

中庸之道

高速8位微控制器的用武之地

个人电脑的软件开发者与嵌入式系统的开发者相比，有许多优越性。不仅是因为他们所面对的系统具有和几年前的超级计算机相媲美的处理能力和存储容量，而且，这些系统通常都是现成的。相比之下，嵌入式系统开发者所面对的对象，不仅是规模小得多的系统，而且他们通常不得不先完成系统设计。

嵌入式系统方案的选择取决于课题的规模。如果该课题是一个相对简单的设计，只涉及到少量的用户交互操作，并控制少量的外设，那么可以选用处理能力较低的8位微控制器，如8051、68HC11、Atmel® AVR®，或是PIC以及它们的变种。这些解决方案通常可以提供足够的能力和灵活性。如果该课题需要相当多的用户交互操作，需要通过以太网通信，或者需要与像数码相机那样的复杂外设通信，那么通常选用PC-104、StrongARM，或是其他类型的“一卡路里个人电脑”。这些解决方案通常能够提供充裕的处理能力、复杂的操作系统，以及大量的RAM。

然而，在这两种解决方案之间还有一片灰色地带。让我们来打个比方，比如说Joe的Security Service想要进军竞争激烈的网络门锁市场，由于近来高度警戒的安全状况，各公司都希望安装的门锁不仅仅依靠用户的ID号码来控制用户的出入。他们希望有这样一种电子门锁，当用户想要开门时，可以拍下用户的面部照片或拇指的指纹。这些图像通过网络传输到中央服务器用于记录，或者更复杂一些，用于图像识别与确认。如果图像得到确认，服务器则对网络门锁作出响应，门便为用户打开。Joe希望他的客户为自己所有的设施都安装有电子门锁的门，因此，将成本控制在较低水平就显得非常重要了。

Joe的竞争者之一，Alex的Security Central，正在研制一种网络相机门锁，采用的方案是8位RISC与以太网控制器相连。Joe认为该方案能力不足而没有采纳。他熟悉许多类似的项目，都采用了这些

Harvard结构的芯片与以太网控制器相连的方案。然而，这些项目中的大部分都远未成熟，没有一个获得商业支持，而且，TCP/IP栈也受到了结构本身的限制。如果说这种方案应用在网络通信方面还勉强合格的话，那么用于与数码相机通信时就无法胜任了。一幅能满足要求的图像需要40kB至60kB的存储空间，还不包括与之竞争的程序存储空间。即便Joe采用非Harvard结构，对于传统的8位微控制器来说，需要处理的工作和数据的量还是太大了。

Joe最主要的对手，Troy的X-Treme Security，也在研制其解决方案。由于Troy对嵌入式系统设计的艺术与精巧毫不关心，所以Joe不喜欢Troy。从浪漫的角度来比喻，我们可以打个比方说，Troy正在与Joe的前女友Amiga约会，Amiga离开了Joe，因为Joe在计算机上花了太多的时间（这被称作嵌入式系统开发者的“职业病”）。Troy研制的方案采用的是Pocket PC使用的StrongARM，它具有高速的I/O以及网络通信能力。Joe则认为这种方案是杀鸡用牛刀。除了拍照的时候，处理器在大部分时间都处于空闲状态。对于这个课题，真正理想的解决方案并不需要大量的存储空间以及很强的处理能力，运行嵌入式Linux或Pocket PC会使系统过于臃肿，对这样一个简单的装置来说，成本太高了。

Joe真正需要的只是一款足以应付网络和相机的微控制器，比32位方案便宜得多，并且功能少得多。如果该微控制器支持比纯汇编高级的语言来简化开发过程的话，对Joe就有极大的帮助。Joe将如何从性能上战胜Alex，从价格上战胜Troy，而赢回他的真爱呢？

TINI®出场了。

通向网络的桥梁

TINI，或微型因特网接口，是Dallas Semiconductor的产品。TINI平台被设计用来作为连接网络的桥梁：PC机可以通过TCP/IP与TINI通信，TINI可以与传感器、传统硬件或其它设备通信。TINI不仅提供大量外部接口，包括1-Wire®、2线、RS-232串口、CAN以及SPI™，而且具有强大的网络工具，能够为IPv4、IPv6、DNS、DHCP、PPP、Telnet以及FTP提供支持。

