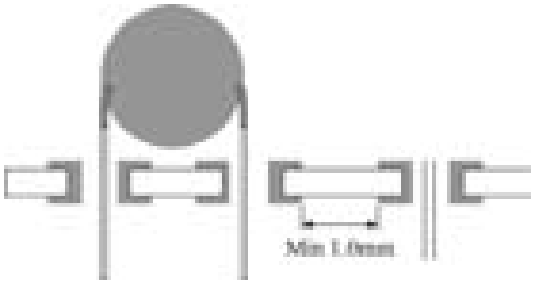

图 3-9　焊盘边缘间距示意图

3）焊盘要求

当插件元器件引脚较多，以焊盘排列方向平行于进板方向布置元器件时，当相邻焊盘边缘间距为 0.6～1.0mm 时，推荐采用椭圆形焊盘或加偷锡焊盘，如图 3-10 所示。

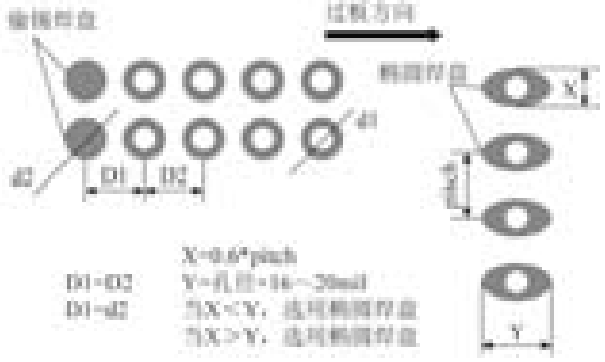

图 3-10　焊盘排列方向平行于进板方向布局时焊盘推荐示意图

可调元器件、可插拔元器件周围应该留有足够的空间供调试和维修，在实际设计中应根据系统或模块的 PCBA 安装布局以及可调元器件的调测方式来综合考虑可调元器件的排布方向、调测空间，可插拔元器件周围空间预留应根据邻近元器件的高度决定。

所有的插装磁性元器件一定要有坚固的底座，禁止使用无底座插装电感。有极性的变压器的引脚尽量不要设计成对称形式，要考虑防呆工艺，以免插件时机械性出错。裸跳线不能贴板跨越板上的导线或铜皮，以避免和板上的铜皮短路，绿油不能作为有效的绝缘。

电缆的焊接端尽量靠近 PCB 的边缘布置以便插装和焊接，否则 PCB 上别的元器件会阻碍电缆的插装焊接或被电缆碰歪。多个引脚在同一直线上的元器件，像连接器、DIP 封装元器件、TO-220 封装元器件，布局时应使其轴线和波峰焊方向平行。较轻的元器件如二极管和 1/4W 电阻等，布局时应使其轴线和波峰焊方向垂直。这样能防止过波峰焊时

因一端先焊接凝固而使元器件产生浮高现象。电缆和周围元器件之间要留有一定的空间，否则电缆的折弯部分会压迫并损坏周围元器件及其焊点。

4）贴片元器件的布局要求

对于贴片元器件而言，一般有着如下的要求。

两面过回流焊的 PCB 的 BOTTOMLAYER 面要求无大体积、太重的表贴元器件，需两面都过回流焊的 PCB，第一次回流焊接元器件重量限制如表 3-1 所示。

表 3-1　回流焊接元器件重量限制表

片式元器件：A≤0.075g/mm^2	A=元器件重量/引脚与焊盘接触面积
翼形引脚元器件：A≤0.300g/mm^2	
J 形引脚元器件：A≤0.200g/mm^2	
面阵列元器件：A≤0.100g/mm^2	

若有超重的元器件必须布在底层面上，并应通过试验验证可行性。焊接面元器件高度不能超过 2.5mm，若超过此值，应把超高元器件列表通知装备工程师，以便特殊处理。

需波峰焊加工的单板背面元器件不形成阴影效应的安全距离应考虑波峰焊工艺的贴片元器件距离，相同类型元器件布局如图 3-11 所示。

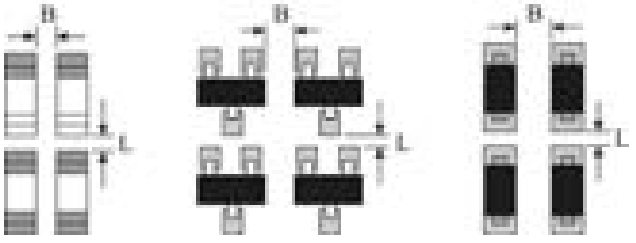

图 3-11　相同类型元器件的布局示意图

相同类型元器件的封装尺寸与距离关系如表 3-2 所示。

表 3-2　相同类型元器件的封装尺寸与距离关系

元　器　件	焊盘间距 L（mm/mil）		元器件本体间距 B（mm/mil）	
	最小间距	推荐间距	最小间距	推荐间距
0603	0.76/30	1.27/50	0.76/30	1.27/50
0805	0.89/35	1.27/50	0.89/35	1.27/50
1206	1.02/40	1.27/50	1.02/40	1.27/50
大于等于 1206	1.02/40	1.27/50	1.02/40	1.27/50
SOT 封装	1.02/40	1.27/50	1.02/40	1.27/50
钽电容 3216、3528	1.02/40	1.27/50	1.02/40	1.27/50
钽电容 6032、7343	1.27/50	1.52/60	2.03/80	2.54/100
SOP	1.27/50	1.52/60	—	—

不同类型元器件在布局时的距离示意图如图 3-12 所示。

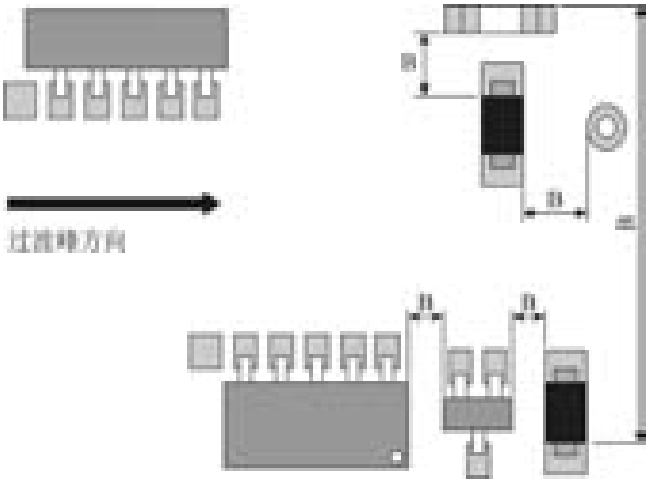

图 3-12　不同类型元器件的距离示意图

不同类型元器件的封装尺寸与距离关系见表 3-3（单位：mm）。

表 3-3　不同类型元器件的封装尺寸与距离关系示意表

元器件	0603	0805	1206	≥1206	SOT 封装	钽电容 3216、3528	钽电容 6032、7343	SOIC	通孔
0603		1.27	1.27	1.27	1.52	1.52	2.54	2.54	1.27
0805	1.27		1.27	1.27	1.52	1.52	2.54	2.54	1.27
1206	1.27	1.27		1.52	1.52	2.54	2.54	1.27	1.27
≥1206	1.27	1.27	1.27		1.52	1.52	2.54	2.54	1.27
SOT 封装	1.52	1.52	1.52	1.52		1.52	2.54	2.54	1.27
钽电容 3216、3528	1.52	1.52	1.52	1.52	1.52		2.54	2.54	1.27
钽电容 6032、7343	2.54	2.54	2.54	2.54	2.54	2.54		2.54	1.27
SOIC	2.54	2.54	2.54	2.54	2.54	2.54	2.54		1.27
通孔	1.27	1.27	1.27	1.27	1.27	1.27	1.27	1.27	

4．PCB 布线

在放置完封装之后，可以使用 CAD 自动布线或者手动布线。对于自动布线则需要将自动布线失败或者不满足需求的地方手工重新布线。

布线的优先次序一般是：电源、模拟小信号、高速信号、时钟信号和同步信号等，关键信号优先布线。

应遵循密度优先原则，即从单板上连接关系最复杂的元器件着手布线。从单板上连线最密集的区域开始布线。

自动布线在布线质量满足设计要求的情况下，可使用自动布线器以提高工作效率。

在自动布线前应准备自动布线控制文件，该文件是为了更好地控制布线质量，一般在运行前详细定义布线规则，这些规则可以在软件的图形界面内进行定义，但软件提供了更好的控制方法，即针对设计情况，写出自动布线控制文件，软件在该文件控制下运行。

电源走线和地线走线之间的电磁兼容性环境较差，应避免布置对干扰敏感的信号。接地系统的结构由系统地、屏蔽地、数字地和模拟地构成；数字地和模拟地要分开，即分别与电源地相连。

环路最小规则，即信号线与其回路构成的环面积要尽可能小，环面积越小，对外的辐射越少，接收外界干扰也越小。针对这一规则，在地平面分割时，要考虑到地平面与重要信号走线的分布，防止由于地平面开槽等带来的问题；在双层板设计中，在为电源留下足够空间的情况下，应该将留下的部分用参考地填充，且增加一些必要的孔，将双面地信号有效连接起来，对一些关键信号尽量采用地线隔离，对一些频率较高的设计，需特别考虑其地平面信号回路问题，建议采用多层板为宜。

具体原则包括：

（1）有阻抗控制要求的网络应布置在阻抗控制层上。

（2）各种印制板走线要在容许的空间短而粗，线条要均匀。

（3）串扰控制，串扰是指 PCB 上不同网络之间因较长的平行布线引起的相互干扰，主要是由于平行线间的分布电容和分布电感的作用。克服串扰的主要措施包括：

- 加大平行布线的间距，遵循 3W 规则；
- 在平行线间插入接地的隔离线；
- 减小布线层与地平面的距离。

（4）最外沿信号线与禁止布线层和机械边缘保持最小 0.7mm 距离。

（5）印制板布线和覆铜拐角尽量使用 45°折线或折角，PCB 设计中应避免产生锐角和直角而不用 90°。

（6）对于经常插拔或更换的焊盘，要适当增加焊盘与导线的连接面积（泪滴焊盘），特别是对于单面板的焊盘，以增加机械强度，避免过波峰焊接时将焊盘拉脱、机械损耗性脱落等。

（7）任何信号都不要形成环路，如不可避免，让环路区尽量小。

（8）对噪声敏感的元器件下面不要走线。

（9）高频线与低频线要保持规定要求间距，以防止出现串扰。

（10）多层板走线应尽量避免平行、投影重叠，以垂直为佳，以减小分布电容对整机的影响。

（11）大面积覆铜需将铜箔制作成网状覆铜工艺，以防止 PCB 在高温时会出现气泡而导致铜箔脱落的现象。

（12）尽量加粗地线，可通过三倍的允许电流。

（13）布板时考虑放置测试点，方便生产线调试，测试点统一为八角形。

（14）同一尺寸板上布不同机种时，两端端子位置尽量保持一致，方便生产线制作工具。

通常情况下，布局基本确定后，应用 PCB 设计工具的统计功能，报告网络数量，网络密度，平均管脚密度等基本参数，以便确定所需要的信号布线层数。

布线层设置在高速数字电路设计中，电源与地层应尽量靠在一起，中间不安排布线。所有布线层都尽量靠近一平面层，优选地平面为走线隔离层。为了减少层间信号的电磁干扰，相邻布线层的信号线走向应取垂直方向。

可以根据需要设计 1～2 个阻抗控制层，如果需要更多的阻抗控制层，应与 PCB 产家协商。阻抗控制层应按要求标注清楚。将单板上有阻抗控制要求的网络布线分布在阻抗控制层上（单面板不用考虑）。

线宽和线间距的设置要考虑的因素：

- 单板的密度。板的密度越高，倾向于使用更细的线宽和更窄的间隙。
- 信号的电流强度。当信号的平均电流较大时，应考虑布线宽度所能承载的电流，线宽可参考以下数据。
- 电路工作电压。线间距的设置应考虑其介电强度。

5．设计规则检查

按照 PCB 设计规则，检查 PCB 设计是否合乎规范。对于元器件、铜线、过孔、覆铜等按照一定规则检查。例如，元器件不可以重叠，布线间距不合乎规范。一般可使用 CAD 对电路进行检查，将不符合规范的设计与未连接的部分查找出来。

PCB 设计检查还应当着重检查热设计要求。PCB 布局时要考虑将高热元器件放在出风口或利于空气对流的位置。较高的元器件应考虑放于出风口，且不阻挡风路，散热器的放置应考虑利于空气对流。

对温度敏感器等元器件应考虑远离热源，对于自身温升高于 30℃的热源，一般要求：

- 在风冷条件下，电解电容等温度敏感元器件离热源距离要求大于或等于 2.5mm；
- 自然冷条件下，电解电容等温度敏感元器件离热源距离要求大于或等于 4.0mm；
- 若因为空间的原因不能达到要求距离，则应通过温度测试保证温度敏感元器件的温升在降额范围内。

大面积铜箔要求用隔热带与焊盘相连，为了保证透锡良好，在大面积铜箔上的元器件的焊盘要求用隔热带与焊盘相连，对于需过 5A 以上大电流的焊盘不能采用隔热焊盘，如图 3-13 所示。

图 3-13　焊盘布线要求示意图

如果使用回流焊的方式，0805 以及封装小于 0805 以下的片式元器件两端焊盘的散热对称性为了避免元器件过回流焊后出现偏位、立碑现象，焊盘与印制导线的连接部宽度不应大于 0.3mm（对于不对称焊盘）。

高热元器件的安装方式及是否考虑带散热器，确定高热元器件的安装方式易于操作和焊接，原则上当元器件的发热密度超过 0.4W/cm^2，单靠元器件的引线腿及元器件本身不足以充分散热。应采用散热网、汇流条等措施来提高过电流能力，汇流条的支脚应采用多点连接，尽可能采用铆接后过波峰焊或直接过波峰焊接，以利于装配、焊接；对于较长汇流条的使用，应考虑过波峰时受热汇流条与 PCB 热膨胀系数不匹配造成的 PCB 变形；为了保证搪锡易于操作，锡道宽度应不大于等于 2.0mm，锡道边缘间距大于 1.5mm。

贴片元器件之间的最小间距满足如下要求。

- 机贴元器件距离要求，如图 3-14 所示。

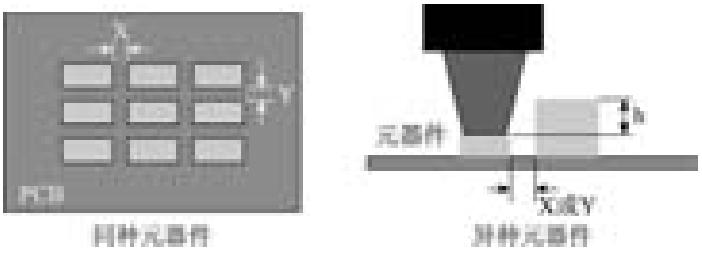

图 3-14　机贴元器件距离要求

- 同种元器件间距大于 0.3mm。
- 异种元器件间距大于 0.13×h+0.3mm（h 为周围近邻元器件最大高度差）。
- 只能手工贴片的元器件之间距离要求大于 1.5mm。

6．PCB 仿真分析

使用 CAD 软件对 PCB 进行仿真分析，确定 PCB 达到需求与设计目标。

7．保存输出

将设计工程保存，并且导出相关文件以便于 PCB 加工。

8．多层 PCB 设计的注意事项及布线原则

在多层 PCB 布线时应注意以下事项：

（1）高频信号线一定要短，不可以有尖角（90°直角），两根线之间的距离不宜平行、过近，否则可能会产生寄生电容。

（2）如果是两面板，一面的线布成横线，另一面的线布成竖线，尽量不要布成斜线。

（3）如果使用自动布线无法完成所有布线，建议设计者首先手工将比较复杂的线布好，将布好的线锁定后，再使用自动布线功能，一般就可以完成全部布线。

（4）一般来说，线宽一般为 0.3mm，间隔也为 0.3mm。但是电源线或者大电流线应

该有足够宽度。焊盘一般应为 64mil。如果是单面板，必须考虑焊盘，否则一般来说生产单面板的工艺都很差，所以单面板的焊盘尽量做得大一些，线要尽量粗一些，表 3-4 给出的是常见的焊盘尺寸。

表 3-4　常用的焊盘尺寸

（单位：mm）

焊盘孔直径	焊盘外径	焊盘孔直径	焊盘外径
0.4	1.5	1.0	2.5
0.5	1.5	1.2	3.0
0.6	2.0	1.6	3.5
0.8	2.0	2.0	4.0

（5）做好屏蔽，铜膜线的地线应该在电路板的周边，同时将电路上可以利用的空间全部使用铜箔做地线，增强屏蔽能力，并且防止寄生电容。多层板因为内层做为电源层和地线层，一般不会有屏蔽的问题。大面积敷铜应改用网格状，以防止焊接时板子产生气泡和因为热应力作用而弯曲。

（6）焊盘的内孔尺寸必须从元器件引线直径、公差尺寸、镀层厚度、孔径公差及孔金属化电镀层厚度等方面考虑，通常情况下以金属引脚直径加上 0.2mm 作为焊盘的内孔直径。例如，电阻的金属引脚直径为 0.5mm，则焊盘孔直径为 0.7mm。当焊盘直径为 1.5mm 时，为了增加焊盘的抗剥离强度，可采用方形焊盘。对于孔直径小于 0.4mm 的焊盘，焊盘外径/焊盘孔直径为 0.5～3mm。对于孔直径 2mm 的焊盘，焊盘外径/焊盘孔直径为 1.5～2mm。焊盘一般应该补成泪滴状，这样线与焊盘的连接强度会大大增强。

（7）地线的共阻抗干扰。电路图上的地线表示电路中的零电位，并用作电路中其他各点的公共参考点。在实际电路中由于地线（铜膜线）阻抗的存在，必然会带来共阻抗干扰，因此在布线时，不能将具有地线符号的点随便连接在一起，这可能引起有害的耦合而影响电路的正常工作。

