嵌入式系统“做中学”

——《单片机应用“做中学”》续

简介：

本文主要探讨嵌入式系统学习如何能更加“易懂”、“实用”、“有趣”，如何能将嵌入式控制所需要的知识融会贯通，在工作中应用自如。

文章以四年前所写的《单片机应用“做中学”》为基础，针对当初的不足，结合技术的发展、社会需求的改变，对“做中学”的重点做了相应调整，放弃了对电子技术部分实践能力的关注，将重点放在与 MCU 和编程上，同时把知识范围从单片机应用延伸至嵌入式系统，将单片机作为理解略显深奥的嵌入式系统的工具，如同计算机原理教学中用的“模型机”。

文中推介了一个循序渐进的学习方式，并构建了一个学习阶梯：

硬件：8 位单片机 —〉Cortex-M3 —〉ARM9

软件：无操作系统编程 —〉uCOSII —〉Linux

一、背景

四年前，我写过一篇《单片机应用“做中学”》，针对我所感受到的“大学生们毕业后单片机应用能力偏弱”的问题，提出了一个设想，并且为配合之设计了一个小车平台 DIY 套件，即“圆梦小车”。

当时初入此行，对如何学习只有一些直觉的认识，且由于自己这方面的知识也是自学所得，不系统、不专业，很多想法不够全面。经过了 4 年多的摸索，对单片机应用以及其高级阶段 —— 嵌入式系统的学习有了更深的体会和认识，对当初小车平台重新审视，在那个基础上，结合客户的反馈，做了较大改进。

新的小车平台设计将嵌入式系统的要素贯穿其中，期望通过小车产生的需求，将嵌入式

系统的精髓逐一呈现。这种“呈现”是自然的、源于需求的，而非那种“如数家珍”式的陈列式展出，看者只能在惊奇之余感叹创造者之才气，无法理解“展品”之“灵气”，更无法产生“如获至宝”的兴奋；因为根本不知道这些精妙的“展品”缘何而生！最多一知半解的记住“解说员”的讲解，无法将其“为我所用”。

之所以将关注的重点从单片机应用转移到嵌入式系统，是因为随着半导体技术的发展，单片机越来越强大，特别是 ARM 公司推出 Cortex-M 系列内核后，将 32 位 MCU 引入了控制领域，促使原来 MCU 的观念发生了巨大变化。以相近的价位，控制用 MCU 也可以选择 32 位架构了。随之而来的就是复杂的功能以及无法回避的操作系统，延续传统的编程方式既费脑、费时，还很难保证可靠。因此，基于操作系统编程将成必然，学习者必须面对。

同时，随着嵌入式系统日渐强大，所谓“后 PC 时代”似乎已经到来，各类移动设备已形成巨大的商机。而物联网的热捧，无疑给嵌入式系统加了一把柴，使嵌入式系统更为火爆。作为正在学习的学生，肯定不能无视这种趋势，必须将自己的知识提升。满足于能开发简单的智能产品已无法应对未来。

二、嵌入式系统学习难在哪里？

嵌入式系统的学习相对较难，因为它最主要特征是引入了操作系统，虽说只是使用，而非设计，但没有足够的计算机专业功底很难参透其中奥妙，即便是计算机专业的学生，都对操作系统课程心存恐惧，何况学习单片机的多数是电子类、控制类专业学生。

此外，在嵌入式系统中编程和在 PC 机上不同，PC 程序的优劣无非是感觉上快慢而已，或是内存消耗大，效率低，但总能实现功能。但在嵌入式系统上，有时“慢”就意味着任务失败！而内存消耗过大或许根本无法运行，所以必须透彻的了解操作系统的工作原理，用其所长，避其所短，否则效果更差。

从我个人学习的体会观察，之所以对计算机专业和非计算机专业的学生而言，操作系统都很难，是因为操作系统核心是用精妙的算法让硬件（MCU、CPU）可靠、高效的运行，是硬件、软件的有机结合。