Atmel和AVR是Atmel Corp.的注册商标。

TINI和1-Wire是Dallas Semiconductor的注册商标。

SPI是Motorola, Inc.的一个商标。

目前有两种TINI参考设计。最常见的是基于DS80C390的TINIm390验证模块。TINIm390是72引脚的SIMM，包括512kB闪存、电池备份的512kB或1MB SRAM、以太网控制器和实时时钟。基于DS80C400的TINIm400验证模块是更新的版本。TINIm400是144引脚SO DIMM，除了以太网MAC已经是DS80C400的组成部分以外，TINIm400具有与TINIm390相近的结构。

390和400都是8位微控制器，实际上都采用了8051微控制器核。然而，它们的性能已经大大扩展了。首先，它们的内核每个机器周期由4个时钟周期组成，而不是标准的12个时钟周期。在相同的时钟频率下，其速度比标准的8051速度提高了3倍。其次，它们具有更大的寻址空间。390支持4MB程序存储区和4MB数据存储区，而400支持连续的16MB地址空间。再次，它们支持更高的时钟频率。390支持的最高时钟频率达到40MHz，而400可以达到75MHz。最后，它们具有针对乘法和除法运算的整型数学加速器。总之，它们提供了处于传统8位微控制器与16位/32位微控制器之间的中等处理能力。

TINI平台的一个独特之处是由Dallas开发的操作系统，它是无版权、多任务、多线程，且拥有Java™运行环境的操作系统，可以通过免费下载获得。512kB的闪存中可以容纳下核心OS和库，在闪存最后一个存储区中还有足够的空间用来存储64kB的应用程序。DS80C400中还包含存放在ROM中的C语言和汇编语言程序库。

网络相机

为了说明网络相机这个课题可能的解决方案，使用TINI来实现一个流网络相机，并作为性能的基准。实际上使用的是用户化的高速DS80C400设计(图1)，而不是TINIm400参考设计。这里讨论的网络相机拍下原始图像，并通过UDP将图像传输到PC主机，与PC间的通信既可以使用主机端的软件实现，也可以通过HTTP运行Java applet来实现。

选用的相机是使用OmniVision 5017 CMOS芯片的M4088模块。该相机是逐行扫描，384 x 288像素

Java 是Sun Microsystems的一个商标。

Intel 是Intel Corporation的一个注册商标。

向Maxim Integrated Products, Inc.或其从属许可名义下的相关公司购买P2C元件，将传递Philips P2C专利许可，允许这些元件用于P2C系统，如果该系统符合Philips定义的P2C标准规范的话。

分辨率的黑白数码相机，提供8根数据线、4根地址线以及片选线，可以方便地使其工作在存储器映像方式下。该相机还提供了一条场同步信号线，在摄取到一帧图像时发出信号；一条行同步信号线，每扫描一行后发出信号；像素时钟，在每个像素到达时发出信号。一旦相机被初始化，就可以很容易地利用软件来访问相机。5017具有内部时钟分频器，可以通过程序来控制帧频；还具有单帧模式，允许在主机的控制下摄取每一帧图像。这种模式非常有用，因为该设计中的处理器能力不足以处理相机的最高速度50帧/秒。

相机的400版本被设计工作在73MHz (18.4MHz x 4)下。400具有内置的以太网MAC，但是需要以太网PHY和磁性元件。400支持许多其它的PHY，比如HomePNA和HomePlug PHY。在该范例中使用Intel® LXT972A通过标准MII接口与400直接连接(图2)，但是需要为该PHY提供25MHz时钟。

在运行中该相机需要12ns存储器。这里使用了Hitachi的HM62W8511H，可以提供足够的访问时间。启动时，处理器执行闪存中的自引导程序，将执行文件的映像从闪存复制到SRAM中，将时钟的四倍频器标志为使能，并跳转到TINI起始地址。由于高速SRAM会使电池消耗太快，因此在该配置中电路板上没有TINIm400中的电池备份的或非易失的存储器。这意味着TINIOS不具备永久的文件系统，但对这个课题来说影响不大，因为TINI将在启动时创建文件系统，下文中还将提到。

