TreeOS

简单、通用的 MCU 操作系统

TreeOS 物联网嵌入式操作系统

使用指南

Rev.02

2017-3

北京光轮电子科技有限公司

www.treeos.com

目 录

第一章 序言

&1.1 物联网需要嵌入式操作系统

根据研究，未来 20 年，随着物联网、人工智能、工业 4.0 的快速发展，预计将有 20 万亿台智能终端得到使用，其中 1 万亿台接入互联网，这相当于全球人均拥有 2000 多台智能终端设备。

想想看，在不远的将来，我们的服装鞋帽、窗帘、咖啡杯、台灯都被“植入”芯片，我们的家里有几百个芯片在工作，我们的汽车里有上千个芯片在工作，附近工厂里有数万个芯片在工作，我们将生活在一个被芯片包围的环境中。

软银公司总裁孙正义断言，物联网（让一切产品智能化、联网化）将会引领下一轮技术爆炸，正如在地球演变历史上寒武纪爆发形成了无数新物种一样，用不了多久，联网的物联网设备数量将会达到 1 万亿。

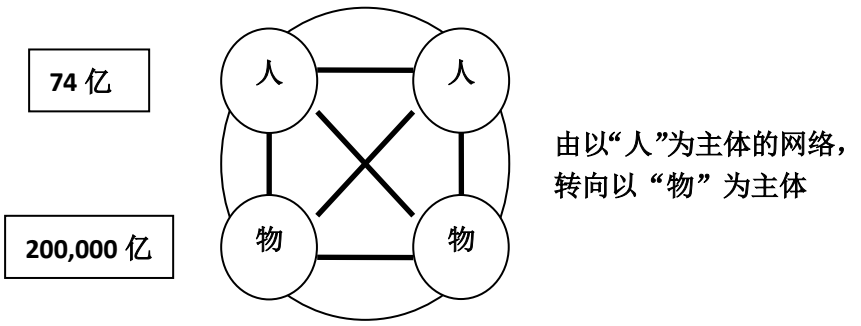


图 1-1 人联网与物联网

80%以上的物联网终端、智能终端将会使用 MCU，而不是像手机

使用 Cortex-A 这样的高端处理器。其中 80%的 MCU 都会是中低档的 MCU，而恰恰这部分 MCU 操作系统是市场空白！

也许有人会说，随着处理器成本不断降低，中低档 MCU 将会被淘汰。其实不然，因为物联网终端的应用场景决定了它需要大量的中低档 MCU。例如，同是产生于上世纪 80 年代的 Intel 286 CPU 和 MCS51 MCU，现在 MCS51 依然在大量使用，而 286 CPU 却早早就绝迹了。

有观点认为，轻内核物联网操作系统很小，技术含量不太高。但 ARM 亚太区物联网部门市场总监潘劭齐不认同这种看法。他说：“一个好的物联网操作系统需要具备易开发、低功耗和安全三个特性，考虑到物联网操作系统要面向如此多的设备和应用，想要同时满足那三个特性并不容易，需要深厚技术积累。”

“碎片化”是物联网的天性，即终端设备众多，类型和功能千差万别，设备的性能要求在不同场景下要求也不一致， 这些特点使得软件开发异常困难。而物联网软件，关乎着终端的数据接入，设备管理，数据安全等等，因此软件发展的滞后，已经成为物联网发展的瓶颈。

要把数量巨大的终端设备统一到一个操作系统或者开发平台上，在技术上是一个巨大的挑战。可以说，现有的编程技术、操作系统很难满足这个需求。因此，必须另辟蹊径，寻找新的对策，在编程技术上寻求新的突破！

与 PC 机及手机操作系统相比，物联网操作系统具有完全不同的特点：

1） 实时系统；

2） 可靠性要求极高；

3） 处理器种类非常多；

4） 周边设备种类非常多；

5） 一般只有一个 APP；

6） 升级频次低；

7） 只有开发人员使用；

从这些特点看，显然嵌入式操作系统更能胜任。

原百度 IDL 常务副院长、地平线机器人创始人兼 CEO 余凯说：“对我个人来说，我当然是认为低功耗、高性能的嵌入式人工智能是最需要的。在不需联网的情况下，通过本地运算可以实现感知、交互和实时决策，并且把体积做的非常小。万物智能不应该是仅仅在云端实现，而是在很多很多的智能终端上实现。这是我认为急需突破的技术！”

嵌入式系统迎来了它的兴盛时代。

综上所述，市场在呼唤一个能够确实针对物联网终端的需求和特点、通用性强、有着良好开发体验的物联网操作系统！

&1.2 嵌入式开发流程需要变革

尽管硬件发展快速，但是嵌入式软件开发方法还比较原始，开发效率很低，已经不能适应形势的发展。

自 1974 年 Intel 推出第一颗 MCU 芯片 4004 以来，一直采用的传统开发流程如下：

1） 确定产品需求，对功能进行分解，选择合适的技术方案和

硬件；

2） 设计产品的硬件电路；

3） 开发软件构架，比如选用实时内核并作配置，或者根据需求自建框架；

4） 开发或选购各种软件模块，如驱动程序、中间件等；

5） 以上两步构成了自己独有的一个基础平台，在此基础上做进一步的应用层软件开发；

6） 软硬件进行整合、调试、测试，完成整个嵌入式产品（实际上第 3~4 步就已经需要不断进行调试）；

7） 后期技术维护，提高产品可靠性，以适应复杂的使用环境。另外，为提高性能或采用新技术，产品常常需要不断更新迭代。

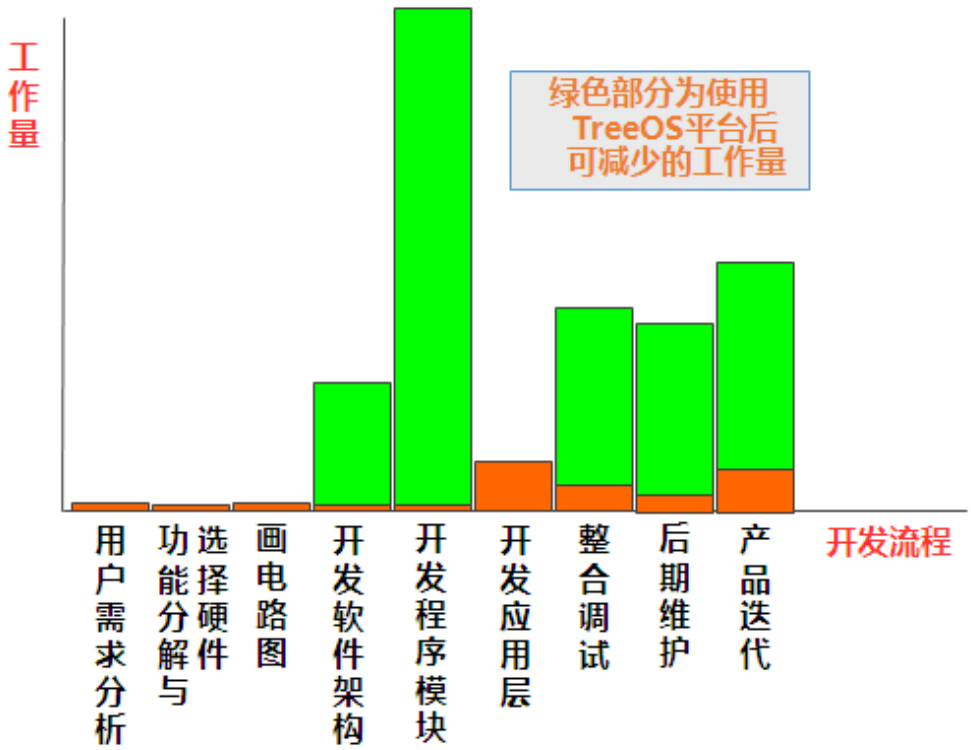


图 1-2 嵌入式系统开发流程

以上流程中，每一步出现问题，都需要工程师付出很多的时间和精力来进行调试修改。另外，与 PC 软件开发相比，嵌入式软件开发难度更大，主要原因有：

- 硬件、软件整合一起，给调试增加许多变数；
- 大量底层开发，要求工程师需要现学习相关硬件知识；
- CPU/MCU 种类繁多，没有好的、通用开发平台或操作系统可用；
- 产品使用环境复杂，需要工程师有丰富的实践经验，等等。

这些原因造成产品的开发周期特别长，开发效率低下。以上这些因素，加上严苛的开发周期、产品成本的掣肘，给工程师造成很大的挑战，使得工程师没有更多的时间来考虑产品的创新和良好的用户体验。另外，开发的不易和求稳的心理也降低了产品更新迭代的速度。

在我们多年的开发实践中，发现整个软件开发流程中工作量最大、最关键的是软件架构开发和软件模块开发（即上述流程第 3~4 步）。后面的整合、维护迭代工作量也很大，但主要是受这两步影响。所以关键把软件架构开发和软件模块开发做好，就可以大大提高开发效率，降低开发难度。

针对这种情况，经过十余年的研发，我们给出了解决方案。

&1.3 TreeOS 开发平台

TreeOS 物联网终端开发平台包括 4 个部分：

- TreeOS 通用 MCU 实时操作系统；
- TreeOS ComLib 软件仓库(TreeOS 的组成部分)；

- AlphaMcu 自动写代码机器人；
- Kepler11 智能硬件开发板；

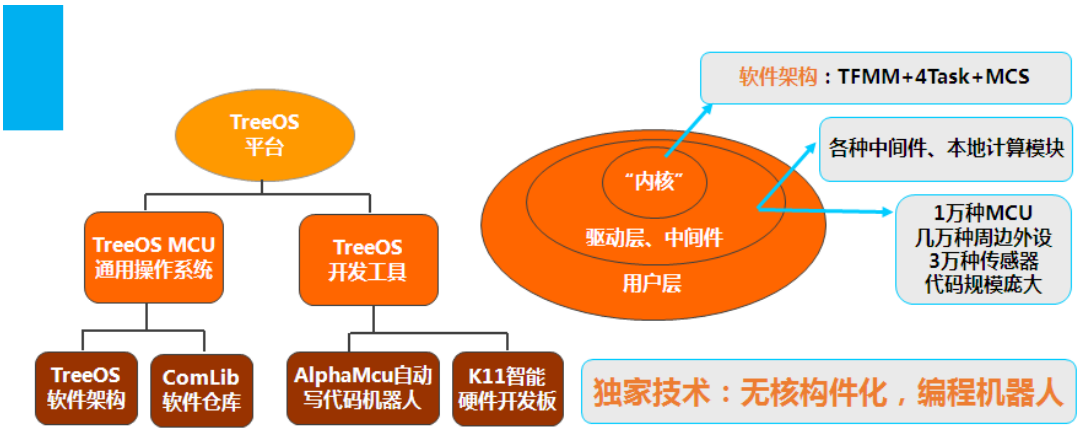


图 1-3 TreeOS 物联网终端开发平台

首先，必须有一个各种档次的 MCU 都能够通用的软件架构或操作系统，这样方便软件模块在各种 MCU 之间移植，最大限度地发挥软件模块的代码复用的作用，也可以使工程师方便、快速地使用各种 MCU，满足嵌入式应用的多样化需求。显然，要把万种 MCU 统一在一个平台上，是一个巨大的挑战。经过多年的努力，我们推出了 TreeOS 通用 MCU 实时操作系统，其软件架构对各种档次的 MCU 皆可通用，这在业界目前是唯一的。这个系统经过十年来多种行业近 300 个实际项目应用，证实是可行、可靠的。

其次，以 TreeOS 软件架构为基础，开发 MCU 的配置软件、芯片或外设的驱动程序、中间件等，形成了一个软件仓库，即 ComLib 软件构件库。软件库中各软件模块可以自由组合，以最快速度满足用户的各种需求。

无疑，ComLib 软件库是巨大的，为了方便工程师使用，我们又设计了 AlphaMcu 自动写代码编程机器人，作为软件库的管理工具。工程师通过输入电路图数据，马上就可以输出项目所需的全部软件模块，这些模块已根据电路图进行配置，且在开发环境中可直接编译通过。过去需要几个月时间编写的代码瞬间即可完成，经过统计，自动生成的代码量可占到整个项目的 70~90%。

为了使工程师可以快速搭建产品硬件原型，我们还推出了 Kepler11 智能硬件开发板，可以像搭积木一样快速满足需求。

综上所述，TreeOS 开发平台已经将软件架构（实时操作系统）、MCU 的初始化配置、驱动程序、中间件、接口协议等都打包调试好了。工程师不必像传统 MCU 软件开发流程那样从最底层的驱动做起，无需搭建自己的平台就可以直接进行应用程序的开发，这将极大缩短开发时间和认证时间，有助于企业缩短产品的上市周期，促进产品的更新换代，降低产品的总拥有成本。

由此可以看出，TreeOS 平台实现了对传统 MCU 开发流程的重大变革，它使 MCU 软件开发从手工模式直接进入高效率的智能化模式。

目前 TreeOS ComLib 软件仓库正在快速扩充之中，以满足物联网时代用户日益增长的设备多样化需求。

&1.4 诞生

之前一直在做工业控制方面计算机软件，后来有机会用单片机开发硬件产品，做了很多项目，发现没有操作系统开发效率太低（为什么不用μcos？因为绝大部分单片机用不了μcos，而且驱动程序需要

自己写。），于是决定要另辟蹊径开发一个好用的 MCU 操作系统。

不断地做项目，不断地总结经验，逐渐形成一套称之为 MCUSDS 的“MCU 标准化开发系统”，用于内部使用。 其时是 2006 年，TreeOS 操作系统基本思想也是成形于那个时期，可以称之为 0.1 beta 版。到了后来，理论上有了更大突破，又历经一百多个单片机项目的提炼和验证，版本不断升级，于 2012 年形成了 TreeOS 1.0 正式版本，并申请了发明专利。

在长期的 MCU 开发实践中，作者发现普遍存在以下问题：

1） 各种项目尽管功能迥异、使用的单片机也不同，但是解决问题思路相似，很多程序模块都是可通用的，因此如果解决好移植问题，可以节省很多时间；

2） 各种程序模块之间为占用 MCU 时间，经常会发生冲突，就是我们常说的任务调度和实时性问题；

3） 网上虽然有很多现成的驱动程序，但是一般无法直接拿来就用，还需要花很多时间进行学习和调试；

4） 由于编写程序的不规范或者不熟练，造成 BUG 一大堆，从而浪费了很多宝贵时间；

5） 由于单片机调试通常要硬件与软件联调，这使得软件调试更令人恼火，因为 BUG 既可能是软件带来的，也可能是硬件引起的，需要分别判定。因此使用成熟的软件模块显得更有意义；

6） 实验室测试完好的产品，拿到现场应用经常会挂掉，这是因

为产品抗干扰性能不过关。单片机多为工业级应用，抗干扰性能要求高。而抗干扰技术主要来源于实践经验，需要长期的积累。如果能把这些前人的经验源源不断注入软件模块中，对于提升产品质量有很大的意义。

……

现在，有了 TreeOS 实时操作系统，这些问题获得了很好地解决。TreeOS 并非是突发灵感式的横空出世，而是一个真正的“从实践中来、到实践中去”的产品，它的形成过程就是一个“从量变到质变”的过程，对于提高开发效率、提升产品质量具有重要的意义！

&1.5 MCUSDS

MCU 是英文 Micro Control Unit 的缩写，称为微控制器，俗称单片机。

MCUSDS 是 MCU Standard Development System 的缩写，称为“MCU 标准化开发系统”，是我们早期的产品。

MCUSDS 是一个单片机开发包，包含三个方面内容：

- 标准化软件；
- 标准化电路；
- 可靠性设计经验。

做这套系统的目的，一是为了提高开发效率，二是为了培训新员工。后来，标准化软件就形成了现在的 TreeOS 实时操作系统。

标准化的意义在于：

- 减少重复性的工作；

- 提高开发效率；
- 减少出错。

&1.6 通用 MCU 操作系统的现实意义

随着“后 PC”时代的到来以及物联网（包括无线传感系统、智能家居等）的兴起，工业智能化的快速发展，微控制器的应用将迎来一个爆发期，对微控制器操作系统的需求愈加迫切。因此，开发一种适用各种 MCU、驱动程序齐全、易学易用、便于互联的实时操作系统，现实意义重大。

操作系统（或统一的开发平台）带来的好处是显而易见的：

- 操作系统提供了完善的编程思路，使大量的任务可以有条不紊地在一起工作，减少了出错的可能性；
- 有了操作系统，可以提供了标准化的驱动程序和常用程序库，所有用户都可以直接拿来就用，大大减少了工作量；
- 对于初学者来说，可以大大缩短自身的积累过程，帮助他快速成长为单片机工程师；
- 程序容易移植、维护、升级；可以多人合作共同完成一个项目；项目管理很方便。

说得直白些，开发人员从底层做起，需要做大量的重复性劳动。使用 TreeOS 之类的操作系统，则只需要编写用户程序，很多现成程序可以拿来就用，更不必将所有任务运行的各种情况记在心中，这就简化了开发的过程，提高了开发效率，也减少了出错的可能性。也就是说，很多工作操作系统替你做了。

事实上，全世界的单片机工程师一直以来都在努力探索解决单片机操作系统问题。多年来已源源不断涌现出一二百个操作系统实时内核，从中可以管窥一斑。遗憾的是，至今并未获得圆满解决。

有人会问，μcos 系列等操作系统不都是可以用于单片机吗？是的。不过适用的单片机种类很少，都是一些内存容量大的高档单片机才能使用，如 Cortex-M3、M4 等，而对于像 51、AVR 等这些大量的中低档单片机（占总量 80%以上）则无法实际应用。而且，这类操作系统称为实时内核，并不提供外围芯片和模块的驱动程序，实际上并非是一个完整的操作系统。

另外，对于占有大量份额的中低档应用，由于现有的实时操作系统不太适用，因此目前一般还是采用传统的大循环(big loop)程序结构，其优点是直观，容易理解。大循环程序结构是一种笼统的说法，并无一定之规，每个成熟的工程师可能都有自己的一套方法和程序积累，因此无法形成统一的开发平台。

&1.7 通用 MCU 操作系统的特殊要求

单片机系统有其特殊性，因此在设计操作系统时，应该充分考虑它的特点：

1、 资源少（包括内存、ROM、寄存器等），操作系统不能占用太多资源；

2、 工作频率较低，运算能力有限，操作系统不应占用太多 MCU 时间；

3、 应用相对简单，如果操作系统过于复杂，有杀鸡用牛刀之

嫌；

4、 单片机种类繁多，操作系统应有利于移植；

5、 应用五花八门，操作系统应有良好的适应性；

6、 单片机应用环境比较恶劣，操作系统应有良好的抗干扰性能和运行可靠性。

&1.8 TreeOS 是怎样的一个操作系统

- 一个可适用于各种单片机的实时操作系统；
- 一个真正适用于 51 单片机的实时操作系统；
- 一个采用软件构件化技术的新型操作系统
- 一个带有驱动程序库的实时操作系统；
- 一个可自动帮你完成 70～90%软件开发任务的操作系统；
- 一个非常原理简单、通用的单片机操作系统。

TreeOS 填补了中低档单片机无适用操作系统的市场空白。由于采用软件构件化技术，使我们可以进一步开发出能够帮助工程师编程的软件机器人，这就是 AlphaMCU。后面你将会看到，通过轻点鼠标，你就可以轻松完成整个项目软件的 70~90%代码（定制化操作系统）！而且这些代码都是产品级的。你所要做的只是编写剩余的用户代码。

&1.9 TreeOS 的突破性技术

- 软件构件化技术；
- 无核设计；
- “面向场景”编程；

- 任务网格化；
- 自动写代码技术；
- 一键生成“定制化操作系统”。

“无核”，并非没有内核功能，而是把内核功能分散在各个构件之中。

“无核”，并非功能就弱，实际是内核功能可大可小，伸缩性极强。

因此 TreeOS 具有通用性，在 8~32 位 MCU 上都有良好表现，甚至在 PC 机（DOS 系统）上也有过应用实例。

可以说，TreeOS 原理简单而通用性强。斯坦福大学著名物理学教授张首晟说过：“最伟大的科学一定有一个共性，那就是简单性(simplicity)和普适性(universality)。”

“无核·构件化”体现了中国古代先贤老子的哲学思想：“道生万物”、“无为而治”。

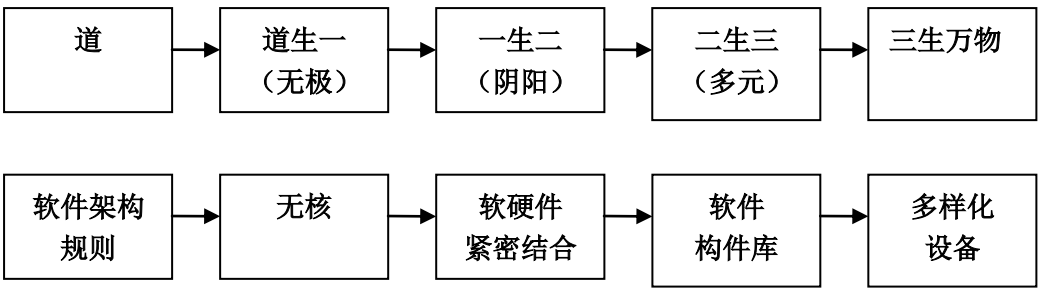


图 1-4　“无核构件化”与老子哲学思想

图 1-5　TreeOS 技术全球领先

第二章 操作系统基本概念

本章以通俗易懂的形式讲解有关实时操作系统、特别是与 MCU 有关的实时系统的一些基本概念。对于初学者，即使不学操作系统，了解这些实时操作系统的基本概念，对于开发 MCU 程序也是大有裨益的。

&2.1 大循环系统（big-loop）

大循环系统，也称前后台系统（foreground-background system），是把应用程序设计为一个无限循环，在循环中按顺序调用各种任务函数（后台程序），对于需要立即处理的紧急事务则放在中断程序中处理（前台程序）。

目前，绝大部分单片机程序都还在使用这种设计方法。大循环系统的优点是无需任务管理程序，因此也不存在系统程序占用单片机宝贵的内存和 ROM 空间等资源的问题。另一个优点是思路简明。这种系统的缺点是整个循环的执行时间不是一个常数，随着任务量的增加，每个任务等待执行的时间也随之增加。

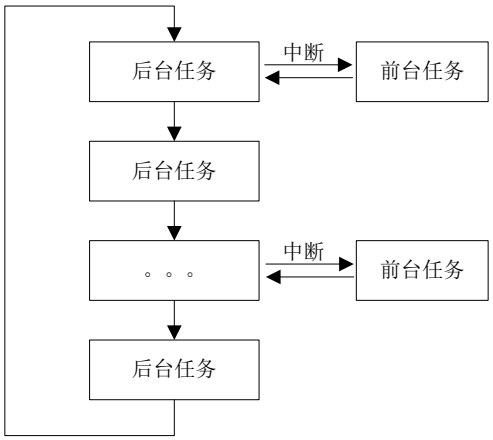


图 2-1 前后台系统

好在单片机的应用相对简单（非常复杂的单片机应用毕竟是极少数），程序中总的任务数不是很多，经过妥善的安排都可以获得完美的结果。在 TreeOS 系统中，把应用系统运行过程划分为一系列的场景，每个场景就是一个小型的大循环系统，这充分利用了这种系统不占用资源的优点。

大循环程序结构是一种笼统的说法，并无一定之规，每个成熟的工程师可能都有自己的一套方法和程序积累，因此无法形成统一的开发平台。

&2.2 多道程序系统

多道程序系统是把多个相互独立的作业保存在计算机内存中，在管理程序控制之下，这些作业相互穿插地运行。如果一个作业运行过程中需要等待另外一个任务（如 I/O 操作）完成，对于单道系统，CPU 就会出现空闲，等到该任务完成后才进行下一个作业。而对于多道系统，管理程序就先切换到其它作业运行，直到第一个作业完成等待并

获得 CPU。这样做的效果是提高了 CPU 的利用率。

在日常生活中，这种例子随处可见，我们称之为统筹安排时间。例如我们准备晚饭的时候，可以先煮上米饭，然后再去洗菜炒菜，等米饭熟了，菜也做好了，节约了很多时间。

多道程序运行的特征是：宏观上多任务并行运行，微观上是一个个任务在串行运行。多道程序的设计思想对实时系统设计很重要，在 TreeOS 中很多地方可看见这种设计思路的影子。例如，在读取键盘键值的程序中，采用每隔 20ms（或以上）查询一下按键是否释放，如果释放，读取键值，否则就转去执行其它的任务。如果一直等待按键释放再去执行其它任务（很多单片机书上或学习板的程序就是这样写的），其它的任务可能就被耽误了，系统的实时性能很差，可靠性差，如果有硬件故障，按键一直无法释放，系统就死机了。

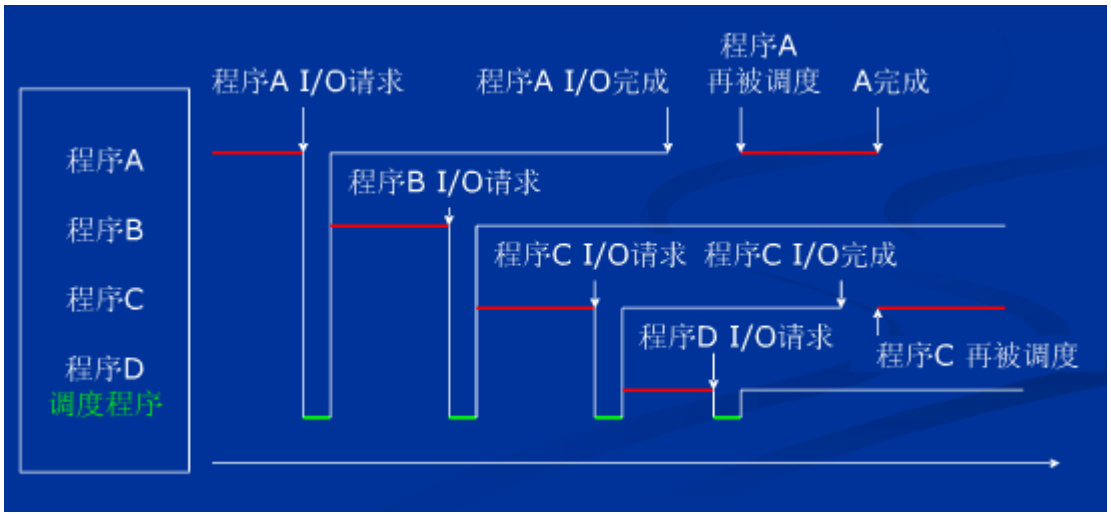


图 2-2 多道程序系统运行过程

如上图所示，首先执行程序 A，当程序 A 出现 IO 请求（IO 请求过程很慢，需要等待较长时间）时，调度程序马上切换任务转去执行

程序 B。当程序 B 出现 IO 请求时转到程序 C，当程序 C 出现 IO 请求时转到程序 D，当程序 D 出现 IO 请求时又回到程序 A。当程序 A 执行完毕后，发现程序 B 还在执行 IO 请求，就转去执行程序 C。如此过程不断循环。宏观上看，程序 A、B、C、D 并行运行，微观上看则是四个程序任务在串行运行。当有程序出现 IO 请求时，CPU 并不一直死等，而是转去执行下一个程序，从而提高了 CPU 的利用效率。

&2.3 操作系统定义

很难给计算机操作系统下一个充分完整的定义。不过从功能方面讲，我们可以这样认为：操作系统是一个有完整算法的、帮助人们更好地使用（应用）计算机的软件系统。

&2.4 嵌入式系统

嵌入式系统是用来控制、监控或者辅助装置、机器或工厂装备运行的设备。

嵌入式系统是一种专用的计算机系统，应用于特定场合，软硬件可剪裁，对功能、可靠性、成本、体积、功耗等要求严格。

嵌入式系统可应用于航天航空、军事电子、工控设备、智能仪表、机器人、汽车电子、智能家居、信息家电、办公自动化设备、通讯设备、移动存储、计算机外设、网络设备等方面。

嵌入式系统的特点包括：特定应用、可配置、可剪裁、高可靠性、实时性、低功耗等。

用于设计嵌入式系统软件的操作系统称为嵌入式操作系统。

嵌入式软件开发正朝着平台化、标准化、可复用的方向发展。

&2.5 实时（real-time）系统

实时系统是对任务运行时间有特定要求的系统。当发生外界事件或者有数据输入时，能够毫无遗漏地响应和接收并快速处理，然后在规定的时间内（截止时间）输出结果。实时系统有明确的和固定的时间约束，任务处理必须在约定的时间内完成。