其目的虽是让“应用程序与硬件无关”，但其本身却和硬件有着密切的关系。虽说有驱动程序（或称为 BSP）来隔离硬件，但其工作特征仍然脱离不了硬件，驱动程序只是将琐碎的硬件操作细节代劳，但如何调度还是操作系统的事，所以才有**互斥、互锁、争抢、优先、原子操作**等概念存在。

计算机专业的学生因为硬件基础偏弱，很少基于硬件编写程序，对操作系统中由于硬件特征而产生的概念很难理解。同时，由于没有编写任务调度的体验，对于操作系统所呈现的那么“复杂”的调度算法和数据结构很难产生共鸣，甚至会认为是自寻烦恼，难以意识到其必然性和重要性，自然学得吃力。

非计算机专业的学生虽说硬件上有些优势，但算法基础太薄弱，基本无法理解操作系统中所使用的那么精巧的数据结构和算法，同样痛苦。

人理解一件事最容易的方式是“**知其然，知其所以然**”，也就是要知道此事为何而生？否则难以明白，更奢谈掌握。

所以，**了解计算机的硬件工作过程是理解操作系统的必要前提**！之后还需掌握算法和数据结构，如链表等，“**软硬兼施**”，这就是难点所在！

不幸的是，随着 PC 机的升级换代，X86 架构如今在 PC 上已“面目全非”，根本无法像 8086 时代，用 Debug 程序就可以观察到计算机的工作过程，PC 机已经成了“黑箱”。

而使用软件构建的“**模型机**”虽然很理想，但对于初学者而言，并不易理解，这种抽象出来的东西作为高手分析用的工具很有用，但作为学习者不太合适，因为计算机的工作过程本来就够抽象，再放在一个抽象平台上，必然入坠云雾。

我个人感受：在一个真实的硬件平台上，观察其内部实在的变化过程，以及对外部的作用，更加容易理解。

三、如何面对？

计算机是一门很特殊的学科，很难界定其属于理科还是工科，因为它需要数学这门纯理科知识作为手段，需要电子技术这门典型的工科课程作为支撑，同时还需要逻辑学等人文学科辅佐，以及一些生活常识作为蓝本，因为其最终的目的是帮人做事，效仿人的做事方式是其必然。

如此宽泛的内涵，使得人们不得不将其分解，结果是产生了软件、硬件两大分支，并且各自衍生出的无数子分支，根本无法纳入到一个专业中。结果就是：学习计算机的学生都成了“盲人摸象”。

能否换个思维？既然无法囊括所有知识，那就干脆放弃，另辟蹊径！

“面向对象”是计算机行业一个比较时髦的概念，其核心在于摒弃原来基于专业知识特征分类的编程方式转换为基于对象需求特征。实质上有些像制造业曾经提出过的“成组技术”、“柔性生产系统”。

早期工厂是以加工工序特征来编排车间，车、钳、刨、铣各自为阵，产品在各个工序间流转，后来发现为何不按产品的需求将这些工序整合，组成一个个生产单元，原材料进入单元后，出来基本就是成品。此方式被称之为“成组技术”。

我觉得这就是编程中“面向对象”的核心理念（但比书籍中的“面向对象”容易理解），只不过车、钳、刨、铣换成了变量、常数、数据结构、函数、方法等编程要素，将这些要素

根据需要解决的问题整合，构成一个“类（Class)”，再基于“类”构建程序。

学习嵌入式系统为何不也借助于这个理念?