TINIOS需要DS2502来存储MAC地址。当不需要MAC地址时，还有一个通过I²C™总线连接的DS1672实时时钟。这就使得TINI不仅可以访问软

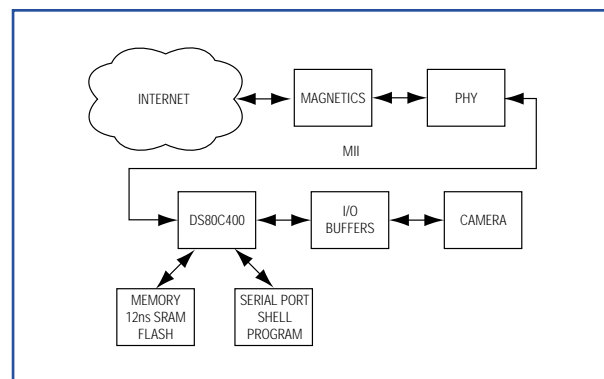


图1. 高速DS80C400设计示意图。

件时钟，而且提供了附加的优点。如果启动时检测到了实时时钟，TINIOS将自动计算系统总线速度，并对内部定时器作出相应的调整，其中包括串口波特率、定时器标记以及网络定时。

相机的连接非常简单：A0-A3，D0-D8以及WEB分别接到相应的信号线上。CE4与相机的CSB相连，使相机映射到0x800000。相机的VSYNC与P1.1相连用于读取，将PSEN与相机的OEB相连。与INT1连接之前必须先将HREF取反，这样就可以使用电平触发方式的中断了。

软件实现

软件开发中要使用到TINI SDK，这可以从Dallas Semiconductor网站上免费下载。有人可能认为Java虚拟机不具备充分的能力来以足够快的速度采集相机数据。然而使用本地方法，TINI允许中

断服务子程序被安装在与应用程序通信的操作系统之下。这正好发挥了TINI平台的强项之一——高层协议(如HTTP)用纯Java实现，而代码中的高速部分用8051本地汇编实现。很大程度上讲，这是两个领域的最佳方案。

TINI支持JDK 1.1中的以下程序包：

```
java.io
java.lang
java.net
java.util
javax.comm
```

TINI中的JVM与PC中的JVM确实存在一些差别。首先，虽然TINI支持无用单元收集，但是不支持终止。这意味着编程者需要明确地关闭使用过的资源，而不是等待系统来完成，不过这种资源管理方式对嵌入式系统开发是有利的。TINI对类的