实时系统又分为“硬实时系统”和“软实时系统”。“硬实时系统”保证关键任务必须在截止时间内完成，否则系统就是失败的，就有可能发生重大的安全事故。例如，回收“神舟”飞船时，当返回舱依靠重力从天而降，到达一定高度后，必须及时打开降落伞，如果打开晚了，返回舱重重摔到地上，就有可能发生机毁人亡的重大事故。“软实时系统”则只要求尽可能快地完成操作即可，当然，拖的时间越长，系统的性能就越低。例如，当我们用键盘输入文字时，计算机反应时间在 0.1 秒我们会觉得很快了，如果反应时间为 1 秒钟也还可以接受。当反应时间为 10 秒时，计算机此时并没有崩溃，只是我们自己先崩溃了。

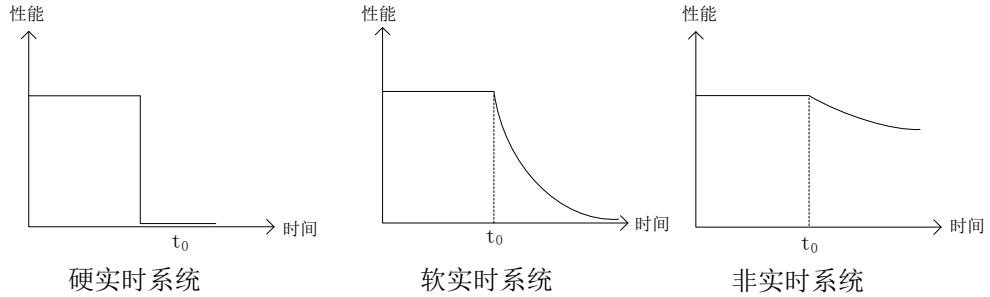


图 2-3 系统失败曲线

从上图可见，硬实时系统在截止时间 t0 之后性能马上降到 0（完

全失败）；软实时系统在 t0 之后性能快速下降；而非实时系统在 t0 之后性能缓慢下降。

实时系统软件应该设计成一个并行的多任务系统，而且要求延时时间尽可能短，只有这样才能够满足实时系统的性能和时间的要求。

硬实时系统的应用：航天、航空、军事、工业控制等；

软实时系统的应用：监控、信息采集等。

实时系统的主要特点：

- 时间约束；
- 可预测：系统能够对实时任务的执行时间进行判断，确定是否能够满足任务的时限要求；
- 可靠性要求高。

实时操作系统是专门用于设计实时系统软件的。其实不用操作系统也可以设计出实时系统，只是使用操作系统更方便更快捷，性能和可靠性更有保证。

&2.6 内核

内核（软件），顾名思义就是操作系统的核心。它提供操作系统的一些最基本功能，包括任务管理和调度、内存管理、任务之间通讯等。内核需要占用系统的内存、ROM 代码空间、CPU 时间等资源。中低档单片机因为内存容量很小，一般无法运行实时内核。

并非所有的操作系统都必须有内核。有些操作系统就完全没有内核，可称之为“无核（zero-kernel）操作系统”。有文献可查的有：DEIMOS、THINK、 JBEOS（北京大学）等，这些操作系统都采用了软

件构件化技术。

TreeOS 也是一种无核的、构件化操作系统。

&2.7 软件复用

软件复用是在软件开发过程中避免重复劳动的解决方案。通过软件复用，软件开发不再采用一切从零开始的模式，而是在已有的工作的基础上，充分利用过去积累的知识和经验，如：需求分析结果、体系结构设计方案、源代码、测试计划等等，从而将开发重点集中于应用的特有构成成分。

软件复用的好处是提高软件开发效率和提升产品质量。近 20 年来，出现了面向对象技术、软件构件技术，并逐渐成为主流技术，为软件复用提供了基本的技术支持。

&2.8 软件构件（component）技术

软件构件技术是软件复用的主流技术，也是目前嵌入式开发中被广泛推广应用的软件技术，它代表着未来软件技术发展的趋势。

软件构件是具有一定的功能和结构，并符合一定的标准，可以完成一个或多个特定服务的软件实体。具体来说是一个具有某种功能的单元，如某个驱动程序、库程序、内核管理模块等。 构件的大小用“颗粒度”来衡量，大尺度的构件可以包含多个小尺度的构件。

以 TreeOS_keyboard 键盘模块为例，这是一个典型的软件构件。该模块可以读取单键、组合键、双击、长按键、加速键、触键前后沿等多功能键值，这些功能基本能够满足各种单片机的应用需求，在一

个软件模块中就能全部实现了。该模块由标准 C 语言编写，可以方便地在不同的单片机系统中移植。同时，为了节约内存，该模块还提供了配置手段，可以选择屏蔽一些用不到的功能。

随着对软件复用技术的深入理解，构件的概念已不再局限于源代码构件，而是延伸到需求、系统和软件的需求规约、系统和软件的体系结构、文档、测试计划、测试案例和数据以及其它对开发活动有用的信息。

&2.9 构件化操作系统

构件化操作系统是基于构件开发方法的操作系统。嵌入式操作系统构件化的意义在于能为开发者提供不同的构件，以使开发者能够配置出适合其特定需要的操作系统，从而达到既可节省资源，又可提高开发效率的效果。

典型的采用构件化技术的嵌入式操作系统有 eCos、TinyOS、JBEOS（北京大学）等。尽管著名的 VxWorks 不能算作构件化操作系统，但也可以视为类构件化，它包括了数百个组件，这些组件可以方便地配置和剪裁。

大多数构件化嵌入式操作系统带有内核或微内核，而真正的构件化操作系统应该是无核的，传统的内核功能由构件之间协作完成。

TreeOS 采用了无核的、构件化的设计方法。

&2.10 任务

任务就是一段具有一定功能的程序体，也称为一个线程。一个任

务除了程序代码之外，还包括任务控制块、任务堆栈等内存数据。由于任何时候单处理器只能有一个任务运行，因此可以认为该任务独占CPU。

&2.11 多任务系统

多任务系统是指系统可以“同时”运行多个任务，这种说法是从宏观上的效果来说的。实际上从微观上看 CPU 在一段时间内只能运行一个任务。CPU 根据实际需要在不同的任务间切换，以提高 CPU 的利用率和快速处理作业。多道程序系统就是一种多任务系统。实时系统一般都是使用多任务设计。

&2.12 抢先式任务

当 CPU 正在运行某个任务时，此时突然出现一个更重要的新任务需要马上执行，CPU 就先保存原来任务的状态，切换去执行新任务，这个新任务就称为抢先式任务。 实时系统把多任务进行分级，以使需要及时响应的任务优先获得 CPU。

TreeOS 把任务进行了简单的分级，把任务分为普通任务和抢先式任务，而且抢先式任务只在中断中执行。

&2.13 任务调度（scheduler）

调度就是选中应该执行的任务。这是内核的主要功能。多数实时内核是基于优先级调度的，算法十分复杂。

TreeOS 各场景采用了前后台系统的算法，因此无需调度运算，对任务的划分和处理也很容易理解。

&2.14 轮询（polling）

轮询就是不断查询一个事件是否发生或者事务是否处理完毕。在轮询过程中 CPU 一直被占用。通常只对很短时间内就能完成的事务才采用轮询的方法，例如等待一次 AD 采集完毕，或者等待串口发送几个字节。采用轮询的优点是程序简单直接，缺点是占用 CPU 时间，是否使用视具体情况而定。采用轮询方法时要注意设定时间限制，避免因故障造成死循环。

&2.15 中断（interupt）

中断是一种对来自外部或自身事件的硬件反应机制。系统出现中断后，CPU 保存当前的运行现场，然后转去执行中断服务程序。当处理完毕之后，回到之前被中断的程序位置继续执行。

中断大大地提高了系统的工作效率。中断对于实时系统是非常重要的，它使紧急事件获得了及时的响应。

&2.16 关中断

有些时候需要关闭中断，以免中断破坏数据或干扰时序。最常见的是保护共享数据。在实时系统中，关中断的时间应尽量短，以免丢失中断请求。在程序中，关中断和开中断应该是成对出现的。

&2.17 资源共享

多个任务共同使用的资源叫做共享资源（如数据、文件、输入输出设备等）。为了防止数据被破坏或设备异常，每个任务使用共享资源时，必须独占该资源，不能被打断。最常用的做法是先关中断，运

行临界区代码，然后重新开中断。

&2.18 任务间通信

任务之间（也包括中断）的协作常常需要相互通信。最主要的办法是使用全局变量，这就是一种共享资源。当然还有其它方法。TreeOS 1.0 就是采用全局变量的方法。使用共享资源时需注意在某个时段让任务独享资源。

&2.19 原子操作（atomic operation）

原子操作是指在执行过程中不会被任务（或线程）打断的操作。原子操作是不可分割的，在执行完毕之前不会被任何其它任务或事件中断。例如：

在时钟节拍为 1ms 的定时器中，使用全局变量 time_ms 对毫秒数进行累加计时。

```
unsigned int time_ms；//双字节的整形变量
time_ms++;  //在中断程序中，ms 计时数加 1
```

若在其它程序中需要取用 time_ms，例如：

```
unsigned int a;
a= time_ms;  //读取并赋值
```

单片机可能需要多条指令才能完成上面这句读取数据的操作。如果 a= time_ms 不是原子操作，则在执行过程中，很可能被定时器中断，time_ms 被改变，从而无法获得正确的结果。而如果是原子操作，不会被中断打断，则可以获得正确的结果。

&2.20 代码临界区（critical section）

任务中的一段代码称为临界区，在该区中可能改变与其它任务共享的资源。因此进入临界区后不能被其它任务中断，一般的做法是先关闭中断，等临界区代码执行完后再打开中断。这样可以确保某个时候只能有一个任务运行代码临界区。

以原子操作中的例程为例，a= time_ms 就是临界区代码，改进如下：

```
DISABLE_INTERUPT;    //关全局中断
a= time_ms;
ENABLE_INTERUPT; //开全局中断
```

&2.21 可重入函数

可重入包含两种情况：一种是在调用该函数过程中发生中断，转去执行中断程序，当中断完成后，返回来可接着从刚才被中断之处继续执行。另一种情况是在多任务系统中，函数可同时被多个任务调用而不会出现数据混乱。

在 TreeOS 中情况比较特殊，只要考虑以上第一种情况即可，因为普通任务运行是独占 MCU 直到运行完毕才让出 MCU 的（运行过程中可被中断），因此普通任务函数可称之为“半可重入”函数。与可重入函数一样，需要对共享数据（主要是全局变量）进行保护。

不可重入函数引起的问题很隐蔽，难以发现，而且有时很难测试出来，需格外小心。

&2.22 递归调用

递归调用就是一个函数自己调用自己，这是一种特殊的嵌套程序。递归调用必须有终止条件，否则就会无穷调用下去。

TreeOS 1.0 是不支持任务的递归调用的。例如当串口接收处理程序 scan_RS232()在接收到数据后，需要发送应答数据，就调用发送函数 send_data()。串口发送数据时，send_data()在等待发送期间，为了不耽误接收串口数据，就轮询 scan_RS232()以免错过接收数据。这样就出现了 scan_RS232()自己调用自己的情况。其结果是通讯出错，甚至是系统崩溃。

与不可重入函数引起的问题相似，递归调用问题很隐蔽，难以发现，而且有时很难测试出来，需格外小心。

&2.23 时钟节拍（clock tick）

时钟节拍就是定时中断。一般使用单片机的 T0 定时器，STM32 单片机专门设有一个 SysTick 定时器。时钟节拍就像是系统的心脏脉动，与时间相关的任务基本都要用到它。时钟节拍犹如程序的节拍器，引领着程序的流动。节拍快了，系统的负荷增加，能耗也加大，但处理速度快了；节拍慢了，系统的负荷减小，能耗也降低，但处理速度慢了。节拍时间视具体应用而定。从某种意义上说，时钟节拍也是反映系统实时能力的一个指标。在 TreeOS 中，节拍时间一般可设为 1~100ms。

```
/*****************************************TreeOS Real-Time Operating System*****
Function Name:
Description   : T0溢出中断处理程序
                计时器;
                各个任务延迟计数器更新;
                需要及时处理的一些事务; 等等
Input         :
Output        : TCNT0, 各个任务延迟计数器更新等
Return        :
*******************************************************************************/
void timer0_isr(void) interrupt 1
{
 //reload counter value
 TH0=0xfc;  //Fosc=11.0592MHz,1ms中断
 TL0=0x66;

 systick_ms++;
 second_stamp++;

 //1s定时
 if(second_stamp >= 1000)    //判断1000ms定时器周期到了没有
    {
        second_flag=1; //秒标志置位
        second_stamp=0;
    }

 //键盘定时扫描
 iSCAN_KEY_TIME;
 //红外遥控器键盘扫描
 iSCAN_IRR_KEY_TIME;
 //数码管动态显示方式扫描
 iscan_LEDDP_dynamic();
 //数码管闪烁扫描
 iSCAN_LEDDP_BLINK_TIME;
 //蜂鸣器计时
 iSCAN_BEEP_TIME;
 //18b20采样计时
 iSCAN_18B20_SAMPLE_TIME;
}
```

图 2-4 一段 T0 中断例程

&2.24 底半处理（bottom half）

尽管嵌入式系统对中断程序处理时间一般不做限制，但是普遍的做法是尽量缩短中断处理时间，以免影响其它中断服务或紧急任务。

一种有效的做法是把中断处理任务过程分为两部分，上半部分（top half）处理紧急事务，例如设置标志或者接收输入数据单元，这部分运行时间很短，放在中断服务程序中进行。然后把一些需耗时处理或对时间要求不太紧急的事务放在之后运行（例如放在后台程序中），这部分程序称为底半部分（bottom half）。

TreeOS 中有很多地方采用这种处理方法。

第三章 初识 TreeOS

本章通过介绍一个开发实例过程，让大家初步认识一下 TreeOS 操作系统。

TreeOS 是一个构件化的操作系统，我们可以把它看成由两部分组成：软件架构方法和软件构件库。软件架构方法是设计软件架构的指导原则，软件构件库则是在软件架构方法基础上编写的各种程序库。

图 3-1 TreeOS 的构成

这里简单介绍使用 TreeOS 开发单片机项目的全过程，以使读者对 TreeOS 有个感性的认识。详细的介绍也可以观看有关的视频。

单片机项目整体开发步骤见下图：

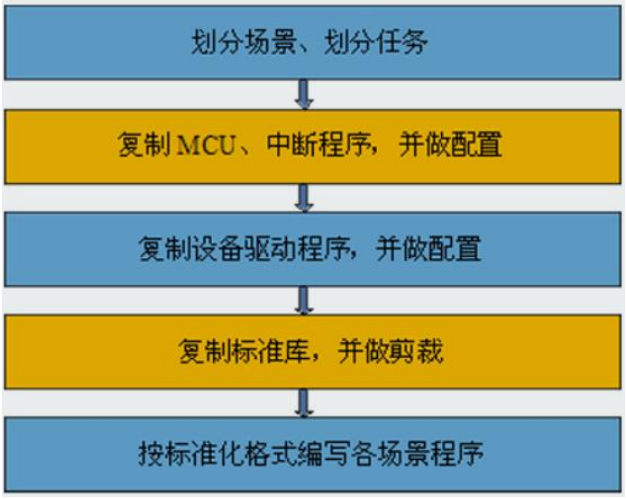


图 3-2 使用 TreeOS 开发步骤

下面，我们结合一个项目实例来进行说明。这个实例的名称为《汽车智能电子表》，它已被收录在 GL210C51 开发板中。

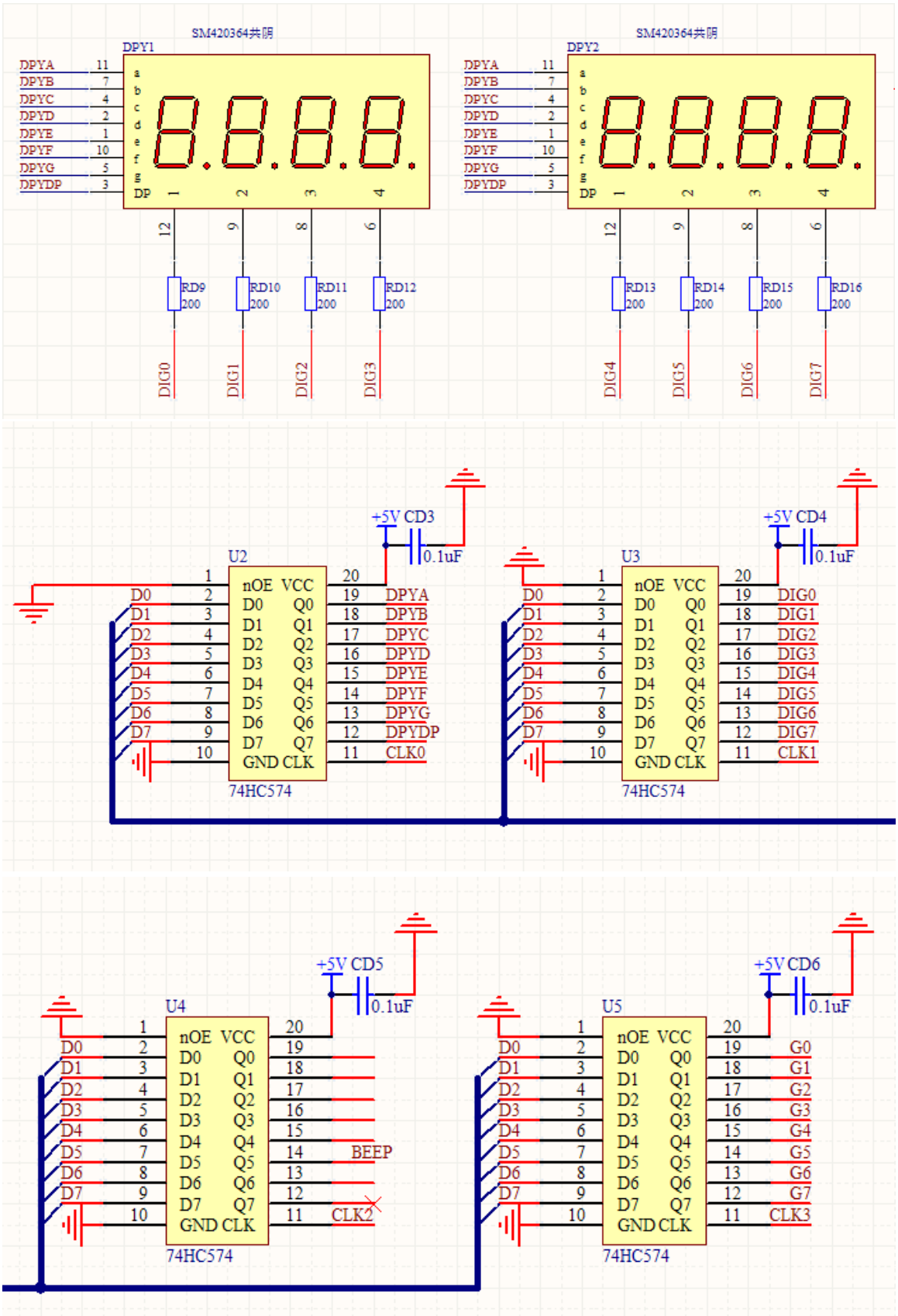
详细的程序清单请参见第九章《应用实例》。

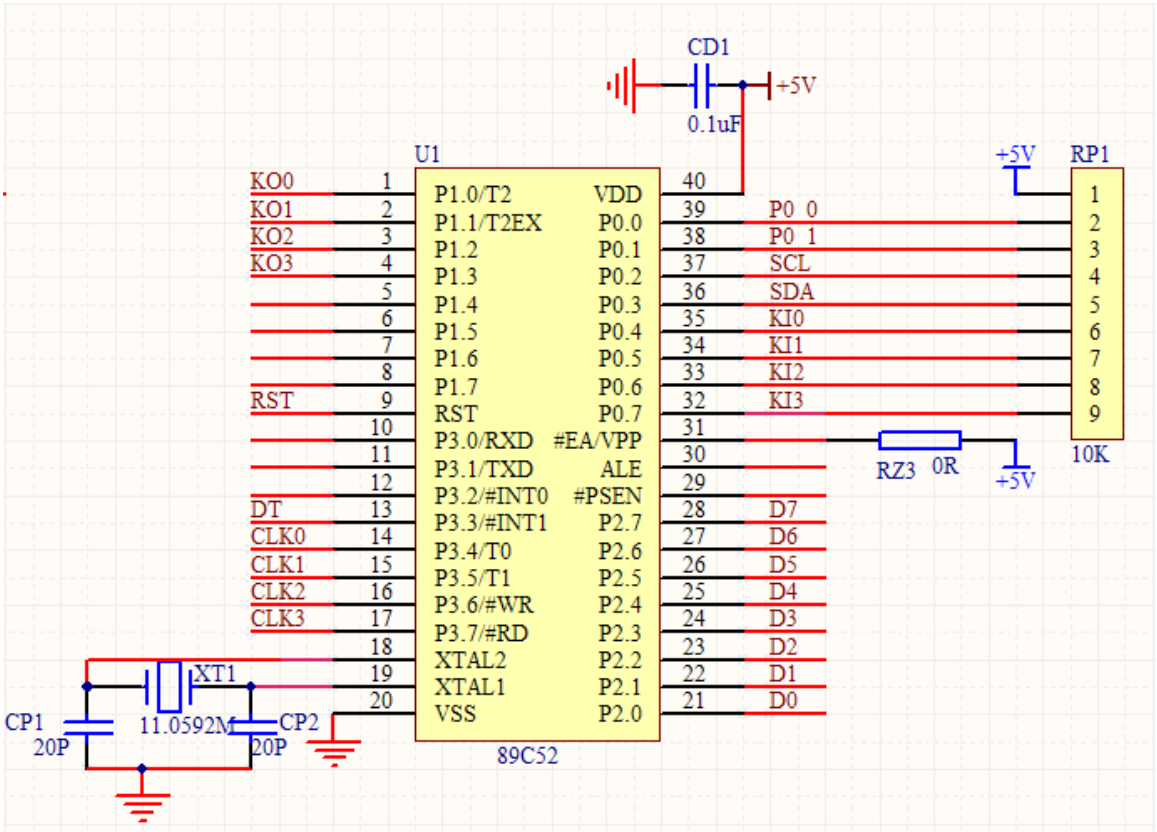
《汽车智能电子表》的主要功能如下所述：

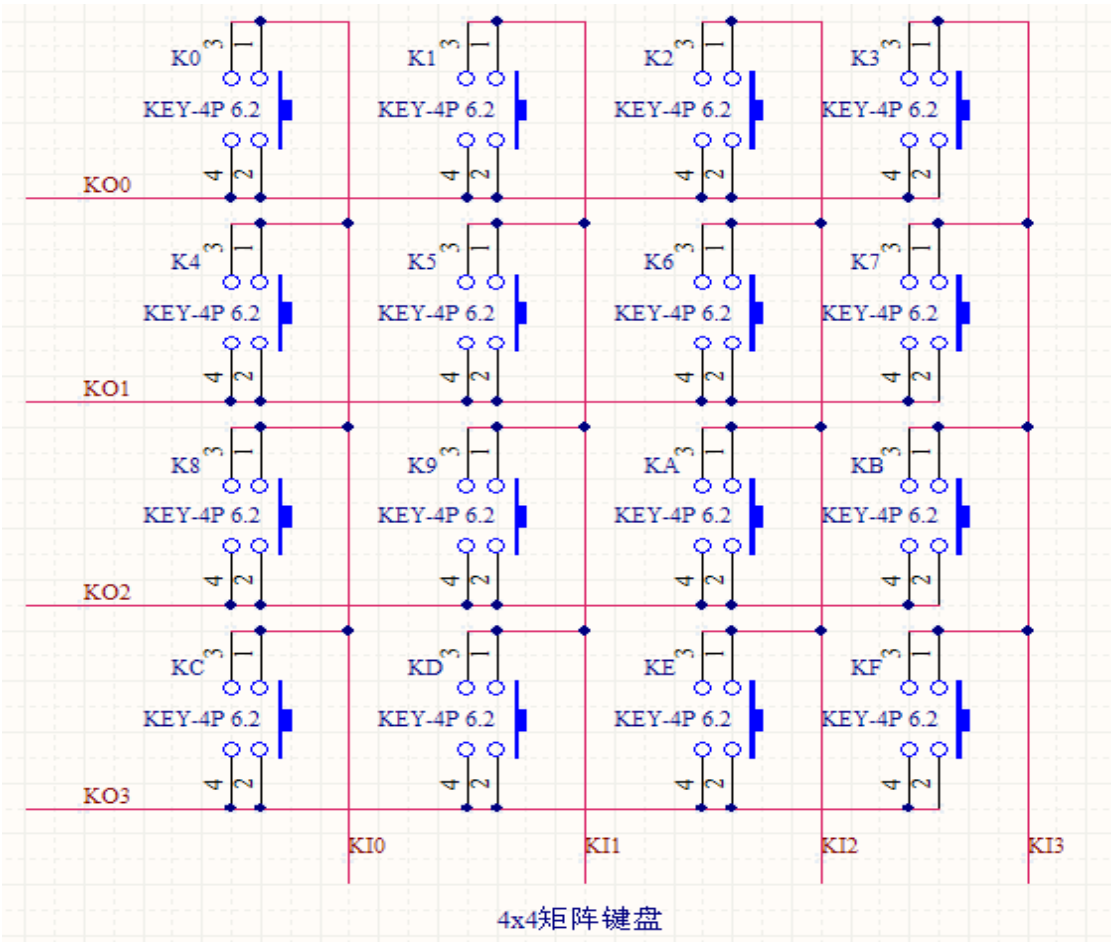
放置在汽车仪表盘内，显示时间（时分）、单次里程、车内温度；时间可以修改，单次里程可以清零。它包括一个 4 位的数码管显示器、一个按键、电子时钟芯片、公里数脉冲测量、温度计等功能模块。

电路原理图如下：

包括 8 位数码管（只用到前 4 位）、4 个扩展 IO 输出芯片 74HC574（未全用到）、4x4 矩阵键盘（只用 1 个键）、1 个单总线温度传感器 DS18B20、蜂鸣器、时钟芯片 PCF8563。


SM420364共阴
DPY1
DPYA 11
DPYB 7
DPYC 4
DPYD 2
DPYE 1
DPYF 10
DPYG 5
DPYDP 3
a b c d e f g DP
1 2 3 4
12 9 8 6
RD9 200
RD10 200
RD11 200
RD12 200
DIG0 DIG1 DIG2 DIG3
SM420364共阴
DPY2
DPYA 11
DPYB 7
DPYC 4
DPYD 2
DPYE 1
DPYF 10
DPYG 5
DPYDP 3
a b c d e f g DP
1 2 3 4
12 9 8 6
RD13 200
RD14 200
RD15 200
RD16 200
DIG4 DIG5 DIG6 DIG7
+5V CD3
0.1uF
U2
nOE VCC
D0 Q0
D1 Q1
D2 Q2
D3 Q3
D4 Q4
D5 Q5
D6 Q6
D7 Q7
GND CLK
74HC574
D0 D1 D2 D3 D4 D5 D6 D7
1 2 3 4 5 6 7 8 9 10
20 19 18 17 16 15 14 13 12 11
DPYA DPYB DPYC DPYD DPYE DPYF DPYG DPYDP CLK0
+5V CD4
0.1uF
U3
nOE VCC
D0 Q0
D1 Q1
D2 Q2
D3 Q3
D4 Q4
D5 Q5
D6 Q6
D7 Q7
GND CLK
74HC574
D0 D1 D2 D3 D4 D5 D6 D7
1 2 3 4 5 6 7 8 9 10
20 19 18 17 16 15 14 13 12 11
DIG0 DIG1 DIG2 DIG3 DIG4 DIG5 DIG6 DIG7 CLK1
+5V CD5
0.1uF
U4
nOE VCC
D0 Q0
D1 Q1
D2 Q2
D3 Q3
D4 Q4
D5 Q5
D6 Q6
D7 Q7
GND CLK
74HC574
D0 D1 D2 D3 D4 D5 D6 D7
1 2 3 4 5 6 7 8 9 10
20 19 18 17 16 15 14 13 12 11
BEEP CLK2
+5V CD6
0.1uF
U5
nOE VCC
D0 Q0
D1 Q1
D2 Q2
D3 Q3
D4 Q4
D5 Q5
D6 Q6
D7 Q7
GND CLK
74HC574
D0 D1 D2 D3 D4 D5 D6 D7
1 2 3 4 5 6 7 8 9 10
20 19 18 17 16 15 14 13 12 11
G0 G1 G2 G3 G4 G5 G6 G7 CLK3