学生所需要的是“在未来工作中运用嵌入式系统知识解决问题的技能”。

注意：**重点不是知识本身，而是运用知识的技能。**

因为没有人能够掌握所有知识，即便是某个特定的专业。而且也没有意义，就像我们在生活中，不会把所有未来可能会用到的东西带到身上一样，知识也是如此，学会如何针对需求找到所需的才是根本。

而学习“如何根据需求找到所需的知识”最好的方式就是自己做一件事，通过做事的过程去锻炼相应的能力，这种能力靠“听”和“看”是无法获取的。

构建一件事，从而产生一个“对象”，面向它构建一个学习的“类”，将嵌入式系统的知识整合，忽略传统的按知识分支的学习方式，基于所构建的“类”完成整个学习过程。类的成员就是学习的内容，完善类的过程就是丰富知识的过程。

知识就像商店中的物品、仓库中的货物，在摆放时需要分门别类，那是为了使用时方便获取。但作为使用者，从来不是按类取物，而是按需!

所以，作为学习者，不应该在意所学的知识属于哪一分支，“不管黑猫白猫，逮着老鼠就是好猫”。先把自己贴上标签，划归一类，应该学某些知识，无异于作茧自缚。选择自己喜欢做、想要做的事，根据这件事的需求获取知识才是正途。

在学习阶段，由于客观条件制约，无法将现实需求作为内容，那么就构建一件容易实现、易于理解、接近现实的事，它应该涵盖嵌入式系统主要的知识点，重要的是它所产生的需求和现实相近，解决问题的方式与工作要求相符。最好能具备趣味性和演示性，以促使学习者肯花时间和精力去完善它，而不是应付了事。工作与学习的差别就在于此，只不过工作中完善的动力源自责任和利益，而非趣味和表现欲。

四、我所构建的对象

嵌入式系统其特征在于“嵌入”，即将计算机嵌入到对象中，所以必须有一个对象作为载体，对象所需完成的任务就是嵌入式系统的需求。

所以，嵌入式系统学习中，一个合适的对象尤为重要。其所产生的需求应能“自然”的引入嵌入式系统的要素，使学习者在解决问题过程中自然的学习到相关知识，而不是为了验证知识而牵强附会的去做一件无前因后果的事。

结合前面的讨论，以及圆梦小车的历程，还是选择了小车作为载体，只是在设计上将重心偏向编程，略过了原来的电子技能部分。详细介绍见“圆梦小车第四代”。

因为构建一个对象需要涉及结构和电子电路，对于学习者而言，自己制作无疑要耗费精力和时间，有时还未必能如愿，所以提供一个便于使用的硬件平台有助于学习者将有限的精力用于学习。

和目前流行的“学习板形式”相比，小车的优势在于其营造了嵌入式系统的真实需求。

学习板更接近计算机，只能算是嵌入式系统中要嵌入的计算机，不是对象，也就无从营造需求。有的也是人机交互，缺少嵌入式系统必须应对的随机、实时、多任务的触发方式，也就无法展示操作系统的互斥、互锁、优先、信号、多任务等看家本领。

小车则不然，其所有控制需求都是随机的、有实时性要求的，且随着你赋予它的任务增加，所需要的操作系统功能就越多，你会为了解决某个问题在系统所提供的资源中挖掘，这样掌握的功能远比传统方式（讲述某个功能，再举例说明如何用？）透彻。

我在编写基于 uCOSII 的小车控制示例程序时，基本将 uCOSII 所具备的功能全部启用，不但如此，连所用 STM32 MCU 的硬件功能也一一唤醒。

新的小车平台将基础部分独立，包括完整的行走机构和驱动、检测，以及电源，只需增加控制器即可使其运转。

为了化解前面所讨论的难点，控制器分成三个层次：

第一层次：简单的 8 位机控制

提供的是标准 100mil 间距的试验板，PCB 上只布了电源。学习者可以选择任何一款 DIP 封装的 MCU，构建一个最小系统（一般是 2 个电容一个晶体）即可。

为何不提供现成的?