9．丝印设计

丝印设计是 PCB 设计中容易被忽视但又十分重要的一个环节。容易被忽视是由于 PCB 并不会因为缺少丝印而不能工作，说其十分重要是因为丝印是 PCB 设计的一个缩影。丝印有效的标记元器件、安装孔、定位孔等 PCB 上关键的元素。

一般丝印设计要求所有元器件、安装孔、定位孔都有对应的丝印标号，PCB 上的安装孔丝印可用 H_1（Hole），H_2，…，H_n 进行标识。同时 PCB 上元器件的标识符必须和 BOM 清单中的标识符号一致。

PCB 板有高压和大电流处，要加上相应的警示标识，并且要保证标识的醒目、清晰、易辨识。同时丝印字符要在元器件本体以外，以避免元器件安装后本体遮住丝印

字符而降低元器件插装和维修效率。丝印字符要与对应元器件保持最近距离，若空间不足，可采用箭头方式在尽可能距离近的位置进行丝印字符标识。丝印字符方向遵循从左至右、从上往下的原则，对于电解电容、二极管等极性的元器件在每个功能单元内尽量保持方向一致。

为了保证元器件的焊接可靠性，要求元器件焊盘上无丝印；为了保证搪锡的锡道连续性，要求需搪锡的锡道上无丝印；丝印不能压在导通孔、焊盘上，以免开阻焊窗时造成部分丝印丢失，影响识别；丝印间距应大于 0.254mm。丝印字符大小在同一板子上要保持一致，参考尺寸为：字高是 1.5mm 字径（笔划的线宽）为 0.2mm，字体是 sans serif。

10．PCB 的可靠性设计

目前电子器材用于各类电子设备和系统时仍然以 PCB 为主要装配方式。实践证明，即使电路原理图设计正确，PCB 设计不当，也会对电子设备的可靠性产生不利影响。例如，若 PCB 两条细平行线靠得很近，则会形成信号波形的延迟，在传输线的终端形成反射噪声。

因此，在设计 PCB 的时候，应注意采用正确的方法，具体的一些参考性设计要点描述如下。

1）地线设计

在电子设备中，接地是控制干扰的重要方法。如能将接地和屏蔽正确结合起来使用，可解决大部分干扰问题。电子设备中地线结构大致有系统地、机壳地（屏蔽地）、数字地（逻辑地）和模拟地等。在地线设计中应注意以下几点：

（1）正确选择单点接地与多点接地。在低频电路中，信号的工作频率小于 1MHz，其布线和元器件间的电感影响较小，而接地电路形成的环流对干扰影响较大，因而应采用一点接地。当信号工作频率大于 10MHz 时，地线阻抗变得很大，此时应尽量降低地线阻抗，应采用就近多点接地。当工作频率在 1～10MHz 时，如果采用一点接地，其地线长度不应超过波长的 1/20，否则应采用多点接地法。

（2）将数字电路与模拟电路分开。电路板上既有高速逻辑电路，又有线性电路，应使它们尽量分开，两者的地线不要相混，分别与电源端地线相连，要尽量加大线性电路的接地面积。

（3）尽量加粗接地线。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，抗噪声性能变差。因此应将接地线尽量加粗，使它能通过三倍于 PCB 的允许电流。如有可能，接地线的宽度应大于 3mm。

（4）将接地线构成闭环路。设计只由数字电路组成的 PCB 的地线系统时，将接地线做成闭环路可以明显提高抗噪声能力。其原因在于：PCB 上有很多集成电路元器件，尤其遇有耗电多的元器件时，因受接地线粗细的限制，会在接地结构上产生较大的电位差，引起抗噪声能力下降，若将接地结构成环路，则会缩小电位差值，提高电子设备的抗噪声能力。

2）电磁兼容性设计

电磁兼容性是指电子设备在各种电磁环境中仍能够协调有效地进行工作的能力。电磁兼容性设计的目的是使电子设备既能抑制各种外来的干扰，使电子设备在特定的电磁环境中能够正常工作，又能减少电子设备本身对其他电子设备的电磁干扰。

（1）选择合理的导线宽度。由于瞬变电流在印制线条上所产生的冲击干扰主要是由印制导线的电感成分造成的，因此应尽量减小印制导线的电感量。印制导线的电感与其长度成正比，与其宽度成反比，因而短而精细的导线对抑制干扰是有利的。时钟引线、行驱动器或总线驱动器的信号线常常载有大的瞬变电流，印制导线要尽可能地短。对于分立元器件电路，印制导线宽度在 1.5mm 左右时，即可完全满足要求；对于集成电路，印制导线宽度可在 0.2～1.0mm 之间选择。

（2）采用正确的布线策略。采用平行走线可以减少导线电感，但导线之间的互感和分布电容增加，如果布局允许，最好采用井字形网状布线结构。具体做法是 PCB 的一面横向布线，另一面纵向布线，然后在交叉孔处用金属化孔相连。

为了抑制 PCB 导线之间的串扰，在设计布线时应尽量避免长距离的平行走线，尽可能拉开线与线之间的距离，信号线与地线及电源线尽可能不交叉。在一些对干扰十分敏感的信号线之间设置一根接地的印制线，可以有效地抑制串扰。

为了避免高频信号通过印制导线时产生的电磁辐射，在 PCB 布线时，还应注意以下几点：

- 尽量减少印制导线的不连续性，例如导线宽度不要突变，导线的拐角应大于 90°，禁止环状走线等；
- 时钟信号引线最容易产生电磁辐射干扰，走线时应与地线回路相靠近，驱动器应紧挨着连接器；
- 总线驱动器应紧挨其欲驱动的总线。对于那些离开 PCB 的引线，驱动器应紧紧挨着连接器；
- 数据总线的布线应每两根信号线之间夹一根信号地线。最好是紧紧挨着最不重要的地址引线放置地回路，因为后者常载有高频电流。

（3）抑制反射干扰。

为了抑制出现在印制线条终端的反射干扰，除了特殊需要之外，应尽可能缩短印制线的长度和采用慢速电路。必要时可加终端匹配，即在传输线的末端对地和电源端各加接一个相同阻值的匹配电阻。根据经验，对一般速度较快的 TTL 电路，其印制线条长于 10cm 以上时就应采用终端匹配措施。匹配电阻的阻值应根据集成电路的输出驱动电流及吸收电流的最大值来决定。

3）去耦电容配置

在直流电源回路中，负载的变化会引起电源噪声。例如在数字电路中，当电路从一个状态转换为另一种状态时，就会在电源线上产生一个很大的尖峰电流，形成瞬变的噪

声电压。配置去耦电容可以抑制因负载变化而产生的噪声，是印制电路板可靠性设计的一种常规做法，一般配置原则如下：

- 电源输入端跨接一个 10～100μF 的电解电容器，如果 PCB 的位置允许，采用 100μF 以上的电解电容器的抗干扰效果会更好；
- 为每个集成电路芯片配置一个 0.01μF 的陶瓷电容器。如遇到 PCB 空间小而装不下时，可每 4～10 个芯片配置一个 1～10μF 钽电解电容，这种元器件的高频阻抗特别小，在 50kHz～20MHz 范围内阻抗小，而且漏电流很小（0.5nA 以下）；
- 对于噪声能力弱、关断时电流变化大的元器件和 ROM、RAM 等存储型元器件，应在芯片的电源线和地线间直接接入去耦电容；
- 去耦电容的引线不能过长，特别是高频旁路电容不能带引线。

4）PCB 的尺寸与元器件的布置

PCB 大小要适中，过大时印制线条长，阻抗增加，不仅抗噪声能力下降，成本也高；过小则散热不好，同时易受临近线条干扰。

在元器件布置方面与其他逻辑电路一样，应把相互有关的元器件尽量放得靠近些，这样可以获得较好的抗噪声效果，如图 3-15 所示。时钟发生器、晶振和 CPU 的时钟输入端都易产生噪声，要相互靠近些。易产生噪声的元器件、小电流电路、大电流电路等应尽量远离逻辑电路，如有可能，应另做 PCB。

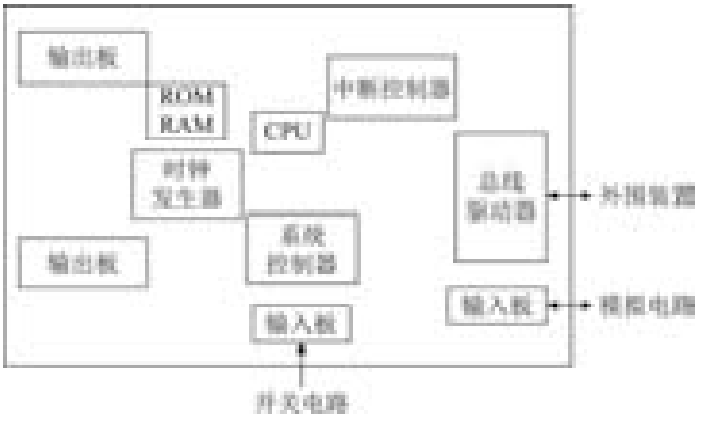

图 3-15　元器件的布置

5）散热设计

从有利于散热的角度出发，PCB 最好是直立安装，板与板之间的距离一般不应小于 2cm，而且元器件在 PCB 上的排列方式应遵循一定的规则：

- 对于采用自由对流空气冷却的设备，最好是将集成电路（或其他元器件）按纵长方式排列；对于采用强制空气冷却的设备，最好是将集成电路（或其他元器件）按横长方式排列；

- 同一块 PCB 上的元器件应尽可能按其发热量大小及散热程度分区排列，发热量小或耐热性差的元器件（如小信号晶体管、小规模集成电路、电解电容等）放在冷却气流的最上游（入口处），发热量大或耐热性好的元器件（如功率晶体管、大规模集成电路等）放在冷却气流最下游；
- 在水平方向上，大功率元器件尽量靠近 PCB 边沿布置，以便缩短传热路径；在垂直方向上，大功率元器件尽量靠近 PCB 上方布置，以便减少这些元器件工作时对其他元器件温度的影响；
- 对温度比较敏感的元器件最好安置在温度最低的区域（如设备的底部），千万不要将它放在发热元器件的正上方，多个元器件最好是在水平面上交错布局；
- 设备内 PCB 的散热主要依靠空气流动，所以在设计时要研究空气流动路径，合理配置元器件或 PCB。空气流动时总是趋向于阻力小的地方流动，所以在 PCB 上配置元器件时，要避免在某个区域留有较大的空域。整机中多块 PCB 的配置也应注意同样的问题。

大量实践经验表明，采用合理的元器件排列方式，可以有效地降低印制电路的温升，从而使元器件及设备的故障率明显下降。

以上所述只是 PCB 可靠性设计的一些通用原则，PCB 可靠性与具体电路有着密切的关系，在设计中还需根据具体电路进行相应处理，才能最大限度地保证 PCB 的可靠性。

3.2.3　电子电路测试基础知识

1．电子电路测试方法

电子电路测试包括内部测试、功能测试、边界扫描与 JTAG。

1）内部测试

电路测试的第一步是在 PCB 装配完成之后对板上各个元器件进行检查：是否正确安装、型号与数值是否正确、焊接是否合格。无论是手工还是机器加工的 PCB 都可能出现错误，因此对 PCB 的检查十分重要。

然后进行的电路内部测试则是使用自动测试与测试程序。PCB 上每个结点都需要进行探测，使用探针床测试夹进行 PCB 测试。自动化工具可以通过 PCB 设计自动计算 PCB 测试工具的设计与各个结点正确的状态，并以此进行测试。

然而这样的测试并不能保证整个系统不会故障，电路的内部测试效果也比较有限，只能保证基本的正确性而无法保证整个系统按照设计运行，因此需要使用功能测试。

2）功能测试

功能测试是在接通电源、激励或者特殊测试信号与输入/输出线之后，测试装配好电路板的功能。使用各类仪器对电路相关部分进行测试，同时进行校准和调整。对于嵌入

式系统，一般需要按照一定流程制订专门的测试步骤。而功能测试可以使用自动化设备进行测试，以简化人工成本。

3）边界扫描与 JTAG

通常来说，数字电路的测试会较为复杂。由于许多数字电路本身太过于复杂，常规的测试无法有效的进行。成百上千个测试点，外加各类 IC 复杂的逻辑与状态无法使用简单的电气特性测试来测试，因此边界扫描和 JTAG 能够较为有效地进行测试。边界测试是常见的硬件测试方式，而 JTAG 则是嵌入式中最常用的方式，它利用与 IC 芯片内部调试模块进行通信的方式进行调试与测试。

2．硬件可靠性测试

以行业标准或者国家标准为基础的可靠性测试包括电磁兼容试验、气候类环境试验、机械类环境试验和安规试验等。

由于网络产品的功能千差万别，应用场合可能是各种各样的，而与可靠性测试相关的行业标准、国家标准一般情况下只给出了某类产品的测试应力条件，并没有指明被测设备在何种工作状态或配置组合下接受测试，因此在测试设计时可能会遗漏某些测试组合。例如机框式产品，线卡种类、线卡安装位置、报文类型、系统电源配置均可灵活搭配，这里涉及到的测试组合会较多，这些测试组合中必然会存在比较极端的测试组合。再如验证该机框的系统散热性能，最差的测试组合是在散热条件机框上满配最大功率的线卡板。如果考虑其某线卡板低温工作性能，比较极端的组合是在散热条件最好的机框上配置最少的单板且配置的单板功耗最小，并且把单板放置在散热最好的槽位上。

总之，在做测试设计时，需要跳出传统测试规格和测试标准的限制，以产品应用的角度进行测试设计，保证产品的典型应用组合、满配置组合或者极端测试组合下的每一个硬件特性、硬件功能都充分暴露在各种测试应力下。这个环节的测试保证了，产品的可靠性才得到保证。

针对不同的产品形态，硬件可靠性测试项目可能有所差异，但是其测试的基本思想是一致的，其基本的思路都是完备分析测试对象可能的应用环境，在可能的应用环境下会承受可能工作状态包括极限工作状态，在实验室环境下制造各种应力条件、改变设备工作状态，设法让产品的每一个硬件特性、硬件功能都一一暴露在各种极限应力下，遗漏任何一种测试组合必然会影响到产品的可靠性。

3.3 Cadence PCB 系统设计

Cadence 公司的 PCB 系统设计提供了从原理图设计输入、分析、PCB 设计、PCB 制造文件输出等一整套工具，为嵌入式系统硬件设计的准确性和高效提供了基础，其设计流程如图 3-16 所示。

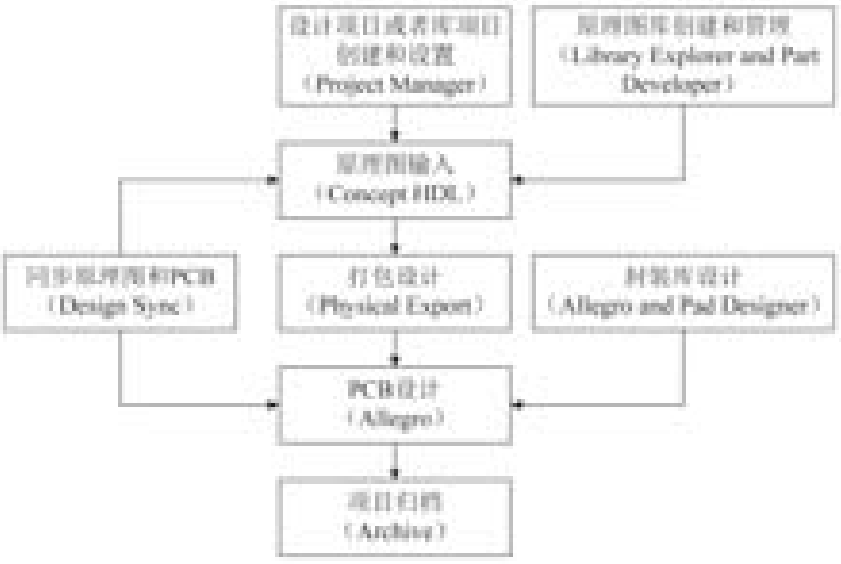

图 3-16　Cadence 的 PCB 系统设计流程示意图

3.3.1　原理图设计输入工具

Cadence 公司的 PCB 系统设计提供了两种原理图输入工具，Concept HDL 和 Capture CIS，Concept HDL 提供了一个高度集成的规则驱动的设计流程，与约束管理器整合提供了整个设计流程中管理电器约束的统一环境，支持团队设计、并发设计、设计重用等。

Concept HDL 提供了传统的平面设计方法和先进的分层次的设计方法，设计者可以根据自己的需要选择合适的设计流程和方法。

1）分层次的设计

Concept HDL 支持自顶向下和自底向上两种设计方法。自顶向下的设计方法就是先创建系统的方框图，分成若干子模块，然后再设计子模块，子模块又可以再往下细分成子模块或者绘制平面原理图。反过来就是自底向上的方法，先创建最底层的原理图，然后将原理图生成各个模块，各个模块又可以组合形成更高层的模块，最后形成一个系统设计。模块和原理图是可以混用的，并且可以分很多层。每个模块可以单独打包（Package，这里所说的打包，即将逻辑从原理图传递到 Allegro）到 Allegro 中，这样多个 PC 设计工程师就可以同时进行布局布线。图形化的分层和配置管理工具加上功能块编辑功能使得分层次设计的实现很容易。同时这些模块又是可以复制的，并且可以标注不同的属性，这样就保证了原理图之下只有一份拷贝，并且当变更模块的原理图时，会将所有的调用全部更新。

2）模块化设计——设计重用

市场压力和设计趋势推进电子产品向着模块化、多功能等级和核心功能派生的方向

发展。但是同时维护同一基础设计的多个版本既耗时又费力，也容易出错。Concept HDL 可以让设计者将与 Allegro 版图有关的原理图完整地作为一个元件（cell）保存到库中，可以像调用一个元件那样使用，省去了重新创建和重复拷贝的麻烦。例如电源电路和时钟电路在一个系统或者多个系统中通常会采用相同的解决方案，就可以采用这些方法来实现，这样就可以提高整体的设计效率。

