讲座题目：VDK 简介

演讲人：Ken Atwell

第 1 章：导言
第 1a 分章：概述
第 1b 分章：Blackfin 操作系统的选择
第 1c 分章： VDK 简介
第 2 章：VDK 的功能
第 2a 分章：线程和优先级
第 2b 分章：关键域/禁止调度域
第 2c 分章：信号量
第 2d 分章：消息
第 3 章：在线演示
第 3a 分章：建立 VDK 项目
第 3b 分章：调试
第 4 章：时序和规模
第 4a 分章：内存空间
第 4b 分章：周期计数
第 5 章：总结
第 5a 分章：附加信息

第 1 章：导言

第 1a 分章：概述

大家好，我是 Ken Atwell，模拟器件公司的产品线经理。今天我要向大家介绍 VisualDSP 内核，更多时候也被称为 VDK。

在这个单元中，我们将讨论 VisualDSP 内核，包括它的一些基本概念和功能。如果您打算听这个单元的讲座，我们建议最好先了解一下软件开发的基本概念。如果以前接触过操作系统的概念，无论是通过使用其它商业 RTOS，还是你所在的公司曾自己开发的操作系统（现在已经开始变得不那么好用），都会对理解本单元的内容十分有帮助。你现在可以将 VDK 当作一个可能的替代品。

本单元的提纲如下：在直接介绍 VDK 之前，将先介绍一下可供 Blackfin 选择的操作系统。接下来将花一些时间来讨论 VDK 的功能，并看一下它的一些 API。随后我将转向 VisualDSP，现场向大家展示一些 VisualDSP 所提供的窗口，可以用来创建、编辑和调试以 VDK 为核心的应用程序。最后，我将通过一些测试结果来把所有的内容结合起来，其中包括对 VDK 应用程序的周期计数和所占空间的测试。

© 2006 Analog Devices, Inc.

第 1b 分章：Blackfin 操作系统的选择

首先，值得注意的是 Blackfin 处理器可以使用很多第三方的操作系统，这一页只列出了其中的一部分。这些操作系统的价格、功能和所能支持的型号都存在着差异。如果有人碰巧正在其它平台上使用这些操作系统中的一种，那么由于你的操作系统已经能支持 Blackfin，从其它平台移植到 Blackfin 的过程将变得十分简单。如果需要关于这些操作系统更详细的信息，右下方的链接提供了 URL — 可以通过它们来获取有关第三方操作系统的信息。

第 1c 分章： VDK 简介

如果这些操作系统中没有一款能很好地满足你的应用要求，那么 VisualDSP 还有一款自己的内核，被称作 VisualDSP 内核或 VDK。这是一款规模很小但却十分健壮的内核，它与 VisualDSP 一起出售，是 VisualDSP 产品的一部分。它与 VisualDSP 一起更新，因此，如果你更换到下一代 VisualDSP 或是需要对现有的 VisualDSP 进行升级或打补丁，VDK 也一样会进行相应的升级或改版。从产品维护的角度来看，使用 VDK 将十分方便，它紧跟工具链更新的步伐。使用 VDK 也不会带来附加成本，因为在购买或使用 VDK 时，都无需支付许可证费用或使用费，并且完全不需要支付专利使用费。VDK 能支持所有现有，以及将来可能出现的 Blackfin 处理器。因此，如果你打算，或你的应用需要换用其它 Blackfin 系列的产品，就可以考虑使用 VDK，因为当你换用不同的 Blackfin 系列的处理器时，API 都是兼容的。

最后，VisualDSP 内核能够与 System Services 互补使用。在这个网站上还有其它 BOLD 培训课题，来详细介绍 System Services 和它的 Device Driver 模型的功能。System Services 的设计目的是帮助 VDK 内核而不是与其竞争。

让我们继续来讨论一些概念，其中一些与 VDK 相关。这些概念包括线程、优先级和时序调度。VDK 提供了三种类型的时序调度。分别为抢占式、协同式和时间片也称为轮转调度式的方法。我们还将讨论关键域和禁止调度域，以及系统中两种协调通信的方法，也就是信号量，包括周期性信号量和消息。