CD1
0.1uF
+5V
U1
KO0
KO1
KO2
KO3
RST
DT
CLK0
CLK1
CLK2
CLK3
P1.0/T2
P1.1/T2EX
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7
RST
P3.0/RXD
P3.1/TXD
P3.2/#INT0
P3.3/#INT1
P3.4/T0
P3.5/T1
P3.6/#WR
P3.7/#RD
XTAL2
XTAL1
VSS
VDD
P0.0
P0.1
P0.2
P0.3
P0.4
P0.5
P0.6
P0.7
#EA/VPP
ALE
#PSEN
P2.7
P2.6
P2.5
P2.4
P2.3
P2.2
P2.1
P2.0
P0_0
P0_1
SCL
SDA
KI0
KI1
KI2
KI3
D7
D6
D5
D4
D3
D2
D1
D0
89C52
+5V
RP1
10K
RZ3 0R
+5V
XT1
CP1
20P
11.0592M
CP2
20P


K0
K1
K2
K3
K4
K5
K6
K7
K8
K9
KA
KB
KC
KD
KE
KF
KEY-4P 6.2
KO0
KO1
KO2
KO3
KI0
KI1
KI2
KI3

4x4矩阵键盘

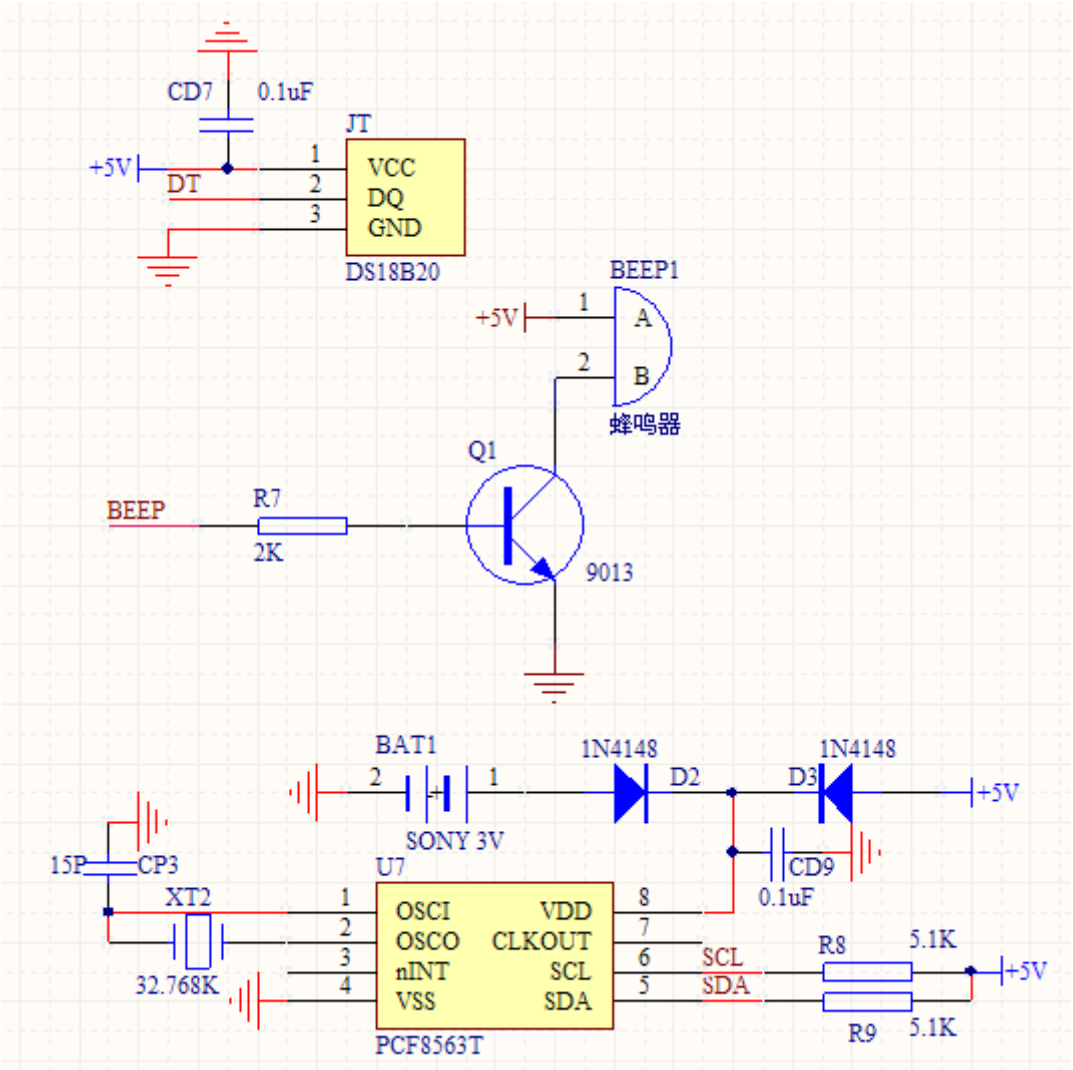


图 3-3 《汽车智能电子表》电路原理图

开发环境如下：

- MCU：89C52 单片机；
- 开发环境：Keil C51；
- 实时操作系统：TreeOS，构件库：TreeOS ComLib A1。

第一步，先按照功能要求画出程序流程图如下：

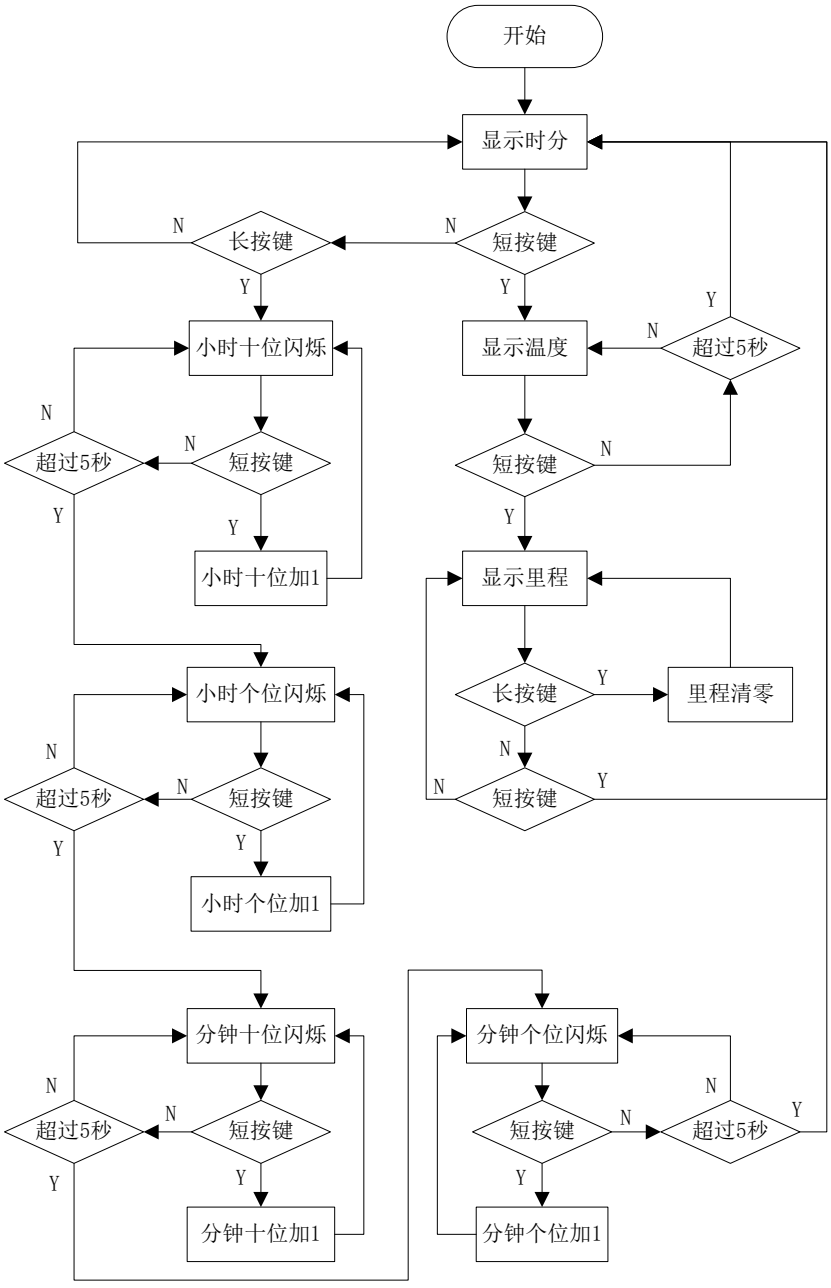


图 3-4 《汽车智能电子表》程序流程图

第二步，根据流程图，我们从中可以划分场景，其树形图如下：

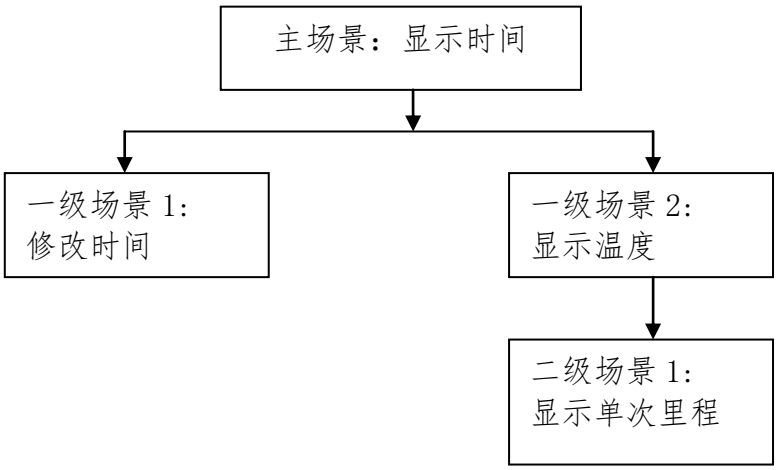


图 3-5 《汽车智能电子表》场景关系图

第三步，在工程文件中建立各个文件组，并从 TreeOS ComLib A1 软件库中提取所需的 C 文件，包括：

1、 主场景 main scene 文件组: TreeOS_main（模板）;

2、 单片机 mcu 文件组：单片机的设置文件 TreeOS_mcu、单片机的中断设置文件 TreeOS_int;

3、 驱动程序 devices 文件组: 根据电路原理图，把各部分设备的驱动程序拷贝进来;

4、 常用函数库 functions 文件组：TreeOS_stdlib 等;

5、 次级场景 sub scenes 文件组：TreeOS_scn_subs（模板）等;

6、 用户程序 user 文件组：如字库，运算程序等。

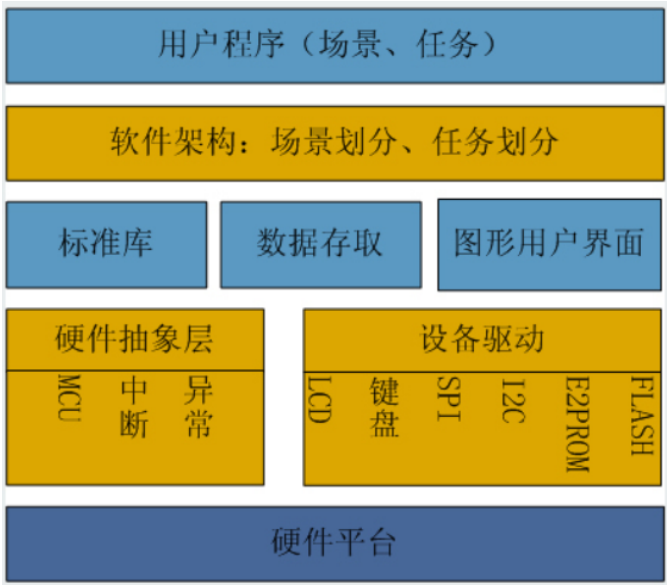


图 3-6 TreeOS 的层次结构

第四步，根据电路原理图以及功能需要对库程序进行配置、剪裁，主要是修改头文件。例如，对于键盘的部分配置程序如下图。

```
//键个数：正确选用可节约内存
#define KEY_DATA_TYPE   ui16  //M/  8/16/32个键分别使用ui8/ui16/ui32
//是否使用长按键
#define USE_LONGKEY    //M/
//是否使用加速键
//#define USE_SPEEDKEY  //M/
//是否使用组合键
//#define USE_UNKEY     //M/
//输入键盘带0~9数字
//#define NUM_KEYBOARD //M/

//键盘扫描周期
//键扫描周期,单位=T0周期(例如 1ms)
#define SCANKEY_CYCLES  20  //M/
//长按键
#define LONGKEY_SCANKEY_CYCLES 100  //M/ 单位=键扫描周期SCANKEY_CYCLES,
                                    //定义按多长时间开始出现longkeyval
#define NOKEY_SCANKEY_CYCLES  1000  //M/ 无按键周期设置
#define NOKEY_SCANKEY_CYCLES_5S  250  //M/ 5秒无按键周期设置
```

图 3-7 键盘的部分配置程序截图

第五步，按照标准格式，编写各个场景的程序。

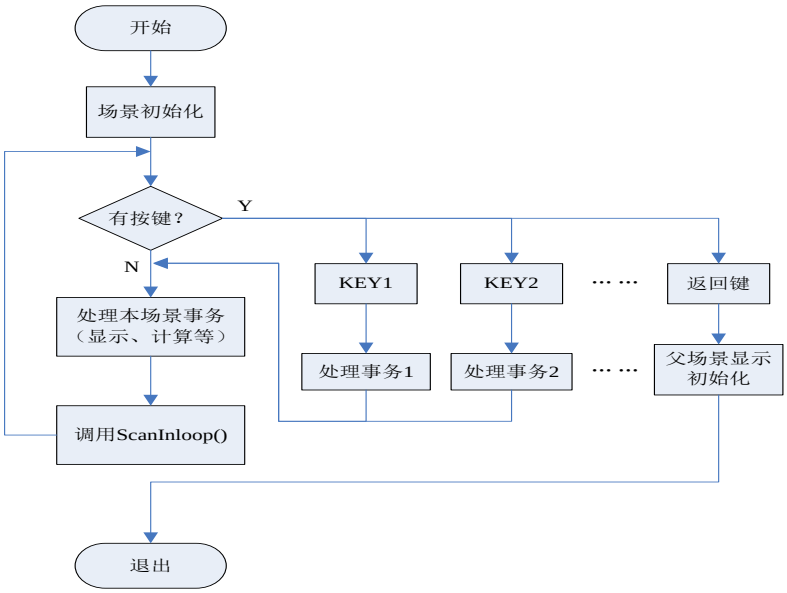


图 3-8 场景程序流程图

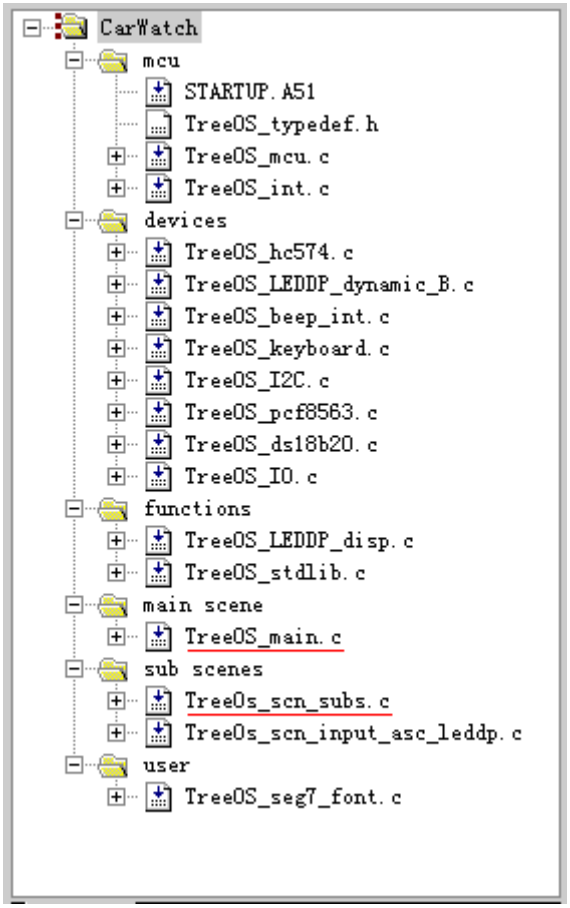
例如，显示温度的场景程序如下：

```
/*****************************************TreeOS Real-Time Operating System*****
Function Name:
Description   : 显示温度（一级场景）
Input         :
Output        :
Return        :
*******************************************************************************/
void scn_sub_show_temp(void)
{

  scene=3;  //M/ 以实际情况，设置本场景编号

  //场景显示初始化
  //显示温度值
    itoa_ltx(Temp_18b20/10);     //温度读数已被放大10倍
    LEDDP_show_strbuff(3, LEDDP0, ALIGN_LEFT);  //左对齐，3位显示
    LEDDP_buff[LEDDP3]=seg7_C;  //"C"

  keyval=0; //清键值
  longkeyval=0;
  while(1)
   {
     WDR();
     if(keyval)
        {
         if(keyval==SINGLE_KEY)     //短按键，进入下级场景
            {
             scn_sub_show_dist();  //显示里程
             return;
            }
          keyval=0; //清键值,等待下一个按键
          longkeyval=0;
```

```
136             //更新显示
137
138           }
139
140
141         //局部任务     //显示温度值
142         itoa_ltx(Temp_18b20/10);  //温度读数已被放大10倍
143         LEDDP_show_strbuff(3, LEDDP0, ALIGN_LEFT);  //左对齐，3位显示
144         LEDDP_buff[LEDDP3]=seg7_C;  //"C"
145
146         //5秒没有按键的话，自动返回
147         if(nokeytimer>NOKEY_SCANKEY_CYCLES_5S)  return;
148
149         //在大循环中需要不断扫描的应用
150         scan_in_while();
151     }
152 }
```

图 3-9 显示温度的场景程序截图

最后的项目工程结果如下：



```
compiling TreeOS_IO.c...
compiling TreeOS_main.c...
linking...
Program Size: data=105.0 xdata=0 code=4220
creating hex file from "CarWatch"...
"CarWatch" - 0 Error(s), 0 Warning(s).
```

图 3-10 《汽车智能电子表》项目工程结果截图

至此，整个编程工作就完成了。我们看到，TreeOS 1.0 完成了绝大部分工作，软件工程师所要做的只是对库程序做适当的配置、剪裁，以及编写用户程序就可以了。

这个程序最后的完成情况如下：

生成 HEX 文件大小：4K 字节；

C 文件总数：16 个；

用户程序 C 文件数：2 个；

程序总行数：4100 行；

用户程序行数：130 行，占比 3%；

TreeOS 提供程序行数：3970 行，占比 97%；

开发周期：约 1 天（设计软件架构、配置剪裁库程序、编用户程序）；

不使用 TreeOS 开发周期估计：15-30 天。

我们惊喜地发现：使用 TreeOS，开发过程变得非常简单，开发周期也大大缩短了。

第四章 TreeOS 的工作原理

&4.1 概述

MCU 小巧灵活，应用范围广泛，应用领域十分复杂。如何把各种复杂多变的应用统一在一个系统框架内，对程序员来说是一个极大的挑战。经过多年的摸索和实践，我们提出了面向场景编程的构件化方法，解决了系统软件架构的问题。

TreeOS 是一种无核的、软件构件化的、实时嵌入式操作系统。

我们可以把 TreeOS 实时操作系统看成由两部分组成：

第一部分：设计软件架构方法，也是构件化方法；第二部分：软件构件库。

图 4-1 TreeOS 的构成

&4.2 解构（deconstruction）系统

按照传统的操作系统思路，操作系统主要是解决系统底层问题，包括与硬件的接口、实时内核、驱动程序等等，是自下而上从底层做起的。

而 TreeOS 是一个构件化的操作系统，首先面临的问题就是如何

对应用系统进行解构。解构的意思就是把一个整体的东西拆解为若干分立的部件，就像孩子把一个玩具拆解为一堆的零部件一样。找到解构的方法实际上就是找到了系统的构件化方法。

系统构件化（模块化）就是把一个复杂的系统分解为若干构件（模块）的过程。每个构件完成一个特殊的功能，构件之间既相互独立又有关联，构件通常是可在不同应用系统中通用的。构件既可从构件库中调用，也可以依实际应用定制，然后把所有构件按照一定原则组装起来，实现系统所要求的功能。

这种构件化系统的方法有两个非常显著的优点，一是可以复用，二是化繁为简，把复杂的问题转化为一系列简单的问题。

图 4-2 构件—建筑工地

TreeOS 解构系统分为两步，包括场景划分和任务划分。

&4.3 场景划分

场景是 TreeOS 中提出的一个非常重要的概念。场景的概念是我

们在长期的 MCU 开发实践中自然而然产生的，它借用了戏剧场景的概念。

TreeOS 按照人机交互界面或控制进程为主线，把应用程序的运行过程分解为一串串场景。

每个场景都有自己独特的故事，场景之间又互相关联。这有如戏剧一样，所不同的是戏剧进程前后顺序连贯，有始有终，故事讲完了，戏剧也就结束了。而应用程序的运行过程则要复杂的多，而且一直运行到关闭系统为止。

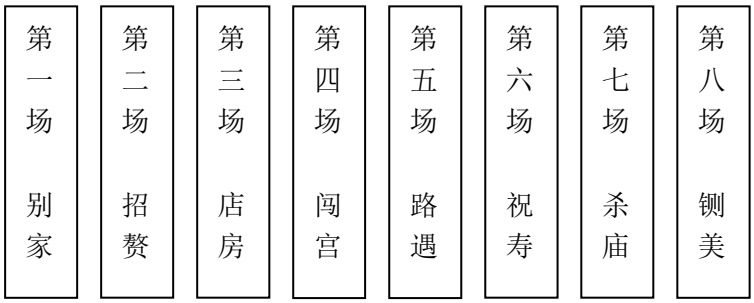


图 4-3 京剧《铡美案》共分八场

划分场景的本质，实际上是一种“分时”处理的方法（注意此处的分时与分时操作系统的概念不同）。在每个划分的场景时段中，我们可以集中精力把发生在该时段中的任务和事件处理好就可以了。这种按时间分段的构件方法，我们称之为“分时构件法”。

通过这种划分场景方法，可把纷繁复杂的应用分解为一个个简明、形象的模块。这种方法符合人类日常思维的方法和习惯，使开发者能更有效地思考问题，并以其他人也容易看得懂的方式把自己的认识表达出来。

学习过状态机的读者以后会发现，划分场景的方法与状态机很相似，场景也是一种比较稳定的特种“状态”。

&4.4 场景（scene）的定义

一个人机交互界面或是一个控制过程，即可称之为场景。

嵌入式应用系统中，绝大部分都需要人机交互。按照人机交互界面来划分场景，十分符合人类的思维习惯。

为了更好理解场景的概念，下面举几个例子：

1） 在有液晶显示的系统里，常常会使用菜单。每一级菜单选择界面就是一个场景；

2） 输入一个参数过程就是一个场景；

3） 步进电机完成一个操作，然会返回起点，这是一个场景；

4） 搬运机器人等待识别目标出现，这可以设为一个场景；然后抓起目标运达目的地，这又是一个场景。

实际上，对于人机交互界面，场景很像是 PC 机的“窗口”。尽管我们开了多个窗口，但只有一个是激活的（可以接受控制）。

从以上例子可以看出，场景大都是以完成某项比较完整的作业过程来划分的。

&4.4.1 场景的三个要素

想想我们在电脑里操作一个窗口的过程：打开一个窗口，然后用键盘输入内容或用鼠标操作，内容显示在窗口上，完成任务后，我们就用鼠标关闭它。

场景的运行过程与“窗口”基本一致。从中可以发现构成场景

的三个要素：显示、按键和生存周期。

&4.4.2 场景的要素之一：显示（display）

场景的显示是供操作人员观察系统运行状况的。绝大多数单片机应用中都要用到它，具有普遍意义。

场景的显示可分为三种类型：

- 直接显示型（如液晶、数码管显示等）；
- 提示显示型（状态指示灯或者声音提示等）；
- 暗箱型（既无显示又无提示，比较少见）。

&4.4.3 场景的要素之二：按键

场景中的按键是广义的。为了区别于普通的键盘，我们把它称为场景按键。场景按键的来源有：

- 键盘设备；
- 来自系统外部的控制信号（例如来自串口的一个指令）；
- 系统内部产生的控制信号（例如机器人已完成一次搬运工作发出的信号）；
- 系统内部的定时信号（例如长时间未操作键盘，自动退出输入界面），等等。

场景按键的作用包括：

- 输入参数；
- 控制设备；
- 改变程序流程；
- 切换场景，等等。

按键是可复用的，同一个按键在不同场景下的作用可能不同。

场景按键是不可或缺的，场景之间的切换必须由场景按键来执行。

&4.4.4 场景的要素之三：场景的生存周期（scene life cycle）

场景的生存周期是指应用程序从进入该场景运行，直到退出该场景，之间所经历的时间。这里我们不必去考虑场景运行途中的转去执行中断的处理时间。

场景的生存周期是在划分场景时一个重要的参考指标。生存周期的长度是相对场景循环体的运行周期来讲的，至少要运行一遍场景循环体程序，场景才有意义。

场景生存周期一般都比较长，一般需要运行数秒或更长时间。当然有时根据实际需要，几十 ms 也是可以的。如果要求再短的话，则可以考虑轮询办法来处理。

场景生存周期的上限则无需考虑，这与应用情况有关，甚至可以无限长，直至系统关机。

&4.5 场景的分类

TreeOS 把场景分为三类：初始场景、主场景和次级场景。

系统的应用程序由一系列场景组成，其结构示意如下：

```
main( )     //主函数
  {
    ……
    init_scene( ); //初始场景
```

```
  ……
  main_scene( ); //主场景
}

sub_scene _1( )   //次级场景 1
 {
   ……
 }
sub_scene _2( ) //次级场景 2
 {
   ……
 }
 ……
```

&4.5.1 初始场景

初始场景是系统上电后对系统运行初始化的一种场景（注意：它并非设备初始化函数）。

例如，有时候设备上电后处于待机状态，需要用户按一下开机键才开始运行，这个待机过程就是初始场景。

再比如，有时上电后需要选择一下运行参数，然后再进入正常工作程序，这个选择运行参数的过程就是初始场景。

实际上，初始场景也可以看做是一种次级场景。为了显得直观和容易理解，我们把它单独列出来。

&4.5.2 主场景

主场景是运行在应用程序主函数 main()中、包含程序主循环的场景。一般情况下，系统主要运行在主场景。系统中必须有而且只能有一个主场景。所有次级场景都是从主场景分支出去的，它处在一个金字塔塔尖，相当于一个家族的最长着。

参见&4.5.1 中的 main_scene()。

&4.5.3 次级场景

从主场景分支出去的场景称为次级场景。系统除了一个主场景和初始场景（一般 0~1 个）之外，其它的场景都是次级场景。次级场景按照运行进程的先后顺序分为一级场景、二级场景等等。

一般中低档单片机应用中，次级场景以 1~3 级居多，4 级以上就很少见了。

在划分场景时，需要注意一个重要问题。因为次级场景级数越多，函数的嵌套调用就越多，系统堆栈需要的内存空间就越大。所以应尽量使次级场景的级数最少！

&4.6 场景之间的关系

这里讲的是主场景、次级场景之间的关系。

按照场景之间的关系来分，场景又可分为父场景和子场景。如图所示，假设系统在场景 A 运行之后进入场景 B，在场景 B 运行之后进入场景 C，那么对于场景 B 来说，场景 A 是它的父场景，场景 C 是它的子场景。每个次级场景有可能既是父场景又是子场景，而主场景就只能是父场景。

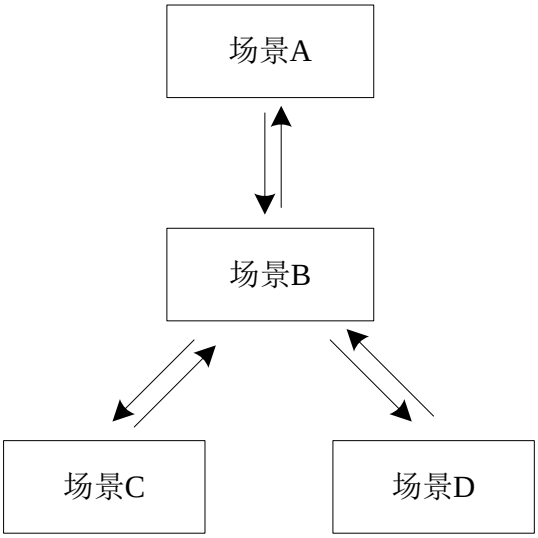


图 4-4 场景关系图

为了更好地实现模块化设计，TreeOS 规定，每个子场景只能有一个父场景，而且，子场景退出运行后，必须回到它的父场景。另一方面，一个场景则允许有多个子场景，例如场景 C、场景 D 都是场景 B 的子场景。也就是说，每个次级场景只能有一个入口，但可以有多个出口。这样就避免了场景之间关系出现混乱，有助于实现模块化设计。