我认为：对单片机的掌握以及对计算机工作的理解，最主要的是能看懂手册，能根据需求合理分配 MCU 的资源。

如果是成品，等于代替学习者消化手册、分配资源，学习者相当于在“看例题”，而不是“解题”，效果会差很多。

这一层次主要目的是理解计算机工作原理，体验没有操作系统时的编程感觉，体会并行处理多个任务时的调度需求。

我编写过多年此类程序，在学习 uCOS 时就感到容易许多，每个系统功能都能找到以往相应的处理方式，在对比中加深理解。

因为 8 位 MCU 很简单，只有几 k 的程序空间和几百字节的 RAM，架构清晰，易于理解，外设有限且功能单一，用于观察计算机的工作过程十分合适。虽然不一定比软件构建的“模型机”全面，但由于有真实的硬件对应，以及和外界有形的交互，更易理解。

至于学软件的能否看懂单片机手册，我认为这是一个理工科学生的基本技能，如果连这个都做不到，那根本没有前途。8 位 MCU 的内容基本不需要任何基础知识，即便需要一些。借助于网络也完全能够应付。

不但要能看懂手册，而且应该是看英文的，最好不要看翻译的中文书籍，会由于翻译的不专业而误入歧途。

第二层次：32 位 ARM Cortex-M3 控制

因为 8 位机内存太小，无法尝试 RTOS（实时操作系统），所以选用 STM32F103 作为第二层次的控制器，目的在于引入相对简单的 RTOS，为嵌入式系统学习入门。

因为 STM32 只有 SMD 封装，手工焊接有难度，所以提供成品。

STM32F103 是基于 ARM 的 Cortex-M3 内核设计的 MCU，32 位，128kBytes Flash，20kBytes RAM，正好适合运行 RTOS。

目前资料最全的开源 RTOS 就是 uCOS，STM32F130 基本符合其内存要求，所以可以以此作为学习嵌入式系统的敲门砖。

有了 8 位机编程和硬件基础后，对 STM32 的理解会容易许多。因为 STM32 很多功能都是 8 位机的升级和完善。如 51 的 IO 口很简单，只有一种准双向模式，无所谓输出和输入，使用虽然方便，但驱动能力却很弱，硬件设计上就会有些麻烦。STM32 的 IO 口则可以设置多种模式，以满足不同需求，要驱动能力可以选择推挽输出，大大简化的硬件设计。定时器更是如此，STM32 增加了很多功能。但没有简单的 51 定时器的铺垫，估计很难理解，一定会因繁杂的设置而郁闷。

软件上也是同样，在 8 位机上，每一条指令都是你所编写，你要为如何在通讯时不丢失码盘采样信号而苦思，要为如何快速改变电机的驱动状态而冥想，要为如何保证主程序的运算不被中断服务破坏而费心。苦心完成后，会为自己的“聪明”而得意。

同样的需求，在 STM32 上基于 uCOS 再次去做，你会豁然开朗，原来还有这么巧妙的解决方式。只有经过上述“磨难”，才会感叹 uCOS 之精妙，也才能理解那些复杂的调度机制为何而生。

这一层次的主要目的是完成“裸奔”到基于操作系统编程的转换，由“大事小事独揽”变为“分工明确、各司其职”的编程思路。

我认为由简入繁，从相对简单的 RTOS 和硬件入手，比直接学习 Linux 之类复杂操作系统更能透彻的理解、掌握基于操作系统编程。Linux 和 uCOS 的关系就有些像 STM32 和 8 位机之间的关系，将 uCOS 作为学习 Linux 等系统的台阶应该是个不错的选择。

uCOS 我才初步尝试，Linux 还未深入，只是略作试探，发现很多机制都可以在 uCOS 中找到对应，只是 Linux 中设计的更为全面，当然也就复杂。如 Linux 的 Select 和 uCOS 的 Flag 功能一样，但 Select 捕捉的信号要全面的多，如果没有 Flag 铺垫，很难理解。

第三层次：引入 ARM9 和 Linux 作为上位机

到了这一步，应该基本从硬件学习中走出，上升到软件层面的学习了。

对于 ARM9，很少有人再去琢磨其硬件、内部资源，这些都应该是操作系统、驱动程序的事，由专门的人员去打理，作为应用程序的编写者，只需选择、使用即可。

此时，前面的硬件知识可以作为你理解 ARM 相应部分的原型，而 uCOS 可以作为理解 Linux 的原型。你所关注的应该是如何用系统所提供的功能编写应用程序，琢磨算法和数据结构，以及你感兴趣领域所需的专业技能，如机器视觉、人工智能、机器学习、自适应控制等。