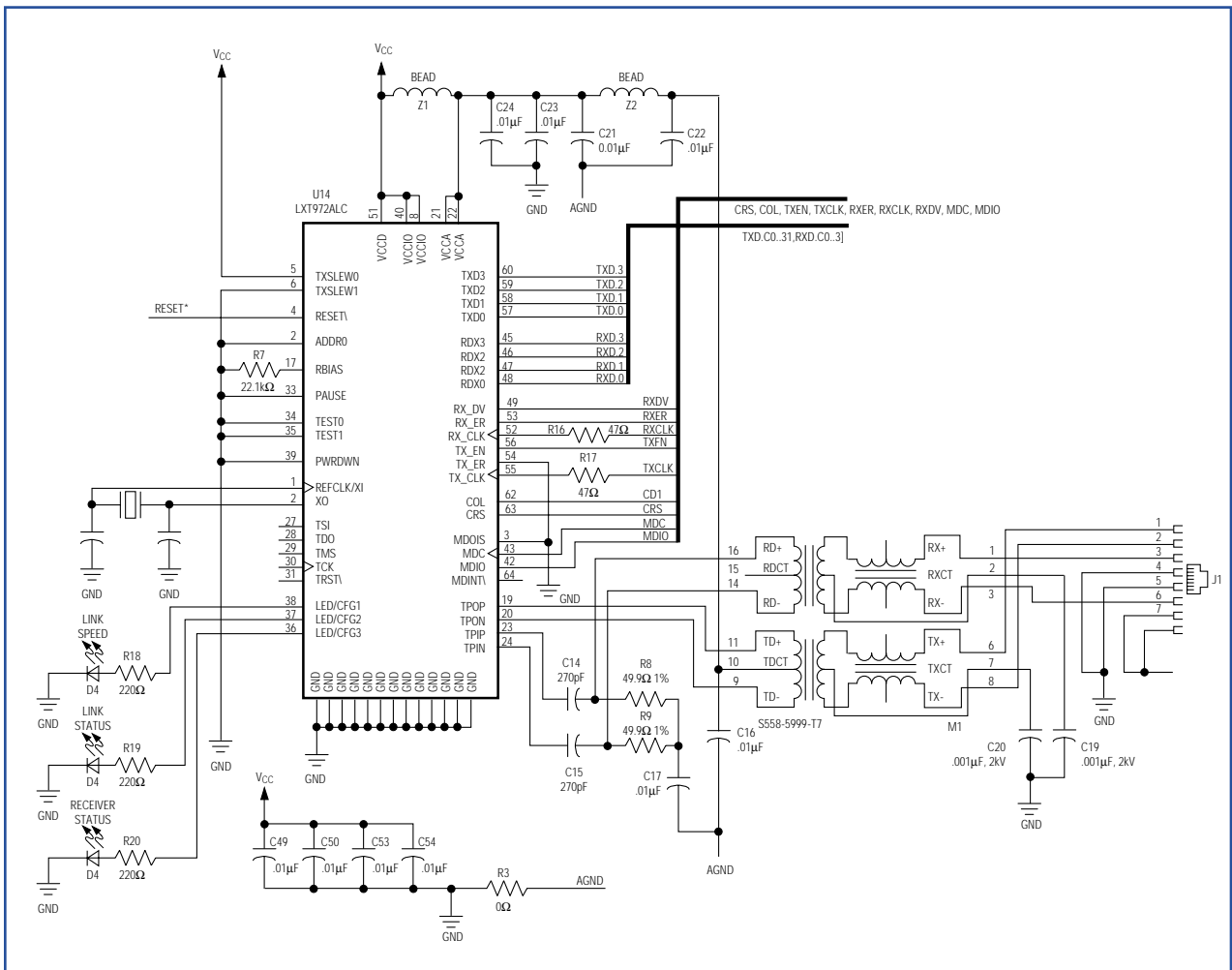


图2. DS80C40通过标准MII接口连接到PHY。

大小也有限制。任何独立的类文件不能超过64kB，一个方法最多可以有63个局部变量，数组的长度不能超过64kB，系统也被限制在8个进程和每个进程32个线程以内。然而，并不是所有区别都是关于限制条件的。比如，1.1 JDK 不支持IPv6，所以TINI的工具来自1.4 JDK。

除了标准的Java类，TINI还提供以下程序包：

```
com.dalsemi.comm
com.dalsemi.fs
com.dalsemi.system
com.dalsemi.tininet
```

这些程序包不仅提供底层的系统访问，而且也支持其他协议，如CAN和I²C。TINI还支持许多高层协议，例如，TINI SDK中的一个范例就是一个轻型(3kB .class 文件)的HTTP服务器。

软件遵循如图3所示的设计大纲。最底层是相机的中断服务处理程序，保持一个指向间接存储区中相机缓冲区的固定指针。相机被设定为单帧模式，等待一个指令来启动图像传输。指令一旦到达，图像就以一定的速率同步传输，该传输速率是通过FRCTL寄存器控制的。在400中，该传输速率被设定为10帧/秒，一帧384 x 288的图像以1080kB/s的传输速率在十分之一秒内被传输。

HTTP SERVER	CAMERA SERVER	CASH
NATIVE METHODS	JVM	
CAMERA ISR	NETWORK	

图3. TINI流相机软件同时支持底层和高层协议。

使用8051汇编语言与相机通信非常容易。图像的每个像素都是从一个相机寄存器中同步读取的。由于相机工作在存储器映像方式下，对相机寄存器的读写需要将数据指针指向相机地址，并执行movx指令。在传统的8051汇编语言中，将一个地址中的数据传送到另一个地址的操作包括如下步骤：载入数据指针，将存储器内容读入累加器，设置新的地址并将其写入数据指针，将累加器内容写入存储器的新地址中。

```
mov R0,#LOW(MEMORY_LOW)
mov R1,#HIGH(MEMORY_HIGH)
camera_loop:
;
```

```
; Move the camera address into the data pointer.
;
mov dptr,#CAMERA_ADDRESS
;
; Move the data into the accumulator.
;
movx a,@dptr
;
; Move to the address we will be writing to. Since
; this will increment every time, we will keep
it stored
; in registers. We will also need to move it one
byte at
; a time using the DPL and DPH SFRs.
;
mov dpl,R0

mov dph,R1
;
; Write the accumulator to the address
movx @dptr,a
;
; Increment the data pointer and store back in
R0 and R1.
;
inc dptr
mov R0, dpl
mov R1, dph
; Do the loop again...
```