&4.7 场景的多级树形结构

通过以上的场景的分类以及场景之间关系的限定，我们发现，整个系统的所有场景组成了一个多级树形结构。参见下图。

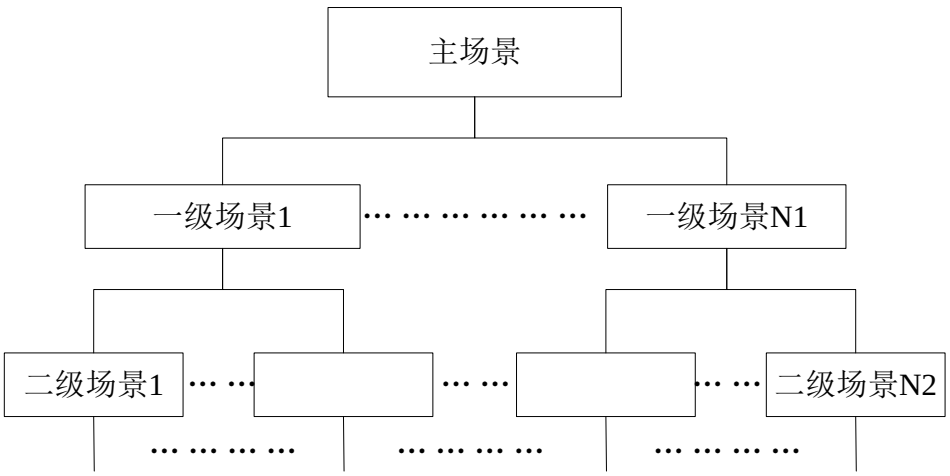


图 4-5 场景的树形结构

系统有且只能有一个主场景。一般情况下，系统都是处在主场景中运行。当需要进行一些具体操作，才进入次级场景。完成相应的操作后，系统最终又回归到主场景。

由主场景分支出多个一级场景，每个一级场景又可以分支出多个二级场景，直至最后全部场景形成了一个多级的树形结构。每一个次级场景只有一个父场景，而子场景则可以有多个。子场景退出运行后，必须回到它的父场景，这就意味着次级场景只能有一个入口，但可以有多个出口。这一点非常重要，它符合模块化设计的要求。

这种结构的特点是等级分明，条理清晰，符合模块化编程的特点。整个系统有如一个金字塔，塔尖就是系统的主场景。

这种多级树形结构，是我们在长期的实践中总结出来的一种适用性强、直观的程序结构，它具有一定的普适性。

&4.8 场景的程序表达

所有的场景，都可以用同一个程序模板表达出来，参见下图。

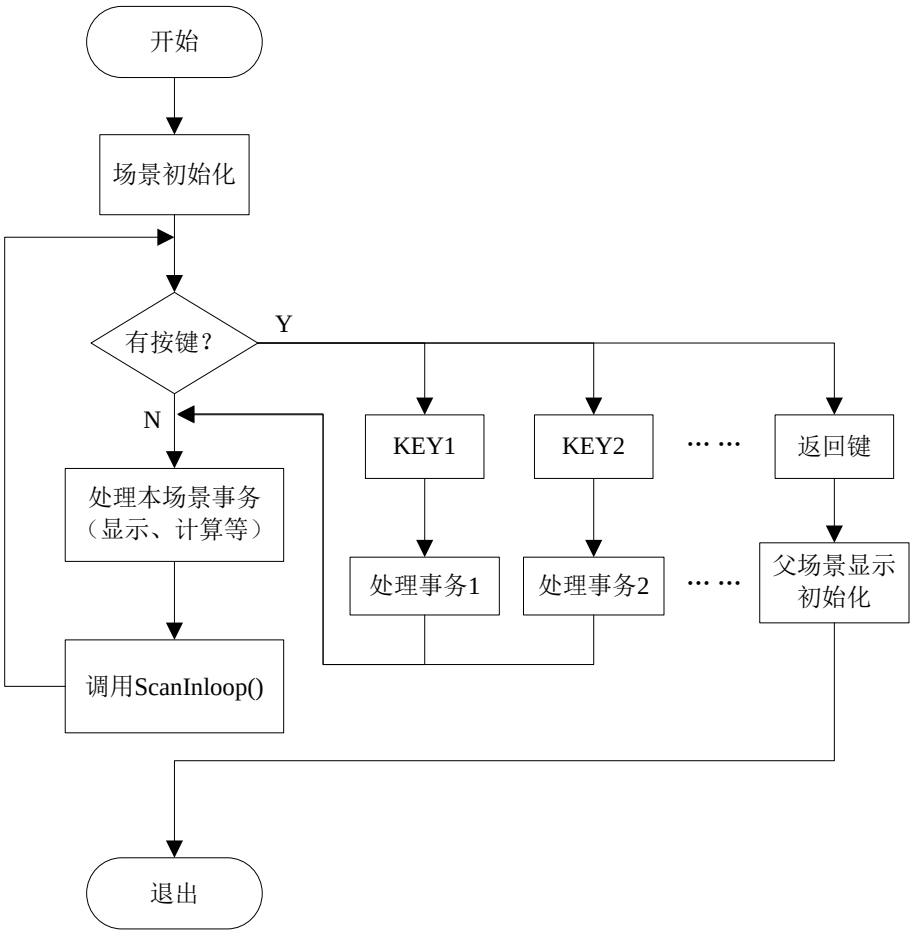


图 4-6 场景的程序表达

图 4-6 是实现一个场景的模板流程图，从中我们可以看到场景的两个特征。首先是每个场景都包含一个无限循环体，即一般需要较长时间的循环运行；其次，必须有按键操作。具体流程是这样的：

1） 首先是场景的初始化，一般是更换显示画面；

2） 进入无限循环体后，不断地判断是否有新键值。若有，进行按键操作处理；若检查到“返回键”，则退出子程序，返回父场景的循环体中。注意，在返回之前，需要调用父场景的初始化程序，这样才能与父场景实现无缝连接。对于主场景，“返回键”则没有作用；

3） 循环体的中间部分是本场景的处理程序，即局部任务，如更新显示内容、计算、控制等，每一次循环都运行一次；

4） 每个循环体都应该包含一个公共调用子程序 ScanInLoop()（或 scan_in_while()），该子程序包含了所有需要及时处理的全局任务，例如读取按键值、读取时钟、信号采集、通讯处理等等，每一次循环都调用一次。

&4.9 一种特殊的场景

这里介绍一种特殊的场景，它由数个单向前进关系的同级场景组成，我们称之为“场景串”，这其实是一种场景组合，参见下图。

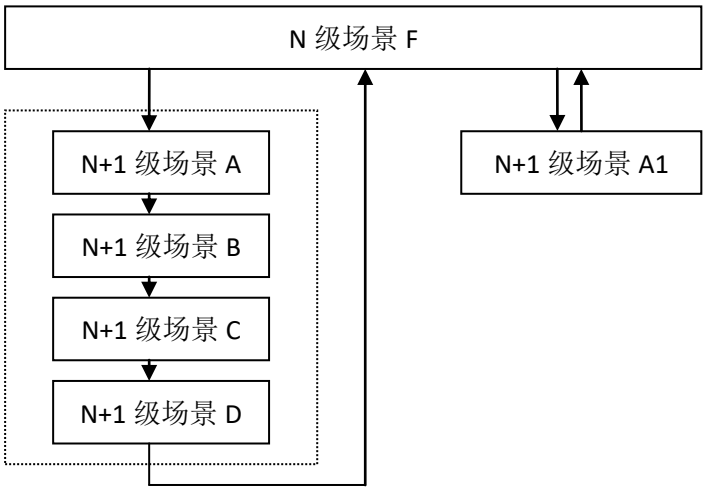


图 4-7 场景串示意图

从上图中可见，场景 A、B、C、D 组成一个场景串。

细心的读者会提出这样的问题，场景 A 是场景 F 的子场景，它退出后应该返回到场景 F 才是，为何却进入场景 B？其实看一下下图就明白了。我们可以这样理解，场景 A 退出后返回到场景 F，然会马上就进入场景 B，这样从效果上看就是从场景 A 直接进入场景 B。其它

的类推。

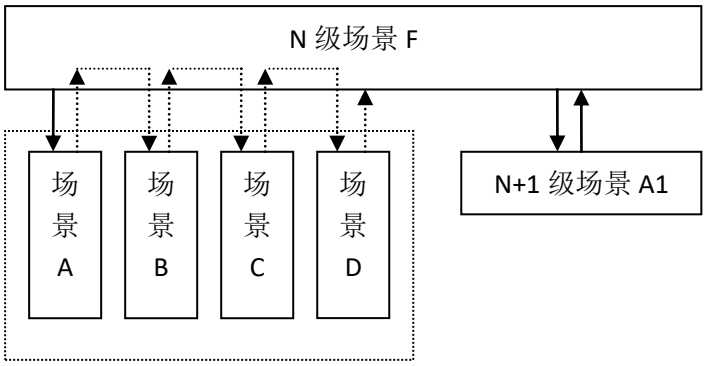


图 4-8 场景串示意图

尽管“场景串”并不是一个实际的场景，但在场景分析和编程过程中，可以把它当做一个组合式场景来看待，这有利于简化分析过程，而且有利于简化程序。实际上，也可以把“场景串”理解成为一个有多个循环体的场景。

需要提醒读者的是，不要把这种“场景串”当做多级场景来处理，参见下图。尽管这种处理方法合理合法，而且还符合思维习惯，程序上也容易实现，但由于存在子程序多级调用，需要更多堆栈从而消耗宝贵的内存，因此我们不提倡这种做法。

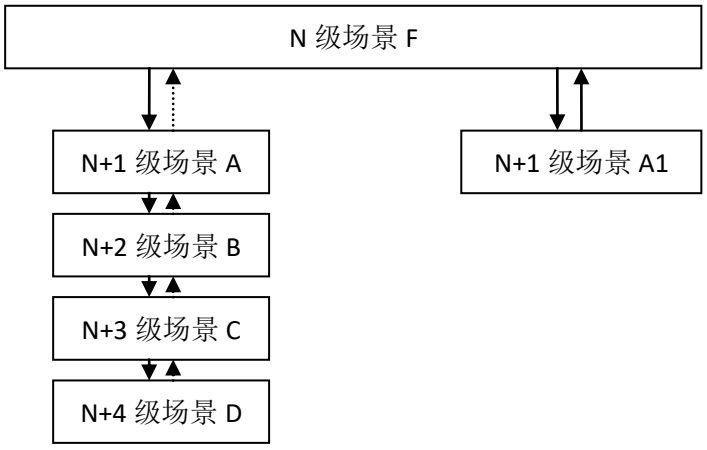


图 4-9 场景串被当做多级场景嵌套处理

“场景串”在实际应用中会经常碰到，最典型的是输入时间。例如，以第三章所述的《汽车智能电子表》案例为例，输入时间就是一个“场景串”，被当做一个场景对待，它包含 “输入时”、 “输入分”2 个场景。参见下图。

```
/*****************************************TreeOS Real-Time Operating System****
Function Name:
Description  : 场景：修改时间
Input        :
Output       :
Return       :

注：这是场景串，由一串单向前进关系的同级场景组成。
*******************************************************************************/
void scn_sub_modify_time(void)
{
            //输入小时 (次级场景)
            bcd_to_ascbuff(DateTime.hour);
            if(scn_input_asc(2, LEDDP0))
              {
               DateTime.hour=ascbuff_to_bcd();
               if(send_pcf8563()) send_pcf8563();  //写入时钟芯片
              }
            else    //取消
              {
               bcd_to_ascbuff(DateTime.hour);
               disp_ascbuff(2, LEDDP0); //显示初值
              }

            //输入分钟 (次级场景)
            bcd_to_ascbuff(DateTime.min);
            if(scn_input_asc(2, LEDDP2))
              {
               DateTime.min=ascbuff_to_bcd();
               if(send_pcf8563()) send_pcf8563();  //写入时钟芯片
              }
            else    //取消
              {
               bcd_to_ascbuff(DateTime.min);
               disp_ascbuff(2, LEDDP2); //显示初值
              }

}
```

图 4-10 场景串软件实例

&4.10 任务划分

通过对场景的划分，并按照树形结构组织起来，系统的软件架构已经搭起来了。但这还远远不够，还不够细致，也没有体现出实时系统的特点。接下来需要对系统的各种任务进行划分。

划分任务，也称为任务管理，实际上是对各种任务的执行顺序进

行排列，这对于实时系统很重要。传统的嵌入式实时操作系统中，这部分任务管理工作属于实时内核的一部分。

&4.11 任务的定义

任务就是一段具有一定功能的程序体，也称为一个线程。一个任务除了程序代码之外，还包括任务控制块、任务堆栈等内存数据。由于任何时候单处理器只能有一个任务运行，因此可以认为该任务独占CPU。

实际上，划分任务是对场景的进一步细分。一个场景是多个任务的集合。这可以把场景看做一个进程，而任务则是该进程下的线程。如果说场景的划分是在时间轴上对应用程序进行划分，那么任务的划分则可以看成是在“程序空间”轴上对程序进行划分。

为了确保系统的实时性，TreeOS 把系统的任务划分为两类，即抢先式任务和普通任务。

&4.12 抢先式任务

当 CPU 正在运行某个任务时，此时一个更重要的新任务需要马上执行，CPU 就先保存原来任务的状态，切换到新任务执行，这个新任务就称为抢先式任务。

对于需要紧急处理的、实时性要求高的任务，如时间定时、外部通讯、脉冲计数、信号捕获等等，可以把它们分配为抢先式任务。TreeOS 把抢先式任务放在系统的中断程序中处理（即中断级任务），其优先级由系统的中断优先级决定。这一部分处理程序与硬件有关。

&4.13 普通任务

有些任务虽然也有实时性要求，但并非那么迫切，这部分任务我们称之为普通任务。在 TreeOS 中，普通任务是放在场景循环中按顺序调度运行（后台运行）。为了简化设计和节省系统资源，TreeOS 规定普通任务需遵守以下三点：

其一，普通任务不分优先级，所有任务排成队列，按顺序逐一处理；

其二，普通任务一直运行到它们主动让出 CPU 为止（中间允许转去执行中断程序）；

其三，系统在调用某个子程序的过程中，不允许强行再次运行该子程序，但允许运行过程中间转去执行中断程序，即子程序是半可重入的。

普通任务采用这种顺序调度任务方式的好处是无需任务管理和内存管理。系统在某一个时刻只能执行一个普通任务，因此无须同时为很多任务开辟内存缓冲区，减小系统堆栈，既节省内存又便于管理。

&4.14 全局任务和局部任务

TreeOS 把普通任务又分为全局任务和局部任务。全局任务在所有场景中都必须调用，而局部任务则是各个场景中所特有的任务，它们只在该场景中运行。这种细分任务的方法，使场景的编程更加简单，进一步体现了模块化设计的特点。

&4.15 利用拆分技术和插入技术提高系统的实时能力

为了提高操作系统的实时能力和应变能力，TreeOS 还提出了拆分技术和插入技术。

对于运行时间长、实时性要求不高的普通任务，可以拆分成两部分或两部分以上的任务，而每部分运行时间都较短。每次循环只运行其中的一部分，多次运行才能获得结果。这实际上是一种多道系统的处理方法，也称为分时调度。例如，对键盘可以通过多次循环扫描的方法，监控按键释放动作出现后再进行处理。这样可以避免占用处理器长时间延时等待的问题。拆分技术可以减少循环总时间。

拆分技术会经常用到。

而对于实时性要求较高的普通任务，可以在任务顺序队列中插入两次或两次以上的运行机会，使任务的实时性要求获得满足。插入技术使循环总时间增加。例如，在一个场景中，任务顺序队列完成一次循环最大时间需要 50ms，而某个任务则需要 30ms 内执行一次，那么就可以采用这种插入技术，使该任务的实时性要求获得满足。

在这里，TreeOS 并没有使用优先调度算法（所谓的优先调度，是指给各个任务设定调用级别，优先级高的任务先调用。），主要是基于以下几点考虑的：

1、通过对一百多个单片机实际应用项目的统计，我们发现需要优先调度的情况极为少数，因此 TreeOS 舍去了这种算法。其好处是系统更为简便，也节约了宝贵的内存；

2、个别需要优先调度的任务，采用上述的插入方法，就可以满足要求。这种方法虽然显得有点笨，但简单实用；

3、如果需要，还可以把需要优先调度的任务提升为抢先式任务，使用中断进行处理。

&4.16 构建系统

构建系统实际上是解构系统的反向操作。

通过以上对应用系统的场景划分和任务划分，我们可以对每个场景和每个任务进行编程，成为一个个具有特定功能的软件模块，然后按照相应的原则把这些模块组装起来，就是一个符合用户需求的应用系统。

&4.16.1 构建系统步骤

TreeOS 按照以下步骤构建应用系统：

第一步，在系统运行框图的基础上，把系统进程分解成一串不同功能的场景；

第二步，把所有场景按照多级树形结构组织起来；

第三步，把系统的实时任务划分为两类，即抢先式任务和普通任务。普通任务又分为全局任务和局部任务；

第四步，MCU 的初始化以及配置系统的中断系统，并在中断程序中编写抢先式实时任务程序；

第五步，编写或调用 TreeOS 的设备驱动程序库、常用程序库等，依实际应用进行剪裁、配置，供各种任务调用；

第六步，组织、编写全局任务的程序，并形成一个汇总调用子程序 ScanInLoop()（或 scan_in_while()）。各个场景都需要在循环中

不断地调用该子程序；

第七步，编写各场景的程序。每个场景都有自己特有的局部任务，它们都需要放在该场景的循环体中不断地循环调用。

至此，整个应用程序编写完成。

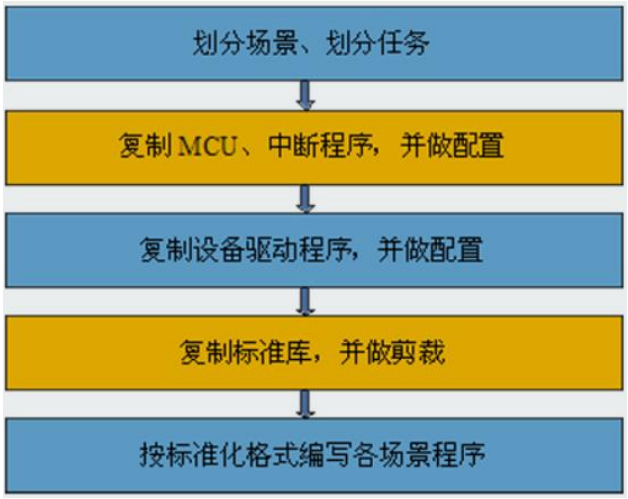


图 4-11 使用 TreeOS 开发步骤

&4.16.2 多级树形的循环嵌套结构

TreeOS 中每个场景的运行都是一个循环，因此整个应用程序就形成一个多级树形循环嵌套结构，如图 4 所示：

1） 程序开始后，首先是各种设备的初始化，如定时器初始化、IO 口初始化、通讯口初始化、AD 初始化、中断设置等等；

2） 然后进入一个初始循环。该循环也是一个场景，如系统自检、显示公司标识等等。当然，初始场景并非是系统必需的；

3） 最后进入主循环，它就是主场景。系统只有一个主场景。一般情况下，系统都是运行在主场景循环内。当需要设置

参数，或者进行一些具体操作，才进入次级场景循环。当完成相应的操作后，系统最终又回归到主场景循环中运行；

4） 次级循环是按照树形结构来组织的，每个次级循环对应一个次级场景。每一个次级场景只有一个父场景，而子场景则可以有多个，子场景退出运行后，必须回到它的父场景，这就意味着次级场景程序只有一个入口。

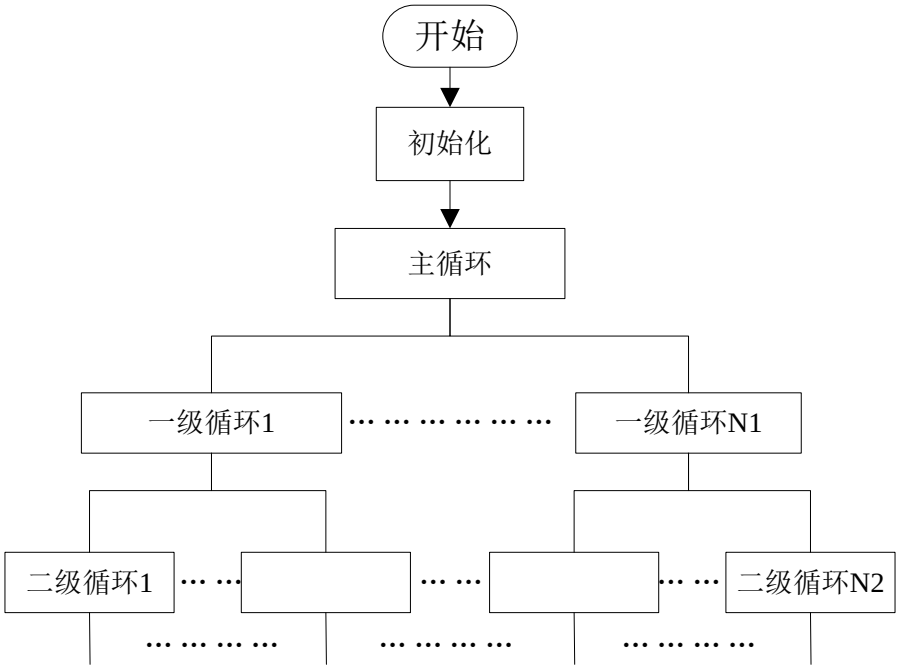


图 4-12 循环的树形结构

第五章 TreeOS 特点分析

本章对 TreeOS 操作系统进行全面分析和总结。

- 一种“无核-构件化”实时操作系统；
- 可以适用 MCS51 等中低档单片机；
- 一种全能型的操作系统：从设备驱动、程序架构到用户层设计，全方位解决方案；
- 提出面向场景编程的新技术；
- 现有统计表明，代码复用率可高达 70~90%！大大减少了开发工作量；
- 初学者亦可轻松掌握。

&5.1 TreeOS 是一个构件化的操作系统

软件构件化是近年来的热门技术。构件化技术特别适合嵌入式应用，其特点是：可配置、可剪裁、可复用。TreeOS 首次把构件化技术引入到 MCU 领域。

嵌入式软件开发正朝着平台化、标准化、可复用的方向发展。

&5.1.1 分时构件法与面向场景编程

TreeOS 的核心技术是提出了“面向场景”编程的软件构件化方法，可以称之为分时构件法。分时构件法就是把一段特定时间内运行的完

成特定功能的一段程序做成一个软件构件。

TreeOS 中的场景就是这样一种分时构件。TreeOS 在业内首次提出了“面向场景”编程的概念。

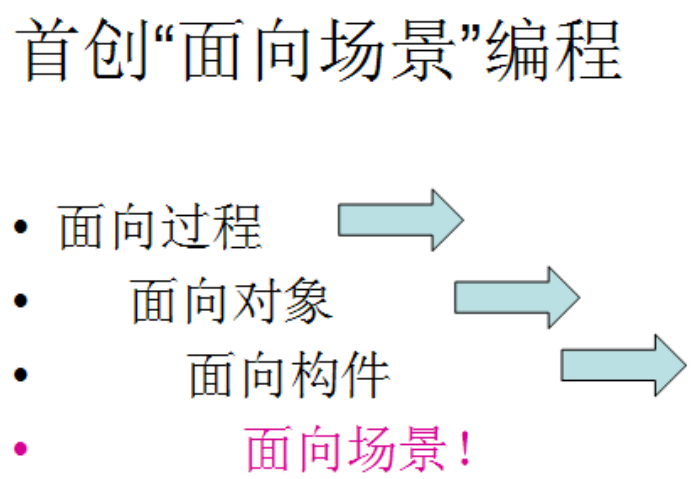


图 5-1 软件技术演变过程

场景是应用在用户层的。它可以作为一种大粒度的、抽象级别高的软件构件进行复用。这种解构系统的方法是通用的，可以快速为用户建立起软件架构，因此可以称之为“方法构件”。它具有可复用的特点，同样可以提高编程效率。

&5.1.2 任务网格化

TreeOS 对系统的解构分为两大部分：场景划分和任务划分。解构之后的效果，是把复杂的应用系统进行“任务网格化”。所谓“任务网格化”，就是把应用系统分成一个个网格，每个网格代表一个任务。就像切西瓜一样，横几刀竖几刀，切成条条块块，方便食用。

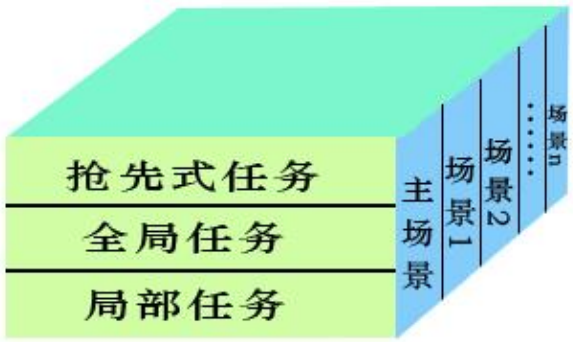


图 5-2 系统结构示意图

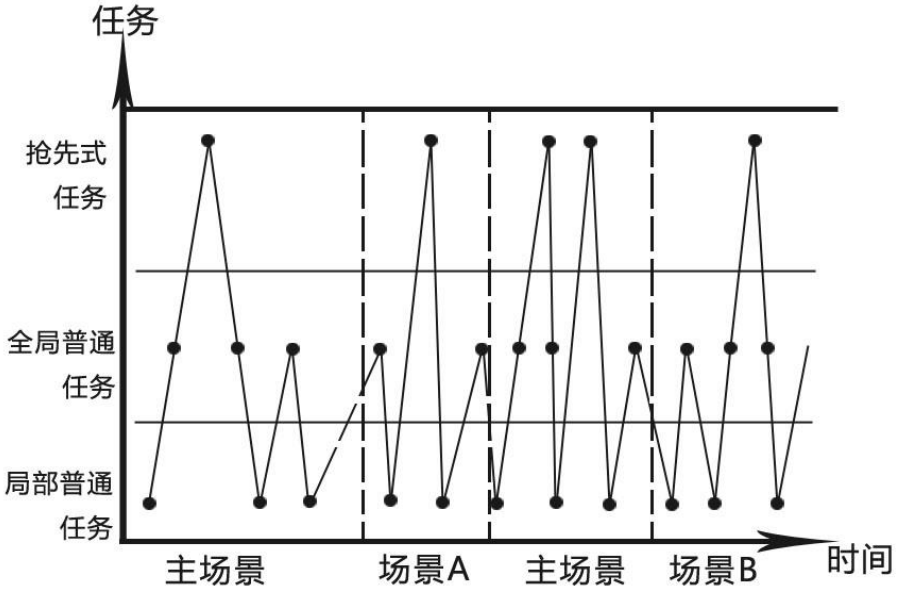


图 5-3 任务运行过程示意图

如上图所示，横向，在程序运行时间方向，分割为一个个场景；纵向，在程序运行空间方向，分割为抢先式任务、普通全局任务和普通局部任务。这样就把整个应用系统的程序就被分解为一个个任务模块。

&5.1.3 建立构件库

另一方面，系统的程序结构明确下来后，在此基础上就可以编写

有关的设备驱动程序、常用库程序等。不仅每个场景、任务都可以用一个构件来描述，其它诸如设备驱动程序、异常处理程序、库程序等也可以用构件表示出来。

TreeOS 的软件构件库，称为 TreeOS ComLib，包含以下四方面的内容：

1) 各种 MCU 的初始化、配置模板（TreeOS_mcu）；
2) 各种周边设备的驱动程序；
3) 常用中间件、边缘计算等库程序；
4) 可通用的场景库程序或模板。

显然，要构筑这样一个构件库，需要极大的工作量。

&5.2 TreeOS 是一个无核的操作系统

传统的实时操作系统，沿用了大型操作系统的做法，都提供了一个操作系统内核，即实时内核。由内核来负责任务调度、内存管理等工作，以确保系统能够实时运行。

TreeOS 所提出的软件建模方法，摈弃了提供内核的做法，简化了内核的功能，并把内核的一些功能转移到构件中来，由构件之间协作完成。这种方法充分考虑了中低档单片机内存资源少、应用相对简单的特点，完全解决了操作系统挤占系统资源的问题，同时又可以满足系统的实时性要求。

实际上，TreeOS 在每个场景中采用了传统的前后台系统。前后台系统的优点是无需任务管理程序，因此也不存在系统程序占用单片机系统宝贵的内存和 ROM 空间等资源的问题。另一个优点是思路简明，