3）并行设计方法

PCB 设计专家提供了真正的并行设计过程。例如在布局时，设计者需要改变连线或者添加元件，在 Allegro 或者 Concept HDL 中都可以实现设计同步，可以帮助用户分析原理图和 PCB 的不同，并且产生一个分层的 ECO 报告自动更新选择的文档。

4）导入物理布局和原理图

Allegro Expert 和 Concept HDL 可以通过 IFF 接口自动导入安捷伦 ADS 物理布局和原理图。导入后，安捷伦 ADS 的设计就如同一个模块，其组件映射到 Allegro 库中。可以选择锁定避免编辑，也可以解锁进行编辑，即使处于锁定状态，模块仍然允许将其连接到设计的其他部分。

5）功能强大的原理图输入方法

- 参数化。如果原理图中需要放置 20 个旁路电容，可以只放置一个电容，然后给这个电容设置参数 size=20，这样就可以减少原理图的篇幅，提高设计的效率，并且原理图看起来更加清晰、整洁。
- 对上下文敏感的菜单。这个功能与 Windows 的功能差不多，当选中一个对象时，右击，就会弹出一个菜单，菜单包含了与当前和上下文有关的命令。
- 群组操作。如果需要对某类元件进行替换或者某些元件需要修改某个属性，可以将这些对象生成一个群组，然后一次替换或者修改这个属性时，节省很多时间。
- 分割元件图形。某些元件管脚非常多，有几百个管脚的元件是非常普遍的，有些都有 1000 多个管脚了，在一页原理图中显示这么多管脚是不切实际的。Concept HDL 的建库工具（Part Developer）可以将这样的元件分为几个图形符号来制作，并且这些符号可以放在不同的原理图页面上。在打包到 Allegro 中时，仍然可以将这些符号打包成一个元件。
- SKILL 和 CAE Views。设计者根据需要写 SKILL 程序来定制 Concept HDL，并且可以共享，全局导航、查找和替换。无论是平面设计还是层次设计，都可以轻松地按几下鼠标键即可找到任何元件或网络。
- 元件列表文件（PPT）。元件列表文件可以让设计者将不同的物理元件映射到同样的原理图符号上。例如常用的电阻和电容等元件，元件原理图的图形是完全一样的，只是封装、标称值等不一样。在建库时，可将一类元件全部输入列表文件中，在原理图中通过选择不同的元件属性来调用它们。
- 脚本（script）和非图形化的 Concept HDL。设计者可以为经常执行的命令设置批

处理，在设计过程中调用。Concept HDL 也可以运行在非图形化的样式，这种模式一般用于自动运行模式。

6）其他功能和特点

- 高性能的图形界面，可以动态移动定制的用户界面，可以命令行输入，热键输入和执行 STROKE（手绘）命令。
- 自动生成 BOM（料单）。Concept HDL 的这个功能方便设计者自动生成料单。料单的格式按照需要也可以定制，并且可以将非电气元件另外生成一个料单与电气元件的料单连接起来。
- 可以进行电气规则检查和生成网表报告。
- 归档。一般原理图中并没有将原理图库信息全部调入，如果将原理图转移到其他计算机上进行编辑，就会出现找不到库的麻烦，归档功能提供可以将原理图所用的组件归档到本地的功能，不用的库就不会拷贝过来。
- 与 Allegro 整合。Concept HDL 不仅仅是一个原理图编辑器，它的作用类似于完整设计环境中的 HUB，无缝地与 Allegro PCB 设计系统和其他仿真工具整合。例如在布局时，设计者可以通过在 Concept HDL 中选中组件而在 Allegro 中放置，也可以一次就放置所有的组件。
- 项目管理器。在项目管理器中，设计者可以启动所有的工具、改变启动工具的设置。
- 与约束管理器整合。Concept HDL 也是设计流程中管理电气约束的统一环境的一部分，所以在 Concept HDL 中也可以利用约束驱动过程传递正确的设计给 Allegro，或者反过来传递给 Concept HDL。

3.3.2　PCB 设计系统

PCB 设计系统可以实现复杂、多层电路板图的创建和编辑，可以方便地输出生产数据，其特点包括：

（1）灵活的驱动布局功能。Allegro Expert 提供约束驱动自动和交互结合的布局模式，可以让工具自动布局，也可以手工调整，在放置元件时高速约束和物理设计规则可以动态地检查元件的放置有没有违反规则，并报告出来。QuickP1ace 可以让设计者对组件进行过滤和预分组，在 PCB 外形图框周围放置他们。Ailegro Expert 使用统一的约束管理器，在布局阶段提供互连线延时的实时图形反馈，使工程师最优化地仿真元件，保证了设计的正确性。

（2）交互式布线编辑器。Allegro 提供基于形状、任意角度和推挤（push/shove）的布线方式，对于有高速规则约束的网络在走线时还可以实时显示还有多少时序裕量。

（3）多种生产加工数据的输出。可以输出多种生产加工数据，包括标准的 Gerber 文件、多种光绘机文件、D 码表、装配图、裸板测试数据等，还能输出 ODB++数据格式。

（4）丰富的平面操作功能。Allegro Expert 提供了功能最强的电源平面创建和编辑功能，包括用户定义分割面、中间层面正片显示以及用户定义部分覆铜区域的功能选项。该电源平面设计工具可以使设计者像观察正片一样显示所见即所得的电源层负片。

（5）高级 SKILL 语言。使用高级 SKILL 语言，设计者可以正确地集成和定义自己需要或喜爱的工具箱。

3.3.3 自动和交互布线工具

Cadence 公司的 PCB 系统设计中的自动布线工具是一流的，针对高密度 PCB 和复杂 IC 封装的自动和交互式互连线布线工具，具体包含 SPECCTRA 布局编辑器、SPECCTRA 交互布线编辑器、SPECCTRA 自动布线器 3 个工具。

1）SPECCTRA 布局编辑器

SPECCTRA 布局编辑器可以方便快捷地帮助设计者完成布局，可以对单个和一组元件进行诸如翻转、旋转、推挤、对齐和移动等操作。SPECCTRA 布局编辑器提供指导布局模式，帮助设计者自动计算出最佳布局位置，设计者也可以调整。SPECCTRA 布局编辑器还提供密度分析功能，以图形方式显示布线阻塞情况，帮助设计者调整布局，提高布通率。

2）SPECCTRA 交互布线编辑器

SPECCTRA 交互布线编辑器提供的推挤功能可以自动按照间距要求移动附近的连线和过孔。当移动连线或过孔时，编辑器会自动推挤它周围的连线并动态地显示出来。设计者也可以通过多级操作放弃所做的动作。

SPECCTRA 交互布线编辑器还提供自动帮助放置过孔、拷贝连线等功能。

3）SPECCTRA 自动布线器

SPECCTRA 自动布线器使用高效的基于形状的布线算法，可以充分利用布线空间。SPECCTRA 自动布线器还提供了电气参数规则控制和电流承载能力的要求。

SPECCTRA 自动布线器还可以提供盲埋孔、焊盘下过孔等的处理，是当今高密度 PCB 设计必需的功能。

3.3.4 库管理

Cadence 公司的 PCB 系统设计的库管理提供 3 个工具，分别是 PCB 库专家、PCB 库、库浏览。

1）PCB Librarian Expert——PCB 库专家

PCB 库专家提供了原理图和 PCB 库的创建、封装和验证功能。它包含了几个工具，原理图库的创建是由 Librarian Expert 和 Part Developer 这两个工具来实现的。Part Developer 用来创建原理图符号、物理引脚与封装的对应和其他关键属性。Padstack Editor 是图形编辑器，用来创建、修改焊盘。Allegro Librarian 用来创建 PCB 封装符号，可以用

手工和向导两种方法来实现。

2）PCB Librarian——PCB 库

PCB Librarian 包含手工创建库的工具，包括 Library Export 和 Part Developer 原理图创建工具，还有创建 PCB 库的 Allegro Librarian 工具，可以完成库的创建和校验。虽然 PCB Librarian 和 PCB Librarian Expert 包含的工具是一样的，但是在 PCB Librarian 中只能使用工具的部分功能，一些高级功能不能实现。

3）Part Browser——库浏览

Part Browser 是基于 Web 的检索和放置元件的工具。库的验证通过 Part Browser 来实现。这个工具可以提供多个方法来检索元件列表（PTF）的内容，并且可以被集成到 MRP/ERP 系统中提供其他商业信息。

3.3.5　约束管理器

约束管理是 PCB 系统设计的核心，提供基于电子数据表格式的约束信息，具有实时显示高速规则和状态的功能，并且可以在设计流程的任意阶段调用。仿真设计人员在做仿真之后，形成了高速约束规则，这些规则一旦加入约束管理器，就可以用来驱动布局布线了。约束管理器包括两个视图，一个视图让设计者观察数据库中不同的电子约束集合相关的约束值；另一个视图提供系统中不同网络以及它们要遵守的约束集名称，并且实时显示约束值的分析结果，通过改变分析结果的颜色来标明成功和失败，一目了然。

第 5 章　嵌入式系统设计与开发

嵌入式系统设计采用硬件和软件协同设计的方法，在设计与开发过程中不仅需要了解软件领域的知识，还需要了解硬件领域的综合知识。设计者必须熟悉并能自如地运用这些领域的各种技术，才能使所设计的系统达到最优。

嵌入式系统设计的主要任务是定义系统的功能、决定系统的架构，并将功能映射到系统实现架构上。系统架构既包括软件系统架构也包括硬件系统架构。一种架构可以映射到各种不同的物理实现，每种实现表示不同的取舍，同时还要满足某些设计指标，并使其他的设计指标也同时达到最优。

本章主要介绍嵌入式系统开发环境和嵌入式软件开发方法，阐述了完整的嵌入式开发流程，最后讨论嵌入式领域软件移植的相关问题。

5.1　嵌入式软件开发概述

嵌入式软件开发需要将软件和硬件更好的结合，因此不同于通用软件的开发。在系统总体开发中，由于嵌入式系统与硬件依赖非常紧密，往往某些需求只能够通过特定的硬件才能实现。本节简要叙述了嵌入式应用开发的流程，具有使用交叉编译工具和资源有限的特点，最后介绍嵌入式软件开发面临的挑战。

5.1.1　嵌入式应用开发的过程

一个嵌入式应用项目的开发过程是一个硬件设计和软件设计的综合过程，也是一个系统过程。一般而言，要经历以下步骤：

（1）硬件的设计与实现，包括元器件选型、原理图编制、印制板设计、样板试制、硬件功能测试等。

（2）设备驱动软件的设计与实现，包括引导加载程序的编写以及各种设备驱动程序的编写。

（3）嵌入式操作系统的选择、移植，以及 API 接口函数的设计。

（4）支撑软件的设计与调试。

（5）应用程序的设计与调试。

（6）系统联调，样机交付。

由此可见，开发一个嵌入式应用其实就是开发一个特定用途的计算机系统，开发时需要综合考虑系统软硬件各个层次上的所有问题。因此，无论是开发过程还是开发环境，

都与一般的桌面系统上的应用程序开发有着显著的不同，需要考虑更多的因素。仅软件部分就要考虑板级支持包 BSP 的开发、操作系统的移植、应用程序的开发和操作系统的接口等问题。即使只开发应用程序，也要在工程项目中将操作系统文件、设备驱动文件和应用程序文件集成在一起，经过修改整理再编译成目标文件。

图 5-1 是一个典型的嵌入式应用程序的生成和加载过程。

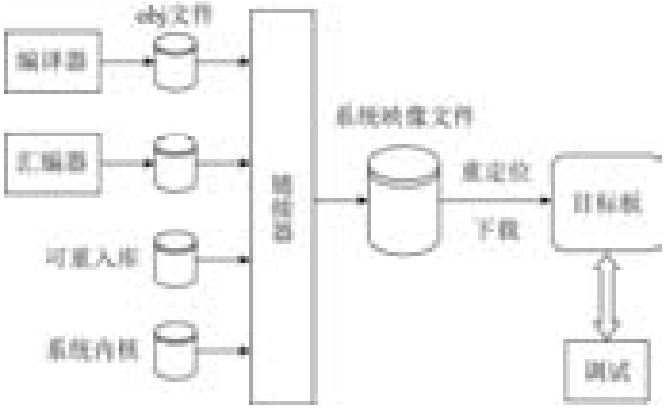

图 5-1　嵌入式应用程序的生成与加载

5.1.2　嵌入式软件开发的特点

嵌入式软件开发有如下的几个特点。

1．需要交叉编译工具

嵌入式系统目标机上的资源非常有限，例如，处理器的结构比较简单，速度较慢；内存和外存的容量小；显示功能较弱；软件资源较少，等等。因此，直接在嵌入式系统的硬件平台上开发和调试应用软件比较困难，有时甚至是不可能的。目前一般采用的解决办法是：将集成开发环境安装在高性能的 PC 上，然后在 PC 上进行嵌入式应用软件的开发。由于 PC 上的 CPU 较多使用 x86 芯片，而嵌入式系统中的处理器芯片种类繁多，如 ARM、MIPS、Power PC 等，两种处理器的指令集是不同的。因此，在集成开发环境下编写的源程序需要经过交叉编译，才能生成目标平台上运行的二进制代码格式。

2．通过仿真手段进行调试

需要在目标机执行的程序经过交叉编译之后，还要经过调试排错，确认能正常运行后才能交付使用。显然，在目标机上进行调试排错是非常困难的，因此实际的做法是仿真调试。也就是通过接口和信号线，把目标机上机器指令的执行结果和 CPU 当前各个寄存器的值传送到集成开发平台，使开发人员能够观察到目标机的执行状况，从而判断出指令执行的正确与错误。

3．开发板是中间目标机

嵌入式应用软件需要在开发板上完成所有的开发任务。系统开发完成之后，才把目标程序安装在目标机上运行。也就是说，开发板只是中间的目标机，其任务就是支持开发和调试。

4．可利用的资源有限

在台式机环境下，程序员拥有大量的硬件和软件编程资源，对内存容量、硬盘容量、可以打开的文件数量等问题几乎不加限制。然而在嵌入式软件开发中，就必须考虑可用资源的问题。嵌入式系统使用的处理器、内存和存储容量与台式机相比有很大的差距。因此，嵌入式代码不仅要提供丰富的功能，还必须满足其他的一些约束条件，如按所需速度运行以满足系统期限、适应内存总量限制、满足功耗要求等等。

5．需要与硬件打交道

在开发桌面应用程序时，很少直接与硬件打交道，除非是开发初级的设备驱动程序。但是在嵌入式系统的开发中，软件与硬件的关系非常密切，经常需要对运算器、寄存器和存储器进行操作。即使是采用了嵌入式操作系统，为了架构的简化以及节省空间，也容许应用程序直接去访问外围的寄存器。特别是当系统发生了难以理解的错误时，可能无法确认到底是程序写错，还是目标平台的硬件电路有问题。因此编程人员除了要了解如何编写高级语言程序（C、C++、Java 等）与低级语言程序（如汇编），还要了解硬件设计及除错的内容。

5.1.3　嵌入式软件开发的挑战

在开发嵌入式软件时，会面临如下的一些挑战和问题。

1．软硬件协同设计

嵌入式系统由硬件和软件组成，因此，在系统设计时，需要考虑哪些功能用硬件来实现，哪些功能用软件来实现。硬件实现的优点是速度快，缺点是芯片成本高，耗电量大，且需要占用额外的空间。软件实现的优点是灵活性高，如果算法发生了改变，那么修改软件是很容易的。例如，以 TCP/IP 协议栈的实现为例。几十年来，都是用软件来实现，因为这种方法为改变协议提供了灵活性。在台式机环境下，TCP/IP 协议栈被绑定在操作系统中，这是可以接受的，因为桌面计算机有大量的内存和外存容量。不过，现在已经出现了 TCP/IP 协议栈的单芯片实现方案，这种方法极大地加速了协议的处理过程。它的另一个优点就是可以把它集成到嵌入式硬件中，从而使嵌入式系统具备网络功能。

2．嵌入式操作系统

嵌入式应用的开发可以分为无操作系统和有操作系统两种情形。

（1）无操作系统的情形。这种情形下，嵌入式软件的设计主要是以应用为核心，应用软件直接建立在硬件上，没有专门的操作系统，软件的规模也很小，基本上属于硬件的附属品。开发人员可以混合使用汇编语言和 C 语言，实现存储管理、输入/输出管理和

任务管理等服务。这种方式的优点是：软件是为特定的应用而专门编写的，因而代码的结构紧凑，体积小、效率高，既提高了运行速度，又节省了存储空间。

（2）有操作系统的情形。开发时首先将一个可用的嵌入式操作系统移植到目标处理器，当程序员在开发应用程序时，不是直接面对嵌入式硬件设备，而是在操作系统的基础上编写，操作系统会提供必要的 API 接口函数来实现各种功能。在这种情形下，开发人员不必操心存储管理、任务管理等一般性的事务，而是将精力集中在应用软件的开发上。因而开发速度更快，编写出来的代码更加可靠。这也是现在广泛采用的一种开发方法。

3．代码优化

一般来说，桌面应用软件的开发人员不必过多考虑代码优化的问题，因为处理器的功能强大，内存的容量也足够用，而且在响应时间上，即使有几秒钟的误差也不会带来显著的区别。但是在嵌入式系统中，存储器容量和执行时间通常是最主要的约束条件。尽管编译器会实现代码上的优化，但编程人员仍必须精心编写代码，并对代码进行优化，开发出运行速度快、存储空间少、维护成本低的软件。有时，为了达到系统所要求的响应时间，编程人员可能需要使用汇编语言来编写部分代码。