除今天所讨论的内容之外，还有一些来不及讨论的功能，如果大家喜欢这些功能，可以参考详细的说明文档。这些功能包括事件和事件位，它与信号量类似，但有更复杂的后台判决。多处理器消息传递，如果你在一块电路板上使用多个 Blackfin，或使用了像 Blackfin 561 处理器这样的双核解决方案，则可以使用 MP 消息传递功能在处理器内核之间或处理器之间传递消息。存储器组合，有点类似于低分片堆（low fragmentation heap）。设备标志是设备驱动器传递状态信息或特定事件返回应用程序的途径。所有这些功能都在 VisualDSP 帮助和 PDF 手册里有详细的说明，如果需要更详细的信息，可以进行查阅。

第 2 章：VDK 的功能

第 2a 分章：线程和优先级

让我们继续来讨论一下 VDK 的功能。线程和优先级，Blackfin 处理器有 31 个优先级，任何线程都能够在这些优先级上运行。在任意时刻，系统上能够运行任意多个线程（可以多于 31 个），实际

上在很多时候，应用程序在某个优先级上都会有不止一个线程在运行 。对不同优先级的调度十分简单，完全由优先权决定，并且强制操作系统执行。也就是说，能够运行且具有最高优先级的线程将一直完全占据处理器，直到发生一些情况迫使这一线程被搁置。

当多个线程运行于同一个优先级上时，判决会变得更复杂，也更有趣。在这张图中，我用右边的第二组矩形条来表示。我将它所显示的内容称作“协同式多处理或多任务”。在这里，有两个互相协同工作的线程，其中的一个线程将一直完全占据处理器直到任务完成。此时它会特意地将应用程序的控制权交给另一个线程。也就是说，这是协同的工作方式，两个线程之间是协同或合作的关系。处理器的控制权在两个线程之间来回传递，使得处理器工作在两个不同的线程之间。

这张图的下方所显示的是被我称为“时间片”或“轮转调度式”的优先级。用过像 Windows 和 Unix 这样的操作系统的人会对它比较熟悉。使用这种优先级处理时，每个应用程序只占用很短的 CPU 时间，用户几乎无法察觉它们在进行轮换。操作系统或 VDK 会自动地将操作系统的控制权以轮转调度或时间片的方式在所有线程之间进行传递。每个线程所得到的处理器控制时间的长度由程序员定义。优先级可以被静态地分配，也可以被动态地分配。静态分配意味着应用程序在创建时就已经被指定好了优先级。动态分配则意味着程序的优先级在其运行时仍能被改变，也就是说在线程实体化时或运行时，其优先级都能被改变。

下面还有一些关于线程的信息。线程能够在系统被引导时进行实体化，也能在随后需要运行时再实体化。实际上在系统被引导时，必须有一个线程被实体化来使得系统能够运行，否则系统将无事可做。VisualDSP 会强制创建至少一个引导线程，之后你可以随意地创建任意多个线程。当然，实际上你所能创建的线程数量会受到你的系统所能提供的内存容量的限制，但 VDK 本身对其没有任何限制。

每个线程都有自己的堆栈，用来供 C 编译器放置本地变量和调用堆栈。因此，你不必担心会出现两个线程意外地共享了同一个本地变量或类似的情况。它们都会有自己的堆栈拷贝，它们自己的堆栈。从最简单的层次上来看，每个线程就是实现了四个函数 — 创建函数、消除函数、运行函数和错误函数。如果大家精通 C++，可以把创建和消除函数分别理解为 C++中的构造器和解构器。也就是说，它们通常是一些小函数，在创建线程时完成一些静态初始化工作。类似地，消除函数或解构器用来完成一些消除线程所必须的少量清除工作。大部分时间花费在执行运行函数上。实际上运行函数通常是不会返回的，它由“while(1) ”循环或一些其它类型的无限循环来实现，这样的循环是永远无法跳出的。如果运行函数一旦退出，线程就会被自动消除。