DS80C400有四个数据指针，可以将数据从一个地址快速地复制到另一个地址中。这样就省去了所有的地址交换，使复制过程快得多。

```
;
; Set up the data pointers. We use the DPS
register to select
; what data pointer we want to use. A data
pointer move allows
; for a 24-bit address to be loaded directly.
;
mov dps,#0
mov dptr,#CAMERA_ADDRESS

mov dps,#1
mov dptr,#MEM_ADDRES

;
; Set data pointer 0 as the current data pointer.
mov dps,#0
camera_loop:
;
; Read from data pointer zero.
;
movx a,@dptr
;
; Switch to the next data pointer. Note that doing
; an inc on this register only affects the data
; pointer-selection bit. This allows one cycle
toggling
; from one data pointer to another.
;
inc dps
;
; Store the data and increment the address.
;
movx @dptr,a
```

```

inc dptr
;
; Switch back to the first data pointer.
;
inc dps
;
; Do the loop again...
;

通过对上述两段程序的比较可以看出，后一段程序的循环执行速度更快，因为大部分地址处理操作都是在循环以外完成的。在存储区的块复制方面，DS80C400 还为实现更快的复制操作进行了优化。首先，所有数据指针的递增操作都是单周期指令。当启用自动递增模式时，数据指针中地址递增的操作将在读或写操作完成之后自动执行。另外，还可以启用自动选择功能，在每次读或写操作完成之后，可以在两个数据指针间自动切换。这些都使存储区复制变得不可思议的方便。
;
; Set the base address of data pointer zero.
;
mov dps,#0
mov dptr,#ADDRESS1
;
; Set the base address of data pointer one.
mov dps,#1
mov dptr,#ADDRESS2
;
; Enable autoselection and autoincrement.
;
mov dps, #(DPS_AID | DPS_TSL)
memory_loop:
;
; Read from data pointer 0, increment the
; data pointer, and toggle the selection bit
; in one instruction.
;
movx a,@dptr
;
; Write to data pointer 1, increment the data pointer,
; and toggle the selection bit in one instruction.

```

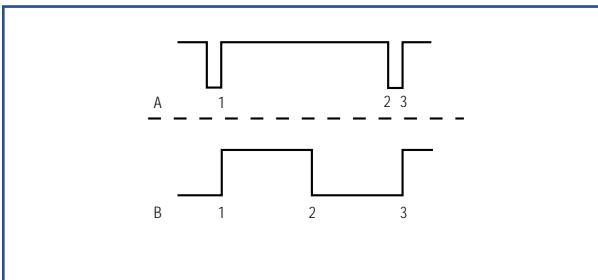


图4. HSYNC 信号在每个单独的扫描行发出。在A中，像素窗口包含全部384个像素点，而B中只有192像素，从左边开始，且两者被设定为同样的帧频。标号1表示相机扫描行开始发出，标号2表示扫描行结束，标号3表示相机启动下一扫描行的时间。尽管B只有A的一半长，但扫描行的周期相同。

```

;
movx @dptr,a
; Loop...

```

回到前面的范例，HSYNC 用来产生中断。每扫描一行后相机通过HSYNC发出信号，ISR 激活并开始采集数据。为了提高性能，将包括调度器在内的其它所有中断都设定为低优先级。这对图像质量的影响很显著。

相机具有一个可调整的窗口，允许由用户决定使用384x 288图像中多大的区域。要找到一个合适的图像尺寸比较困难，大的图像质量更好，但需要更多的传输时间并降低了帧频。在该应用中，Joe将相机的分辨率设定为240x 180，一方面这是因特网视频的标准分辨率，另一方面它还有其它硬件方面的优势。从图4可以看到，当相机传输图像时，实际上就是列举图像阵列中的每个像素，但是只有当像素落在指定窗口中，HSYNC线才会发出信号。这意味着在100ms周期里，一帧图像的采集消耗了3/5的CPU时间。