4．有限的输入/输出功能

在台式机环境下，一般都使用键盘和鼠标作为输入设备，显示器作为输出设备。但是在嵌入式系统中，不一定存在这些外设。事实上，大多数嵌入式系统的输入/输出功能是有限的。例如，只有小键盘（具有 8 个或 12 个功能键）可用于输入数据。在输出设备上，可能只有少量的 LED，或每行 8 到 12 个字符且仅有两行的小型 LCD 显示器。而有些嵌入式系统根本就没有键盘或显示器这样的 I/O 设备来与用户进行交互。例如，在许多过程控制系统中，采用电信号来作为输入并产生电信号来作为输出。因此，开发、测试和调试这一类的系统更具有挑战性，必须采用特殊的程序来测试这些系统。

总之，嵌入式软件的开发具有很大的挑战性，需要开发人员具有扎实的软、硬件基础，能灵活运用不同的开发手段和工具，具有较丰富的开发经验。如果要设计出可靠、稳定、高效的嵌入式软件，就必须综合考虑多种因素，如并发性、兼容性、实时性、层次性、可扩展性、有限的资源、多样性和可读性等。

5.2 嵌入式软件开发环境

在早期的嵌入式开发中，大多数嵌入式软件的开发是直接在硬件平台上进行的，采用处理器的汇编语言进行编程，直接对各种硬件设备进行控制和访问。用户除了要编写具体的应用程序外，还要编写各种监控程序和调试工具软件来构建相应的调试环境。尤其是对于多任务和实时性处理，必须编写出性能优化的系统软件，根据各个任务的重要性进行统筹兼顾和合理调度，以确保每个任务能及时执行，满足系统的实时性要求。随着嵌入式产品规模越来越大，功能越来越复杂，这种手工作坊式的开发方式越来越不能

满足需要。

面对这些困难，人们提出以下要求：

（1）嵌入式系统开发需要专门的开发工具和调试环境，支持多种软硬件平台，提供一种高级编程语言如 C 或 C++。由开发工具提供针对多种处理器的编译系统，使开发代码易于移植扩充。调试环境提供的调试手段丰富，易于发现问题。

（2）嵌入式软件需要一个较好的操作系统开发平台，提供性能完备的实时控制、任务管理、存储管理和资源分配等功能。应用软件的开发是在这个平台上进行，编程人员不必去考虑底层的实现细节。

（3）嵌入式系统开发工具要求易学、易用、可靠、高效、人机用户界面友好，为编程人员提供一个方便好用的开发环境。

（4）支持嵌入式系统开发可剪裁的要求。针对不同的嵌入式系统应用，可以根据需要来剪裁出一个大小适宜的系统。

在实际的嵌入式开发中，根据项目的需要，既可以采用一组相互独立的软件开发工具，如编辑器、编译器、调试器和仿真器等，也可以采用一些商业化的集成开发环境，将各种软件开发工具集成在一个用户界面友好、功能强大、使用方便、适用性广、覆盖产品开发全周期的平台环境中。

5.2.1　宿主机和目标机

嵌入式应用开发需要良好的开发环境的支持。在嵌入式系统中，由于目标机的资源有限，不可能在其上建立庞大、复杂的开发环境，因而通常的做法是把开发环境和目标运行环境进行分离，如图 5-2 所示，嵌入式应用软件的开发方式一般是：首先在宿主机（Host）上建立开发环境，进行应用程序编码和交叉编译，然后在宿主机和目标机（Target）之间建立连接，将应用程序下载到目标机上进行交叉调试，经过调试和优化，最后将应用程序固化到目标机中实际运行。

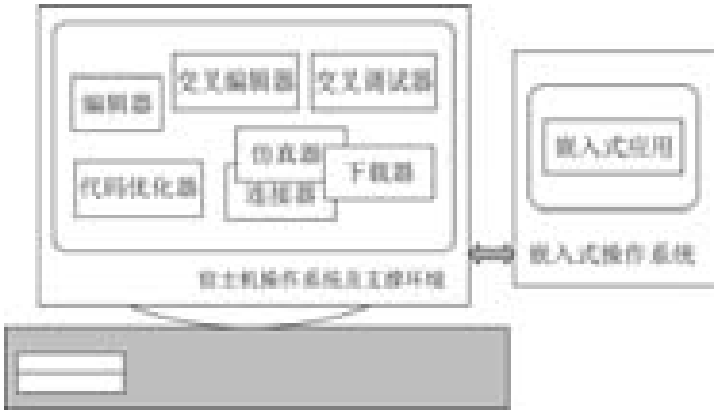

图 5-2　宿主机与目标机的开发模式

1．宿主机

宿主机是用于开发嵌入式系统的计算机，它通常是拥有大容量内存和硬盘、支持打印机等外设的 PC 机或工作站。在宿主机端（其操作系统可以是 Windows 系列、Linux 或 Solaris 等）运行的工具包括文本编辑器、交叉编译器、交叉调试器、集成环境以及各种分析工具。其中集成环境是其他工具的总入口，被集成的工具一般有它自己独立的图形界面，例如交叉调试器和分析工具等。

2．目标机

目标机一般在嵌入式应用软件的开发和调试期间使用，它可以是嵌入式应用软件的实际运行环境，也可以是能够替代实际运行环境的仿真系统。目标机的软硬件资源通常都比较有限，主要用来运行包含应用程序代码和嵌入式操作系统的可执行映像。

在开发过程中，目标机端须接收和执行宿主机发出的各种命令，如设置断点、读内存和写内存等，并将结果返回给宿主机，配合宿主机各方面的工作。所有需要与目标机进行信息交互的工具在目标机端都有自己的代理，有的代理是软件实现的（如目标机监控器），有的代理是硬件实现的（如 BDM、JTAG 等）。在目标机端运行的这些代理，负责解释并执行从宿主机端发送过来的各种命令。

3．宿主机与目标机的连接

在宿主机和目标机之间必须建立连接，这样就可以从宿主机向目标机下载、运行可执行映像，或者进行远程调试。宿主机和目标机之间的连接可以分为两类：物理连接和逻辑连接。

物理连接是指宿主机与目标机上的一定物理端口通过物理线路连接在一起。其连接方式主要有三种：串口、以太网接口和 OCD（On Chip Debug）方式（如 JTAG、BDM）等。物理连接是逻辑连接的基础。

逻辑连接是指宿主机与目标机之间按某种通信协议建立起来的通信连接，目前逐步形成了一些通信协议的标准。

要顺利地建立起交叉开发环境，需要正确地设置。在物理连接上，要注意使硬件线路正确连接，且硬件设备完好，能正常工作，连接线路的质量要好。在逻辑连接上，要正确配置宿主机和目标机的物理端口参数，并与实际的物理连接一致。

在实际嵌入式开发中，最常用的连接方式是以太网上的 IP 网络连接，这种连接不但有很高的带宽，而且具有网络连接的所有优点。至于串口连接方式，主要适用于以下两种情形：

- 在嵌入式应用中并不需要支持网络，同时在代码规模上又有限制，此时可删除嵌入式操作系统中的网络部分。
- 进行嵌入式操作系统内核调试，而有些嵌入式操作系统的网络驱动程序并不支持这种调试模式。

实际上，这两种连接方式是可以并存的。例如，在下载可执行映像时可以使用以太

网接口，在进行操作系统内核调试时可以使用串口。

5.2.2　嵌入式软件开发工具

嵌入式软件的开发可以分为几个阶段：源代码程序的编写；将源程序交叉编译成各个目标模块；将所有目标模块及相关的库文件链接成目标程序；代码调试等。在不同的阶段需要用到不同的软件开发工具，如编辑器、编译器、调试工具、软件工程工具等。

1．编辑器

从理论上来说，任何一个文本编辑器都可以用来编写源代码。但是为了提高编程的效率，一个好的编辑器应该具备如下一些特点：

（1）支持 C 语言、汇编语言等程序设计语言的语法高亮显示；

（2）支持文件管理操作（如打开文件、保存文件、关闭文件等）、文件编辑操作、文件打印、文本查找等功能；

（3）编辑窗口可以同时作为调试时源代码执行的跟踪窗口；

（4）通过“编译结果输出窗口”可以直接定位到相应的源代码编辑窗口；

（5）提供一系列辅助编辑工具；

（6）编辑器可以同时打开多个窗口进行编辑，可编辑的文件大小理论上无限制；

（7）编辑器的编辑命令和编辑操作最好与标准的 Windows 编辑器功能一致，以便熟悉 Windows 的用户使用。

在各种集成开发环境中，一般都会提供一个功能强大的编辑器。UltraEdit 和 Source Insight 是两个常用的独立编辑器。

UltraEdit 是一个功能强大的文本编辑器。它可以取代记事本，用来编辑文本文字，也可以用来编写各种语言的源代码。它内建英文单词检查、C++及 Visual Basic 语法加亮显示，可同时编辑多个文件。即使打开一个很大的文件，速度也不会慢。UltraEdit 附有 HTML Tag 颜色显示、搜寻替换以及无限制的还原功能。它支持二进制和十六进制编辑，可以用来直接修改 EXE 或 DLL 文件。

Source Insight 是一款面向工程项目的源码编辑和查看软件，其用户界面友好，变量和函数名都以特定的颜色表示出来，非常直观。对于各种语言的源文件，如 C/C++、C# 和 Java，它能自动解析程序的语法结构，动态地保持符号信息数据库，并主动显示有用的上下文信息。Source Insight 不仅是一个功能强大的程序编辑器，它还能显示参考树、类继承图和调用树等信息。它具有快速源代码导航功能，用户可以使用各种搜索命令，在各个源文件的不同函数和变量定义之间来回跳转，非常方便，因此它很适合于编辑大型软件。

2．编译器

编译阶段要做的工作是用交叉编译或汇编工具处理源代码，产生目标文件。在嵌入式系统中，宿主机和目标机所采用的处理器芯片通常是不一样的。例如，目标机采用的

CPU 是 DragonBall M68x 系列或 ARM 系列，而宿主机采用的是 x86 系列。因此，为了把宿主机上编写的高级语言程序编译成可以在目标机上运行的二进制代码，就需要用到交叉编译器。

与普通 PC 中的 C 语言编译器不同，嵌入式系统中的 C 语言编译器要进行专门的优化，以提高编译效率。一般来说，优秀的嵌入式 C 编译器所生成的代码，其长度和执行时间仅比用汇编语言编写的代码长 5%～20%。编译质量的不同，是区别嵌入式 C 编译器工具的重要指标。因此，硬件厂商往往会针对自己开发的处理器的特性来定制编译器，既提供对高级语言的支持，又能很好地对目标代码进行优化。

GNU C/C++（gcc）是目前比较常用的一种交叉编译器，它支持非常多的宿主机/目标机组合。宿主机可以是 Unix、AIX、Solaris、Windows、Linux 等操作系统，目标机可以是 x86、Power PC、MIPS、SPARC、Motorola 68K 等各种类型的处理器。

gcc 是一个功能强大的工具集合，包含了预处理器、编译器、汇编器、连接器等组件。它在需要时会去调用这些组件来完成编译任务，而输入文件的类型和传递给 gcc 的参数决定了它将调用哪些组件。对于一般或初级的开发者，它可以提供简单的使用方式，即只给它提供 C 源码文件，它将完成预处理、编译、汇编、连接等所有工作，最后生成一个可执行文件。而对于中高级开发者，它提供了足够多的参数，可以让开发者全面控制代码的生成，这对于嵌入式系统软件开发来说是非常重要的。

gcc 识别的文件类型主要包括：C 语言文件、C++语言文件、预处理后的 C 文件、预处理后的 C++文件、汇编语言文件、目标文件、静态链接库、动态链接库等。以 C 程序为例，gcc 的编译过程主要分为 4 个阶段：

（1）预处理阶段，即完成宏定义和 include 文件展开等工作；

（2）根据编译参数进行不同程度的优化，编译成汇编代码；

（3）用汇编器把上一阶段生成的汇编码进一步生成目标代码；

（4）用连接器把上一阶段生成的目标代码、其他一些相关的系统目标代码以及系统的库函数连接起来，生成最终的可执行代码。

用户可以通过设定不同的编译参数，让 gcc 在编译的不同阶段停止下来，这样可以检查编译器在不同阶段的输出结果。

在 gcc 的高级用法上，一般希望通过使用编译器达到两个目的：检查出源程序的错误；生成速度快、代码量小的执行程序。这可以通过设置不同的参数来实现，例如，“-Wall”参数可以发现源程序中隐藏的错误；“-O2”参数可以优化程序的执行速度和代码大小；“-g”参数可以对执行程序进行调试。

3．调试及调试工具

在开发嵌入式软件时，交叉调试是必不可少的一步。嵌入式软件的特点决定了其调试具有如下特点：

（1）对于通用的计算机，调试器与被调试程序一般位于同一台计算机上，操作系统

也相同，调试器进程通过操作系统提供的调用接口来控制被调试的进程。而在嵌入式系统中，由于目标机的资源有限，调试器和被调试程序运行在不同的机器上。调试器主要运行在宿主机上，而被调试程序则运行在目标机上。

（2）调试器通过某种通信方式与目标机建立联系。通信方式可以是串口、并口、网络、JTAG 或专用的通信方式。

（3）在目标机上一般有调试器的某种代理，这种代理能配合调试器一起完成对目标机上所运行程序的调试。这种代理可以是某种软件，也可以是支持调试的某种硬件。

总之，在交叉调试方式下，调试器和被调试程序运行在不同的机器上。调试器通过某种方式能控制目标机上被调试程序的运行方式，并能查看和修改目标机上的内存、寄存器以及被调试程序中的变量。在嵌入式软件的开发实践中，经常采用的调试方法有直接测试法、调试监控器法、ROM 仿真器法、在线仿真器法、片上调试法及模拟器法。

1）直接测试法

直接测试法是嵌入式系统发展早期经常采用的一种调试方法。这种方法需要的调试工具非常简单，比较适合当时的实际情况。采用这种方式进行软件开发的基本步骤是：

（1）在宿主机上编写程序的源代码。

（2）在宿主机上反复地检查源代码，直到编译通过，生成可执行程序。

（3）将可执行程序固化到目标机上的非易失性存储器（如 EPROM、Flash 等）中。

（4）在目标机上启动程序运行，并观察程序的运行结果。

（5）如果程序不能正常工作，则在宿主机上反复检查代码，查找问题的根源，然后修改代码，纠正错误，并重新编译。

（6）重复执行（3）～（5），直到程序能正常工作。

从这些开发步骤可以看出，这种调试方法基本上无法监测程序的运行。虽然也有人提出了一些调试的小窍门，例如，从目标机打印一些有用的提示信息（通过监视器、LCD 或串口等输出信息），或者利用目标机上的 LED 指示灯来判断程序的运行状态。但这些窍门的作用有限，如果一个程序在运行时没有产生预想的效果，那么开发者只能通过检查源程序来发现问题。显然，这种调试方法的效率很低，难度很大，开发人员也很辛苦。但由于开发条件特别是开发工具的限制，在嵌入式系统的早期阶段，程序的开发只能采用这种方法。甚至目前在开发一些新的嵌入式产品时，也往往要采用这种方法。

2）调试监控器法

调试监控器法的工作原理如图 5-3 所示。在这种调试方式下，调试环境由三部分构成，即宿主机端的调试器、目标机端的监控器（监控程序）以及两者之间的连接（包括物理连接和逻辑连接）。

监控器是运行在目标机上的一段程序，它负责监视和控制目标机上被调试程序的运行，并与宿主机端的调试器一起，完成对应用程序的调试。监控器预先被固化到目标机的 ROM 空间中，在目标机复位后将被首先执行。它对目标机进行一些必要的初始化，

然后初始化自己的程序空间，最后就等待宿主机端的命令。监控器能配合调试器完成被调程序的下载、目标机内存和寄存器的读/写、设置断点以及单步执行被调试程序等功能。一些高级的监控器能配合完成代码分析、系统分析、ROM 空间的写操作等功能。

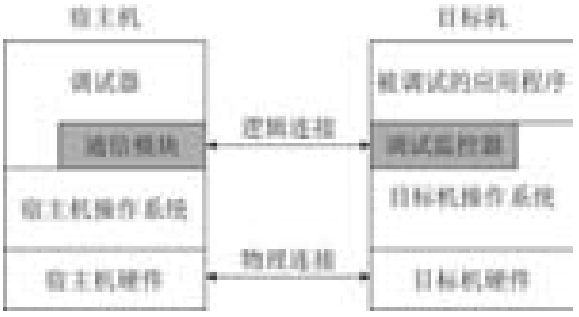

图 5-3　调试监控器法的工作原理

利用监控器方式作为调试手段时，开发应用程序的步骤如下：

（1）启动目标机，监控器掌握对目标机的控制，等待与调试器建立连接。

（2）调试器启动，与监控器建立起通信连接。

（3）调试器将应用程序下载到目标机上的 RAM 空间中。

（4）开发人员使用调试器进行调试，发出各种调试命令。监控器解释并执行这些命令，并通过目标机上的各种异常来获得对目标机的控制，将命令执行结果回传给调试器。

（5）如果程序有问题，则开发人员在调试器的帮助下定位错误。修改之后再重新编译链接并下载程序，开始新的调试。如此反复直到程序能正确运行为止。

监控器方式明显地提高了程序调试的效率，降低了调试的难度，缩短了产品的开发周期，有效地降低了开发成本。而且这种方法的成本也比较低廉，基本上不需要专门的调试硬件支持。因此它是目前使用最为广泛的嵌入式软件调试方式之一，几乎所有的交叉调试器都支持这种方式。

3）ROM 仿真器法

ROM 仿真器可看作是一种用于替代目标机上 ROM 芯片的硬件设备。它一边和宿主机相连，一边通过 ROM 芯片的插座和目标机相连。对于嵌入式处理器，它就像一个只读存储芯片；而对于宿主机上的调试器，它又像一个调试监控器。由于仿真器上的地址可以实时地映射到目标机的 ROM 地址空间中，所以在目标机上可以没有 ROM 芯片，而是用仿真器提供的 ROM 空间来代替。

实际上 ROM 仿真器是一种不完全的调试方式，它只是为目标机提供 ROM 芯片，并在目标机和宿主机之间建立了一条高速的通信通道。因此它经常和调试监控器法相结合，形成一种功能更强的调试方法。