在右边，你能够看到一张使用的 API 的不完整列表。其中的一些会被经常用到，如清除线程或创建线程。其余的 API，特别是用来处理错误的那些，是专门用来进行调试的，在最终的应用程序中可能没有太大的用处。值得指出的是列在这一页上的最后两个 API，Sleep()和 Yield()，能够使应用程序在一段特定的时间内休眠，随后 VDK 会自动将其唤醒，并使其继续运行。Yield()还能被用在协同多任务的情况下，像我前面所说的那样，此情况下可能会有两个线程，并且处理器的控制权会在它们之间来回传递。

© 2006 Analog Devices, Inc.

第 2b 分章：关键域/禁止调度域

关键域和禁止调度域能被用来完成一些少量的工作，这些工作非常重要，并且不能被中断。使用过 RTOS 的人一定遇到过这样的情况，你需要完成一些小操作，它们非常关键以至于在执行期间操作系统必须被隔离。也就是说在关键时期内，操作系统不能再响应其它的任务。VDK 提供了关键域和禁止调度域来完成这些操作。

关键域所采取的行动十分极端，它们简单地关闭调度程序和所有的中断，从而让当前进程取得处理器全部的控制权，而不管其具有什么样的优先级，也不管系统当时正在做什么。当你需要完成一些非常小并且不能被中断的工作时，采用这一方法十分方便。但是这一方法在使用时必须十分小心，因为如果让处理器在关键域停留太长的时间，将无法执行中断。其它一些副作用，像中断延时，会随着在关键域停留时间的增加而等量增加。通常，此功能被用于测试和设置或读取—修改—写入操作，例如你的应用程序正在查询一个全局变量值，根据它来做出判断，然后再改变这个全局变量值。如果这一操作十分重要，并且在这段时间内系统不能被中断，那么关键域可能是你可以使用的一个好方法。

禁止调度域所采取的行动就不那么极端，但它们之间是相互联系的。它所做的仅仅是禁止线程切换，因此本质上 VDK 将做出让步，不允许进行时序调度，但中断还能够被执行。在右边能看到一组用于关键域和禁止调度域的 API，它们十分简单。注意它们使用的是堆栈形式的操作，你需要将关键域推入堆栈，并将其从堆栈中取出。使用堆栈形式的操作是因为，如果你正在编写一段需要进入关键域的代码，则这段代码不必去关心它自己是否已经处于一个关键域。你只需简单地让关键域出栈、压栈。完成时让关键区出栈，而不必关心当前这个函数的调用者自己是否也处于一个关键域中。

第 2c 分章：信号量

信号量：信号量可能是在线程之间进行同步的最简单方法。本质上，它是线程之间或一个线程的 ISR 之间的一种协同同步的方法。信号量通常被用来控制对一些共享资源的访问。经典的例子是我可能有一个音频信号的缓存，一个线程正在对它写入，而另一个线程同时在对它进行读取，我要确保下一个音频缓存不会覆盖这个正在被另一个线程读出的缓存。因此，我需要一个信号量来控制对缓存的访问。正在读取音频信号缓存的线程会提示，“在这段时间内请不要进行写操作。”接着要对缓存进行写操作的线程会遵守这个提示，一直等待直到信号量被清除。这样的话，两个不同的进程之间就不会出现争着访问同一块缓存或外设或其它资源的情况。

大多数信号量是布尔类型的，只能表示是或否这两种情况。在我给的缓存的例子中，它只需表示能被访问或不能被访问这两种情况。但可能会出现这样的情况，一个共享资源能同时被不止一个线程访问，可能是两个或三个，或更多。在这种情况下就需要使用计数信号量，它的实现方式在本质上与一个布尔信号量相同，只是共享计数的值比 1 大。最后，值得注意的是信号量可以被设置成周期性的。那样操作系统就会在由应用程序定义的一段时间内，自动发布一个信号量。如果你有一个背景动画，希望在一秒钟内对其更新 24 次，或进行其它诸如此类的操作，你就可以使用一个周期信

© 2006 Analog Devices, Inc.