初始化时，相机软件从存储器管理器中分配出95kB连续的存储区，用来完成双重缓冲。在Java中，一个线程用来处理图像采集，而另一个线程则用来通过网络传输图像。Java提供所有必须的线程及锁定机制来简化上述工作。

在软件设计中位于ISR上层的是为Java虚拟机提供相机驱动的本地方法。TINI平台使用TINI本地接口(TNI)可以使开发者从Java访问底层代码。本地方法可调用TINIOS本地API，允许他们访问存储器管理器、调度器和其它操作系统的内部资源。他们可以传递参数，甚至象其它Java方法那样引发异常。它们通过TINI动态链接库(TLIB)与Java可执行程序链接，TLIB可以用TINI开发套件建立。

本地方法允许Java向相机发送指令。相机驱动类具有如下定义：

```

public static final int IMAGE_BUFFER_0 = 0;
public static final int IMAGE_BUFFER_1 = 1;

/**
takePhoto takes a photograph and stores it in
the memory.
@param buffer Species what image buffer to use.
Use IMAGE_BUFFER_0 or IMAGE_BUFFER_1
*/

```

```

static native void takePhoto(int buffer);

/**
getScanlines pulls a fixed number of
scanlines out of the memory buffer. This
allows the Java application
the ability to work with fixed pieces of the
image.

@param start first scanline to copy from.
@param end last scanline to copy from
@param offset offset into the data array to
copy to
@param data array to copy into
@param buffer selects the image buffer to read
from. Useful IMAGE_BUFFER_0 or IMAGE_BUFFER_1
*/
public static native int getScanlines(int
start, int end , int offset, byte []data, int
buffer);

```

主方法takePhoto用于采集一幅图像到图像缓冲区中。首先，它开放中断，向相机发送摄取一帧图像的指令。这里会出现一个小的僵局。在这个时刻Java线程最好处于休眠状态，但是它却不能通过ISR被唤醒，这是因为TINIOS的函数是不可重入的。为解决这个问题，TINIOS允许开发者注册查询子程序，由系统每4ms调用一次。查询子程序快速查看图像摄取的完成情况，图像摄取完成后唤醒该线程。在将该线程置于休眠状态之前，注册查询子程序。该线程被唤醒后进入Java。正如上文提到的，相机是被双重缓冲的，因此必须指定将要被覆盖的图像缓冲区。为了使Java访问下层的图像缓冲区，还需要执行getScanlines，将扫描行数据块复制到Java字节数组中去。

接下来出现的是关于存储的问题。TINI运行环境占用了512kB闪存中的7个区，只为用户程序留下1个区。正如上文提到的，在高速设计中没有非易失的文件系统，所以必须创建文件系统。比较理想的办法，是将上电时需要执行的全部程序放在闪存区中，包括HTTP服务器将要用于web浏览器的Java applet。为了将applet包含在可执行文件中，applet的二进制文件被转换为与汇编程序兼容的格式，并被包含在库的数据段中。然后，由一个本地方法将其从库拷贝到Java字节阵列中。启动时，Java代码读取applet的大小，并创建一个相应的阵列，然后将applet复制到阵列中，最后将内容写入文件系统中。这个过程的确有点麻烦，但是这意味着可以从一个干净的启动环境开始，只运行所有需要的任务。以下就是用来完成这项任务的方法：

```

/**
Extracts the sample jar file from the native
library. The demo application had a jar file
embedded inside the native library.
This allowed the jar file, the application, and
the native library all to be embedded in flash
format.

@param dummy Array to copy jar image into.
Must be of greater size than that specified in
getJarFileSize()
*/
static native void getJarFile(byte []dummy);

/**
Gets the size of the embedded jar file
*/
static native int getJarFileSize();

```

运行在相机驱动器上层的Java就简单多了，它可以运行许多线程，这些线程中的大部分都是相互独立的。首先是HTTP服务器。HTTP服务器的代码来自包含在TINI SDK中的HTTP服务器范例。该HTTP服务器是非常轻型的，它并不是为servlet、cgi-bin进程或其他功能设计的。文件由TINI文件系统读取，它是一个运行于TINI非易失存储器中的层次化文件系统。启动后，相机applet从闪存读出，并写入文件系统，index.html页便产生了。