与简单的监控器方法相比，ROM 仿真器的优点是：

- 在目标机上可以没有 ROM 芯片，因此也就不需要用其他的工具来向 ROM 中写入

数据和程序。

- 省去了为目标机开发调试监控器的麻烦。
- 由于是通过 ROM 仿真器上的串行接口、并行接口或网络接口与宿主机相连，所以不必占用目标机上通常很有限的资源。

4）在线仿真器法

在线仿真器（In Circuit Emulator，ICE）是一种用于替代目标机 CPU 的设备。对目标机来说，在线仿真器就相当于它的 CPU。事实上，ICE 本身就是一个嵌入式系统，有自己的 CPU、RAM、ROM 和软件。它的 CPU 比较特殊，可以执行目标机 CPU 的所有指令，但有更多的引出线，能将内部信号输出到被控制的目标机上。在线仿真器的存储器也可以被映射到用户的程序空间。因此，即使没有目标机，仅用 ICE 也可以进行程序的调试。

ICE 和宿主机一般通过串口、并口或网络相连。在连接 ICE 和目标机时，需要先将目标机的 CPU 取出，然后将 ICE 的 CPU 引出线接到目标机的 CPU 插槽上。在使用 ICE 来调试程序时，在宿主机上也有一个调试器用户界面。在调试过程中，这个调试器将通过 ICE 来控制目标机上的程序。

采用在线仿真器，可以完成如下的调试功能：

- 同时支持软件断点和硬件断点的设置。软件断点只能到指令级别，也就是说，只能指定程序在读取某一指令前停止运行。而在硬件断点方式下，多种事件的发生都可使程序在一个硬件断点上停止运行。这些事件不仅包括取指令，还包括内存读/写、I/O 读/写以及中断等。
- 能够设置各种复杂的断点和触发器。例如，可以让程序在“当变量 m 等于 100，同时 AX 寄存器等于 0”时停止运行。
- 能实时跟踪目标程序的运行，并可实现选择性的跟踪。在 ICE 上有大块 RAM，专门用来存储执行过的每个指令周期的信息，使用户可以得知各个事件发生的精确次序。
- 能在不中断被调试程序运行的情况下查看内存和变量，即非干扰的调试查询。

在线仿真器特别适用于调试实时应用系统、设备驱动程序以及对硬件进行功能测试。它的主要缺点就是价格昂贵，一般都在几千美元，有的甚至要几万美元。这显然阻碍了团队的整体开发，因为不可能给每位开发人员都配备一套在线仿真器。所以，现在 ICE 一般都用于普通调试工具解决不了的问题，或者用它来做严格的实时性能分析。

5）片上调试法

片上调试（On Chip Debugging，OCD）是 CPU 芯片提供的一种调试功能，可以把它看成是一种廉价的 ICE 功能。OCD 的价格只有 ICE 的 20%，但却提供了 80%的 ICE 功能。

最初的 OCD 是一种仿调试监控器方式，即将监控器的功能以微码的形式来体现，如

Motorola 的 CPU 32 系列处理器。后来的 OCD 摒弃了这种结构，采用了两级模式的思路，即将 CPU 的工作模式分为正常模式和调试模式。

当满足了特定的触发条件时，CPU 就可进入调试模式。在调试模式下，CPU 不再从内存读取指令，而是从调试端口读取指令，通过调试端口可以控制 CPU 进入和退出调试模式。这样在宿主机端的调试器就可以直接向目标机发送要执行的指令，通过这种形式调试器可以读/写目标机的内存和各种寄存器，控制目标程序的运行以及完成各种复杂的调试功能。

OCD 方式的主要优点是：不占用目标机上的通信端口等资源；调试环境和最终的程序运行环境基本一致；支持软硬件断点；提供跟踪功能，可以精确计量程序的执行时间；支持时序分析等功能。

OCD 方式的主要缺点是：调试的实时性不如 ICE 强；不支持非干扰的调试查询；使用范围受限，目标机上的 CPU 必须具有 OCD 功能。

目前比较常用的 OCD 的实现有：后台调试模式（Background Debugging Mode，BDM）、连接测试存取组（Joint Test Access Group，JTAG）和片上仿真器（On Chip Emulation，OnCE）等，其中 JTAG 是主流的 OCD 方式，OnCE 是 BDM 和 JTAG 的一种融合方式。

6）模拟器法

模拟器是一个运行在宿主机上的纯软件工具。它通过模拟目标机的指令系统或目标机操作系统的系统调用来达到在宿主机上运行和调试嵌入式程序的目的。

模拟器主要有两种类型：一类是在宿主机上模拟目标机的指令系统，称为指令级的模拟器；另一类是在宿主机上模拟目标机操作系统的系统调用，称为系统调用级的模拟器。指令级模拟器相当于在宿主机上建立了一台虚拟的目标机，该目标机的 CPU 种类与宿主机不同。例如，宿主机的 CPU 是 Intel Pentium，而虚拟机是 ARM、Power PC 或 MIPS 等。比较高级的指令级模拟器还可以模拟目标机的外部设备，如键盘、串口、网口和 LCD 等。系统调用级的模拟器相当于在宿主机上安装了目标机的操作系统，使得基于目标机操作系统的应用程序可以在宿主机上运行。两种类型的模拟器相比，指令级模拟器所提供的运行环境与实际的目标机更接近；而系统调用级的模拟器本身比较容易开发，也容易移植。

使用模拟器的最大好处是：可以在实际的目标机环境并不存在的条件下开发其应用程序，并且在调试时可以利用宿主机的资源来提供更详细的错误诊断信息。但模拟器也有许多不足之处，包括：

- 模拟环境与实际的运行环境差别较大，无法保证在模拟条件下调试通过的程序就一定能在真实环境下顺利运行。
- 不能模拟所有的设备。嵌入式系统中经常包含许多外围设备，但除了一些比较常见的设备之外，多数设备是不能模拟的。

- 实时性差。在使用模拟器调试程序时，被调试程序的执行时间和在真实环境中的运行时间差别较大。

尽管模拟器有许多不足，但是在项目开发的早期阶段，尤其是在还没有任何硬件可供使用时，模拟器还是非常有用的。对那些实时性不强，没有特殊外设，只需验证其逻辑的程序，用模拟器基本可以完成所有的调试工作。而且在使用模拟器调试程序时，不需要额外的硬件来协助，因此降低了开发成本。

4．软件工程工具

软件工程工具是指在分布式开发环境或大型嵌入式软件项目中使用的各种管理软件，如 CVS、GNU make 等。

1）CVS

CVS（Concurrent Version System）是一个版本控制软件，用来记录源码文件和其他相关文件的修改历史。对于一个文件的各个版本，CVS 只存储版本之间的区别，而不是把每个版本都完整地保存下来。当一个文件的内容发生变化时，CVS 会在一个日志中记录每一次修改的作者、修改的时间以及修改的原因。CVS 能够有效地管理软件的发行版本，以及多位程序员同时参与的分布式开发环境。它把一个软件项目组织成一个层次化的目录结构，里面包含了与项目有关的所有文件，如源文件、文档文件等。这些目录和文件合并起来，就构成了该软件项目的一个发行版本。

2）GNU make

GNU make 是一种代码维护工具，在大中型软件开发项目中，它将根据程序各个模块的更新情况，自动地维护和生成目标代码。make 的主要任务是读入一个文本文件（默认的文件名是 makefile 或 Makefile），并根据这个文件所定义的规则和步骤，完成整个软件项目的维护和代码生成等工作。在这个文本文件中，定义了一些依赖关系（即哪些文件的最新版本是依赖于哪些其他的文件）和需要用什么命令来产生文件的最新版本或管理各种文件。有了这些信息，make 会检查文件的修改或生成时间戳，如果目标文件的时间戳比它的某个依赖文件要旧，那么 make 就会执行 makefile 文件中描述的相应命令，来更新目标文件。make 工具的特点如下：

- 适合于文件较多的大中型软件项目的编译、连接、清除中间文件等管理工作；
- 只更新那些需要更新的文件，而不重新处理那些并不过时的文件；
- 提供和识别多种默认规则，方便对大型软件项目的管理；
- 支持对层状目录结构的软件项目进行递归管理；
- 对软件项目，具有渐进式的可维护性和扩展性。

5.2.3　集成开发环境

嵌入式软件开发环境起初主要由专门开发工具的公司提供，这些公司根据不同操作系统和不同处理器版本进行专门定制，如美国 Microtec 公司的交叉开发工具曾经被

VRTX、pSOS 等定制采用。随着用户对开发工具套件的需求增加，一些著名的操作系统供应商开始发展本系列操作系统产品的开发工具套件，如 WindRiver 公司的 Tornado、微软的 Windows CE 嵌入式开发工具包等。

在国际上，嵌入式软件开发环境的另一支研发队伍是 GNU。GNU 在因特网上提供免费的相关研究和开发成果，成为自主开发嵌入式软件开发环境的重要资源。一些公司已在 GNU 软件的基础上，经过集成、优化和测试，推出更加成熟、稳定的商业化嵌入式软件开发环境。

随着嵌入式系统的发展，嵌入式软件开发环境越来越重要，它直接影响到嵌入式软件的开发效率和质量。目前的开发环境已向开放性、集成化、可视化和智能化的方向发展，将各种类型且功能强大的软件工具，如编辑器、编译器、连接器、调试器、版本管理、用户界面等，有机地集成在一个统一的集成开发环境（Integrated Development Environment，IDE）中。

1．Tornado

Tornado 是 WindRiver 公司推出的一个集成开发环境。它由三个高度集成的部分组成：运行在宿主机和目标机上的交叉开发工具和实用程序；运行在目标机上的实时操作系统 VxWorks；用来连接宿主机和目标机的各种通信介质，如以太网、串口、在线仿真器 ICE 或 ROM 仿真器等。

Tornado 提供的交叉开发工具和实用工具主要有：源代码编辑工具、图形化的交叉调试工具、工程配置工具、集成仿真工具、诊断分析工具、C/C++编译工具、宿主机-目标机连接配置工具、目标机系统状态浏览工具、命令行执行工具、多语言浏览工具及图形化内核配置工具等。在 Tornado 中，宿主机上的工具与目标机之间的通信由目标服务器和目标代理共同完成。如图 5-4 所示，在形式上目标代理是 VxWorks 上的一个任务。调试命令通过宿主机上的目标服务器发送给目标代理。这些调试请求决定了目标代理应如何控制目标机上的其他任务。

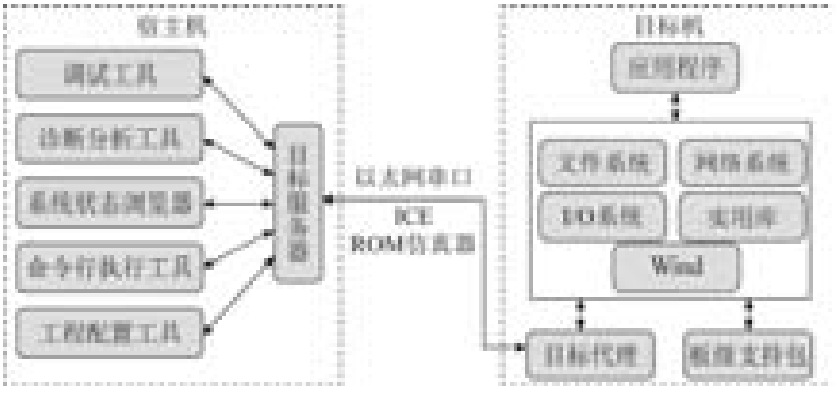

图 5-4　Tornado 环境中宿主机与目标机之间的关系

- 图形化的交叉调试器 CrossWind/WDB：支持任务级和系统级两种调试方式，支持混合源代码和汇编代码显示，支持多目标同时调试，具有良好的图形用户界面。
- 工程配置工具 Project：用于对 VxWorks 操作系统及其组件进行自动配置，进行依赖性分析和代码容量计算，自动生成 Makefile 文件。
- 集成仿真工具 VxSim：提供与真实目标机完全一致的调试和仿真运行环境。
- 诊断分析工具 WindView：一个图形化的动态诊断和分析工具，主要是向开发者提供在目标机上运行的应用程序的许多详细情况。
- C/C++编译工具：Tornado 提供以下支持 C 语言和 C++语言的工具和类库：Diab C/C++编译器、GNU C/C++编译器及 iostreams 类库。
- 宿主机-目标机连接配置工具 Launcher：位于 Tornado 环境的最上层，开发者可以通过它来设置开发环境。
- 目标机系统状态浏览工具 Browser：一个图形化工具，能随时提供目标系统的全面状态信息。
- 命令行执行工具 WindSh：一个功能强大的命令行解释器，可以直接解释、执行 C 语言表达式，调用目标机上的 C 函数及访问已在系统符号表中定义的变量。
- 多语言浏览工具 WindNavigator：浏览源程序代码，用图形化的方式显示函数调用关系，从而实现快速的代码定位。
- 图形化内核配置工具 WindConfig：通过 WindConfig 提供的图形向导，用户可以方便地配置 VxWorks 内核及其组件的参数。

Tornado 的特点：

（1）友好的开发环境。Tornado 可以运行在不同的系统中，支持 UNIX、Windows NT、Windows 98/95 等。

（2）适用于开发不同类型的目标机。针对不同的目标机，Tornado 为开发者提供了一个一致的图形接口和人机界面。这样，当开发人员转向新的目标机时，不必再花费时间去学习或适应新的开发工具。事实上，Tornado 的所有工具都驻留在开发平台上。

（3）工具齐备，具有丰富的交叉开发工具和实用工具。

（4）开放的、可扩展的开发环境。Tornado 是一个完全开放的环境，开发人员或第三方厂商可以很容易地把自己的工具集成到 Tornado 框架下。

2．Windows CE 应用程序开发工具

如图 5-5 所示，Windows CE 应用程序开发工具包括：①Platform Builder；②eMbedded Visual Tools；③eMbedded Visual Basic；④eMbedded Visual C++。它们都是专门针对 Windows CE 操作系统的开发工具。

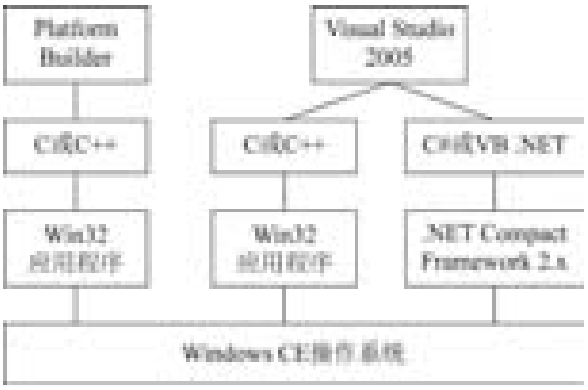

图 5-5　Windows CE 应用程序开发工具

Microsoft Windows CE Platform Builder 为开发商迅速创建一个嵌入式系统提供了全部相关工具。Platform Builder 集成开发环境使开发者能够对新一代高度模块化的设计进行配置、创建与调试，以实现嵌入式系统的灵活性与可靠性，并与 Windows 和 Web 功能特性紧密结合。它的特点主要包括：

- 通过使用改进的目标-宿主集成与连接特性来提高工作效率，节省嵌入式系统的创建时间。这些特性包括：集成化连接与下载、集成化目标控制、状态监视器、灵活的创建选择、简化操作系统配置等。
- 通过使用先进的系统级调试功能来提高调试速度。Platform Builder 的系统级调试器目前可为硬件辅助和系统级调试提供支持，从而大大扩展了调试功能的作用范围。具体包括：硬件辅助调试、源点级调试、新型的内核追踪器、远程系统信息、远程性能监视器、改进的调试器用户界面、调试区间等。
- 提供了一个新型扩展模型，可以帮助开发商将各类特性集成到开发环境当中。包括：微处理器控制单元、嵌入式开发工具控制单元等。

Microsoft eMbedded Visual C++和 eMbedded Visual Basic 是开发下一代 Windows CE 通信、娱乐及信息访问等应用程序时的功能强大的工具。它们所提供的全面高速应用程序开发环境能够帮助开发者在各类不同设备上，迅速就相关的 Windows CE 应用程序进行创建、调试和部署，并且在不牺牲控制功能、性能表现及有关灵活性的前提下，提高 Windows CE 的开发效率。这两个产品的特点主要包括：

- 开发工作效率比较高。这两个集成开发环境与传统的 Windows 应用开发环境非常相似，因此开发人员无需额外的培训即可熟练掌握它们的使用方法。而编程时的在线提示辅助功能（如语句完成、参数信息和语法错误检查等），能够大大提高程序员的工作效率。另外，通过创建可重复使用的 ActiveX 组件，可以将软件开发的复杂程度降至最低水平。
- 开发过程与集成化调试得以简化。允许 eMbedded Visual Tools 在编译完成后，从

移动设备或模拟器上的 IDE 中自动复制并启动相关的应用程序，从而实现应用程序的迅速测试和执行。在调试程序时，可以使用集成化调试器，当应用程序在 Windows CE 设备上（或模拟器内）运行的同时对其错误予以消除。另外，在测试时可以先在 Windows CE 设备模拟器上对应用程序进行测试，以避免高昂的硬件成本投资。

- 针对 Windows CE 平台的全面访问。包括 TCP/IP 通信机制、COM 组件模型、ActiveX 控件、设备的 API 接口函数等。
- 面向最新的 Windows CE 设备创建相应的解决方案。例如，Handheld PC Pro、Palm-size PC 及 Pocket PC 等 Windows CE 设备。
- 迅速、灵活的数据访问。数据存储可通过相关连接来实现与远程数据源之间的同步。

3．Linux 环境下的集成开发环境

在 Linux 环境下也有一些很好的集成开发环境，如 Kdevelop、Eclipse 和 Anjuta 等。

1）Kdevelop

Kdevelop 是 KDE 小组开发的 Linux/UNIX 操作系统上的 C/C++集成开发环境，为快速开发 C/C++应用程序提供了强有力的开发工具。

Kdevelop 的操作界面类似于微软的 Visual Studio，提供编辑、编译、连接、除错、版本管理及计划管理等基本的 IDE 功能。此外它还内建了一个可以产生 Qt 图形界面的资源编辑程序。对于 C++程序，它额外提供了类浏览器。此外其文件管理程序内建了所有有关 KDE 发展所需的文件，并提供搜寻的功能。