号量来实现。可以每隔 1/24 秒发布一个信号量来更新动画，而不必通过像让一个线程休眠这样不恰当的方法来实现这一功能。在右边是一组用于信号量的 API，像我说的那样，它们的创建和消除都十分简单。你也可以静态地创建信号量，也就是在创建应用程序的过程中创建信号量，稍后我们将关注一下这个问题。

第 2d 分章：消息

消息是在线程之间进行协作的一种更复杂的方法。消息是从一个线程到另一个线程有目标的信息传送。因此，本质上会有一个发送线程和一个接收线程。发送线程将消息按某种格式进行封装，然后传递给接收线程，接着接收线程会将消息解封，并按照应用程序的要求来进行操作。这一过程是通过空指针来实现的。如果你熟悉 C 语言，会发现其类似于 C 语言中的后门。你能够将空指针指向任意类型任意长度的数据，这样就能在系统中随意地传递数据。还有一个非常方便的地方 — 本质上你能够在线程间传递任何东西。这一切都是假设接收端的接收者已经对消息的内容有所了解，因此接收者必须能够将数据再映射回到正确的数据类型上。

虽然下面的内容已经超出了今天讨论的范围，但我还是要简单地提一下。当熟练掌握了消息传递的 API 后，在多核心和多处理器之间使用消息传递是十分方便的，你能够通过一个串行端口或通过像 Blackfin 561 这样的双核系统中的共享缓存，来传递消息。

在右边是一组用作消息传递的 API，使用十分方便，你看有 API 专门用来获取线程所接收到的消息的负载和一些其它类型的信息。

第 2 章：在线演示

第 3a 分章：建立 VDK 工程

下面我将返回 VisualDSP 本身。我将用几分钟时间，来讨论一些 VisualDSP 所提供的窗口。我们先来看一看项目创建窗口，本质上它被用来设置一些系统运行所需的线程和你可能会做的一些创建时判决。接着来看一看线程创建模板。如果你需要创建一个新的线程，我们来看看 VisualDSP 能够帮你做些什么。最后再看一下两个调试窗口。一个是状态窗口，从中能够查看应用程序的当前状态，另一个是历史窗口，从中能够查看应用程序中最近发生的 N 个事件。

这里，我已经将 VisualDSP 自带的项目窗口最大化。如果你曾使用过集成编辑器，很可能以前就看到过这样的窗口。这是我在应用程序中所使用的一组基本的源文件。值得注意的是，底部这里是内核的标签。如果你已经创建了一个 VDK 应用程序，你将会自动获得这个内核标签。当我点击这里时，会给出我的 VDK 设置。让我们花点时间来浏览一下这些设置。我的系统，我能够知道系统运行的速度。在这里我能够控制历史窗口的大小，在有可用的内存的前提下，我可以按自己的需要来确定窗口的大小。还有测试设备的选项。测试设备是 VDK 放入 API 中的错误检查代码，用来在开发时捕捉错误。通常当你发布应用程序时，要把测试设备全部关闭。

如果需要创建或查看线程，可以在这里进行操作。再次强调一下，这些不是运行系统的线程，而是我在应用程序内定义的线程。因此，在这里可以有不同的线程类型，我在这个应用程序里创建了 4

© 2006 Analog Devices, Inc.