小车可以移动，而且还配备了摄像头，所以上述技能都能设计出相应的项目包含之，从而使你能从简单的“让硬件动起来”，上升到基于嵌入式系统解决高层次应用的水准。

现实中很多需求抽象出实质后，都可以在小车上构建出相应的任务，通过小车平台实现，从而学习、锻炼相应的技能。

例如，机器视觉是目前比较热的领域，但单独用摄像头去学习、研究很无趣。借助于小车上的摄像头需求就很丰富了，可以做导航、避障、跟踪以及测量，基于这些需求，编写程序就不会枯燥、乏味，而且由于有了与现实的交互，很容易去评价算法的优劣，明确完善的方向。因为，每一需求都有其特殊之处，能准确把握是一个嵌入式系统工程师的必备技能。

通过小车上的象征性练习，可以锻炼这方面的能力。

如果再要提升一步，可以尝试 Web 应用编程，ARM9 核心板有 USB Host 接口，可以方便的用 WiFi 接入网络。Linux 提供了丰富的 Web 应用基础。

如果这个过程能全部走完，嵌入式系统的要素基本一网打尽。

选择上、下位机方式，继续围绕相关的需求练习，原来所做的成为新任务的基础。这种模式还有一个好处是：

通过一件连贯的事情，将学习的内容有机的整合在一起，在练习新的内容时，仍会不断触及以往所学的内容，随着新知识的掌握，会反过来加深对前面所学知识的理解，使已学的知识更加巩固。

编过程序的人都有过这样的体会，在编写新的程序时，会想到以往编过的类似程序，发现其不足，在内心中感叹：要是在现在，决不会编出那么臭的代码！可时过境迁，一般没有机会再去弥补了，只能说说而已。

但在此不同，你随时可以回去完善以往的程序，在完善的过程中得到提升，关键是这样做并非没有意义，它能使你的小车控制更加完善，不是无效劳动。人本能有惰性，无明显收获的事情一般懒于去做。

五、结语

本文所针对的是想成为嵌入式系统应用工程师的学习者，以知识的实用性为前提，特别是那些将在中小企业工作的。企业规模小，无法承受那么明确、细致的分工，所以要求工程师具备广泛的适应性，其知识需求取决于产品，而非专业。那些未来将成为科学家、教授的不在此列，他们要在某一分支精益求精。

对于多数应用工程师而言，将已有的知识和现实需求对接，是其最需要的能力，也是企业赖以生存的基础。所谓创新，多数是某项知识的恰当应用，真正革命性的创新是少数奇人的事，那应该属于创造范畴了，如晶体管、激光、计算机等。

所以，“了解知识并能把握需求需要哪些知识”是应用工程师必备的技能，这种技能源于对知识的透彻理解。

算法、数据结构、操作系统等知识本身就是从解决现实问题的过程中抽象、归纳出来的，如果学习时不将其放回到对象中，只是面对抽象的概念，只能得到一堆空洞的词汇。

我接触过一个学习嵌入式软件的学生，毕业设计做的就是基于小车的项目，在小车上构建了一个 WebServer，可以用 IE 控制。用的是 ARM9 和 Linux。因为那时我还没有接触过 Linux，在看他们程序时请教了 Select 函数的作用机制，他给我的解释基本不对，虽然这个程序已经能工作，但如果作为产品或工程中的实用程序，我决不敢信任。这就是目前教学模式的问题，学生一知半解的运用所学的概念，程序根本经不起推敲。

所以，在做中学习吧，通过需求激励掌握知识，而不是被动接受知识变成装饰。

南京嵌入之梦

2010 年 12 月 30 日星期四