2）Eclipse

Eclipse 是替代 IBM Visual Age for Java（简称 IVJ）的下一代 IDE 开发环境，但它未来的目标不仅仅是成为专门开发 Java 程序的 IDE 环境。根据 Eclipse 的体系结构，通过开发插件，它能扩展到任何语言的开发，甚至能成为图片绘制的工具。目前，Eclipse 已经开始提供 C 语言开发的功能插件。而且它是一个开放源代码的项目，任何人都可以下载其源代码，并在此基础上开发自己的功能插件。

3）Anjuta

Anjuta 是 GNU/Linux 平台下的 C/C++集成开发环境，它主要是为了开发 GTK/GNOME 程序而设计的。Anjuta 利用 GLADE 来生成优美的用户界面，加之以自己强大的源程序编辑功能，使之成为应用程序快速开发的集成开发环境。以前，人们使用 GLADE 做界面，用 emacs 或 vi 来编辑源程序，再用某种终端模拟器编辑开发项目。而现在使用 Anjuta，所有这些繁杂零散的任务都可以在一个统一的、集成的、自然而然的环境下完成。

5.3 嵌入式软件开发

当确定完嵌入式软件开发环境后，就可以真正的进行嵌入式软件的开发，本节按照嵌入式平台选型、架构软件设计、软件设计方法、特性设计技术、编码、测试、下载和运行的顺序阐述嵌入式软件开发的流程。

5.3.1 嵌入式平台选型

按照常规的工程设计方法，嵌入式系统的设计可以分为 3 个阶段：分析、设计和实现。分析阶段是确定要解决的问题及需要完成的目标，也常常被称为需求阶段；设计阶段主要是解决如何在给定的约束条件下完成用户的要求；实现阶段主要是解决如何在所选择的硬件和软件的基础上进行整个软、硬件系统的协调和实现。在分析阶段结束后，开发者通常面临的一个棘手问题就是软硬件平台的选择，因为它的好坏直接影响着实现阶段的任务完成。

通常，硬件和软件的选择包括处理器、硬件部件、操作系统、编程语言、软件开发工具、硬件调试工具和软件组件等。

1．硬件平台的选择

嵌入式系统的核心部件是各种类型的嵌入式处理器。据不完全统计，目前全世界嵌入式处理器的品种总量已经超过 1000 多种，流行的体系结构有 30 多个系列。由于嵌入式系统设计的差异极大，因此选择是多样化的。

设计者在选择处理器时要考虑的因素主要有以下几个方面：

（1）处理性能：对于许多需用处理器的嵌入式系统来说，目标不是在于挑选速度最快的处理器，而是在于选取能够完成作业的处理器和 I/O 子系统。

（2）技术指标：当前许多嵌入式处理器都集成了外围设备的功能，减少了芯片的数量，降低了整个系统的开发费用。开发人员首先考虑的是：系统所要求的一些硬件能否无需过多的逻辑就连接到处理器上。其次考虑该处理器的一些支持芯片的配套。

（3）功耗：对于手持设备、PDA、手机等消费类电子产品，在选购微处理器时要求高性能、低功耗。

（4）软件支持工具：选择合适的软件开发和支持工具对系统的实现会起到至关重要的作用。

（5）是否内置调试工具：处理器如果内置调试工具，可以大大缩短调试周期，降低调试难度。

2．软件平台的选择

软件平台的选择涉及到操作系统、编程语言和集成开发环境 3 个方面。

1）操作系统

编写嵌入式软件有两种选择：一是自己编写内核；二是使用现成的操作系统。如果嵌入式软件只需要完成一项非常小的工作，例如在电动玩具、空调中，就不需要一个功能完整的操作系统。但如果系统的规模较大、功能较复杂，那么最好还是使用一个现成的操作系统。可用于嵌入式系统软件开发的操作系统有很多，但关键是如何选择一个适合开发项目的操作系统，可以从以下几点进行考虑：

（1）操作系统提供的开发工具。有些实时操作系统只支持该系统供应商的开发工具，因此，还必须从操作系统供应商处获得编译器、调试器等；而有的操作系统应用广泛，且有第三方工具可用，因此选择的余地比较大。

（2）操作系统向硬件接口移植的难度。操作系统到硬件的移植是一个重要的问题，是关系到整个系统能否按期完工的一个关键因素。因此，要选择那些可移植性程度高的操作系统，以避免因移植带来的种种困难。

（3）操作系统的内存要求，有些操作系统对内存有较大要求。

（4）操作系统的可剪裁性、实时性能等。

2）编程语言

尽管高级语言能够完成大部分的嵌入式软件开发工作，但汇编语言仍然不可替代。汇编语言可以直接对硬件进行操作，代码效率高，所以经常应用在系统移植以及直接控制硬件的场合。此外，良好的汇编基础也有助于程序的调试。

越是高级的语言，其编译和运行的系统开销就越大，应用程序也越大，运行越慢。因此一般来说，编程人员都会首选汇编语言和 C 语言，然后才会考虑 C++语言或 Java 语言。

3）集成开发环境

集成开发环境是进行开发时的重要平台，在选择时应考虑以下因素：

（1）系统调试器的功能，包括远程调试环境。

（2）支持库函数。许多开发系统提供大量的库函数和模板代码，如 C++编译器就带有标准的模板库。与选择硬件和操作系统的原则一样，应尽量采用标准的 glibc（GNU 标准 C 库函数）。

（3）连接程序是否支持所有的文件格式和符号格式。

5.3.2　软件设计

1. 软件设计的任务

在给定系统的需求规格说明书后，需要对软件的结构进行设计，并对设计的过程进行管理。在嵌入式系统的软件设计过程中，需要完成以下一些任务。

1）准备工作计划

在软件设计之前，首先要制订详细的工作计划，其内容包括：

- 过程管理方案：包括软件开发的进度管理、软件规模和所需人年的估算、开发人员的技能培训等；
- 开发环境的准备方案：包括开发工具的准备、开发设备的准备、测试装备的准备、分布式开发环境下的开发准则等；
- 软硬件联机调试的方案：联调的起始时间、地点、人员和具体的准备工作；
- 质量保证方案：包括质量目标计划、质量控制计划等；
- 配置控制方案：包括配置控制文档的编写、配置控制规则的制订等。

2）确定软件的结构

设计软件的各个组成部分，包括：

- 任务结构的设计：使用操作系统提供的函数，设计出一个最佳的任务结构；
- 线程的设计；
- 公共数据结构的设计：在确保系统一致性的基础上，设计出所需的公共数据；
- 操作系统资源的定义；
- 类的设计；
- 模块结构设计：在设计时要充分考虑模块的划分、标准化、可重用和灵活性等；
- 内存的分配与布局。

3）设计评审

对于软件设计的结果，进行一次设计评审，并在必要时对设计进行修正。具体内容包括：

- 确认每件工作的执行方法是否恰当，其内容是否完善；
- 确认该设计完成了系统需求规格说明书所要求的功能和服务；
- 评估任务结构设计、评估类的设计、评估模块结构设计；
- 对软件设计的结果进行总结，编写出相应的文档。

4）维护工作计划

执行软件设计工作控制，在每日、每周和每月的时间粒度上对进度进行控制，确保软件设计能够如期完成。

5）与硬件部门密切合作、相互协调

根据工作计划中的安排，定期与硬件部门召开会议，协调各自的进展。如果软件规格说明书发生了变化，立即进行调整，重新进行软件设计。

6）控制工作的结果，把工作记录存档

掌握当前的工作进展情况，尽早地发现和分析问题，并采取相应的措施。对各种事件进行跟踪记录，包括：

- 执行过程控制，跟踪进展情况并定期记录、存档。
- 执行质量控制，保留质量记录。
- 记录产品的配置、版本变化、bug 的发现和处理等信息。

2．软件架构设计

软件架构也称为软件体系结构，需要考虑如何对系统进行分解，对分解后的组件及其之间的关系进行设计，满足系统的功能和非功能需求。软件架构形成过程如图 5-6 所示。

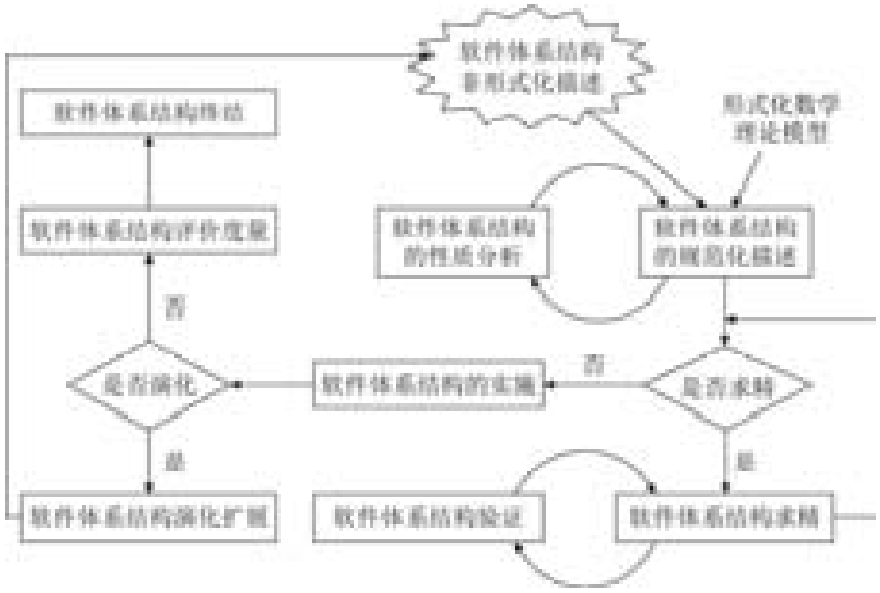

图 5-6　架构的形成过程概要

软件架构设计需要从用户业务需求、未来应用环境、需求分析、硬件基础、接口输入、数据处理、运算或控制规律、用户使用等方面进行综合、权衡和分析基础上产生。面向某种问题的架构一旦确定就很难改变，随后的架构设计需要通过一系列的迭代开发完善，使得软件架构日趋成熟、稳定。

软件架构的重要作用也在于控制一个软件系统的使用、成本和风险。好的架构要求是和谐的软件架构，包括与上一级系统架构相互和谐、与系统中同一级的其他组件架构互相和谐，确保系统满足性能、可靠性、安全性、信息安全性和互操作性等方面的关键要求，也具有可扩展、可移植性，从而为一个软件带来长久的生命力。

在大量开发实践中，有很多广泛使用并被普遍接受的软件架构设计原则，这些原则独立于具体的软件开发方法，主要包括抽象、信息隐藏、强内聚和松耦合、关注点分离等。

（1）抽象：这是软件架构的核心原则，也是人们认识复杂客观世界的基本方法。抽象的实质是提取主要特征和属性，从具体的事务中通过封装来忽略细节，并且运用这些特征和属性，描述一个具有普遍意义的客观世界。软件架构设计中需要对流程、数据、行为等进行抽象。复杂系统含有多层抽象，从而有多个不同层次架构。

（2）信息隐藏：包括局部化设计和封装设计。局部化设计就是将一个处理所涉及到的信息和操作尽可能地限制在局部的一个组件中，减少与其他组件的接口。而封装设计是将组件的外部访问形式尽可能简单、统一。

（3）强内聚和松耦合：强内聚是指软件组件内的特性，即组件内所有处理都高度相关，所有处理组合在一起才能组成一个相对完整的功能。而松耦合是指软件组件之间的特性，软件组件之间应尽量做到没有或极少的直接关系，使其保持相对独立，这样使得未来的修改、复用简单，修改之后带来的影响最小。

（4）关注点分离：所谓关注点是软件系统中可能会遇到的多变的部分。如为适应不同运行接口条件，需要进行适应性的参数调整和驱动配置。关注点分离设计是将这部分组件设计成为相对独立的部分，使未来的系统容易配置和修改。而核心的部分可以保持一个相对独立的稳定状态。如果功能分配使得单独的关注点组件足够简单，那么就更容易理解和实现。但“展示某些关注点得到满足时，可能会影响到其他方面的关注点，但架构师必须能够说明所有关注点都已得到满足”。

以上的原则中，删除需求细节或对细节进行抽象是最重要的工作，为用户的需求创建抽象模型，通过抽象将特殊问题映射为更普遍的问题类别，并识别各种模式。

软件架构设计使用纵向分解和横向分解两种方式。纵向分解就是分层，横向分解就是将每一个层面分成相对独立的部分。经过分解之后，可以将一个完整的问题分解成多个模块来解决。模块是其中可分解、可组装，功能独立、功能高度内聚、之间低耦合的一个组件。

类似于建筑架构，软件架构也决定了软件产品的好用、易用、可靠、信息安全、可扩展、可重用等特性，好的软件架构也给人完整、明确、清晰等赏心悦目的感觉，具有较长的生命力。

架构设计是围绕业务需求带来的问题空间到系统解决空间第一个顶层设计方案。按照抽象原则，在这个阶段进行的架构设计关注软件设计环节抽象出来的重要元素，而不是所有的设计元素。在架构设计时将软件这些要素看作是黑盒，架构设计需要满足黑盒的外部功能和非功能需求的目标。一个软件的架构设计首先为软件产品的后续开发过程提供基础，在此基础上可将一个大规模的软件分解为若干子问题和公共子问题。而一般意义的软件设计是软件的底层设计，开发人员需要关注各子问题或要素的进一步分解和实现，是根据架构设计所定义的每个要素的功能、接口，进一步实现要素组件内部的配置、处理和结构。在遵守组件外部属性前提下，考虑实现组件内部的细节及其实现方法。对于其中的公共子问题，形成公共类和工具类，从而可以达到重用的目的。

一般的软件构架是根据需求自上而下方式来设计，即首先掌握和研究利益相关方的关键需求，基本思路是首先进行系统级的软件架构设计，需要将软件组件与其外部环境属性绑定在一起，关注软件系统与外部环境的交联设计；其次将一个大的系统划分成各组成部分，这些部分可以按照架构设计的不同方法，分为层次或成为模块；之后再开始

研究所涉及到的要素，再实现这些要素以及定义这些要素之间的关系。

在实际工作中，软件构架也可采用自底向上的方法，前提是已经建立了一个成熟稳定的软件架构，也可以称之为“模式”。模式是组织一级设计某一类具体问题的顶层思路，是为了解决共有问题解的方案模板，但并不是一个问题的设计或设计算法。

模式常常整合在一起使用，提供解决更大、更复杂问题的解决方案，而组成一个解决问题的通用框架。框架往往提供统一平台和开发工具，而且已经高效地利用了已经经过验证的模式、技术和组件。在新软件系统的设计中指定沿用或重用这种架构框架，这时其他重要元素可以在这个架构基础上针对新的需求进行扩展，有时是针对性地进行参数化设计。所以在架构设计中可以借用模式的概念进行设计，采用成熟的先进的设计框架和工具提高开发的效率，保证设计正确性。

图 5-7 所示是针对架构设计中非功能需求的多维度分析，从中可知任何一个因素的变化都会带来对其他因素的影响。实际上软件架构设计属于软件设计过程的一部分，但超越了系统内部的算法和数据结构的详细设计。

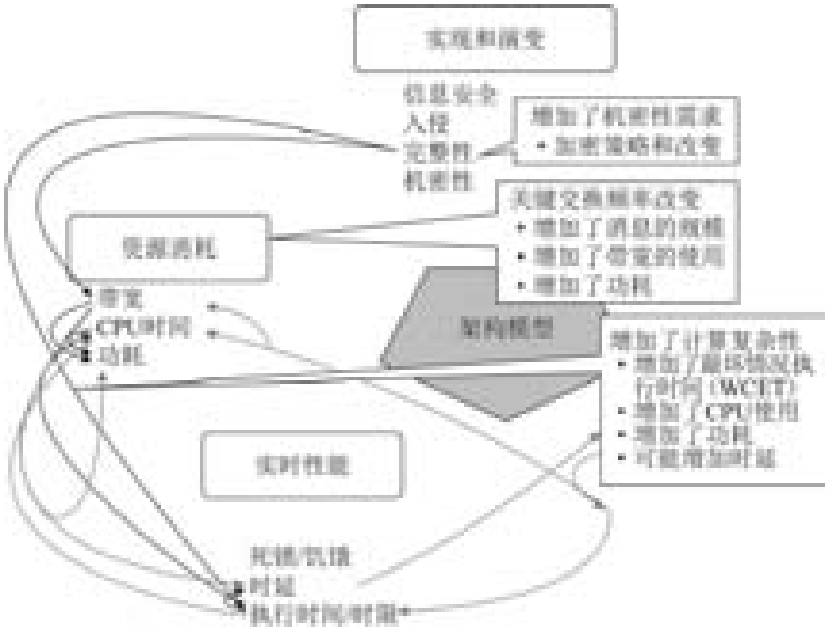

图 5-7　架构的多维度分析

在架构设计阶段，需要定义边界条件、描述系统组织结构、对系统的定量属性进行约束、帮助对模型进行描述并基本构造早期的原型、更准确地描述费用和时间的评估。

3．软件设计方法

在将系统分解为各个组件的过程中，需要采取不同的策略，而每个策略则关注不同的设计概念。根据分解过程中所采用的不同策略，设计方法有基于功能分解的设计方法、

基于信息隐藏的设计方法和基于模型驱动开发的设计方法等分类。

（1）基于功能分解的设计方法。实时结构化分析与设计采用了功能分解，系统被分解为多个函数，并且以数据流或控制流的形式定义函数之间的接口；基于并发任务结构化的设计（Design Approach for Real-Time Systems，DARTS）提供了任务结构化标准，辅助人员确定系统中的并发任务，并指导定义任务接口。