种线程类型，这个应用程序的具体功能并不重要，我只是想在这里演示一下 VDK 的使用。每个线程都有一个优先级。在这个实例中，引导线程无疑具有最高优先级，优先级 1 代表最高优先级。我还能设置其它一些与线程性质相关的参数。堆栈大小，是一个重要的参数，我曾说过每个线程都有自己专用的 C 语言堆栈，我能够使用这个参数来定义这个堆栈的大小。我可以这样做：在这里创建一个任意多个线程类型。

引导线程 — 像我前面所提到的那样，你必须有至少一个引导线程类型，或引导线程。在这里我已经有了这样的线程。我已经简单地给它取名为 BootThread，并且在这里做了申明。轮转调度式优先权，任意一个优先级都可以是协同式或轮转调度式的。在这个应用程序中，除了优先级 6 是轮转调度式的，其余的优先级都被设置为协同式的。轮转调度的周期是 10，即 10 个节拍，可以在这里的系统设置选项中设定节拍。也就是说，处于优先级 6 的线程，在转交控制权之前，操作系统需要等待 10 个节拍。

前面所讨论过的信号量在这里定义。在本实例的应用程序中，最多只能有 5 个信号量同时运行。实际上我已经在这里定义了四个静态的信号量。我也可以在程序运行时定义，但在本实例中，在这里静态地定义信号量会更加方便。这里有一项 UARTAvailableSemaphore，初始时默认为正确 — 可用的 — 并且它的最大计数值是 1。如果我要将计数值设为 4，使其不再是一个简单的布尔值（是/否），我可以通过指定那个值来改变这个值。如果我要进行周期设置，可以这样做，当然也可以静态地进行设置。

在这里的下方，能看到一些关于更高级特性的设置，虽然我们今天没有对它们进行讨论，但本质上它们的创建与我们刚才看到的一些工程中的操作是十分类似的。

让我们来看一看如何创建一个新的线程。如果我要创建一个新的线程，我只需要在那里点击一下鼠标右键，我要创建一个新的线程类型，给它取名为“NewThreadType”。在这个实例中，我可以选择我想使用的语言，我将选择 C 语言。需要让更新自动存档吗？是的，在本实例中我需要，这样就创建了一个新线程。实际上，会创建 NewThreadType.C 和.H 这两个源文件。如果我在这里双击一下，就会进入这个线程，你能看到这就是 VDK 或 VisualDSP 环境为我所创建的代码。我在前面曾提到过，在任意一个线程中都有四个最基本的函数，它们在这里。这个是运行函数，这些是错误函数、创建和消除函数、构造器和解构器，以及更多的 C++风格的用法。所有这些都是我来创建的。我在前面提到过在许多应用程序中，运行函数永远不会停止，实际上那是在这里所做的一个假设。事实是，这个运行函数中的 while (1)循环已经被添加进去。当然如果不需要这一行为，我只需将其删除即可。

第 3b 分章：调试

让我们假设已经完成了应用程序的开发，现在正在运行和调试。我已经让这个程序运行了大约 30 秒，因此有了足够的历史和状态的信息。让我们来看看这些信息。首先，看一下状态窗口。这里是一张快照。如果我想使处理器暂时停止工作，只需按一下暂停按钮，此时没有特别的事件发生。这是一张目前系统状态的快照。让我们来看一下。能够在上面找到 10 个线程和 4 个信号量。我还能

© 2006 Analog Devices, Inc.

够进一步深入以得到更多的信息，在本实例中，让我稍微调整一下。我还有一些线程用来控制连接在硬件电路板上 LED 的闪烁。我还有一个音频线程和几个其它的线程，它们在本质上不受控制，占用了全部 CPU 时间。我还能得到关于每个线程更多的信息，这个音频线程或许是最有趣的，你可以在这里看见它的状态，它被信号量 ID 3 阻止了。记住这一点，因为后面我们还将看到它。我可以将这个打开，就能够得到各种信息，例如实际上使用了多少个堆栈，这些堆栈的位置、优先级，以及它们已经运行了多久，等等。如果遇到某一类型的错误，将会在这里下面的最近错误类型和最近错误值中给出报告。我们看见它处于被信号量 ID 3 阻止的情况。因此如果我们看一下信号量，就应该在那里能看见一些匹配的信息。事实上我们确实看见了。这些是我曾创建的信号量，这个是信号量 ID 3，它被称作 kBuffer2 可用。如果我将那个打开，就会看见有一个被挂起的线程，线程 ID 10。这就反映了我们刚在几秒钟之前所看到的东西。因此，你既可以从以信号量为中心的角度，也可以从以线程为中心的角度，来考察你的应用程序。如果在这个应用程序中，我使用了消息（实际上我没有使用），我同样可以做这样的关联。