对初学者来说很容易掌握。有得必有失，这种系统的缺点是整个循环的执行时间不是一个常数，随着任务数的增加，任务等待执行的时间也随之增加。

将来，在 TreeOS 其它版本中，不排除采用实时内核技术，以适应更加复杂的应用。

&5.3 TreeOS 是一个实时多任务操作系统

实时系统是对任务运行时间有特定要求的系统。当发生外界时间或者有数据输入时，能够毫无遗漏地响应和接收并快速处理，然后在规定的时间内（截止时间）输出结果。实时系统有明确的和固定的时间约束，任务处理必须在约定的时间内完成。

单片机的应用绝大多数是实时系统，因此 TreeOS 必须能够满足系统的实时性要求。

首先，对于实时性要求高的任务，TreeOS 利用系统的中断系统即可得到即时的处理。

其次，对于普通任务，其实时能力与一次循环的总时间有关。任务越多，循环总时间就越长，实时能力就越差。考虑到对于中低档的单片机系统来说，任务数不多，也都比较简单，这种循环调度方式的实时能力完全可以满足应用要求，长期的实践也证明了这一点。事实上，由于取消了内核程序，系统无需花费很多时间去运行内核程序，客观上使系统的实时能力提高了，这也是一种补偿。

采用拆分技术，是一种多道系统的处理手法，可以提高 MCU 的利用率。另外，利用插入技术，对个别对于实时性要求较高的普通任

务，可以在任务顺序队列中插入两次或两次以上的运行机会，使任务的实时性要求获得满足。

&5.4 TreeOS 是一个真正适用于 MCS51 的操作系统

TreeOS 是一个构件化操作系统，设计的出发点是软件复用，这与传统的实时操作系统有所不同，但它们的目的都是相同的，都是为了简化软件开发过程，提高软件质量。TreeOS 在场景中采用传统的前后台系统，无需任务管理，几乎不占用单片机的资源（尤其是内存资源），因此即使 51 单片机也可以轻松地使用 TreeOS。

&5.5 “680 程序结构”

“任务网格化”形象地说明了 TreeOS 的软件体系结构。为了使初学者更好地掌握 TreeOS 的工作原理，我们还提出了“680 程序结构”，它是一种形象说法，如下图所示。

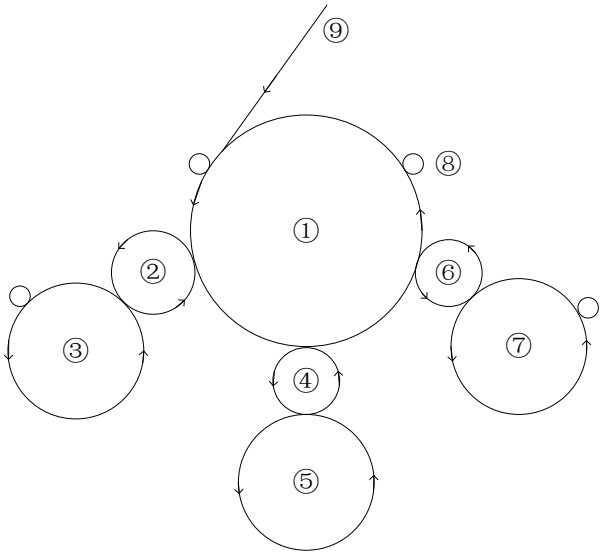


图 5-4 “680 程序结构”示意图

这是一种无限循环的嵌套程序结构。“6”代表主循环，“6”的头

部表示系统程序的起始点，初始化程序就放在这里，提醒循环需要初始化。主循环是一个无限循环；

“8”表示次级循环连在一起。其实可以有很多次级循环连在一起。次级循环也是一个无限循环，在一定条件下可以退出循环，返回上一级循环，而且规定只能返回上一级循环。

“0”是表示运行中断程序。每次中断发生，它只运行一次，并非无限循环。它可以出现在任一个循环中。

“680 程序结构”只是一种形象的说法，把这张图放在脑子里，有助于理解系统的软件架构。

&5.6 TreeOS 的设计原则

在开发 TreeOS 操作系统的时候，针对嵌入式系统的特点，我们制定了以下四项原则：

- 标准化：模块化、可配置、可剪裁、可移植；
- 绿色：节约每个字节，节省点滴电力；
- 友好：易学、易用；
- 开放：开源、吸收。

&5.7 TreeOS 的实践验证

TreeOS 操作系统从 0.1 版本到 1.0 正式版本，经过了 7 年时间。与其它嵌入式操作系统的开发模式不同，我们并没有在设计之初就开始发布产品，而是采取“使用、验证、改进、升级”同时进行的开发模式。截止到目前为止，TreeOS 已成功应用在工业及民用方面的二百

多个 MCU 项目上，有的产品已运行多年。结果表明，TreeOS 可以适用多种应用系统，系统的实时能力完全能够满足实际要求，产品长时间运行的稳定性、可靠性也得到了验证。

&5.8 TreeOS 的适用范围

TreeOS 目前已成功应用在 MCS51、STC、AVR、MSP430、STM8、STM32 等多种单片机上。由于在场景中采用了前后台系统，这种做法比较适合单片机使用。但实际上，TreeOS 并无具体针对某一类计算机系统（构件库可能不同）。因此 TreeOS 同样可适用于任何其它符合其设计要求的计算机系统，比如 ARM7 或 PC 机。例如，我们曾把 TreeOS 应用在安装了 DOS 操作系统的 PC 机上，使用 Borland C 2.0 IDE，为用户开发了一种称为“电子秤配料”的工业控制系统，目前还稳定地在生产线上运行。

TreeOS 的应用范围包括：工业控制、仪器仪表、汽车电子、民用电器、 医疗仪器、通讯等多个行业。

目前，随着物联网（包括无线传感网路、智能家居等）、工业 4.0、人工智能的蓬勃兴起，MCU 作为这些应用的主力军，其发展即将迎来一个爆发期。物联网可以看做是大量 MCU 构成的一个庞大而复杂的网络体系，互联互通，需要标准，迫切需要通用型实时操作系统。解决其操作系统问题，绕不过使用量占 80%以上的中低档 MCU！因此，TreeOS 必将在这些新兴领域一展身手。

&5.9 TreeOS 的升级展望

TreeOS 将会不断升级，主要沿两条线路，一是不断扩充构件库以及构建专用的构件库，这个工作量很大；二是对任务的管理可以考虑加上实时内核或微内核，以适应更复杂的嵌入式应用。

&5.10 TreeOS 是电子工程师的好助手

- 可套用格式化的软件架构；
- 提供大量的经过实践检验的程序库；
- 可方便在不同系统间移植；
- 大大缩短开发周期和测试周期；
- 减少 BUG，提高程序的稳定性、可靠性；
- 加速自身积累进程；
- 注重工程师体验，易学易用，完全克服了操作系统晦涩难懂的缺点；
- 为学习其它嵌入式操作系统打好基础。

&5.11 TreeOS 是初学者的好老师

- 编程思路和处理任务之间错综复杂的关系，是初学者遇到的最大挑战；
- TreeOS 提供了完整的编程思路，并使任务之间的关系简单化；
- TreeOS 提供大量的程序库，可即学即用，或者不学亦可用；
- 细节决定成败，优秀的开源程序是学习的好材料，可培养良好编程习惯；
- 大大加速自身积累进程；

- 注重工程师体验，易学易用，完全克服了操作系统晦涩难懂的缺点；
- 从零起步,快速入门。

&5.12 TreeOS 的优点总结

1、 采用无核设计，解决了操作系统挤占单片机资源的问题，提高了系统的利用效率；

2、 采用无核设计，去掉了那些不是一般工程师所能轻易掌握的内核技术，提高了操作系统程序的可读性和易用性，使广大的工程师队伍都可以方便地使用本发明技术；

3、 构件化设计，使开发者可以采用“拿来主义”，直接利用现有的标准构件构筑自己的系统，大大减轻了开发者的劳动强度，缩短了新产品的开发周期；

4、 构件化设计，通过使用那些经过大量实用验证的标准构件，可以提高产品的稳定性和可靠性，同时也缩短了新产品的磨合期；

5、 构件化设计，利用其可剪裁的特点，可以根据项目需要仅保留那些有用的构件，从而设计出符合需求的最紧凑、节约的应用系统；通过灵活配置机制，使标准构件非常适合在不同系统间的移植。这一点对单片机尤为重要。单片机种类繁多，各种开发环境五花八门，标准化构件可通过合理配置来适应各种系统；

6、 多级树形结构的场景体系结构，是一种可适合绝大多数应

用的系统运行机制。它把一个复杂的系统分解为一系列简单的构件，其设计思路清晰，易于理解，而且体现了构件化设计的特点；

7、 面向场景编程，以人机交互为主线，符合人类日常思维的方法和习惯，体现了注重工程师体验的人性化设计思想。它可使开发者提高设计效率，也降低了出错的可能性；

8、 对于单片机初学者来说是一个学习编程的好工具。初学者通过学习前人提供的例程（构件），可以快速掌握规范的编程技术，甚至是在只要会配置构件的情况下就可以编程，使他们可以快速成长为合格的单片机工程师。

第六章 TreeOS 与其它操作系统比较

本章把 TreeOS 与传统的一些嵌入式实时操作系统做了比较，并分析了它们的优缺点。

实际上，TreeOS 是一个以“软件复用”为出发点的无核、构件化操作系统，其工作原理和代码体量与带有实时内核的传统嵌入式实时操作系统差别很大，因此也难以做深入的比较。为了使读者更好地理解 TreeOS，本章对它们的特点和应用范围进行了一些比较。

&6.1 TreeOS 与大循环系统的比较

对于占有大量份额的中低档应用，由于现有的实时内核操作系统不太适用，因此目前一般还是采用传统的大循环(big loop)程序结构，其优点是直观，容易理解。大循环程序结构是一种笼统的说法，并无一定之规，每个成熟的工程师可能都有自己的一套方法和程序积累，因此无法形成统一的开发平台。

大循环系统的优点是无需任务管理程序，因此也不存在系统程序占用单片机系统宝贵的内存和 ROM 空间等资源的问题。另一个优点是思路简明。这种系统的缺点是整个循环的执行时间不是一个常数，随着任务数的增加，任务等待执行的时间也随之增加。

TreeOS 就是为结束大循环系统这种无序的开发状态而提出的。TreeOS 不仅提供了可通用的软件体系架构，而且还提供了丰富的构件

库，因此成为了一个通用的单片机开发平台。不过，TreeOS 的场景也采用了大循环系统的方法，因此也继承了其优缺点。

&6.2 TreeOS 与 μcos 的比较

可用于 MCU 的实时内核操作系统有很多，比较知名的有美国 Micrium 公司 μC/OS、德国 Keil 公司（现被 ARM 公司收购）RTX51、FreeRTOS 等。

这些传统的嵌入式实时内核操作系统沿用了传统大型操作系统的思路，即提供了一个操作系统内核，由内核来负责任务调度、内存管理等工作，以确保系统实时、多任务运行。它们应用在中低档的单片机上，有以下几个方面的缺点：

①本来单片机的内存容量、程序空间等资源就非常有限，而操作系统内核需要占用相当一部分的内存和程序空间，这就制约了系统的性能，也限制了大部分单片机无法使用这些操作系统；

②由于运行这些内核程序需要占用处理器时间，因此降低了系统的实时能力，也不利于降低功耗；

③操作系统内核程序技术非常复杂，学习和使用起来比较困难，若非必要，一般不会去选用这些操作系统；

④这些单片机操作系统只是提供了传统的内核功能，而对于需要大量设计工作的驱动程序、常用程序库等中间件程序，还是需要设计人员自己设计和不断积累，因此它们并非是一个完整的操作系统。

以下是 TreeOS 与 μcos 的对照表。

操作系统名称	μcos	TreeOS
多任务	是	是
实时性	好	好
内核	**有**	**无**
提供编程架构	**无**	**有**
提供设备驱动程序	**无**	**全面**
提供功能程序模块	**无**	**全面**
内存管理	有	无
任务优先管理	有	有
占用内存	几百字节以上	~0
占用程序空间	4K 以上	~0
适合 51 单片机	不太适合	适合
适用 ARM	是	是
需用汇编	是	否
开放源代码	是	是
提供配置工具	**无**	**有**
学习难度	大	很小

（Keil 公司针对 51 单片机还推出 RTX51 Tiny 版本，同样是不开源，而且最多只能有 16 个任务，没有任务优先级。）

μcos-II、RTX51、FreeRTOS 比较适合于中大型项目和关键性应用。对于中低档的应用，这些操作系统便落入“英雄无用武之地”。

TreeOS 操作系统因其“无核”的设计思想解决了占用系统资源（内存、程序空间、MCU 运行时间）的问题，使 MCS51 等中低档单片机也能用上操作系统。实际上绝大多数的单片机应用都是中低档的应用。

不仅如此，TreeOS 最大的好处就是提供了可复用的软件体系架构以及可复用的构件库，这大大简化了软件开发过程，为开发人员节省了大量的工作量，也缩短了开发周期。

第七章 TreeOS ComLib 软件构件库

TreeOS ComLib 软件构件库，是 TreeOS 构件化操作系统的主要组成部分。这也是 TreeOS 区别于其它嵌入式操作系统的重点之一。

&7.1 概述

TreeOS ComLib 软件构件库主要包含以下四方面的内容：

1） 各种 MCU 的初始化、配置程序；

2） 各种周边芯片或模块设备的驱动程序；

3） 常用中间件、边缘计算库程序；

4） 可通用的场景库程序或模板。

也就是说，TreeOS ComLib 内容涵盖了硬件抽象层、驱动程序、中间件以及用户层。

TreeOS ComLib 主要包括：

- TreeOS_MCU.c（MCU 配置模板、各种 MCU 的常用配置）；
- 各种常用设备驱动：例如 TreeOS_KeyBoard.c、TreeOS_I2C.c、TreeOS_RS232.c 、 TreeOS_RS485.c 、 TreeOS_HC595.c 、TreeOS_HD61202.c、TreeOS_PCF8563.c 等等；
- 常用程序库：如边缘运算、数字字符转换、通讯协议栈、UI 库函数等；
- 用户界面显示：如显示图形、显示多种字体文字、输入文本框、时间显示、简单作图等；

- 数据存取：主要是 E2PROM、FLASH、SD 卡等存储器的数据读写模板；
- 常用的场景库及模板，如 TreeOS_main、TreeOS_scn_menu、TreeOS_scn_input_asc 等；
- 用户层程序模板。

显然，要构筑这样一个构件库，工作量及其庞大，长期的实践积累非常关键。

而对于使用者来说，需要掌握的内容却并不多。使用者花费少量的时间，就可以轻松掌握一些常用软件构件。以后若有一些其它需求，再随时补充学习相应的内容。

这些由专家编写的、经过长期实践检验的软件组件，经过配置就可以直接使用于不同的平台，可以极大地提高软件开发效率，同时也是初学者学习规范编程的好例程。

&7.2 TreeOS ComLib 的特点

- 符合 TreeOS 软件架构规范；
- 可方便移植到不同的单片机平台；
- 专家编写，经过实践检验；
- 不断升级、扩展。

&7.3 如何使用 TreeOS ComLib

1、 复制：根据项目需要，把 C 文件和 H 文件复制到项目文件夹中，

并把 C 文件调入工程；

2、 配置：一般配置都在 H 文件中进行，包括条件编译选择、属性、IO 口定义、特殊语句等。需要配置之处，带有“//M/”标志；

3、 剪裁：有些子程序用不着，必须把它注释掉，以免占用内存和 ROM 空间。一般通过 H 文件中的“条件编译选择”进行剪裁。需要剪裁之处，带有“//M/”标志；

4、 引用：包括调用子程序和引用“特殊语句”。最常见的引用“特殊语句”是放在 T0 中断中进行倒计时，例如 TreeOS_keyboard 中的 iSCAN_KEY_TIME；

5、 修改与补充：有些子程序需要根据实际情况进行编写，例如 TreeOS_keyboard 中的 ReadPress(void)。需要修改与补充程序之处，带有“//M/”标志；

6、 模板：有些文件只是一个模板，一般只提供架构，不提供具体内容（有时提供例程），可以按照该架构来编写程序。同样带有“//M/”提示标志。

每个库文件中都有使用说明和提示，请注意查看。

事实上，AlphaMCU 自动写代码机器人就是 TreeOS ComLib 的配置工具。有了 AlphaMCU，使用 TreeOS ComLib 非常方便快捷。

&7.4 TreeOS ComLib A2 构件库介绍

在 Kepler11 开发板中，我们提供了 TreeOS ComLib A2 构件库。该构件库主要围绕 Kepler11 开发板设计，它包括了 36 个常用的构件文

件，参见下图。

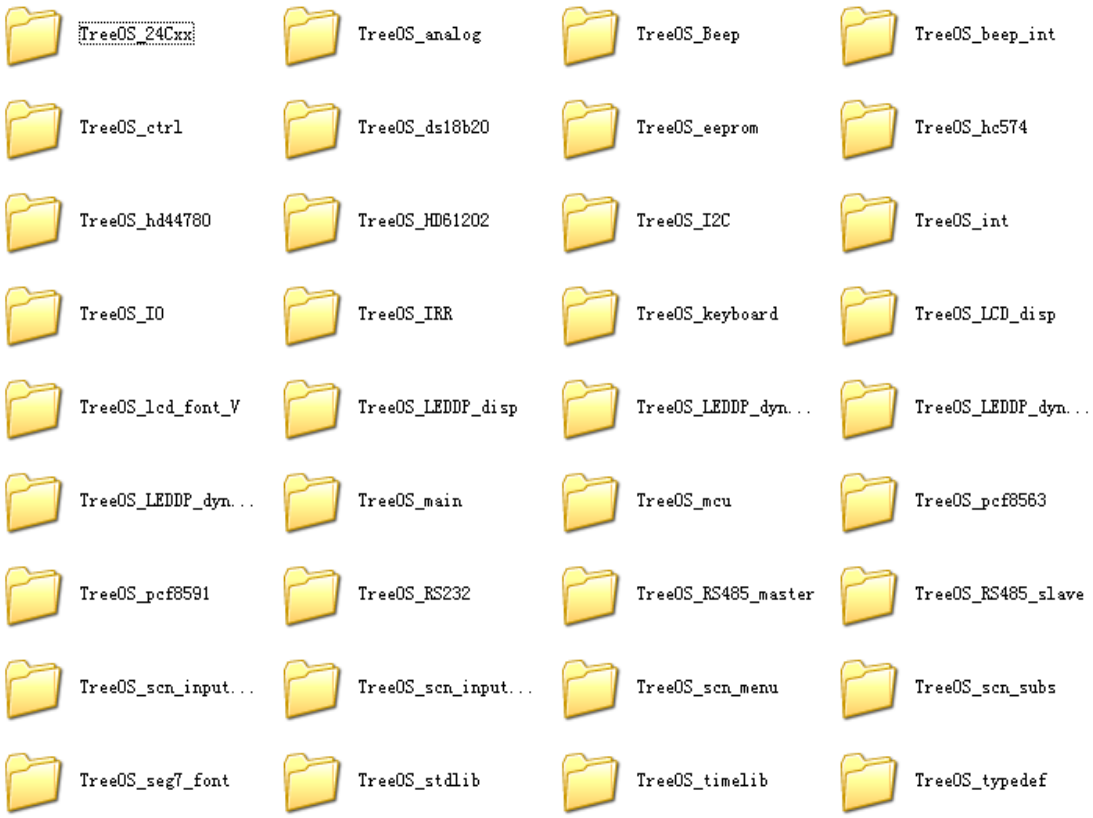


图 7-1 TreeOS ComLib A2 构件库内容

包括：

1、 单片机文件

TreeOS_MCU ：51 单片机初始化配置；

2、 设备驱动

TreeOS_24Cxx：at24c32/64 EEPROM 芯片驱动程序；

TreeOS_beep：蜂鸣器驱动程序（查询方式）；

TreeOS_beep_int：蜂鸣器驱动程序（中断方式）；

TreeOS_ds18b20：ds18b20 温度传感器驱动程序；

TreeOS_hc574：IO 口扩展芯片 74hc574 驱动程序；

TreeOS_hd44780：字符液晶控制芯片 hd44780 驱动程序；

TreeOS_HD61202：点阵液晶控制芯片 hd61202 驱动程序；

TreeOS_I2C：I2C 总线驱动程序；

TreeOS_IO：IO 口开关量输出、开关量读取程序（模板）；

TreeOS_IRR：红外遥控器解码程序（NEC 码）；

TreeOS_keyboard：键盘处理程序；

TreeOS_LEDDP_dynamic：数码管动态驱动程序；

TreeOS_LEDDP_dynamic_B：数码管动态闪烁驱动程序（中断方式）；

TreeOS_LEDDP_dynamic_B1：数码管动态闪烁驱动程序（查询方式）；

TreeOS_pcf8563：日历时钟芯片 pcf8563 驱动程序；

TreeOS_pcf8591： 8 位 AD/DA 芯片 pcf8591 驱动程序；

TreeOS_RS232：RS232 串口驱动程序；

TreeOS_RS485_master：RS485 串口驱动程序（主机模式）；

TreeOS_RS485_slave：RS485 串口驱动程序（从机模式）；

3、 常用程序库

TreeOS_timelib：时间运算库程序；

TreeOS_stdlib：标准库程序（数据、字符转换处理等）

TreeOS_analog：AD 数据采集及处理库程序；

4、 用户界面显示

TreeOS_LCD_disp：点阵液晶显示常用库程序；

TreeOS_LEDDP_disp：数码管显示常用库程序；

5、 数据存取

TreeOS_eeprom：参数或运行数据存取程序（模板）；

6、 常用的场景库及模板

TreeOS_main：主场景程序及全局任务函数 scan_in_while()等（模板）；

TreeOS_scn_menu：菜单场景库程序（模板）；

TreeOS_scn_subs：子场景库程序（模板）；

TreeOS_scn_input_asc_lcd：LCD 显示输入 ASC 字符场景库程序；

TreeOS_scn_input_asc_leddp：数码管显示输入 ASC 字符场景库程序；

7、 用户程序模板

TreeOS_ctrl：运行控制或数据处理（模板，无内容）；

TreeOS_lcd_font：点阵液晶字体（模板）；

TreeOS_seg7_font：数码管 7 段代码字库；

第八章 自动写代码工具 AlphaMCU 简介

最新使用方法请以官网 www.treeos.com 公布为准。

AlphaMCU 可以根据电路原理图自动生成“定制化操作系统”的源代码，包括 MCU 片上资源代码、周边器件（芯片或模块）驱动程序、中间件、通讯协议、以及边缘计算代码等。

AlphaMCU 是 ComLib 软件构件库的管理工具，是 TreeOS 物联网开发平台的重要组成部分。

AlphaMCU 的使用步骤如下：

1、 按照《AlphaMCU 原理图设计规则》设计硬件原理图，并生成电路图网络文件（《AlphaMCU 原理图设计规则》参见附录 C）；

2、 进入 TreeOS 官网：www.treeos.com；

3、 按照要求登录 AlphaMCU 系统。如果是新用户，需要注册才能登录；

4、 界面如下图：

5、 在本地找到电路图网络文件路径，导入文件；

6、 点击“输出代码”；

7、 进入你的注册邮箱，马上就可以收到自动生成的代码文件包，它们就是你要的“定制化操作系统”；

8、 把代码文件导入 IDE，可以直接编译通过。在此基础上花点时间添加你的用户程序，你的产品软件很轻松就完成了。

第九章 Kepler11 单片机开发板简介

Kepler-11 是开普勒空间望远镜在距地球 2000 光年处发现的一个极不同寻常的小太阳系。最新的结果是发现了“地球的表哥”--与地球相似度达 83%的 Kepler-452b。本开发板命名为 Kepler11 是为了向不断探索太空的科学家们致敬！

Kepler11 开发板的创新，不仅在于它搭载了专利产品 TreeOS 构件化实时操作系统，还在于它采用了业内首创的子母板结构的可堆叠设计（电子积木），不仅有各种外围功能模块可供选择，MCU 也可轻松替换，用户可以像搭积木一样快速搭建自己所需的应用系统！

一、Kepler11 开发板特点

1、 子母板结构：母板集成了一些最常用的功能，包括数码管、LCD、键盘、红外遥控器、串口、AD 输入、蜂鸣器、时钟、E2PROM、继电器、电源等；子板包含各种 MCU 板（如 51、STC、AVR、MSP430、STM8、STM32 等）、各种外围功能板，如 PWM、DA、4~20mA、GPS、短信、GPRS、RS485、WiFi、蓝牙、无线模块、USB、触摸屏、TFT LCD、MP3、语音、FLASH、SD 卡、RFID、各种传感器等等；

2、 可堆叠设计：上述这些模块可以像搭积木一样随意组合，满足用户的各种需求；

3、 模块化设计：预装 TreeOS ComLib 软件构件库，不仅软件

可配置可剪裁，硬件也做到了可配置，符合嵌入式系统的特点，效率极高；

4、 可扩展：用户需要增加某种功能，可购买相应的模块，与系统无缝连接，即装即用；

5、 +5V 与+3.3V 电源并存：方便设计不同电压的系统；

6、 价格低廉。

以前，用户为了学习不同的单片机，需要购买不同的开发板，如果某些功能不全，还得购买更多的开发板，成本很高，而且有很多功能重复，造成浪费。不仅如此，不同的开发板之间的软件无法直接移植，用户需要花很多时间来消化，效率很低。而若使用电子积木式的 Kepler11 开发板，可以按需配置，不仅节省成本，而且学习与开发效率也大为提高。

当然，Kepler11 开发板最有价值之处还是其预装了 TreeOS ComLib 软件库。

二、Kepler11 开发板与其它学习板的比较

	软件架构	功能组件	编程经验
Kepler11	√	√	√
其它学习板		√	

纵观现有的单片机学习板，存在两方面问题：

1、过分强调硬件功能模块，软件粗编滥造。通常仅提供一些演示用途的硬件驱动程序，而对于编程思路、编程技巧、以及培养良好

的编程习惯等问题却鲜有提及；

2、MCU 固定，功能固定，难于灵活扩展。

比较内容	其它 51 学习板	Kepler11 开发板
软件来源		实际项目，专家编写
软件架构	无	完整的可通用架构
软件可读性	一般	注释充分、条理清晰
编程规范	无	有一套完整编程规范
软件模块化	没有考虑	极好
软件可移植性	没有考虑	可迅速移植到其它单片机
软件可用性	仅限于学习	可直接在实际项目中使用
软件可靠性	没有考虑	好
软件健壮性	没有考虑	好
软件后续维护	没有考虑	非常方便维护
适用操作系统		TreeOS，带 ComLib 软件构件库
可升级	否	是
软件整体评价	差	非常好
硬件配置	功能固定，可扩展性差	电子积木式设计，各种 MCU、各种功能模块齐全
电路设计	仅供学习	贴近实际产品

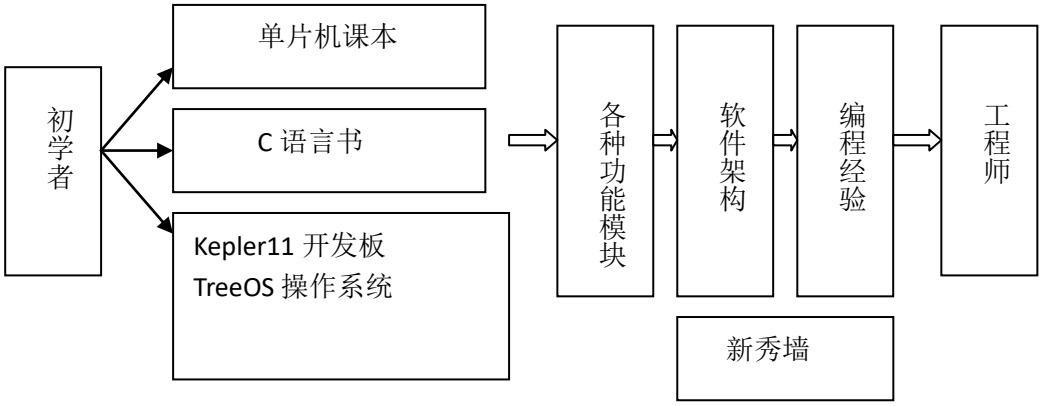
需要跳线	需要很多跳线	跳线少，使用方便
性价比	一般	极高

三、Kepler11 助您跨过“新秀墙”

软件架构问题可以说是童鞋们学习单片机道路上要遇到的第一道坎，借用 NBA 的说法就是“新秀墙”。开发单片机项目，软件架构设计至关重要。设计合理，编程就比较顺当，开发也容易成功。反之编程就会很别扭，来来回回修改，既耗费大量时间，产品的质量还有问题。很多童鞋尽管 C 语言学得好，单片机及各种周边设备也玩得很熟，但是编出来的软件总是漏洞百出，无法使用，主要原因是没有处理好软件架构问题。从一些到我们公司实习或工作的大学生身上我们发现，不懂得合理设计软件架构是个普遍存在的问题。