（2）基于信息隐藏的设计方法。面向对象（Object Oriented，OO）设计方法将数据和数据上操作封装在对象实体中，对象外界不能够直接对对象内部进行访问和操作，只能通过消息间接访问对象，符合人类思维方式，提高软件的扩展性、维护性和重用性。

（3）基于模型驱动开发的设计方法。通过借助有效的（Model Driven Development，MDD）工具，构建和维护复杂系统的设计模型，直接产生高质量的代码，将开发的重心从编码转移到设计。当前使用较为广泛的 MDD 工具有 IBM 公司的 Rhapsody。

5.3.3 特性设计技术

由于嵌入式系统的特殊性，嵌入式软件除正常功能要求外，在实时性、安全性、可靠性、可扩展性、可定制性、标准符合性等方面都有要求。下面简要介绍实时性、可扩展性、可定制性等方面的设计技术。

1．实时性的设计

嵌入式应用通常都有实时性的要求，即系统部分功能需要满足时间的限制。根据系统对时间的要求不同，可分为软实时要求和硬实时要求。对于软实时要求的系统，希望系统运行越快越好，不局限于特定任务在多长时间完成。对于硬实时系统，则有明确的任务执行时限的要求，如果系统运行不能满足该要求，则必须采取处理措施。

通常情况下，嵌入式系统对于硬实时和软实时都有要求，是两者的结合。在进行软件实时性设计时，需要考虑如下几个方面因素：

（1）通过合理划分实时单元和分时单元，提高系统的实时性能。例如，对于信号处理系统，对于信号的翻译、解释、转移、传递和应答是实时单元，通常要放到实时任务中处理。而对于运行过程中信息输出或故障信息记录，可以是分时单元，可以利用操作系统数据通信机制传递信息，或直接采用共享内存方式传递数据，由分时任务或系统后台任务进行数据处理。

（2）合理划分系统中的实时任务，提高系统运行效率、实时性和吞吐量。任务是实时系统运行的调度单元，具体管理系统中的各类资源，可以使用或等待 CPU、存储空间或 I/O 设备等。任务彼此之间按照系统的调度策略执行，对于没有操作系统的系统，需自行编写调度算法。通常，任务与函数形式差异不大，但有确定的任务入口点、私有数据区、以及主体结构，表现为循环体或明确的中止状态（任务没有返回值）。任务划分粗细程度，对系统影响较大，划分过细则会引起任务频繁切换，如果划分不彻底，又会造成原本可并行的操作只能串行开展。为了达到效率和吞吐量之间的平衡与折衷，应遵循

一定的任务分解规则（假设下述任务的发生都依赖于唯一的触发条件，如果两个任务能够满足下面的条件之一，则可以将其合理地分开）：

- 时间：两个任务所依赖的周期条件具有不同的频率和时间段。
- 异步性：两个任务所依赖的条件没有相互的时间关系。
- 优先级：两个任务所依赖的条件需要有不同的优先级。
- 清晰性/可维护性：两个任务可以在功能上或逻辑上相互分开。

同时，在设计过程中尽量减少模块（任务）之间的数据通信，特别是控制耦合（即一个任务可控制另一个任务的执行流程或功能），如果必须出现，应采取相应措施（如，任务间通信）实现彼此之间的同步或互斥，以避免可能引起的临界资源冲突，防止死锁现象出现。

（3）在程序设计上采取措施，提高程序执行效率。比较常用的优化手段有：系统关中断/关调度范围尽可能最小化，采用简短的中断服务程序，循环体工作量最小化，将频繁使用的变量设置为寄存器变量，采用经典高效的算法（如查找、排序）等。

2．可扩展性的设计

出于系统的升级、维护和重用的考虑，对软件的可扩展性提出了要求，需要在设计过程中规划好系统的架构，并采用模块设计方法实现。例如，在面向航空领域应用的特定系统，其系统功能由软硬件共同实现，如何区分软硬件功能分配以及两者之间的界限非常重要，同时应用中涉及大量驱动软件、功能组件，就需要对这些部分进行模块化设计。

1）采取混合编程的方式

混合编程模式是指同时利用汇编语言和高级语言进行嵌入式软件设计。这主要是利用汇编语言对硬件操作的方便性和汇编语言的高效执行特点来编写与硬件紧密相关，或实时要求严格的代码；而利用高级语言接近人的思维、处理逻辑关系功能强大的优势编写逻辑功能函数。

通常，高级语言实现的代码比汇编语言实现相应功能的代码具有更好的移植性。在系统设计时，要从系统整体的性能和效率考虑，合理分配高级语言和汇编语言的实现比例，不过分强调某一方面而忽略另一方面。

2）硬件驱动管理机制

嵌入式领域软件开发时，常常需要开发大量的硬件设备驱动软件，包括各类处理器的驱动、各类外部设备的驱动软件。由于驱动软件需要与硬件进行深度结合，并且部分是采用汇编语言实现，如果没有合理硬件驱动管理机制的支持，很难做到软件在不同硬件平台上的迁移与扩展。

类似于 Windows 系统的硬件设备驱动，在嵌入式系统的软件开发中引入了硬件驱动层，对系统运行的各类设备驱动进行封装。上层的系统软件通过标准的接口进行访问，实现系统软件与硬件的隔离，降低系统软件的开发难度，缩短了开发时间。硬件驱动层

包括 CPU 片内资源的硬件驱动和板子上外围设备硬件的驱动，如图 5-8 所示。

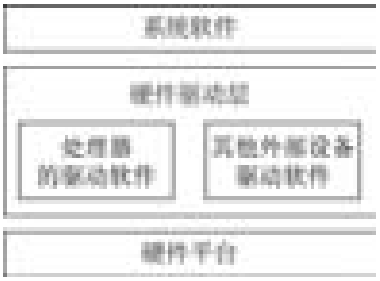

图 5-8　硬件驱动层结构

CPU 的硬件驱动通常包括寄存器、时钟、中断、异常、存储管理单元等。在系统引导过程中，要配置好各种寄存器，进行时钟、中断、异常、存储管理等部件初始化，后续系统软件则通过子程序的方式调用相关硬件驱动。

其他外部设备驱动通常包括串口、网口、鼠标、键盘、存储器等。在系统开发时，需要为目标机所有的外部设备逐一编写驱动程序，以供其他应用软件进行调用。例如键盘，硬件会记录每次按下的键码，放入输入键码队列中，编制的驱动程序即从键码队列中取出按下的键，根据键值的不同执行不同的操作。通过编制硬件驱动层，并通过标准的接口向上提供访问，使得上层软件的编写就与硬件无关了，只要软件之间逻辑关系正确，就不需要改动。即便是驱动程序需要移植到其他硬件上，只要硬件设计基本相同，也可以直接重用硬件驱动程序，使得整个软件方便地移植。

3）软件的模块化设计

模块化设计方法适用于通用软件和嵌入式软件的设计，是提高系统可扩展性和软件复用性的通用方法。模块化设计包括四个方面：模块、数据、体系和程序设计。

模块设计降低了系统的复杂性、使得系统易于修改、且支持了各部分的并行开发。针对模块的操作特性，则是通过时间历史、激活机制和控制模式进行体现。在程序结构内部，模块可分类：

（1）顺序模块，由应用程序进行引用和执行，运行过程中不能被打断。

（2）增量模块，运行过程中可被其他应用程序打断，而后再从断点重新开始。

（3）并行模块，在多处理器环境下可以与其他模块同时执行。

由于单个模块的功能已被划分出来，开发起来相对较为容易，更多要从独立性的角度关注模块之间的界限。功能的独立性可以使用内聚性和耦合性这两个特性要素进行衡量：内聚性衡量模块功能强度的相关性，耦合性衡量模块间的相互依赖的相关性。

数据设计至关重要，并被有些人认为是最重要的设计行为。数据设计主要取决于数据结构的设计和程序复杂性的设计，通常采用如下办法保证数据设计的质量：

（1）用于功能和行为分析的系统分析原理也适用于数据。

（2）模块涉及的数据结构及基于数据结构的操作应被确定。

（3）创建数据词典并用来详细说明数据和程序的设计。

（4）较低层次的数据设计策略延迟至设计过程的后期。

（5）数据结构的信息应只被需要使用此结构内数据成员的模块知道。

（6）可在适当时候借鉴有用的数据结构和操作库。

（7）设计和编程语言应支持抽象数据类型的规范和实现。

体系设计的主要目标是定义适合模块化开发的程序结构，并描述出模块之间的控制相关性。体系设计应融合程序结构与数据结构，并对数据在程序中流动的界限进行定义。体系设计要关注系统的整体设计而不是单独组件。进行体系设计有许多不同的方法，但这些方法无一例外都是从软件的全局性出发，逐步接近设计的原点。

过程设计通常是在数据、程序结构及算法被确定后（通常是类似英语的自然语言），再进行程序过程设计。

3．可定制性的设计

系统的可定制性通常包括可剪裁性和可配置性两个方面。

1）可剪裁性

系统的开发者需要完成系统的全集，而系统的使用者由于资源限制或使用需求只要求获得系统的子集，对于不需要的部分要进行剪裁。系统的剪裁性取决于模块之间的耦合度，耦合度越小的系统，剪裁力度越大。对于嵌入式系统，通常可以剪裁到只包含如下内容：

（1）一个用作引导的可用设施；

（2）一个具备任务管理和定时功能的最基本内核；

（3）一个初始任务。

例如，一般的嵌入式系统设计中，通常是按照功能对代码进行了细致地划分，抽象出一部分公共函数作为实现其他功能的基础，不可被剪裁，其他功能代码之间由于比较独立，具有良好的可剪裁性。但是，由于操作系统本身的管理功能要求，类似任务管理和定时功能这些从技术上看可以剪裁的模块，必须保留在内核之中。

2）可配置性

用户对于已经选择的模块，需要通过配置进一步调整系统的功能和规模。通常在系统设计时要定义一组参数来实现对软件规模的配置，也可以通过系统调用中的参数对系统功能进行配置。

5.3.4　嵌入式软件的设计约束

嵌入式软件在设计过程要遵循一些国家标准、军用标准、行业标准、企业标准或特定型号标准，在这些标准中通常会对软件的设计提出一些约束，包括模块接口设计约束、中断设计约束、异常设计约束、数据安全设计约束、余量设计约束及其他方面的设计约束。

1．接口设计约束

（1）模块的参数定义应与该模块接受的输入参数定义一致，包括参数的个数、属性、单位和次序。

（2）传送给被调用模块的参数定义应与该模块的参数定义一致，包括参数的个数、属性、单位和次序。

（3）模块内部函数调用时，参数的个数、属性、单位和次序一致。

（4）不能对仅作为输入值的参数进行修改。

（5）全局变量在所有引用它们的模块中有相同的定义。

（6）模块间传递的参数个数不超过 5 个。当需要传递的参数过多时，采用结构体传递参数。

（7）模块间的数据通信可采用操作系统提供的通信接口实现。

2．中断设计约束

（1）系统初始化阶段屏蔽无用中断，并对无用中断设置入口并返回。

（2）中断初始化要初始化中断所需的全部资源，包括中断向量号、触发方式、中断服务程序等。

（3）分析系统出现的假中断或频繁中断的影响，给出处理措施。

（4）程序中开关中断位置要仔细分析，避免关闭范围过大或过小。

（5）尽量避免使用中断嵌套的功能，进入中断后关掉不希望嵌套的中断。

（6）避免在中断服务程序中使用跳转语句或子程序返回直接跳出中断。

（7）与操作系统配合完成中断现场的保存和恢复，对于操作系统未处理的现场，在用户连接的中断处理程序中要辅助保存和恢复。

（8）应考虑不同优先级的中断处理程序，以及中断处理与普通程序之间临界区保护问题。

（9）中断处理程序尽可能简短，不要在中断处理程序中调用操作系统的资源申请或时间等待服务。

3．模块设计约束

（1）除中断服务程序外，模块应采用单入口和单出口的控制结构。

（2）控制模块的扇入扇出数。将模块在逻辑上划分为层次结构，并在不同层次上定义不同扇入扇出。

（3）通过提高模块内聚度和降低耦合度，提高模块独立性。通常采用模块变量局部化、限制模块间参数传递、采用模块调用等方式实现。

（4）对于模块间耦合方式，按照数据耦合、控制耦合、外部耦合、公共数据耦合、内容耦合的优先顺序进行处理。

（5）对于模块内聚，按照功能内聚、顺序内聚、通信内聚、时间内聚、逻辑内聚、偶然内聚的优先顺序进行处理。

（6）禁止使用递归设计。嵌入式系统的资源有限，任务栈通常不会很大，递归调用容易产生栈溢出问题。

4．异常设计约束

（1）软件设计时应对所有可能发生的异常状态进行接管。

（2）设计或使用统一的异常处理机制，使得发生异常时系统能够转入安全状态。

（3）要充分分析外购软件或重用软件中的异常处理，是否与系统的异常处理相适应。

5．数据安全设计约束

（1）规定数据的合理范围，包括值域、变化速率。如果数据超出范围，应进行出错处理。

（2）数值运算时注意数值范围及误差，保证输入、输出及中间计算结果不超过机器数值表示范围。

（3）保证运算所要求的精度，充分考虑到计算误差、舍入误差，选定足够的数据有效位。

（4）在软件入口、出口和关键点上，对重要物理量范围进行合理性检查，并定义出错时的隔离措施。

（5）考虑浮点数接近零时的处理方式，避免下溢。避免对于浮点数进行相等关系的判断。

（6）定期检查存储器、指令和数据总线。测试指令序列的设计必须确保单点或很可能的复合失效能够被检测出来。加载时必须进行数据传输的检查及程序加载验证检查，并在此后定期进行，以确保安全关键代码的完整性。

6．余量设计约束

（1）在资源分配和余量要求上，应确定软件模块存储量（包括内存、固存）、输入/输出吞吐率及处理时间，满足系统规定的余量要求。通常系统未确定情况下，一般应留有不少于 20%的余量。

（2）在时间安排和余量要求上，应根据被控对象确定各种周期，包括控制周期、数据处理周期、采样周期、自检测周期、输入/输出周期等。在同一个时间轴上进行各个周期的安排，当安排不下时，应提高系统硬件处理能力或采用并行处理方式。

7．其他设计约束

（1）如果应用任务对系统响应时间要求非常高，应使用抢占式的调度方法。抢占式调度总能保证最高优先级的任务在运行，并且该最高优先级任务的执行是确定的，其任务级响应时间可以最小化。

（2）如果系统采用并发处理方式，那么就应该考虑函数的可重入性。正在执行非可重入函数的任务被抢占后，可能会导致数据的破坏。采用可重入函数可以解决上述问题。可以通过仅使用局部变量（CPU 寄存器变量或栈中的变量），或者在使用全局变量时对其进行互斥保护来实现可重入。

（3）如果软件需要使用互斥机制，那么优先考虑使用操作系统提供的互斥机制。当多个任务间通过共享数据结构进行通信时，需要实现互斥，以保证数据不会被破坏。如果选用了操作系统，优先考虑使用操作系统提供的互斥机制，如互斥信号量等。

（4）任务与任务之间、任务与中断之间需要访问共享资源时应使用互斥机制。共享资源包括全局数据、端口、内存地址等，对这些共享资源进行访问需采用互斥机制。

（5）调用函数后，检查返回值。不应认为调用的函数一定是正确执行的，建议调用后立即检查其返回值，并做相应处理。

（6）对于计算系统的安全关键子系统编写故障检测和隔离程序。故障检测程序必须设计成在这些有关安全关键功能执行之前检测潜在的安全关键失效。故障隔离程序必须设计成将故障隔离到实际的最低级，并向操作员或维护人员提供这个信息。

（7）通过数据隐藏的方式实现全局数据的保护。借鉴面向对象设计中数据封装的思想，将全局数据封装为抽象数据类型，不允许任务直接访问该全局数据，而是通过与该全局数据配套的函数来访问，并且在访问该全局数据时进行互斥保护。

（8）仅在软件初始化时完成空间分配。软件正常状态不释放内存。动态的内存释放与分配会导致产生内存碎片，可能存在空闲空间总和满足要求，却申请不到内存的情况，从而导致软件行为的不确定性。

（9）变量使用前应初始化。特别是静态变量，不能依赖编译器对其进行初始化设置，应将其人工设置为特定的值。

（10）应将硬件设备初始化为确定状态。不能想当然地认为硬件设备上电后就处于某种状态，而应该人工将其设定为某种状态。

（11）延时设计应避免采用循环的方法。采用循环方法延时在代码优化时可能出现错误。建议借助硬件高精度时钟实现延时。如由于硬件条件限制只能采用循环延时方法，应考虑编译器优化和硬件特性问题，并在软件设计文档中进行特别说明，如增加防止编译优化，及要求使用何种硬件。

（12）应确保延时使用的时钟精度满足延时误差的要求。

5.3.5 编码

1．编码过程

在给定了软件设计规格说明书后，下一步的工作就是编写代码。一般来说，编码工作可以分为四个步骤：

（1）确定源程序的标准格式，制订编程规范。

（2）准备编程环境，包括软硬件平台的选择，包括操作系统、编程语言、集成开发环境等。

（3）编写代码。

（4）进行代码审查，以提高编码质量。为提高审查的效率，在代码审查前需要准备

一份检查清单，并设定此次审查须找到的 bug 数量。在审查时，要检查软件规格说明书与编码内容是否一致；代码对硬件和操作系统资源的访问是否正确；中断控制模块是否正确等。

2．编码准则

在嵌入式系统中，由于资源有限，且实时性和可靠性要求较高，因此，在开发嵌入式软件时，要注意对执行时间、存储空间和开发/维护时间这三种资源的使用进行优化。也就是说，代码的执行速度要越快越好，系统占用的存储空间要越小越好，软件开发和维护的时间要越少越好。