最后一个值得一看的调试窗口是状态历史窗口。这是一张状态历史窗口的图示。同样地，应用程序的细节是不重要的，图中很好地用颜色进行了编码。如果你是第一次使用，可能需要打开图例来了解这些颜色所代表的含义。由于在这里受到分辨率的限制，我暂时将图例关闭了。这里显示了所有的线程，来回移动的绿线代表控制权在线程之间的传递。橙色的剑号和所有其它这些符号显示了应用程序中的信息量或历史。如果我想查看这一块的活动，我只需拖动指针将这里放大即可，或许我需要将图放得更大来仔细查看，现在我已经找到一组有用的信息的子集。在这点上，我可能想看一下这些单独的事件。如果我点击鼠标右键，并选择“数据指针”，现在就可以使用键盘上的左、右箭头在应用程序的历史信息中移动。如果将注意力集中到窗口的左下角，实际上显示了更多关于我正在寻找的实际事件的信息。这样我就能够追踪应用程序的一切活动。如果显示的内容过于杂乱，而我又希望专注于所能提供的活动类型的一个子集，我可以在这里点击鼠标右键，来使用过滤器。我可以取消对于不是十分感兴趣的事件的选定，或者如果我只想查看比如说仅仅两个线程间的相互作用，可以在这里进行过滤。

© 2006 Analog Devices, Inc.

第 4 章：时序和规模

第 4a 分章：内存占用

我现在打算再返回到已经准备好的材料上。我们将集中讨论一些关于规模和时序的信息。首先，我们经常被问到的一个问题是，“VDK 的资源开销是多大?使用 VDK，我需要在代码规模或性能上付出多大的代价?”这是一个非常难回答的问题，因为问题的答案与你的应用性质密切相关。因此，我在这里所做的，只是把一些我觉得对于一般的应用具有代表性的数字列在了一起，但对于一个特定的应用，答案还是会随着应用要求的不同而变化。

首先，我列出了一些静态的内存占用。这是在这些场景中，当链接时，VDK 所消耗的代码和数据的总量。这只是 VDK 所消耗的，而不是整个应用程序，仅仅是可执行的程序中 VDK 的那部分所消耗的。这是使用 VisualDSP 4.5 测得的，已经去除了无用的代码和数据。所有这些大小都以字节为单位。如果我们看这张表中的第一行，这是一个人为制造的最佳场景，我只使用了一个 C 语言的线程，并且根本没有调用 API。这种情况下我需要 5KB 的代码和 1KB 的数据。这样你就对 VDK 总体的资源开销有了一些概念。向下看，在接下来的这几行里，我开始加入越来越多的功能。如果我有两个线程，并且在它们之间使用一个静态信号量来进行协作，我将需要大约 7KB 的代码，但数据量基本上保持不变，和开始的那行相比只增加了 10%左右。如果我使用消息传递功能，将需要大约 9KB 的代码。在最后一行里，代码规模急剧增加，这是一个真实的调试场景。在一个发布后的应用程序中，你不会遇到这样的情况，但当你在调试时，想要打开所有的测试设备，并需要一个历史窗口来使用那些我们几分钟前刚看到的重要特性，则代码的规模将增大到大约 13KB，数据量也增大到接近 10KB。值得注意的是，共有 512 个事件，用它们乘以 16 字节/事件，如果做一下简单的计算，就会发现仅仅历史窗口就带来 8KB 数据，占了数据空间的绝大部分。这些数据量与具体的应用相关，你可以按照调试任务的需求，将数据量减小或增大。