尽管单片机资料汗牛充栋，但是有关软件架构方面的资料却很少，这主要是因为软件架构隐含在程序中，往往要结合程序才能说得清楚。每个成熟的工程师都有自己一套经过多年摸索出来的做法，他们是不会轻易拿出来共享的。

现在好了，TreeOS 提供了一套完整的具有普适性的软件架构，童鞋们可以借助 TreeOS 轻松地越过这堵“新秀墙”。

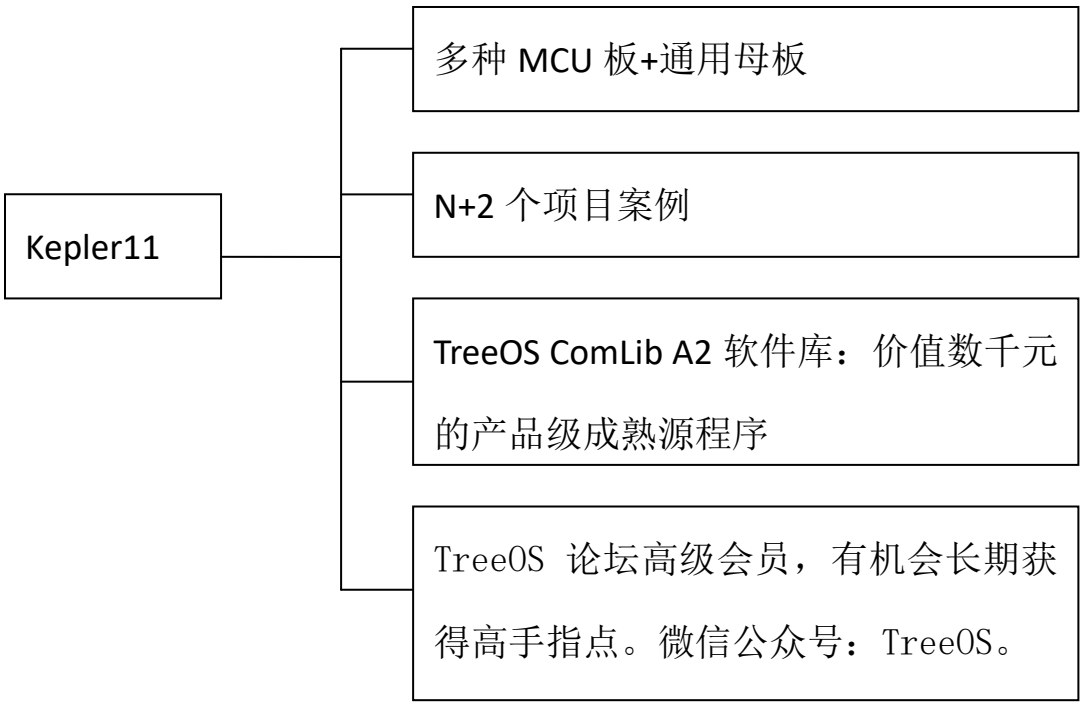


四、 案例学习法

学习单片机有没有好的方法？有！这就是“案例学习法”。其做法是完全解读一个实际产品案例，把每一行程序都搞懂，从中不仅可以学到各种知识，还可以学到作者的编程思路以及各种技巧。有兴趣的童鞋可以参阅这篇博客文章《我是如何学习 C 语言的》，相信您能从中受益。

但是，实际产品的源程序属于商业机密，一般的初学者根本搞不到。我们推出的 Kepler11 开发板，就为童鞋们解决了这个问题。首先，它把每个功能模块的实验例程都做成一个个小“项目”的形式，从中可以学习各种功能模块如何配合工作；其次它集成了《汽车智能电子表》和《智能家居控制中心》两个比较典型的项目案例，这些案例都是来自于我们做过的实际产品（适当简化），比较适合初学者学习。其中最关键的是我们提供了近万行的成熟软件库。

当童鞋们学完这总共 20 多个案例，相信就可以自己动手做项目了。另外，通过 AlphaMCU 组合可以轻松派生出各种案例！



五、学习单片机，就从 TreeOS 开始

对于初学单片机的人，很多时候，通过网络便可以轻松获取到所需要的学习资料。但随着学习的深入或是自己做项目，就会发现，网络提供的有实用价值的东西是十分有限的。尽管网络上单片机资料铺天盖地，但是大部分的资料都只是初级的学习资料，而且也只是仅仅作功能性的演示。学习板软件也是抄来抄去、大同小异。因此，单片机爱好者们为了突破学习瓶颈，不得不开始了艰难的摸索历程。

TreeOS 操作系统的出现，改变了这种状况。

很多童鞋会问，操作系统那么复杂，初学者能学得懂吗？我们可以负责任地说：没问题。详情可登陆官方网站 www.treeos.com 做深入了解。这里我们着重介绍 TreeOS 两方面功能，这些功能将给童鞋们学习单片机提供莫大的帮助。

第一：它提供了一套完整的、具有普适性的软件架构。这种架构是以大循环（一般的单片机编程都是用大循环结构）为基础的，所以非常容易理解。

第二：它提供了一套内容丰富的成熟软件库。软件库包含了各种设备的驱动程序和单片机配置，以及常用的标准函数库等，这些由专家编写的、经过实践检验的软件源程序，不正是童鞋们学习编程的好材料嘛！事实上，即使您一时看不懂这些程序，您只要会调用即可！既然作为一种正规的软件产品，我们所提供的软件是非常严谨的，童鞋们从中也可以体会到诸如软件规范、软件移植性、健壮性、可靠性等多方面的知识和技巧。

使用 TreeOS 的好处：一是使编程变得简单多了，二是大大减少了工作量。例如，在使用熟练的情况下，《汽车智能电子表》案例大概只需要 1 天、《智能家居控制中心》案例大概只需要 3 天就能完成！购买多功能 Kepler11 就相当于找到了一位长期的好老师！

学习单片机或者独立开发项目的过程中经常会遇到各种难题，这时就非常需要别人的帮助，很多童鞋正是由于经常不能独自解决问题而最终选择了放弃单片机学习，非常可惜。

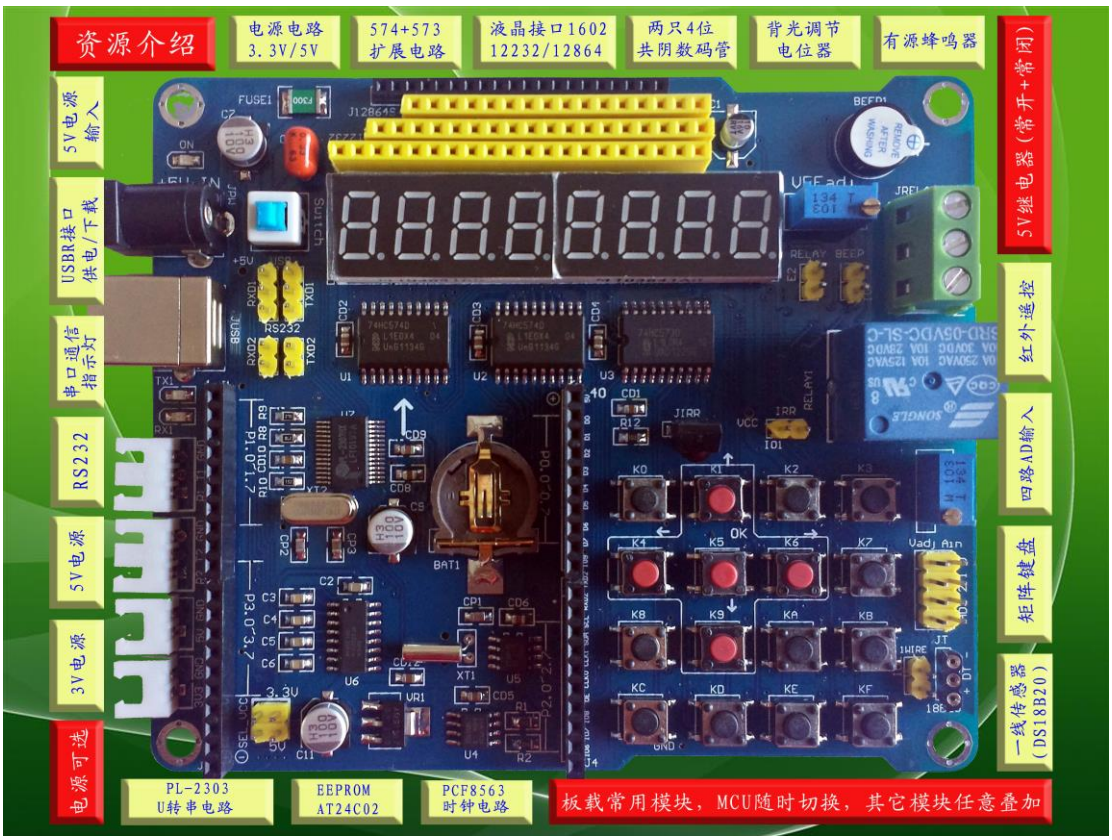
购买 Kepler11 的用户同时也获得一份保证：成为 TreeOS 论坛的高级会员，将有机会长期获得高手的指点；

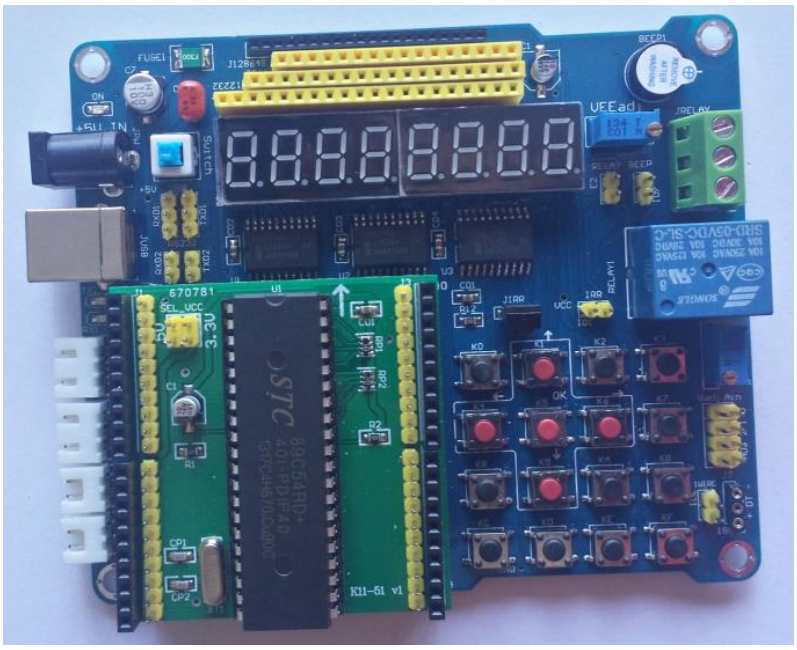
TreeOS 也是一位好老师，它教您如何设计软件架构、如何编写规范的高质量的程序；

使用 TreeOS 还容易在网上获得别人的帮助，因为大家都有“共

同语言”了。

六、 Kepler11 开发板的硬件配置





母板提供的功能包括：

--2 个扩展输出芯片 74HC574

--1 个 74HC573 芯片

--4X4 矩阵键盘

--8 位动态数码管

--1602 字符型液晶电路

--12232 字符型液晶电路

--12864 点阵型液晶电路（小屏）

--12864 点阵型液晶电路（大屏）

--I2C EEPROM 芯片 AT24C64

--I2C 时钟芯片 PCF8563

--继电器

--纽扣电池及插座

--一体化红外线接收管、遥控器

--DS18B20 温度传感器座

--有源蜂鸣器

--RS232 芯片 SP3232

--USB-232 转换芯片 PL2303

--可直接通过 USB 接口供电和下载程序

--单片机 40 脚外扩接口

--5.0V、3.3V 电源输出

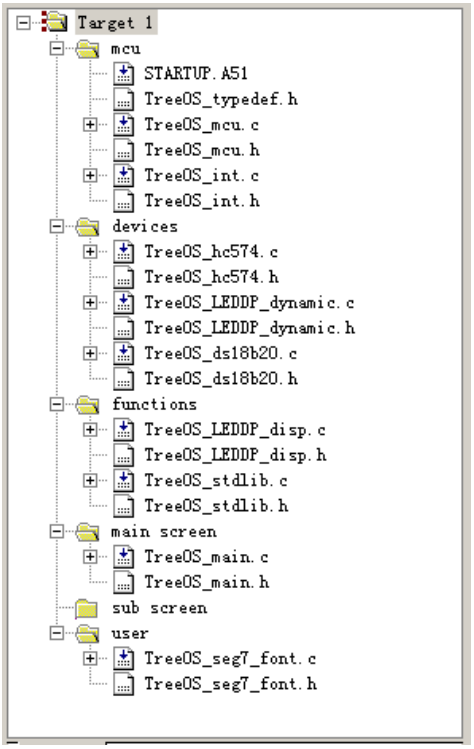
--可选用 STC、AVR、STM32F10x 等各种常用单片机

--可选用各种外围功能模块

七、Kepler11 开发板的软件配置

第一部分：小项目（基础模块实验），共 19 个。适用多种单片机。通过 AlphaMCU 可派生出非常多各种功能的项目！

Kepler11 开发板实验项目虽然不多，但信息量却一点都不少。如果按照软件的行数来计算的话，可能比其它学习板要多得多！例如：对于 DS18B20 温度传感器的实验例程，整个工程文件如下：



共包含了 9 个 C 文件和 10 个 H 文件。

第二部分：实战案例

Kepler11 开发板还提供了 2 个完整的项目案例，这些案例把各个功能模块有机地结合起来，用户可以完整地学习到整个开发过程。这种完整的项目案例，其它学习板大部分都没有，也不具备实验条件，因为它们一般都是采用分立模块设计。

	程序总数（行）	软件库提供(行)	需编写程序（行）	占比	开发周期	生成 HEX 文件大小（字节）
智能家居控制中心	4970	4370	600	12%	约 3 天	8K
汽车智能电子表	4100	3970	130	3%	约 1 天	4K

我们看到，使用 TreeOS 操作系统，需要工程师编写的程序量已非常少！更重要的是，程序的可靠性有保证，省去了来回调试的麻烦，节省的时间还要多的多！

第十章 应用实例

本章详细介绍一个开发项目实例程序，这个实例的名称为《汽车智能电子表》，它已被收录在 Kepler11 开发板中。

具体的功能、开发过程、结果等在第三章《初识 TreeOS》已有描述，此处不再赘述。本章主要是讲解程序。

限于篇幅，本章仅列出几个软件模块供参考，更多、更新软件可以访问官网 www.treeos.com，通过 AlphaMCU 下载。

&9.1 ： 数据类型定义：TreeOS_typedef.h

TreeOS 数据类型定义头文件，主要是为了简化书写，而且用位数来统一数据类型，以免在不同软件平台移植发生混淆。另外还定义了一些常用的数据量。

```
/*****************************************************************************/
/*                                                                           */
/*                      TreeOS_typedef.h 程序清单                             */
/*                                                                           */
/*****************************************************************************/
typedef signed long  i32;
typedef signed int   i16;    //M/
//typedef signed short i16;
typedef signed char  i8;

typedef code long  i32c;  //code data,read only
typedef code int   i16c;  //M/code data,read only
//typedef code  short i16c;  //code data,read only
typedef code char  i8c;   //code data,read only

typedef unsigned long  ui32;
typedef unsigned int   ui16;   //M/
//typedef unsigned short ui16;
```

```
typedef unsigned char  ui8;

typedef code unsigned long  ui32c;  //code data,read only
typedef code unsigned int   ui16c;  //M/code data,read only
//typedef code unsigned short ui16c;  //code data,read only
typedef code unsigned char  ui8c;   //code data,read onl

/********开关量、状态********/

typedef enum BOOL{FALSE = 0, TRUE = !FALSE} ;
typedef enum FLAG{RESET = 0, SET = !RESET} ;
typedef enum ERROR{ERR= 0, SUCCESS = !ERR} ;
typedef enum SWITCH{OFF = 0, ON = !OFF} ;

/********实时时钟：压缩 BCD 码*****************************************************/

typedef struct {
        ui8 sec;      //须屏蔽 bit7
        ui8 min;      //0~59
        ui8 hour;     //0~23
        ui8 day;      //1~31
        ui8 week;     //0~6
        ui8 mon;      //须屏蔽 bit7
        ui8 year;     //0~99
}TREEOS_CLOCK;

typedef struct {   //时分
        ui8 hour;     //0~23
        ui8 min;      //0~59
}TREEOS_HHMM;

#endif

/***** (C) COPYRIGHT 2012  Beijing GuangLun Electronic Technology Co., Ltd.****/
```

&9.2 ：单片机配置：TreeOS_mcu

包括 TreeOS_mcu.c 和 TreeOS_mcu.h 文件。

本模块为 MCU 配置文件 ，不同的 MCU 完全不同。TreeOS ComLib 数据库将提供大量常用的 MCU 配置文件。

本软件使用的 MCU 为 89C52，选用外部晶振 11.0592MHz。

常用功能包括：

- IO 口初始化
- 中断初始化
- 看门狗初始化
- 系统节拍,即 T0 时钟设置
- 延时函数 Delayms()，等等。

使用方法：

根据项目需求在.h 文件中进行相应的配置，同时.c 文件中部分子程序也需要修改。

在系统初始化阶段调用 MCU_init()函数。

```
/*****************************************************************************/
/*                                                                           */
/*                          TreeOS_mcu.c 程序清单                            */
/*                                                                           */
/*****************************************************************************/
/********includes*************************************************************/
#include<reg52.h> //M/

#include "TreeOS_mcu.h"

/********全局变量*************************************************************/

ui16 systick_ms=0;  //系统节拍计数器，一般单位为 1~10ms

ui16 second_stamp=0; //秒计时戳
ui8 second_flag=0;  //秒计时标志

/*****************************************TreeOS Real-Time Operating System****
Function Name:
Description  : 上电后，IO 口马上先进行初始化，很有必要！
               包括：没有用到的 GPIO 缺省配置
                     各种周边设备的 GPIO 口初始化
```

```
Input        :
Output       :
Return       :
*******************************************************************************/
void port_init(void)    //M/
{
 //先把所有 GPIO 配置为：为 51 单片机的准双向口
 //重要的设备需要立即初始化：例如 LCD 先关闭使能、485 先设为接收状态、继电器先关闭等
 P0 = 0xff;
 P1 = 0xef;      //P1.4=0 (LCD.E)
 P2 = 0xff;
 P3 = 0xff;

 //  各种周边设备的 GPIO 口初始化 ：上电时关键任务放在这里
 //CLR_TXEN;  //上电时就把 485 设为接收状态，防止干扰总线上其它机器的工作

}

/*****************************************TreeOS Real-Time Operating System****
Function Name:
Description  : 上电后，芯片内部看门狗初始化
Input        :
Output       : WDTCR
Return       :
*******************************************************************************/
void watchdog_init(void) //M/
{
 WDR(); //this prevents a timout on enabling
}

/*****************************************TreeOS Real-Time Operating System****
Function Name:
Description  : 芯片内部看门狗喂狗子程序
Input        :
Output       :
Return       :
*******************************************************************************/
void WDR(void)          //M/
{
 // _nop_();

  //AT89S52 WATCHDOG 喂狗子程序
  //在 8191 个机器周期内必须喂狗,否则复位
  //sfr WDTRST  = 0xA6;   //WATCHDOG SFR OF AT89S52
```

```
  //WDTRST=0x1e;   //喂狗
  //WDTRST=0xe1;
}

/*****************************************TreeOS Real-Time Operating System****
Function Name:
Description  : 定时器 T0 初始化。
               T0 作为系统节拍(system tick)，一般定时为 1～10ms；
Input        :
Output       : T0 相关寄存器
Return       :
*******************************************************************************/
void timer0_init(void)     //M/
{
  //定时器 T0/T1 模式设定
  //sfr TMOD ： GATE1 C/T1  M1_1  M1_0  GATE0 C/T0  M0_1  M0_0
  TMOD|=(1<<0); //M0_1=0,  M0_0=1 模式 1

  //T0 作为定时器，为系统的滴答时钟
  //(0xffff+1-X)*12/Fosc=dt ==> X=0xffff+1-Fosc*dt/12
  //TH0=0xee;  //Fsoc=18.432MHz,3ms 中断
  //TL0=0x00;
  //TH0=0xfe;  //Fsoc=3.686MHz,1.5ms 中断
  //TL0=0x33;
  TH0=0xfc;  //Fsoc=11.0592MHz,1ms 中断
  TL0=0x66;
  TR0=1;     //START T0
  //ET0=1;     //ENABLE T0 INTERUPT
}

/*****************************************TreeOS Real-Time Operating System****
Function Name:
Description  : 定时器 T1 初始化。

Input        :
Output       : T1 相关寄存器
Return       :
*******************************************************************************/
#ifdef ENABLE_T1
void timer1_init(void)      //M/
{
  /*  //*********作为波特率发生器************
  //定时器 T0/T1 模式设定
  //sfr TMOD ： GATE1 C/T1  M1_1  M1_0  GATE0 C/T0  M0_1  M0_0
```

```
  TMOD|=(1<<5);  //M1_1=1, M1_0=0 . T1 MODE 2 FOR BAUDRATE
  TH1=TH1_VALUE;
  TL1=TL1_VALUE;
  TR1=1;     //START T1     */

  /*  //*********作为定时器************
  //定时器 T0/T1 模式设定
  //sfr TMOD : GATE1 C/T1  M1_1  M1_0  GATE0 C/T0  M0_1  M0_0
  TMOD|=(1<<4);  //M1_1=0, M1_0=1 . T1 MODE 1: 16 位计数器
  TR1=1;     //START T1
  */
}
#endif
/*****************************************TreeOS Real-Time Operating System****
Function Name:
Description  : 定时器 T2 初始化。

Input        :
Output       : T2 相关寄存器
Return       :
*******************************************************************************/
#ifdef ENABLE_T2
void timer2_init(void)    //M/
{
  //定时器 T2 模式设定
  //T2CON : TF2  EXF2  RCLK  TCLK  EXEN2  TR2  C/T2n  CP/RLn
  //T2MOD : X    X     X     X     X      X    T2OE   DCEN

  /* //********捕获方式************
  T2CON|=(1<<0);   //CP/RLn =1
  EXEN2=1;            //T2EX 管脚发生 1 至 0 的跳变时引起捕获，并置位 EXF2
  TR2=1;     //START T2    */

  /* //********T2 作为串口波特率发生器*************
  RCAP2L=0xfa;  //Fsoc=11.0592MHz, BaudRate=9600
  RCAP2H=0xfa;
  TR2=1;     //START T2    */

}
#endif
/*****************************************TreeOS Real-Time Operating System****
Function Name:
Description  : 串口初始化。
```

```
Input        :
Output       : T1 相关寄存器
Return       :
*******************************************************************************/
#ifdef ENABLE_UART0
void uart0_init(void)        //M/
{
  //PCON=0;          //波特率倍速 X1: SMOD=0
  PCON=0x80;      //波特率倍速 X2: SMOD=1
  SCON=0x50;  //mode 1,REN=1:8 bits data ;对应 N.8.1
  //SCON=0xd8; //11011000B  ,serial port MODE 3  REN=1 8+1BIT DATA,TB8=1 用来模拟停止位
  //                                    ;对应于 E,7,2. D7 用做效验位
  RI=0;  //清接收中断
  ES=1;   //ENABLE SERIAL PORT INTERUPT    使能串口接收中断
}
#endif
/*****************************************TreeOS Real-Time Operating System****
Function Name:
Description  : 上电后，对系统中用到的中断进行配置

Input        :
Output       :
Return       :
*******************************************************************************/
void int_init(void)       //M/
{
 //外部中断 INT0 设置
  //IT0=1;      //INT0 EDGE TRIGGER，=1 下降沿触发方式；=0 低电平触发方式
  //IE0=0;      //开放之前,清零中断标志,以防触发一次中断
  //EX0=1;      //ENABLE INT0 INTERUPT
//外部中断 INT1 设置
  //IT1=1;      //INT0 EDGE TRIGGER，=1 下降沿触发方式；=0 低电平触发方式
  //IE1=0;      //开放之前,清零中断标志,以防触发一次中断
  //EX1=1;      //ENABLE INT0 INTERUPT

  ET0=1;     //ENABLE T0 INTERUPT 使能 T0 溢出中断
  //ET1=1;      //ENABLE T1 INTERUPT 使能 T1 溢出中断
  //ET2=1;      //ENABLE T2 INTERUPT 使能 T2 溢出中断

  //ES=1;   //ENABLE SERIAL PORT INTERUPT 使能串口接收中断

  //PT0=1;      //T0 中断优先
  //PT1=1;      //T1 中断优先
  //PT2=1;      //T2 中断优先
```

```
  //PX0=1;     //INT0 中断优先
  //PX1=1;     //INT1 中断优先

  //PS=1;      //串口中断优先
}
/*****************************************TreeOS Real-Time Operating System****
Function Name:
Description  : 上电后，单片机设备的初始化
               开系统中断
Input        :
Output       : MCU 相关寄存器等
Return       :
*******************************************************************************/
void MCU_init(void)         //M/
{

  CLR_SYS_INT();      //关系统总中断

  port_init();       //GPIO 初始化
  watchdog_init(); //看门狗初始化
  timer0_init();   //T0 初始化
#ifdef ENABLE_T1
  timer1_init();
#endif
#ifdef ENABLE_T2
  timer2_init();
#endif
#ifdef ENABLE_UART0
  uart0_init();
#endif

  int_init();     //中断设置初始化

  SET_SYS_INT();     //开系统中断

}

/*****************************************TreeOS Real-Time Operating System****
Function Name:
Description  : 微秒级延时函数：用循环语句延时 n us（近似值）,与晶振频率有关
Input        :
Output       :
Return       :
```

```
//用软件实验检测 T1 计数值（时钟周期）：（1 计数=12/11.0592=1.085us,fosc=11.0592Mhz)
  //delayus(0);     //17（显示的实验结果: T1 的计数值）  ~=18.4us @11.0592M
  //delayus(1);     //26          ~=28.2us @11.0592M
  //delayus(2);     //35
  //delayus(3);     //44
  //结果表明：增加 1，就增加 9 个计数;调用函数及返回的操作，占用 17 个计数

  //delayus(10);  //107 =10*9 +17       ~=116us @11.0592M
  //delayus(100);  //917 =100*9 +17     ~=995us @11.0592M
  //delayus(500);  //4518     ~=4902us @11.0592M
  //delayus(1000); //9020     ~=9787us @11.0592M
  //delayus(5000); //45036    ~=48864us @11.0592M

*******************************************************************************/
#ifdef USE_DELAYUS
void delayus(ui16 n)    //M/
{
  while(n--);
}
#endif
/*****************************************TreeOS Real-Time Operating System****
Function Name:
Description  : 毫秒级延时函数：用循环语句延时 n ms（近似值）,与晶振频率有关
Input        :
Output       :
Return       :
*******************************************************************************/
#ifdef USE_DELAYMS
void delayms(ui16 n)        //M/
{
 ui16 i;

  for(i=0;i<n;i++)
     {
       delayus(100);  //995us @11.0592M 12T
       WDR(); //喂看门狗
     }
}
#endif
/***** (C) COPYRIGHT 2012  Beijing GuangLun Electronic Technology Co., Ltd.****/
/*******************************************************************************/
/*                                                                             */
/*                          TreeOS_mcu.h 程序清单                              */
```

```
/*                                                                              */
/********************************************************************************/
/********防止冲突 prevent recursive inclusion*********************************/

#ifndef _TREEOS_MCU_H
#define _TREEOS_MCU_H

/********includes************************************************************/

#include "TreeOS_typedef.h"

/********条件编译选择********************************************************/
//选择是否使用 51 单片机中断
//#define ENABLE_T1            //M/
//#define ENABLE_T2            //M/
//#define ENABLE_UART0    //M/
//#define ENABLE_INT0     //M/
//#define ENABLE_INT1     //M/

//是否使用延时函数
#define USE_DELAYUS       //M/
//#define USE_DELAYMS      //M/ 注意：使用 delayms()，同时也要使用 delayus()

/********属性****************************************************************/
#ifndef WAIT_NOP
#define WAIT_NOP _nop_()  //M/     延时一个时钟周期
#endif

#define SET_SYS_INT()  EA=1    //M/ 开中断
#define CLR_SYS_INT()  EA=0    //M/ 关中断

#define SET_SYSTICK_INT()  ET0=1 //M/ 使能系统节拍时钟 T0 溢出中断
#define CLR_SYSTICK_INT()  ET0=0 //M/ 关闭系统节拍时钟 T0 溢出中断

//串口通讯波特率
//#define TL1_VALUE  0xd0    //fosc=11.0592Mhz,baud=1200 SMOD=1
//#define TH1_VALUE  0xd0
//#define TL1_VALUE  0xe8    //fosc=11.0592Mhz,baud=2400
//#define TH1_VALUE  0xe8
//#define TL1_VALUE  0xf4    //fosc=11.0592Mhz,baud=4800
//#define TH1_VALUE  0xf4
#define TL1_VALUE  0xfa    //fosc=11.0592Mhz,baud=9600
#define TH1_VALUE  0xfa
//#define TL1_VALUE  0xfd    //fosc=11.0592Mhz,baud=19200
```

```
//#define TH1_VALUE  0xfd


/********IO 定义**********************************************************************/

/********特殊语句**********************************************************************/

/********全局变量声明******************************************************************/
extern ui16 systick_ms;  //系统节拍计数器，一般单位为 1~10ms

extern ui16 second_stamp; //秒计时戳
extern ui8  second_flag;  //秒计时标志

/********函数声明**********************************************************************/
void port_init(void);
void watchdog_init(void);
void WDR(void);
void timer0_init(void);
#ifdef ENABLE_T1
void timer1_init(void);
#endif
#ifdef ENABLE_T2
void timer2_init(void);
#endif
#ifdef ENABLE_UART0
void uart0_init(void);
#endif
void int_init(void);

void MCU_init(void);

#ifdef USE_DELAYUS
void delayus(ui16 n);
#endif
#ifdef USE_DELAYMS
void delayms(ui16 n);
#endif

#endif

/***** (C) COPYRIGHT 2012  Beijing GuangLun Electronic Technology Co., Ltd.****/
```