具体来说，在编写代码时，需要做到以下几点：

- 保持函数短小精悍。一个函数应该只实现一个功能，如果函数的代码过于复杂，将多个功能混杂在一起，就很难具备可靠性和可维护性。另外，要限制函数的长度，一般来说，一个函数的长度最好不要超过 100 行。
- 封装代码。将数据以及对其进行操作的代码封装在一个实体中，其他代码不能直接访问这些数据。例如，全局变量必须在使用该变量的函数或模块内定义。对代码进行封装的结果就是消除了代码之间的依赖性，提高了对象的内聚性，使封装后的代码对其他行为的依赖性较小。
- 消除冗余代码。例如，将一个变量赋给它自己，初始化或设置一个变量后却从不使用它，等等。研究表明，即使是无害的冗余也往往和程序的缺陷高度关联。
- 减少实时代码。实时代码不但容易出错、编写成本较高，而且调试成本可能更高。如果可能，最好将对执行时间要求严格的代码转移到一个单独的任务或者程序段中。
- 编写优雅流畅的代码。
- 遵守代码编写标准并借助检查工具。用自动检验工具寻找缺陷比人工调试便宜，而且能捕捉到通过传统测试检查不到的各种问题。

3．编码技术

1）编程规范

在嵌入式软件开发过程中，遵守编程规范，养成良好的编程习惯，这是非常重要的，将直接影响到所编写代码的质量。

编程规范主要涉及的三方面内容：

- 命名规则。从编译器的角度，一个合法的变量名由字母、数字和下画线三种字符组成，且第一个字符必须为字母或下画线。但是从程序员的角度，一个好的名字不仅要合法，还要载有足够的信息，做到“见名知意”，并且在语意清晰、不含歧义的前提下，尽可能地简短。
- 编码格式。在程序布局时，要使用缩进规则，例如变量的定义和可执行语句要缩进一级，当函数的参数过长时，也要缩进。另外，括弧的使用要整齐配对，要善

于使用空格和空行来美化代码。例如，在二元运算符与其运算对象之间，要留有空格；在变量定义和代码之间要留有空行；在不同功能的代码段之间也要用空行隔开。

- 注释的书写。注释的典型内容包括：函数的功能描述；设计过程中的决策，如数据结构和算法的选择；错误的处理方式；复杂代码的设计思想等。在书写注释时要注意，注释的内容应该与相应的代码保持一致，同时要避免不必要的注释，过犹不及。

2）性能优化

由于嵌入式系统对实时性的要求较高，因此一般要求对代码的性能进行优化，使代码的执行速度越快越好。以算术运算为例，在编写代码时，需要仔细地选择和使用算术运算符。一般来说，整数的算术运算最快，其次是带有硬件支持的浮点运算，而用软件来实现的浮点运算是非常慢的。因此，在编码时要遵守以下准则：

- 尽量使用整数（char、short、int 和 long）的加法和减法。
- 如果没有硬件支持，尽量避免使用乘法。
- 尽量避免使用除法。
- 如果没有硬件支持，尽量避免使用浮点数。

图 5-9 是一个例子，其中两段代码的功能完全一样，都是对一个结构体数组的各个元素进行初始化，但采用两种不同的方法来实现。图 5-9（a）采用数组下标的方法，在定位第 i 个数组元素时，需要将 i 乘以结构体元素的大小，再加上数组的起始地址。图 5-9（b）采用的是指针访问的方法，先把指针 fp 初始化为数组的起始地址，然后每访问完一个数组元素，就把 fp 加 1，指向下一个元素。在一个奔腾 4 的 PC 上，将这两段代码分别重复 10 700 次，右边这段代码需要 1ms，而左边这段代码需要 2.13ms。

```
struct {
    int a;
    char b;
    int c;
} foo[10];
int i;
for(i = 0; i < 10; ++i)
{
    foo[i].a = 77;
    foo[i].b = 88;
    foo[i].c = 99;
}
```

（a）乘法

```
struct {
    int a;
    char b;
    int c;
} *fp, *fend, foo[10];
fend = foo + 10;
for(fp = foo; fp != fend; ++fp)
{
    fp->a = 77;
    fp->b = 88;
    fp->c = 99;
}
```

（b）加法

图 5-9　算术运算性能优化的例子

5.3.6　下载和运行

如前所述，嵌入式应用软件采用宿主机/目标机模式来开发，然后通过串口或网络等通信线路，将交叉编译生成的目标代码传输并装载到目标机上，在监控程序或操作系统的支持下利用交叉调试器进行分析和调试，最后目标机可以在特定环境下脱离宿主机单独运行，如图 5-10 所示。

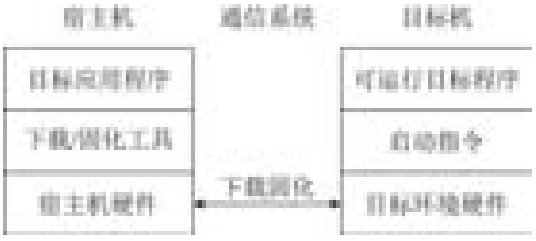

图 5-10　嵌入式应用程序下载/固化

根据嵌入式系统硬件的配置情况，固化的方式有多种，可以固化在 EEPROM 和 FLASH 这类存储器中，或者固化在 DOC 和 DOM 等电子盘中。比较常见的方式还是使用编程器将二进制映像文件写入到目标机的 EEPROM 或 FLASH 中，或者是使用 TFTP 协议进行远程文件传送。

编程器上面有各种形状和大小不同的芯片插座，可以通过通信线路与宿主机连在一起。在进行固化时，一般是先把存储芯片插入编程器上某个大小及形状合适的插座上，并通过软件选择芯片的型号，然后将被固化的程序文件传到编程器上。整个固化过程可能需要几秒钟到几分钟，所需时间取决于文件的大小和所用的芯片型号。

简单文件传输协议（Trivial File Transfer Protocol，TFTP）可以看作是一个简化的 FTP，主要用于下载映像文件。它和 FTP 的主要区别是没有用户权限管理的功能，也就是说，TFTP 不需要认证客户端的权限。这样，远程启动的目标板在启动一个完整的操作系统之前，就可以通过 TFTP 下载启动映像文件，而不需要证明自己是合法的用户。一般来说，在目标机初次配置时，需要通过 BootLoader 的 TFTP 客户端下载启动映像，这个映像包含了嵌入式操作系统和应用程序代码，然后将下载得到的映像烧写至闪存，这样每次启动时就可以直接从 Flash 中载入。当然，在有些嵌入式系统中，把嵌入式操作系统也固化在目标机的 ROM 中，然后把各个应用程序做成可加载的模块。当目标机操作系统启动后，可以根据自己的需要，从宿主机下载相应的应用程序模块。

5.4　嵌入式软件移植

嵌入式软件与通用软件的不同在于，嵌入式软件高度依赖于目标应用的软硬件环境。软件的部分功能函数由汇编语言完成，与处理器密切相关，可移植性差。一般来说，嵌

入式应用软件追求正确性、实时性，因此使用编译效率高的汇编语言有时是难以避免的，但这就会导致应用软件的可移植性大打折扣。另外，对于一个运行良好的嵌入式软件或其中的部分子程序，在今后的开发中可能会再次用在类似的应用领域。由于原有的代码已被反复地应用、测试和维护，具有很好的稳定性。因此，在原有代码的基础上进行移植将会缩短开发周期，提高开发效率，降低开发成本。基于以上原因，在嵌入式软件开发中，必须高度关注应用软件的可移植性和可重用性。

不过，可移植性和可重用性的程度应该根据实际的应用情况来考虑。由于嵌入式应用软件有许多自身的特点，如果追求过高的可移植性和可重用性，可能会降低应用软件的实时性能，并增加软件的代码量。这对于资源本来就有限的嵌入式应用环境来说，是得不偿失的。尽管如此，我们仍然可以在资源有限、满足系统需求的情况下，尽可能把可移植性和可重用性作为第二目标，致力于开发正确性、实时性、代码量、可移植性和可重用性相对均衡的嵌入式应用软件。

嵌入式应用软件的开发可以分为无操作系统和有操作系统两种情形。与之相对应，在移植嵌入式软件的时候，也可以分无操作系统的软件移植和有操作系统的软件移植两种情形。对于后者，又可以细分为两种方式：一是把操作系统和应用软件作为一个整体进行移植；二是把应用软件移植到一个新的操作系统上。

5.4.1 无操作系统的软件移植

如果在一个嵌入式系统中，软件的开发是直接在硬件平台的基础上进行的，没有使用操作系统。那么当处理器等硬件设备发生变化时，需要把原有的应用软件移植到新的硬件平台上运行。一般来说，对于这一类的嵌入式系统，它们的应用软件通常都比较简单，软件的代码量不是很大。在移植的时候，如果编程语言使用的是与处理器密切相关的汇编语言，那么移植将会变得非常困难，甚至是不可能的。此时的移植类似于重新开发，当然在数据结构和算法的层面上还是可以重用的，这里不考虑这种情形。

如果软件大部分是用 C 语言开发的，那么可以考虑移植问题。一个好的程序设计应该是模块化和层次化的，而 C 语言的最大优点是可移植性比较好。

基于层次化的嵌入式应用软件通常设计成图 5-11 所示的结构，其中，软件可分为两层结构，I/O 模块属于设备驱动程序层，它以嵌入式硬件为运行平台，实现了各种 I/O 设备的输入/输出功能，并向上层的应用软件提供相应的 I/O 接口函数 API，如控制功能、数据读写等。这些接口函数被设计成与硬件无关的，因此在移植系统时，只需要重新编写与处理器有关的 I/O 模块即可，不需要修改该模块的 API。移植的工作量主要体现在 I/O 的编码工作上。

可以对上述软件结构作进一步的细化设计，如图 5-12 所示。在这种体系结构中，在处理器的硬件层之上添加了一个硬件抽象层，这层软件把不同类型的硬件进行了封装和抽象。对于 I/O 模块，它不再是直接面对处理器硬件，而是基于硬件抽象层。也就是说，

它被设计为与硬件无关的。这样的 3 层软件结构的优点是需要移植的代码进一步减少，移植的工作量也进一步减少。

图 5-11　基于模块化的嵌入式软件结构

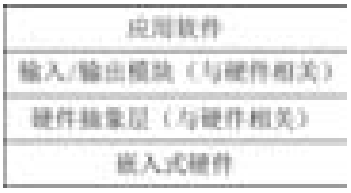

图 5-12　具有硬件抽象层的软件结构

5.4.2　有操作系统的软件移植

这里讨论的有操作系统的软件移植，是指把操作系统和应用软件作为一个整体，移植到一个新的嵌入式硬件平台上。

嵌入式软件的体系结构可分为四个层次，即设备驱动层、操作系统层、中间件层和应用软件层。当需要把一个嵌入式软件系统整体移植到一个新的硬件平台时，真正需要移植的是与硬件直接打交道的部分，包括设备驱动层的软件和操作系统当中的部分代码。而其他的软件，如操作系统内核、中间件和应用软件，不用做任何修改。当然，在有些嵌入式操作系统中，如单体结构的操作系统，把设备驱动层的软件也集成在系统内核中，这时就要把相应的软件摘取出来进行修改。总之，在系统移植时，真正需要移植的主要是：引导加载程序 BootLoader、设备驱动程序以及操作系统中与处理器密切相关的代码。

为提高可移植性，BootLoader 的实现一般分为 stage1 和 stage2 两大部分。依赖于 CPU 体系结构的代码，如设备初始化代码等，通常都放在 stage1 中，用汇编语言来实现。而 stage2 则采用 C 语言来实现。在移植时，主要的工作量在 stage1 的移植，基本上要重新编写。

一般来说，一个嵌入式操作系统在设计的时候，就已经充分考虑了可移植性，所以它的移植相对来说是比较容易的。以 μC/OS-Ⅱ为例，为了方便移植，μC/OS-Ⅱ的大部分代码都是用标准的 C 语言编写的，不需要改动。只有少部分代码，尤其是那些与 CPU 寄存器打交道的代码，需要针对具体的 CPU 类型进行修改，这些代码一般都是用汇编语言来写的。

如图 5-13 所示，μC/OS-Ⅱ操作系统的代码被分为三大部分。第一部分是与处理器无关的代码，如任务管理、任务调度、存储管理、信号量、邮箱、消息队列等；第二部分与系统的配置有关，应用程序开发人员可以通过修改这些配置文件来裁剪内核，选择自己需要的系统服务；第三部分是与处理器相关的代码，包括三个文件：OS_CPU.H、OS_CPU_A.ASM 和 OS_CPU_C.C。在移植 μC/OS-Ⅱ操作系统时，主要修改的就是这三个文件。

- OS_CPU.H：该文件包括三部分的内容，首先是一个符号常量，用来设定处理器

的栈的增长方向；其次是 3 个宏定义，用来关闭和打开中断；最后是 10 个数据类型的定义，用来定义与编译器无关的数据类型。在操作系统内部，只使用这些数据类型，而不使用标准 C 的数据类型。

- OS_CPU.ASM：用汇编语言编写 OS_CPU.ASM 文件中的四个与处理器相关的函数，包括任务切换、时钟中断服务程序等。
- OS_CPU_C.C:用 C 语言编写 OS_CPU_C.C 文件中的十个与操作系统相关的函数，一般只需要改写其中的一个函数，即任务堆栈的初始化函数。

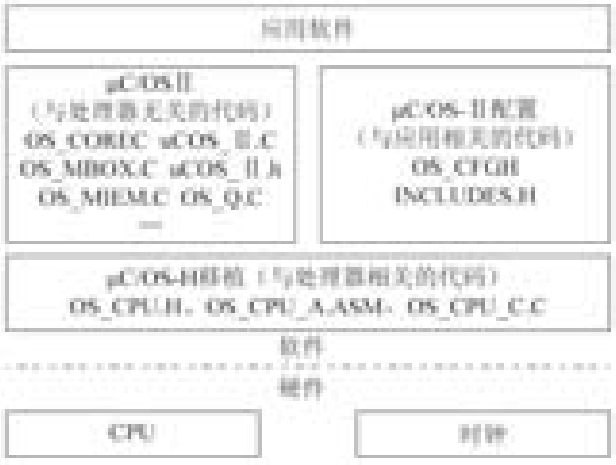

图 5-13　μC/OS-II 移植的示意图

根据目标处理器的不同，一个移植实例可能需要编写或改写 50 至 300 行的代码，需要的时间因人而异。

5.4.3　应用软件的移植

嵌入式应用软件的移植指的是把应用软件从一个嵌入式操作系统平台移植到另一个操作系统平台。

一个应用软件的实现涉及两个方面的问题：

（1）这个应用软件必须用某种编程语言来编写，如汇编语言、C 语言、C++语言。

（2）这个应用软件必须在某个平台上运行，该平台一般是一个操作系统，如 Windows XP、Linux 等。

当然，也有一些软件系统，它们既是编程语言，又是运行平台，如 Java。

因此，移植一个应用软件时，既要考虑编程语言的因素，也要考虑运行平台的因素。对于 PC 上的应用软件来说，它的运行平台比较有限，主要有两大类，即 Windows 系列和 UNIX 系列。相应的，每一类平台都有各自的一套应用程序编程接口。在嵌入式系统中，编程语言的问题不大，因为大多数嵌入式开发都是采用移植性较好的 C 语言。但是

在运行平台上，嵌入式操作系统的选择是非常多的，目前已经开发了数以百计且各具特色的嵌入式操作系统产品。从理论上说，每一个操作系统都会定义一组 API 接口函数，因此，如果要在嵌入式平台上进行应用软件的移植，难度是比较大的。

为了提高嵌入式应用软件的可移植性，在软件开发时需要遵守以下的一些原则：

（1）在软件设计上，要采用层次化设计和模块化设计。所谓层次化，指的是软件设计的纵向结构，下层为上层提供服务，上层调用下层提供的服务。每一个层次都应该定义清晰的接口和功能，分层的数量要合适。层次化结构设计的优点是：在进行系统移植时，通常只需要修改底层软件，而不需要去修改上层软件。所谓模块化，既体现在整体软件的设计上，又体现在同一层的软件结构上。模块化不同于层次化，一般来说，软件模块之间是相互独立的，一个模块的实现不依赖于其他模块的实现。良好的模块化设计，可以很容易地进行软件模块的裁减和更新。

（2）在软件体系结构上，可以在操作系统和应用软件之间引入一个虚拟机层，或者叫操作系统抽象层，把一些通用的、共性的操作系统 API 接口函数封装起来。在编写一个应用程序时，不是直接去调用实际操作系统的 API，而是使用虚拟层所提供的 API。这样，在移植这个应用程序的时候，只要针对新的操作系统平台，去实现这个虚拟层即可，其他的代码不用做任何的修改。在定义这个虚拟层时，要综合考虑现有的各种嵌入式操作系统的功能和特性，尽量采用标准的操作系统接口，如 POSIX 标准。

（3）在功能服务的调用上，要尽量使用可移植的函数，如标准的 C 语言函数，或自己编写的函数。尽量不要使用依赖于特定操作系统的 API 函数。

（4）在数据类型上，由于 C 语言的数据类型与机器的字长和编译器有关，因此可以用宏定义的方式来定义一组可移植的数据类型，然后在应用程序的内部，只使用这些数据类型，而不使用 C 语言的数据类型。例如，可以用 INT32U 来表示无符号的 32 位整型数据。对于实际的编译器，可以定义为：

```
#define INT32U unsigned int
```

（5）将不可移植的部分局域化。对于想进行软件移植的程序设计人员来说，如果应用软件的各个地方都散布着不可移植的代码，就必须从软件中一一找出它们，然后修改。这将是一件非常费时又费力的事情，而且这种修改也容易导致新的问题。为了提高移植的效率，可以把不可移植的代码通过宏定义和函数的形式，分类集中于某几个特定的文件之中。这样，对不可移植代码的使用，就可转换成对函数和宏定义的使用。在以后的移植过程中，既有利于迅速地对需要修改的代码进行定位，又可方便地进行修改。

（6）提高代码的可重用性。在进行嵌入式软件开发时，要有意识地提高代码的可重用性，不断积累可重用的软件资源。例如，可以更好地抽象软件的函数，使之更加模块化，功能更专一，接口更简洁明了。