第 4b 分章：周期计数

最后，我想讨论一下 VDK 对于特定事件的周期计数，它对于性能十分敏感。首先说明一下我的运行环境，这里所有的应用程序都装载在 L1 中。如果你某些部分的应用程序装载在 L3 中，那么性能可能会变差，但如果你能恰当地管理缓存，你得到的周期数将与这里所得到的周期数十分相近。这里所使用的应用程序实际上与我在前面所展示的表中所使用的很相似，其中有 5 个线程运行在一些不同的优先级上。我所使用的处理器是 Blackfin 533，硅版本 0.5。引导时间需要 15,000 个周期，如果想要将结果转换成真实的时间，你知道自己的处理器的运行频率，你可以自己做一下计算，来得到以毫秒或微秒为单位的值。节拍只占用了 67 个周期，如果我不改变线程的位置的话。但在我处于时间片的情况下，我必须改变到另一个线程，这就需要长达 722 个周期。发布信号量的情况与之相似，如果发布一个信号量，并且执行没有发生改变，只需要 76 个周期。发布一个信号量，而执行出现了改变，就需要接近 300 个周期。关键域，也就是将关键域压栈，增加一个全局变量，再使其出栈的行动，总共需要占用大约 200 个周期。最后，如果我创建一个新的线程，这里我在堆上使用 malloc()来创建，共需要 2300 个周期。

第 5 章：总结

第 5a 分章：附加信息

© 2006 Analog Devices, Inc.

总而言之，使用 VDK 不会增加 VisualDSP 的成本，并且不需要支付专利使用费和附加的维护费用。但不要忘记 Blackfin 也可以使用许多商业操作系统。今天我们看到了 VDK 的许多便利的功能，其中包括线程、优先级、信号量、消息传递、关键域和禁止调度域。我在前面还提到过，VDK 还提供其它一些便利的功能，只是由于时间的关系，我们今天无法讨论，但在 VisualDSP 的文档中对它们进行了详细的讨论，如果你想学习这些功能，可以参考 VisualDSP 的文档。

像我们在现场演示中所看到的那样，VDK 被很好地集成到了 VisualDSP 中。项目的静态设置，如我们所创建的线程和信号量，管理起来十分容易。当创建一个新的线程类型时，VisualDSP 还提供了模板，模板给出了代码的框架，因此我可以立即对其进行编译和连接。在调试方面，提供了一个状态窗口和一个历史窗口，在本实例中，状态窗口使我对应用程序的状态有了一个详细的了解，接着使用历史窗口查看应用程序最近的历史。

欲获取任何附加信息，可浏览其它 BOLD 主题，特别是 System Services，有一些培训课程是基于 System Services 及其驱动模型的，它能很好地为 VDK 提供帮助。VisualDSP 可以试用，提供了一个 90 天的免费评估版本。你能够用很低的花费得到一个 EZ-KIT Lite。你可以从下面的 URL 得到试用版本。你也能在 Windows Explorer 中打开这个地址来得到 VDK 的实例。值得一提的是，绝大多数演示 VDK 功能的实例都没有硬件要求，因此如果你还没有准备购买 EZ-KIT Lite，免费的试用版本将使你能了解到 90% VDK 所提供的功能。

VisualDSP 提供了详细的文档，当你安装完试用版本时，在线帮助系统会显示这些文档，文档还能以 PDF 形式被独立下载，此外与 VisualDSP 一起出售的 CD 上也会有这些文档

最后，再次提醒您看一下可供使用的第三方操作系统。从前面的幻灯片上你已经看到有大量不同价格、不同功能的产品。可以通过这个 URL 来获取更多的信息。

最后，在你们的屏幕上有一个“询问问题”的按钮。如果点击这个按钮，就能直接向模拟器件公司发送问题。

感谢大家花时间来参加这个讲座，我是 Ken Atwell，模拟器件公司的产品线经理。感谢参加这个关于 VisualDSP 内核的讲座，希望工作开心，再次表示感谢。

© 2006 Analog Devices, Inc.