&9.3 ：单片机中断程序：TreeOS_int

包括 TreeOS_int.c 和 TreeOS_int.h 文件。

该模块列出了 MCU 的主要中断服务子程序。本案例使用的 MCU 为 89C52。用户可根据实际应用进行配置，并填写相应的中断子程序。

```
/*****************************************************************************/
/*                                                                           */
/*                          TreeOS_int.c 程序清单                            */
/*                                                                           */
/*****************************************************************************/
/********includes*********************************************************/
#include<reg52.h> //M/
//#include<intrins.h>

#include "TreeOS_int.h"
#include "TreeOS_mcu.h"

//#include "TreeOs_IRR.h"
#include "TreeOS_keyboard.h"
//#include "TreeOS_LEDDP_dynamic.h"
#include "TreeOS_LEDDP_dynamic_B.h"
//#include "TreeOS_LEDDP_dynamic_B1.h"
//#include "TreeOs_Beep.h"
#include "TreeOs_beep_int.h"
//#include "TreeOS_RS232.h"
#include "TreeOS_IO.h"
#include "TreeOS_ds18b20.h"


/*****************************************TreeOS Real-Time Operating System****
Function Name:
Description  : T0 溢出中断处理程序
               计时器；
               各个任务延迟计数器更新；
                 需要及时处理的一些事务；等等
Input        :
Output       : TCNT0，各个任务延迟计数器更新等
Return       :
*****************************************************************************/
void timer0_isr(void) interrupt 1 //using 1
{
```

```
 //reload counter value
 TH0=0xfc;  //Fosc=11.0592MHz,1ms 中断
 TL0=0x66;

 systick_ms++;
 second_stamp++;

 //1s 定时
 if(second_stamp >= 1000)   //判断 1000ms 定时器周期到了没有
    {
        second_flag=1; //秒标志置位
         second_stamp=0;
    }

 //键盘定时扫描
 iSCAN_KEY_TIME;
 //红外遥控器键盘扫描
 //iSCAN_IRR_KEY_TIME;
 //数码管动态显示方式扫描
 iscan_LEDDP_dynamic();
 //数码管闪烁扫描
 iSCAN_LEDDP_BLINK_TIME;
 //蜂鸣器计时
 iSCAN_BEEP_TIME;
 //串口接收结束判断
 //iscan_uart_rcv_end();
 //开关量检测扫描
 iSCAN_READ_IN_TIME;
 //18b20 采样计时
 iSCAN_18B20_SAMPLE_TIME;
}

/*****************************************TreeOS Real-Time Operating System****
Function Name:
Description  : T1 溢出中断处理程序
Input        :
Output       :
Return       :
*******************************************************************************/
#ifdef ENABLE_T1
void time1_isr(void) interrupt 3
{

}
```

```
#endif
/*****************************************TreeOS Real-Time Operating System****
Function Name:
Description  : T2 溢出中断处理程序
Input        :
Output       :
Return       :
*******************************************************************************/
#ifdef ENABLE_T2
void time2_isr(void) interrupt 5
{

}
#endif
/*****************************************TreeOS Real-Time Operating System****
Function Name:
Description  : INT0 外部中断处理程序
Input        :
Output       :
Return       :
*******************************************************************************/
#ifdef ENABLE_INT0
void int0_isr(void) interrupt 0
{

}
#endif
/*****************************************TreeOS Real-Time Operating System****
Function Name:
Description  : INT1 外部中断处理程序
Input        :
Output       :
Return       :
*******************************************************************************/
#ifdef ENABLE_INT1
void int1_isr(void) interrupt 2
{

}
#endif
/*****************************************TreeOS Real-Time Operating System****
Function Name:
Description  : 串口接收中断处理程序
Input        :
```

```
Output       :
Return       :
*******************************************************************************/
#ifdef ENABLE_UART0
void uart0_isr(void) interrupt 4
{
  //uart has received a character in SBUF
  if(RI==1)
    {
     RI=0;  //清接收中断标志
     uart_rx();
    }
  //character has been transmitted
  else
    {
     TI=0;  //清发送中断标志
     uart_tx();
    }
}
#endif

/***** (C) COPYRIGHT 2012  Beijing GuangLun Electronic Technology Co., Ltd.****/
/*******************************************************************************/
/*                                                                             */
/*                          TreeOS_int.h 程序清单                              */
/*                                                                             */
/*******************************************************************************/
/********防止冲突 prevent recursive inclusion*********************************/

#ifndef _TREEOS_INT_H
#define _TREEOS_INT_H

/********includes*************************************************************/

#include "TreeOS_typedef.h"

/********条件编译选择*********************************************************/

/********属性*****************************************************************/

/********IO 定义**************************************************************/

/********特殊语句*************************************************************/
```

```
/********全局变量声明************************************************************/


/********函数声明************************************************************/

//void timer0_isr(void);
//void timer1_isr(void);
//void timer2_isr(void);
//void int0_isr(void);
//void int1_isr(void);
//void uart0_isr(void);

#endif

/***** (C) COPYRIGHT 2012  Beijing GuangLun Electronic Technology Co., Ltd.****/
```

&9.4 ： 74HC574 驱动程序：TreeOS_hc574

包括 TreeOS_hc574.c 和 TreeOS_hc574.h 文件。

该模块为并口输出扩展芯片 74HC574 的驱动程序，用于扩展输出口，可多个级联在一条总线上

使用方法：

1、系统初始化时需要调用 HC574_Output_Init(void);

2、改变某个 574 芯片的输出值（1 字节）Change_HC574_Output(ui8 chip, ui8 _data);

3、改变某位输出值（1 bit）Change_HC574_Output_Bit(ui8 chip, ui8 bit, enum SWITCH onoff)。

```
/*****************************************************************************/
/*                                                                           */
/*                        TreeOS_hc574.c 程序清单                             */
/*                                                                           */
/*****************************************************************************/
```

略。

&9.5 ： 数码管驱动程序：TreeOS_LEDDP_dynamic_B

包括TreeOS_LEDDP_dynamic_B.c和TreeOS_ LEDDP_dynamic_B.h文件。

本模块为数码管显示驱动程序，动态显示模式。数字、小数点可以控制是否闪烁。

使用方法：

1、LEDDP_buff[LEDDP_NUMBER]为显示存储区；

2、直接刷新显示存储区的内容，即可显示出来；

3、有闪烁显示需求时，需在 scan_in_while() 调用Scan_LEDDP(void)。

```
/*****************************************************************************/
/*                                                                           */
/*                    TreeOS_ LEDDP_dynamic_B.c 程序清单                     */
/*                                                                           */
/*****************************************************************************/
```

略。

&9.6 ： 蜂鸣器驱动程序：TreeOS_beep_int

包括 TreeOS_beep_int.c 和 TreeOS_beep_int.h 文件。

该模块为有源蜂鸣器的驱动程序。

只鸣一声，延续时长可调。调用 set_beep(ui16 beeptime)即可。

需在系统节拍时钟（T0）中断中执行 iScan_Beep_Time()。

```
/*****************************************************************************/
/*                                                                           */
/*                        TreeOS_beep_int.c 程序清单                         */
/*                                                                           */
```

```
/*****************************************************************************/
/********includes*****************************************************************/
#include<reg52.h> //M/
//#include<intrins.h>  //M/

#include "TreeOs_beep_int.h"
#include "TreeOS_hc574.h"  //M/
#include "TreeOS_mcu.h"

/********全局变量*****************************************************************/

ui16 beep_time_stamp=0;  //时间戳

/*****************************************TreeOS Real-Time Operating System****
Function Name:
Description  : 设置有源蜂鸣器的发声时长（鸣一声）
Input        :
Output       :
Return       :
*****************************************************************************/
void set_beep(ui16 beeptime)
{
  CLR_SYSTICK_INT(); //关系统节拍中断，准备修改 beep_time_stamp（临界区）
  beep_time_stamp=beeptime;
  SET_SYSTICK_INT();

  BEEP_ON;

}

/*****************************************TreeOS Real-Time Operating System****
Function Name:
Description  : 蜂鸣器扫描
               放在系统节拍时钟中断中执行！

Input        :
Output       :
Return       :
*****************************************************************************/
void iScan_Beep_Time(void)   //M/
{
 ui8 busval;
   //必须先保护数据总线的数据！
   busval=DATABUS;
```

```
 if(beep_time_stamp) beep_time_stamp--;
 else BEEP_OFF;

 //恢复数据总线的数据
   DATABUS=busval;
}


/*****************************************TreeOS Real-Time Operating System****
Function Name:
Description  : 上电时，蜂鸣器输出初始化
               需要根据各输出连接的设备来定
Input        :
Output       :
Return       :
*******************************************************************************/
void beep_init(void)  //M/
{
 beep_time_stamp=0;
 //SET_BEEP_GPIO_OUT;
}

/***** (C) COPYRIGHT 2012  Beijing GuangLun Electronic Technology Co., Ltd.****/

/*******************************************************************************/
/*                                                                             */
/*                          TreeOS_beep_int.h 程序清单                          */
/*                                                                             */
/*******************************************************************************/
/********防止冲突 prevent recursive inclusion*********************************/

#ifndef _TREEOS_BEEP_INT_H
#define _TREEOS_BEEP_INT_H

/********includes*************************************************************/

#include "TreeOs_typedef.h"

/********条件编译选择*********************************************************/

/********属性*****************************************************************/

/********IO 定义**************************************************************/
```

```
//#define SET_BEEP_GPIO_OUT   //设置 IO 口为输出口
#define BEEP_ON   Change_HC574_Output_Bit(HC574_CHIP3, BEEP, ON)   //M/
#define BEEP_OFF  Change_HC574_Output_Bit(HC574_CHIP3, BEEP, OFF)   //M/

/********特殊语句*************************************************************/

//刷新扫描时间戳 beep_timestamp，此句放在 T0 中断程序中
#define iSCAN_BEEP_TIME  iScan_Beep_Time()

/********全局变量声明*********************************************************/

extern ui16 beep_time_stamp;  //时间戳

/********函数声明*************************************************************/

void set_beep(ui16 beeptime);
void iScan_Beep_Time(void);
void beep_init(void);

#endif

/***** (C) COPYRIGHT 2012  Beijing GuangLun Electronic Technology Co., Ltd.****/
```

&9.7 ： 键盘驱动程序：TreeOS_keyboard

包括 TreeOS_keyboard.c 和 TreeOS_keyboard.h 文件。

本模块为键盘驱动程序。

功能：

1.键盘扫描 scan_key()放在大循环 scan_in_while()中定时进行;

2.扫描间隔一般 20~40ms(机械)或 2~5ms(薄膜键盘),以躲开抖动区;

3.数字滤波：连续 2 次扫描都非空,才能认为有键按下;

4.键盘计时器:有键按下开始计时,为键保持时间;键释放后开始

计时,则为空键期;

5.可以检测多键,键值以确认有键按下后,扫描过程中键值最大(最多键)的为键值;

6.可以检测长按键（可选）;

7.可以检测加速键（尚未加上）;

8.可以检测组合键（尚未加上）;

本程序既适用于矩阵键盘，也适用于多位独立键(公共端接地)。

键盘扫描具体做法:

1)设扫描时间戳 keystamp,扫描间隔 SCANKEY_CYCLE(T0 计时单位);

2)在 T0 中断程序中判断倒计时 keystamp;

3)在大循环中判断若 keystamp=0,调用一次;

真正的扫描周期等于 SCANKEY_CYCLE+(0~1)个大循环时间,平均周期等于 SCANKEY_CYCLE+0.5 个大循环时间。

使用方法:

- 键值 keyval(可多键)、长按键 longkeyval（可选）、加速键（可选）、加速键（可选）;
- 数字键值 numkeyval;
- 可判断按键瞬间、释放瞬间;
- 有按键时间、空键时间记录;

注意：本子程序并不清零键值(包括加速键、长按键等)，必须在主程序使用后马上清零 keyval、longkeyval 等键值!

移植时在 h 文件中进行配置，c 文件中 KeyPress()、ReadPress() 也要做相应修改（针对不同的 IO 口配置）。

```
/*****************************************************************************/
/*                                                                           */
/*                          TreeOS_keyboard.c 程序清单                        */
/*                                                                           */
/*****************************************************************************/
/********includes*********************************************************/
#include<reg52.h> //M/

#include "TreeOs_keyboard.h"
#include "TreeOS_mcu.h"

#include "TreeOs_beep_int.h"

/********全局变量*********************************************************/

//每个键状态用 1 位表示:1 表示按下,0 表示未按下
KEY_DATA_TYPE lastkey=0;  //上次扫描的临时键值
KEY_DATA_TYPE maxkey=0;   //有键按下后,记录最大的键值,用于确认多键
KEY_DATA_TYPE keyval=0;   //对外输出的键值
#ifdef NUM_KEYBOARD //有数字键盘
ui8 numkeyval=0xff; //辅助数字键值 0~9,非数字键时=0xff
#endif

ui16 keytimer=0; //按键计时器（近似值）,单位=2 次键扫描间隔时长
ui16 nokeytimer=0; //无按键计时器（近似值）,单位=2 次键扫描间隔时长
ui8 key_timestamp=SCANKEY_CYCLES; //扫描时间戳

#ifdef USE_UNKEY  //使用组合键（选用）
KEY_DATA_TYPE lastkeyval=0;
KEY_DATA_TYPE unkeyval=0; //组合键值
#endif

#ifdef USE_LONGKEY  //使用长按键（选用）
KEY_DATA_TYPE longkeyval=0;
#endif

/*****************************************TreeOS Real-Time Operating System****
Function Name:
Description  : 键盘初始化
Input        :
```

```
Output        :
Return        : 0：无按键; 非 0：有按键动作
*****************************************************************************/
void keyboard_init(void)    //M/
{
  //所有 KO 线设为输出
  SET_ALL_KO_OUT;  //51 单片机不需要
  //所有 KI 线设为输入
  SET_ALL_KI_IN;   //51 单片机把它们置 1 即可
  //拉高所有 KO 线
  SET_ALL_KO_HIGH;

}

/*****************************************TreeOS Real-Time Operating System****
Function Name:
Description  : 检查是否有按键动作
Input        :
Output       :
Return       : 0：无按键; 非 0：有按键动作
*****************************************************************************/
ui8 KeyPress(void)      //M/
{
  //拉低所有 KO 线
  SET_ALL_KO_LOW;
  //读所有输入线(d0~d3 为代表 KI0~KI3 的值)
  if(READ_KEYIN!=0) return 0xff; //有线被拉低,说明有按键
  else return 0;
}

/*****************************************TreeOS Real-Time Operating System****
Function Name:
Description  : 读入当前键状态
               1 表示按下,0 表示空
Input        :
Output       :
Return       : 返回:各键状态。0：无按键; 非 0：有按键动作
*****************************************************************************/
KEY_DATA_TYPE ReadPress(void)    //M/
{
 ui8 i;
 KEY_DATA_TYPE input,val=0;

 SET_ALL_KO_HIGH; //拉高所有 KO 线
```

```
 //逐条拉低 KO 线，并读取所有 KI
 for(i=0;i<KEY_KO_NUM;i++)
   {
    KO_L(i);
    //读输入 KI。注意若 MPU 速度太快，需要延时！
    KO_L(i);  //起延时作用
    KO_L(i);
    input=READ_KEYIN; //结果在 input d0~d3
    val|=(input<<(i*KEY_KI_NUM));  //每路 KI 占 4bits
    KO_H(i); //复位
   }

  return val;
}

/*****************************************TreeOS Real-Time Operating System****
Function Name:
Description  : 键盘扫描程序
               在 ScanInWhile()中调用（查询方法）
Input        :
Output       : keyval 等各键盘变量
Return       :
*******************************************************************************/
void scan_key(void)
{
  KEY_DATA_TYPE key;
  ui8 t;   //数据类型同 key_timestamp

  //CLR_SYSTICK_INT(); //关系统节拍中断，准备读取系统计时 ms 值    （临界区）    //对于 8 位单片
机，单字节数据读写都是原子操作，无需关中断！
  t=key_timestamp;
  //SET_SYSTICK_INT();

  if(t) return;          //确保扫描周期>=READ_IN_CYCLES
  else
    {
     //CLR_SYSTICK_INT(); //关系统节拍中断，准备修改 key_timestamp 值  （临界区）
     key_timestamp=SCANKEY_CYCLES;
     //SET_SYSTICK_INT();
    }

  //判断是否有键按下
  if(KeyPress()) //有按键
```

```
    { key=ReadPress(); //读取当前键值
      if(lastkey) //上次已非空键
         {
           if(!maxkey) keytimer=0;   //刚确认有键按下,开始持键计时
            if(key>maxkey) maxkey=key; //整个持键过程中，取出现过的最大值，以获得多键
            //有按键开背光或者开蜂鸣器
           //OPEN_BACKLIGHT;
            set_beep(200);
         }
      lastkey=key;
      keytimer++;
    }
  else //空键
    {
      //键已经释放,输出键值 keyval
      if(maxkey)
         { keyval=maxkey;
#ifdef USE_LONGKEY  //使用长按键
              if (keytimer>LONGKEY_SCANKEY_CYCLES) longkeyval=keyval;
#endif
#ifdef NUM_KEYBOARD //有数字键盘
           //给出数字键辅助键值，便于计算
            if(keyval==KEYD) numkeyval=0;
             else if(keyval==KEY0) numkeyval=1;
             else if(keyval==KEY1) numkeyval=2;
             else if(keyval==KEY2) numkeyval=3;
             else if(keyval==KEY4) numkeyval=4;
             else if(keyval==KEY5) numkeyval=5;
             else if(keyval==KEY6) numkeyval=6;
             else if(keyval==KEY8) numkeyval=7;
             else if(keyval==KEY9) numkeyval=8;
            else if(keyval==KEYA) numkeyval=9;
             else  numkeyval=0xff;
#endif
            maxkey=0;
             //此时可以开始空键计时
             nokeytimer=0;
         }
      lastkey=0;
      nokeytimer++;
      //长时间无按键关背光
      //if(nokeytimer>BACKLIGHT_DELAY) CLOSE_BACKLIGHT;
    }//else
```

```
}

/***** (C) COPYRIGHT 2012  Beijing GuangLun Electronic Technology Co., Ltd.****/

/*****************************************************************************/
/*                                                                           */
/*                          TreeOS_keyboard.h 程序清单                        */
/*                                                                           */
/*****************************************************************************/
/********防止冲突 prevent recursive inclusion*********************************/
#ifndef _TREEOS_KEYBOARD_H
#define _TREEOS_KEYBOARD_H

/********includes*************************************************************/
#include "TreeOs_typedef.h"

/********条件编译选择*********************************************************/

//键个数：正确选用可节约内存
#define KEY_DATA_TYPE   ui16  //M/  8/16/32 个键分别使用 ui8/ui16/ui32
//是否使用长按键
#define USE_LONGKEY   //M/
//是否使用加速键
//#define USE_SPEEDKEY  //M/
//是否使用组合键
//#define USE_UNKEY    //M/
//输入键盘带 0~9 数字
#define NUM_KEYBOARD //M/

/********属性*****************************************************************/

//矩阵键盘线数
#define KEY_KO_NUM  4  //M/ KO 线数量
#define KEY_KI_NUM  4  //M/ KI 线数量，要求不超过 8 条

//键盘扫描周期
//键扫描周期,单位=TO 周期(例如 1ms)
#define SCANKEY_CYCLES  20  //M/

//长按键
#define LONGKEY_SCANKEY_CYCLES 100  //M/ 单位=键扫描周期 SCANKEY_CYCLES，定义按多长时间开始出现 longkeyval

#define NOKEY_SCANKEY_CYCLES  1000  //M/ 无按键周期设置
```

```
#define NOKEY_SCANKEY_CYCLES_5S  250  //M/ 5 秒无按键周期设置

//键值定义
//通用 4*4 薄膜键盘键位定义(keyval 中 D0~D15)
#define KEY0  (1<<0)   //M/ 坐标(KI0,KO0)
#define KEY1  (1<<1)   //M/ 坐标(KI0,KO0)
#define KEY2  (1<<2)   //M/ 坐标(KI0,KO0)
#define KEY3  (1<<3)   //M/ 坐标(KI0,KO0)

#define KEY4  (1<<4)   //M/ 坐标(KI1,KO1)
#define KEY5  (1<<5)   //M/ 坐标(KI1,KO1)
#define KEY6  (1<<6)   //M/ 坐标(KI1,KO1)
#define KEY7  (1<<7)   //M/ 坐标(KI1,KO1)

#define KEY8  (1<<8)   //M/ 坐标(KI2,KO2)
#define KEY9  (1<<9)   //M/ 坐标(KI2,KO2)
#define KEYA  (1<<10)  //M/ 坐标(KI2,KO2)
#define KEYB  (1<<11)  //M/ 坐标(KI2,KO2)

#define KEYC  (1<<12)  //M/ 坐标(KI3,KO3)
#define KEYD  (1<<13)  //M/ 坐标(KI3,KO3)
#define KEYE  (1<<14)  //M/ 坐标(KI3,KO3)
#define KEYF  (1<<15)  //M/ 坐标(KI3,KO3)

//复用定义
#define SINGLE_KEY         KEYF  //M/  单一按键

#define UP          KEY1  //M/  上
#define DOWN        KEY9  //M/  下
#define LEFT        KEY4  //M/  开
#define RIGHT       KEY6  //M/  关
#define OK          KEY5  //M/  确定
#define ESC         KEYE  //M/  取消

//#define KEY_SPACE  KEYA  //M/ 空格键
//#define KEY_MINUS  KEYB  //M/ -键
//#define KEY_DOT    KEYC  //M/ .键

//多键(可根据需要自行添加)
//#define UP_DOWN  (UP|DOWN)  ///M/
//#define LEFT_RIGHT  (LEFT|RIGHT)  ///M/


/********IO 定义*****************************************************************/
```

```
//4X4 键盘
#define SET_ALL_KO_OUT                //M/所有 KO 线设为输出 P1.0~1.3
#define SET_ALL_KI_IN     P0|=0xf0    //M/所有 KI 线设为输入 P0.4~0.7

#define KO_H(x)    P1|=(1<<x)   //M/
#define KO_L(x)    P1&=~(1<<x)  //M/

#define SET_ALL_KO_HIGH    P1|=0x0f   //M/
#define SET_ALL_KO_LOW     P1&=0xf0   //M/

#define READ_KEYIN  ((~P0>>4)&0x0f) //M/ 取反,有按下的键置 1 .结果放在 d0~d3

/********特殊语句*************************************************************/

//刷新扫描时间戳 key_timestamp, 此句放在 T0 中断程序中
#define iSCAN_KEY_TIME  if(key_timestamp) key_timestamp--

/********全局变量声明*********************************************************/

extern KEY_DATA_TYPE lastkey;  //上次扫描的临时键值
extern KEY_DATA_TYPE maxkey;   //有键按下后,记录最大的键值,用于确认多键
extern KEY_DATA_TYPE keyval;   //对外输出的键值

extern ui16 keytimer; //键盘计时器,单位=2 次键扫描间隔时长
extern ui16 nokeytimer; //无按键计时器,单位=2 次键扫描间隔时长
extern ui8 key_timestamp; //扫描时间戳

#ifdef USE_LONGKEY  //使用长按键（选用）
extern KEY_DATA_TYPE longkeyval;
#endif

#ifdef NUM_KEYBOARD //有数字键盘
extern ui8 numkeyval; //辅助数字键值 0~9,非数字键时=0xff
#endif

/********函数声明*************************************************************/
void keyboard_init(void);
ui8 KeyPress(void);
KEY_DATA_TYPE ReadPress(void);
void scan_key(void);

#endif

/***** (C) COPYRIGHT 2012  Beijing GuangLun Electronic Technology Co., Ltd.****/
```

&9.8 ： I2C 总线驱动程序：TreeOS_I2C

包括 TreeOS_I2C.c 和 TreeOS_I2C.h 文件。

本模块为 I2C 总线的驱动程序，用 IO 口模拟。

```
/*****************************************************************************/
/*                                                                           */
/*                        TreeOS_I2C.c 程序清单                              */
/*                                                                           */
/*****************************************************************************/
```

略。

&9.9 ： PCF8563 驱动程序：TreeOS_pcf8563

包括 TreeOS_ pcf8563.c 和 TreeOS_ pcf8563.h 文件。

本模块为 I2C 总线日历时钟芯片 PCF8563 的驱动程序。

使用方法：

上电时启动时钟 start_PCF8563();

读时钟 read_PCF8563(),时钟值存放在 DateTime;

写时钟 write_PCF8563(),时钟存放值在 DateTime。

```
/*****************************************************************************/
/*                                                                           */
/*                       TreeOS_pcf8563.c 程序清单                           */
/*                                                                           */
/*****************************************************************************/
```

略。

&9.10 ： DS18B20 驱动程序：TreeOS_ds18b20

包括 TreeOS_ds18b20.c 和 TreeOS_ds18b20.h 文件。

本模块为单总线温度传感器 DS18B20 的驱动程序。

软件使用说明：

void scan_ds18b20(void)：读取温度扫描，要放在 scan_in_while()中进行循环扫描；

在 T0 中断服务程序 timer0_isr(void) 中调用 iSCAN_18B20_SAMPLE_TIME；

注意，软件运行过程中多处关闭了系统中断，最长关闭时间达 1ms 左右。

```
/*****************************************************************************/
/*                                                                           */
/*                        TreeOS_ds18b20.c 程序清单                          */
/*                                                                           */
/*****************************************************************************/
```

略。

&9.11 ：开关量输入输出程序：TreeOS_IO

包括 TreeOS_IO.c 和 TreeOS_IO.h 文件。

本模块集合了开关量信号的输入输出函数，这与具体应用紧密相关。一般包括：控制指示灯、控制继电器、开关量信号读取(扫描方式)，等等。

输入信号采集采用了每个大循环采集一次的办法，因此采集周期较长，仅适用于缓慢变化的开关量！

```
/*****************************************************************************/
/*                                                                           */
/*                        TreeOS_IO.c 程序清单                               */
/*                                                                           */
/*****************************************************************************/
```

略。

&9.12 ：数码管显示常用库程序：TreeOS_LEDDP_disp

包括 TreeOS_LEDDP_disp.c 和 TreeOS_LEDDP_disp.h 文件。

本模块集合了数码管显示常用的显示子程序，如显示时间、显示某个整数、或字符串等。

```
/*****************************************************************************/
/*                                                                           */
/*                        TreeOS_LEDDP_disp.c 程序清单                       */
/*                                                                           */
/*****************************************************************************/
```

略。

&9.13 ： 标准库程序：TreeOS_stdlib

包括 TreeOS_stdlib.c 和 TreeOS_stdlib.h 文件。

本模块主要是常用的数据-字符转换函数库。尽管有些函数在编译器里已有，但不够简明。本模块提供的函数经过了简化，可以节省代码空间和内存。

```
/*****************************************************************************/
/*                                                                           */
/*                        TreeOS_stdlib.c 程序清单                           */
/*                                                                           */
/*****************************************************************************/
```

略。

&9.14 ： 主场景程序：TreeOS_main

包括 TreeOS_main.c 和 TreeOS_main.h 文件。

该模块包括主场景程序（主函数 main）、初始场景 init_scene()（一般的项目都不需要）、全局任务函数 scan_in_while()、以及系统上电初始化等等。本模块由用户编写。

```
/*********************************************************************************/
/*                                                                               */
/*                          TreeOS_main.c 程序清单                                */
/*                                                                               */
/*********************************************************************************/
/********includes*************************************************************/
#include<reg52.h> //M/
//#include<intrins.h>

#include "TreeOS_main.h"

#include "TreeOS_mcu.h"
#include "TreeOS_int.h"

#include "TreeOS_hc574.h"
//#include "TreeOS_LEDDP_dynamic.h"
#include "TreeOS_LEDDP_dynamic_B.h"
//#include "TreeOS_LEDDP_dynamic_B1.h"
#include "TreeOS_seg7_font.h"
#include "TreeOS_LEDDP_disp.h"

//#include "TreeOS_hd61202.h"
//#include "TreeOS_hd44780.h"
//#include "TreeOS_LCD_disp.h"

#include "TreeOS_I2C.h"
//#include "Tree0s_24Cxx.h"
#include "TreeOS_pcf8563.h"
//#include "Tree0s_pcf8591.h"

//#include "TreeOS_RS232.h"
//#include "TreeOS_RS485_slave.h"
//#include "TreeOS_RS485_master.h"

//#include "Tree0s_Beep.h"
#include "Tree0s_beep_int.h"

#include "TreeOS_keyboard.h"
//#include "Tree0s_IRR.h"


#include "TreeOS_IO.h"
//#include "TreeOS_analog.h"
#include "TreeOS_ds18b20.h"
```

```
//#include "TreeOS_stdlib.h"

#include "TreeOS_scn_input_asc_leddp.h"
#include "TreeOS_scn_subs.h"

/********全局变量**********************************************************************/

ui8 scene=0;   //当前激活场景编号

ui16 dist=0;   //单次行驶公里数

/****************************************TreeOS Real-Time Operating System****
Function Name: Scan_In_While
Description  : 系统全局普通任务集合函数
               每个场景的 while 循环都需要调用,故称 Scan In While;
                 以扫描的方式顺序执行各全局任务，包括数据采集和输出;
Input        :
Output       :
Return       :
******************************************************************************/
void scan_in_while(void)
{
 //读取实时钟
 read_pcf8563();
 //键盘扫描
 scan_key();
 //红外遥控器扫描
 //scan_IRR_key();

 //开关量检测
 scan_in();
 //ADC 检测
 //scan_analog();
 //DS18B20 检测
 scan_ds18b20();

 //蜂鸣器扫描
 //scan_beep();
 //数码管闪烁显示
 scan_LEDDP();

 //串口通讯
 //scan_RS232();
```

```
 //scan_RS485();

 //更新显示时间或者秒运算,每秒 1 次
 /*if(DateTime.sec!=old_sec)
   {
    old_sec=DateTime.sec;

   }*/
 /*if(second_flag) //判断秒标志是否置位
   {
     second_flag=0; //清除秒标志

   }*/
}

/*****************************************TreeOS Real-Time Operating System****
Function Name:
Description  : 系统主程序
               系统各设备初始化、自检；
                 一般包括初始场景、主场景；
Input        :
Output       :
Return       :
*******************************************************************************/
void main(void)
{
  ui8 counter=0;
  ui8 last_min;

  MCU_init(); //单片机初始化

  //==========devices 周边设备初始化===========

  HC574_output_init();  //开机后，再使能所有 74HC574

  LEDDP_init();    //数码管初始化
  //lcd_init();    //液晶初始化

  I2C_init();
  if(start_pcf8563()) start_pcf8563();
  //init_24Cxx();
  //eeprom_init();

  beep_init();
```

```
  keyboard_init();

  IO_init();
  //analog_init();
  ds18b20_init();


  //==================================

  set_beep(500);   //上电鸣一声
  read_pcf8563();
  LEDDP_show_hhmm(DateTime.hour, DateTime.min, LEDDP0);  //显示时分
  LEDDP_dot_blink_set|=(1<<1);  //小数点闪烁

  while(1)   //主场景循环
     {
       WDR();  //喂看门狗

       //处理按键
       if(keyval)
         {
          if(longkeyval==SINGLE_KEY)   //长按键 修改时间
            {
              LEDDP_dot_blink_set&=~(1<<1);  //取消小数点闪烁
               scn_sub_modify_time();
             LEDDP_dot_blink_set|=(1<<1);  //小数点闪烁
              LEDDP_show_hhmm(DateTime.hour, DateTime.min, LEDDP0);
            }
          else if(keyval==SINGLE_KEY)   //短按键    显示温度及行驶里程
            {
              LEDDP_dot_blink_set&=~(1<<1);  //取消小数点闪烁
               scn_sub_show_temp();
             LEDDP_dot_blink_set|=(1<<1);  //小数点闪烁
              LEDDP_show_hhmm(DateTime.hour, DateTime.min, LEDDP0);
            }
          keyval=0;  //按键处理完毕后，必须清空键值!
           longkeyval=0;
         }

       //局部任务
       if(DateTime.min!=last_min)  //每分钟更新一次时间
        {
         last_min=DateTime.min;
        LEDDP_show_hhmm(DateTime.hour, DateTime.min, LEDDP0);
```

```
        }

        //全局任务扫描
        scan_in_while();
    }
}

/***** (C) COPYRIGHT 2012  Beijing GuangLun Electronic Technology Co., Ltd.****/

/*****************************************************************************/
/*                                                                           */
/*                          TreeOS_main.h 程序清单                            */
/*                                                                           */
/*****************************************************************************/
/********防止冲突 prevent recursive inclusion*********************************/

#ifndef _TREEOS_MAIN_H
#define _TREEOS_MAIN_H

/********includes*************************************************************/

#include "TreeOS_typedef.h"

/********条件编译选择*********************************************************/

/********属性*****************************************************************/
//场景定义
#define SCENE_NULL  0  //上电后程序初始化阶段，未进入场景
#define SCENE_INIT  1  //初始场景
#define SCENE_MAIN  2  //主场景

/********IO 定义**************************************************************/

/********特殊语句*************************************************************/

/********全局变量声明*********************************************************/

extern ui8 scene;   //当前激活场景编号

extern ui16 dist;   //单次行驶公里数

/********函数声明*************************************************************/
//void init_scene(void);  //初始场景函数
void scan_in_while(void);  //全局任务函数
```

```
#endif

/***** (C) COPYRIGHT 2012  Beijing GuangLun Electronic Technology Co., Ltd.****/
```

&9.15 ： 子场景程序：TreeOS_scn_subs

包括 TreeOS_scn_subs.c 和 TreeOS_scn_subs.h 文件。

该模块集合了所有子场景程序。本模块由用户编写。

```
/*****************************************************************************/
/*                                                                           */
/*                        TreeOS_scn_subs.c 程序清单                          */
/*                                                                           */
/*****************************************************************************/
```

略。

&9.16 ：输入 ASC 字符库程序：TreeOS_scn_input_asc_leddp

包括 TreeOS_scn_input_asc_leddp.c 和 TreeOS_scn_input_asc_leddp.h 文件。

该模块用于输入 ASC 字符(密码或数字)。用//UP/DN/L/R/OK 5 键输入模式，使用数码管作为显示设备。scn_input_asc()是一个场景程序。输入结果存放在 ascbuff[INPUT_BUFF_SIZE]。本模块比较通用。

在《汽车智能电子表》中仅用 1 个按键来完成 HH:MM 时间的输入。

```
/*****************************************************************************/
/*                                                                           */
/*              TreeOS_scn_input_asc_leddp.c 程序清单                         */
/*                                                                           */
/*****************************************************************************/
```

略。

&9.17 ： 数码管 7 段代码字库：TreeOS_seg7_font

包括 TreeOS_seg7_font.c 和 TreeOS_seg7_font.h 文件。

该模块列出了常用的数码管七段代码（共阴型）。用户可以根据项目的需求进行剪裁，以节省 ROM 空间。

```
/*****************************************************************************/
/*                                                                           */
/*                        TreeOS_seg7_font.c 程序清单                        */
/*                                                                           */
/*****************************************************************************/
```

略。

附录 A：部分成功案例

空气净化器集中控制

小区供热温度监测

建筑工地爬架荷载控制系统

大型祭祀控制管理系统

大型智能寄存管理控制系统

博物馆集控

厨余垃圾处理控制柜

小型无土栽培集控器（系列）

开放式小型无土栽培系统

电站锅炉煤粉取样

电站锅炉风粉监测

核子秤计量与配料（系列）

电子秤计量与配料（系列）

电子倾角仪（系列）

机床通讯控制器、存储器

股市资讯通讯监视报警器

原油产量流量积算仪

视频字符叠加器

电梯楼层显示控制板

CAN 总线通讯系统

IC 卡读写器

激光打标机操控板

激光打标机控制器

医用等离子体碎石机能量输出控制板

坦克大战智能游戏系统

网络型超薄灯箱电子值班显示屏

超薄 LED 显示条屏（系列）

电机专用数字电子开关控制器

三相电无功补偿控制器

工业计数器

温度巡检仪

蒸汽洗车机控制器

环保厕所自动控制器

麦克风自动控制器

甲烷检测仪

电机位发控制板

张力显示单元

小灵通接入控制台

彩色流水灯控制器

智能家居集控器

电热锅炉控制器

家用蒸汽吸尘器报警器

矢量电压百分表

消防水箱控制箱

运算数码屏

内投遥控器

切纸机控制器

铁轨测距器

机床位置控制器

展柜 MP3 自动播放器

升降控制器

火焰信号放大器

时间事件记录仪

汽车防盗报警器

汽车甲醇发动机控制器

汽车电子钟

甲烷氧气两用仪

盲人求助器

自净式药品柜控制器

学习型红外遥控器

学习型无线遥控器

行程控制器

太阳能洗浴控制器

消防报警器

PCR 基因实验仪控制器

加油机油量油速显示器

工业干燥机自动控制器

脉搏检测器

催款时间锁

净化柜控制器

杂交生物实验仪

炼苗室控制器

啤酒泡沫测试仪

大容量存储手持测温仪

485 集线器、中继器

RFID 门禁控制器（系列）

光透率测量控制装置

汽车分时租赁管理监控器

脑电波赛车

磁导航智能小车

智能农业温室控制系统

家庭智能无土栽培设备

微型电机无线遥控装置

工业信号无线采集系统

太阳能充电控制器

高精度 GPS 航拍控制系统

人体经络信号检测仪

附录 B：TreeOS 编程规范

前言：

编程规范可使程序更加简明、整齐干净、易读、易理解、易移植，从而提高移植性、兼容性、易维护等，提高编程效率。

当然，若规矩太多，难以记得全，反而对程序员是个负担。因此把规范尽量精简，抓住关键之处即可。

编程规范要求贯穿始终。TreeOS 是开源软件，也要求对其修改或扩展的软件也必须遵守相同的编程规范。

以下是 TreeOS ComLib 软件构件库的编程规范。

一 基本原则

- 所有的代码必须采用 ANSI C 标准
- 严格遵守编程规范
- 尽可能使用简单的语法
- 避免使用复杂语句
- 不使用晦涩难懂的表述
- 不要使用 GOTO 语句
- 修改代码，需随时更新相关文档

二、 源代码文件

我们把具有一定功能的源代码文件（如某种设备的驱动程序）称为一个模块，它包括两个文件：执行文件（.c 文件）和头文件（.h 文件）。

执行文件的布局:

- 文件头
- 包含文件(#include)
- 变量，包括全局变量和局部变量
- 表单
- 函数，包括全局函数和局部函数

头文件布局:

- 文件头
- 包含文件(#include)
- Global macros (全局宏定义)

  包括：条件编译选择、属性（常量定义）、IO 管脚定义、特殊语句。
- 全局变量外部声明
- 函数声明(前一部分为局部函数原型，后一部分为全局函数原型)

注意：头文件必须保证避免在包含文件中重复定义一个数值。

三、文件头

文件头是放在源代码文件头部的一个注释块，其内容包括版权、公司信息、程序员、文件的描述信息、软件版本及修改历史记录等。

注意每次修改软件，应该做相应的版本修改记录。

例如：

文件名称：TreeOS_main.h

适用 TreeOS：本软件适用于 TreeOS 实时操作系统。

配置：缺省配置适用 TreeOS Kepler11 开发板。

版本：隶属 ComLib A2 构件库, 2014

功能：

*TreeOS_main.c 的配置文件

修改记录：

1、完成日期：2014-10-1 ；作者：TreeOS 老林

****
**********/

四、文件包含

TreeOS 要求每个模块的 C 文件必须配备一个同名头文件。

模块 C 文件可以只列出需要包含的头文件，也可以用一个名为 TreeOS_inc.h 的头文件把所有文件包含进去，这样虽然省事（所有模块都只需包含 TreeOS_inc.h 即可），但是会使编译时间加长，而且随着 TreeOS ComLib 的不断扩大，这个 TreeOS_inc.h 也需要根据需求配置。另外，模块若仅包含所需的头文件，可以直观地看到该模块与其它哪些模块出现了耦合。

例如：

#include<reg52.h> //M/

```
#include "TreeOS_main.h"
#include "TreeOS_mcu.h"
#include "TreeOS_int.h"

#include "TreeOS_hc574.h"
#include "TreeOS_hd61202.h"
#include "TreeOS_lcd_disp.h"

#include "TreeOS_I2C.h"
#include "TreeOS_24Cxx.h"
#include "TreeOS_pcf8563.h"

#include "TreeOS_RS232.h"
#include "TreeOS_beep.h"
#include "TreeOS_keyboard.h"
```

五、函数声明

模块中所有定义的函数都在头文件中进行声明。分为两部分，前一部分为局部函数原型，后一部分为全局函数原型。

需要调用其它模块中定义的函数，必须包含该模块的头文件。

六、注释

注释写在语句后面，一般使用//符号。太长了可以分多行写。之所以用//，一是书写简单，二是有时方便把一段程序用/* */注释掉（比使用 #if 0 和 #endif 更直观）。

当然，也不排斥注释使用/*..*/，但要求同一个模组中(即对应的.c 和.h 文件)，使用同一种注释风格。

七、缩进格式

记住函数的“{}”直接靠近左栏边沿，语句则相对左栏边沿缩进 4 个空格。

对于 if、for、while、switch 等语句，其语句块大括号“{”与“}”不缩进，且单独放一行，并成对处在同一列。这样便于区分众多的{}。大括号{}内的语句缩进 4 格，把{}露出来。

例如：

```
if ((t - t0)  >  500)  //500ms 计时
 {
      t0 = t;
     for ( i  =  0;  i  <  5;  i++)
     {
          disp_chinese(hao + i * 32,  0);
      } //for
} //if
```

对于 case 语句，缩进 4 个空格，把 case 露出来。

例如：

```
case UP:
    if (item)
     {
        ……
     }
     break;
case DOWN:
    if (item   <   5)
     {
        ……
     }
     break;
……
```

八、命名规则

- 宏定义

宏定义用大写表示，用下划线分隔单词。特殊情况允许使用 1 个小写字母，且只能放在最前面，主要是为了避免与其它引进程序产生冲突。

宏定义一般以所在的文件的文件名的一部分作为前缀（开头）。

例如：

#define UART_BUFFER_SIZE_SEND 20 //M/ 发送缓冲区字节数。

注意：#define 和宏名之间空一个空格就可以了。

- 变量名与函数名

命名要尽量表达清楚，尽量让人一眼能猜出其意义。有两种命名方式：一种是词的首字母大写，以此来分开不同词；二是所有词都用小写字母，中间用下划线“_”分开。两种方法都可用，第二种方法虽然多了“_”，但更有利于阅读。

变量名与函数名一般以所在的文件的文件名的一部分作为前缀（开头）。

例如：

ui8 UartSendPrt = 0;

void lcd_init(void)。

九、数据类型

为了通用于各种处理器的 IDE 以及书写方便提高效率，TreeOS 用 typedef 定义了所有数据类型，这些定义放在一个专门的头文件 TreeOS_typedef.h。

有些平台 char 等价于 signed char，另一些则等价于 unsigned char，为此，要在代码中明确的使

用 signed char 或 unsigned char 。

不能直接使用 int 类型，要使用 short 和 long。

例如：

```
typedef signed long    i32;
typedef signed short    i16;
typedef signed char    i8;

typedef code signed long    i32c;    //code data,read only
typedef code signed short    i16c;    //code data,read only
typedef code signed char    i8c;      //code data,read only

typedef unsigned long    ui32;
typedef unsigned short    ui16;
typedef unsigned char    ui8;

typedef code unsigned long    ui32c;    //code data,read only
typedef code unsigned short    ui16c;    //code data,read only
typedef code unsigned char    ui8c;      //code data,read only
```

十、语句与表达式

- 每行只放一个语句；

  例如：

```
if（a < b）
    return 1;
    else
        return 0;
```

- 代码块之间加空行分隔；
- 每个逗号或分号后面要有一个空格符分隔；

例如：ui8 send_data_24Cxx(ui16 SubAdr, ui8 ByteCnt ui8 *p)

for (i = 0; i < 100; i++)

注意：函数名后的括号不可以有空格符；

括号()与括号内的字符间不要有空格。

- 一元操作符与操作数之间不可以包含空格；

例如：!a, ~b, i++, i--, *p, &x, (ui16)y 等。

- 二元和多元操作符与操作数之间至少要有一个空格；

例如：a + b; a = b; a | b; a > b; a >= b; 等。

- 部分关键字后面要跟随一个空格符

例如：for (i = 0; i < 100; i++)，if (a > b)，else {… }，switch (a)，return (a)等。

十一、结构体与共用体

结构类型用大写字母表示（允许带 1 个小写字母），规则同宏定义。

例如：

```
typedef struct       //定义时分 hh:mm
{
ui8 hour;          //0~23
ui8 min;           //0~59
} tHHMM;
```

十二、功能配置

TreeOS 是一个可配置、可剪裁的构件化操作系统。对于具体应用，需要对功能进行配置（在.h 文件中修改），甚至对一些软件进行修改（在.c 文件中修改）。所有这些需要修改的地方，都带有“//M/”（M 即 Modify 的意思），提醒用户此处可能需要修改。

例如：

```
#define UART_BUFFER_SIZE_SEND    20    //M/ 发送缓冲区字节数。
```

//M/在该处提醒修改串口发送缓冲区的大小。太小了不合适，太大了浪费内存。

附录 C: AlphaMCU 原理图设计规则

(2017-2　Rev.001，更新版本详见官网)

&1 概述

AlphaMCU 可以根据电路原理图自动生成定制化操作系统的源代码，包括 MCU 片上资源代码、周边器件（芯片或模块）驱动程序、中间件、通讯协议、以及边缘计算代码等。

AlphaMCU 是根据网络表文件来识别原理图的。本文档阐述了器件命名、网络命名等设计规则。

AlphaMCU 只能识别按照此规则设计的原理图。

&2 适用 PCB 设计工具软件

目前适用：Altium Designer、 Protel99SE。

其它 PCB 设计工具可以设法转换为以上软件格式。

&3 总则

1、 描述清楚使用了 MCU 哪些片上资源；

2、 描述清楚 MCU 与外围芯片或模块之间各管脚的连接关系；

3、 有复用功能的管脚，必须通过网络命名来加以选用；

4、 有些关键字已有固定用途，参见&5 节；

5、 芯片或模块命名必须是在&6 节列表中可以找到，否则无效；

6、 网络命名必须全部使用大写字符，芯片或模块命名则不要求；

7、 所有名称不允许使用空格符号；

8、 本文档版本不断在升级，版本向上兼容，不会因升级而影响用户使用。

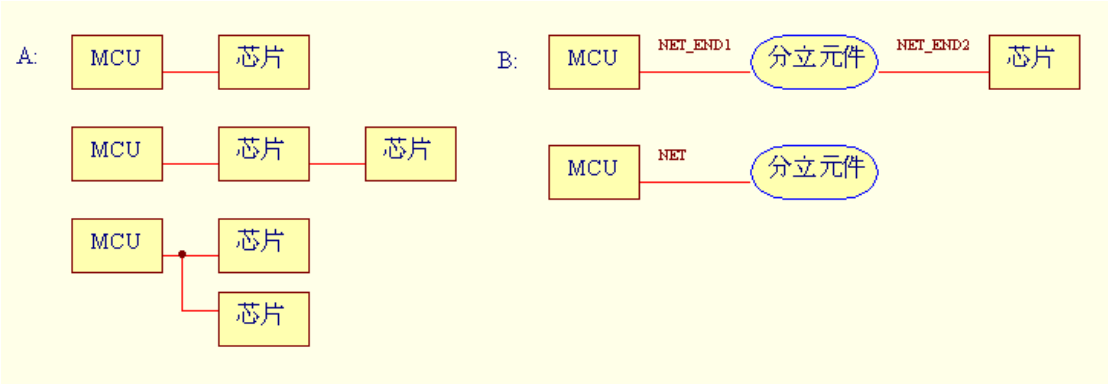
&4 MCU 与芯片或模块连接关系

一个原则就是使 AlphaMCU 能够识别 MCU 管脚在原理图中的作用。

以下 A 类几种连接情况，一般无需标注网络（使用专用网络名称、管脚复用等情况除外），AlphaMCU 可以依据外围芯片或模块的管脚定义，反推 MCU 相应管脚的设置以及使用所需的片上资源。

B 类的情况，由于分立元件电路情况比较复杂，因此要求通过网络来说清楚 MCU 管脚的功能定义。外围芯片或模块与 MCU 间接连接的管脚，需要加“END”或“REVEND”网络关键词加以说明（参见&5 节）。

其它情况可以参照以上两类情况。



&5 网络专用关键字列表

以下列出的网络命名关键字，都有特殊用途，不能把它们用作其它用途！

表一：网络专用关键字

关键词	命名	举例（备注）
分隔符	下划线“_”：用于分隔不同的关键字	INT1_UP
网络对接	ENDx(x=0,1…)：一条线路中间被元器件隔断，两边网络加上该关键字，表示它们实际是同一条网络。	例如，PA1_END1 连接 MCU，PA1_END2 连接外围芯片，这表示芯片的该管脚是与 MCU 的 PA1 管脚相连的。
MCU 主频	fosc：振荡器主频，单位 MHZ fclk：MCU 工作主频，单位 MHZ 放在 MCU 名称后面， MHZ 简写为 M。 fosc 在前，fclk 在后。两项一致时只写一项。	80C52-PDIP40_3.6864M; XX32F103C8T6_8M_72M
IDE 工具	放在 MCU 工作主频后面。 目前支持的 IDE 有： KeilC51-V955 MDK520 IAR-EW-MSP430-V6.5 IAR-EW-STM8-V2.2 ICCAVR-V7.22	XX32F103C8T6_8M_72M_KeilC51V955
MCU IO 口类型	DI：数字量输入，规定放在最前面 DO：数字量输出，规定放在最前面	DIN_PA1
数字信号类型	UP：上升沿 DN：下降沿 HI：高电平 LO：低电平	一般配合其它关键字使用。 INT1_UP
总线	DBx：数据总线 ABx：地址总线	x 为数字编号，非关键字，参见注释。 DB0，DB1，…
模拟信号	ADx：模拟输入 DAx：模拟输出	MCU 管脚必须有此功能。 AD0，AD1，…
外部中断	INTx(x=0,1…)	MCU 管脚必须有此功能。 INT0_UP
串行通讯	串口：TXDx,RXDx I2C：SDAx,SCLx SPI：MOSIx,MISOx,SPICLKx,SPICSx	对于串口，MCU 管脚必须有此功能。 TXD0,RXD0
PWM	PWMx	MCU 管脚必须有此功能。 PWM0
矩阵键盘	KIx：行信号引脚（输入） KOx：列信号引脚（输出）	KI0~3、KO0~3 表示 4X4 矩阵键盘。 如果只有一行，则只用 KIx 表示
有源蜂鸣器	BEEPx	BEEP_HI(高电平发声) BEEP_LO(低电平发声)
红外接收	IRRx(x=0,1…)	
继电器	RELAYx(x=0,1…)	RELAY_HI(高电平吸合)

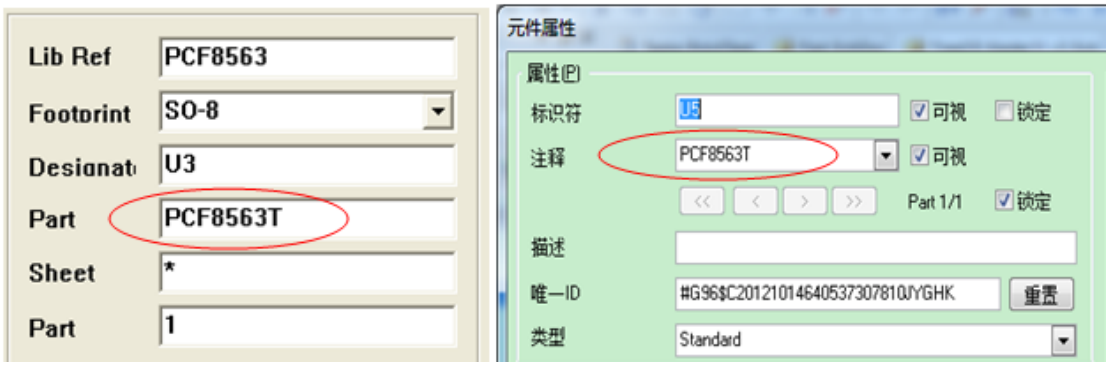
		RELAY_LO(低电平吸合)
电子开关管	SWx(x=0,1…)	SW_HI(高电平打开) SW_LO(低电平打开)

注：x(x=0,1,2…)为数字编号，非关键字。如果只有 1 路网络，x 可以不写。例如，仅有一路继电器，写 RELAY_HI 即可。

&6 芯片或模块名称列表

本表列出 AlphaMCU 目前能够识别的、需要驱动程序的芯片或模块名称。AlphaMCU 不能识别的芯片或模块将被忽略。

在 PCB 设计工具中的设置方法参见下图：



芯片或模块的型号一般与生产厂家命名一致。但为了简化，有时用一个型号代表一个系列或多种类似产品。相同功能而生产厂家不同的器件，可以用现有的器件名称替代。

注意：管脚排列次序以生产厂家提供的原理图为准!!! 本规则是以厂家原理图来判断电路网络的，这样可以省去大量复杂的网络定义。因此，描述清楚芯片或模块的封装形式非常重要！

此表正在不断扩冲中。

表二：芯片（模块）名称表

编号	芯片（模块）	型号

1	MCU	MCS51: 8051-PDIP40,8052-PDIP40
		STC: STC89C54RD+-PDIP40, STC89C54RD+-LQFP44 STC12C5A60S2-PDIP40, STC12C5A60S2-LQFP44, STC12C5A60S2-LQFP48, STC12C5A60S2-PLCC44,
		ATMEL: AT89S51-24A, AT89S51-24J, AT89S51-24P AT89S51-33A, AT89S51-33J, AT89S51-33P AT89S52-24A, AT89S52-24J, AT89S52-24P AT89S52-33A, AT89S52-33J, AT89S52-33P ATmega8L-8A, ATmega8L-8P, ATmega8L-8M, ATmega8-16A, ATmega8-16P, ATmega8-16M ATmega16L-8A, ATmega16L-8P, ATmega16L-8M, ATmega16-16A, ATmega16-16P, ATmega16-16M ATmega32L-8A, ATmega32L-8P, ATmega32L-8M, ATmega32-16A, ATmega32-16P, ATmega32-16M
		TI: MSP430F149
		ST: STM8S207RBT6 STM32F103C8T6，STM32F103RBT6,STM32F103RCT6
		MICROCHIP:
		NXP:
		RENESAS:
2	存储器	EEPROM: AT24C01,AT24C02,AT24C04, AT24C08,AT24C016,AT24C032,AT24C064 NAND FLASH: NOR FLASH:
3	实时钟	PCF8563T
4	IO 口扩展	74HC373, 74HC573，74HC374，74HC574， 74HC138
5	ADC 与 DAC	
6	有线通讯	232: MAX232,MAX3232
		485: MAX485,MAX3485
		CAN:
		USB:

7	无线通讯	2.4G： Sub 1G:
8	功率器件	
9	液晶	1602-HD44780, 2002-HD44780, 12232-SED1520, 12232-S1330, 12864-HD61202,12864-KS0108B
10	数码管	共阴： MT03641AR, SM420364 共阳：
11	红外遥控与键盘	YG3818
12	打印机	
13	语音	
14	传感器	温湿度： 18B20
15	其它	