接下来是相机图像服务器。相机服务器有两个主线程。第一个线程在42877端口打开一个TCP服务器套接字，并等待来自applet的连接。在一个嵌入式系统中，服务器套接字是如何被打开的呢？实际上，这和PC上完成的过程很类似。

```
sockpuppet = new ServerSocket(42877);
```

注意到服务器套接字被同时绑定到了IPv4和IPv6端口上。这意味着不需要作任何改动就可以使相机兼容IPv6。IPv6大得多的地址空间将给网络设备带来好处，目前，它们仍要同PC、蜂窝电话和其它网上设备争夺地址。

当进程连接时，发送'A'表示连接，或发送'D'表示断开连接。执行连接指令后，IP地址被加入已连接地址的共享向量，而断开连接指令将把地址从该向量中去掉。

```

sock = sockpuppet.accept();
ch = (char)sock.getInputStream().read();

switch (ch)
{
case 'A':

cwt.addAddress(sock.getInetAddress());
break;

```

```
case 'D':  
  
cwt.removeAddress(sock.getInetAddress());  
break;  
}
```

```
sock.close();
```

另一个线程是图像传输。如果相机向量中有一个地址，那么采集的图像将以UDP数据包的形式传输到该地址中。相机的采集和传输经过优化可以并行运行。采集是双重缓冲的，允许传输使用其中一个缓冲区，而采集发生在另一个缓冲区。由于传输时相机只使用了CPU时间的50%，因此这是可以实现的。异步锁存全部在Java内完成。

```
//  
// Notify the camera thread we are ready for  
// the next frame.  
//  
synchronized(stopper)  
{  
    stopper.notify();  
}
```

由于没有使用图像压缩硬件，所以传输的图像是原始的。数据包的组成非常简单，每个包有一个2字节的头，随后是5个扫描行的数据。第一个字节是帧号，它是一个滚动计数器，每传输一帧后加1；第二个字节是垂直偏移除以5。

最后，有一个配置工具在串口上运行。该应用，CAmera SHell 或CASH，是一个菜单驱动的实用程序，用于设置IP地址和显示已连接的用户。这些功能中的大部分都来自TINI SDK中的Slush外壳。要想配置相机，用户可将相机上电，并通过串口与其通信。CASH提供了一个简单的用户界面来设置IP地址。

一旦配置完成，相机就驻留在网络中，等待用户访问。当有人通过web浏览器接入时，相机会提供applet，产生与相机服务器的连接并显示图像。虽然TINIOS每次支持24个同时打开的套接字，但是由于速度的原因，将对相机的访问限制在每次4个用户。采用组播可以解决这个问题，但是Java applet不支持这种方法。

网络相机使用200kB/s的平均网络传输速率，每秒采集并传输4.5帧图像。要知道这是在极少硬件支

持的情况下获得的。没有图像采集和编码硬件，DS80C400同时操持着图像采集、图像传输、网络通信、web服务以及通过串口的用户交互等等操作。

结语

回过头再来说Joe。由于了解到TINI是针对他的产品的理想解决方案，他继续开展网络门锁的研制工作。由于他低成本高性能的解决方案，Joe成为网络门锁市场的统治者。尽管Troy的解决方案号称拥有全球软件专利权，但仅仅因为它成本太高了，所以被Joe轻易地打败了。而Alex的解决方案还面临着相机和网络两方面的问题，很难真正推向市场。于是Amiga离开了Troy回到Joe身边，而Joe则许诺他将不会再让计算机夹在两人中间。由于事业的成功，Joe和Amiga退休居住在明尼苏达的一栋小木屋里，当然，每个门上都有网络门锁。

嵌入式设备需要针对特定的问题进行设计。如何找到功能与成本之间合理的平衡点是其中的难点。尤其是要为嵌入式装置添加网络功能时，问题就更复杂了。一种解决方法是扩展8位微控制器，使其融入网络世界。尽管这种方法是可行的，但它的速度慢将不可避免的。另一种解决方法是使用嵌入式Linux、PC-104，或Pocket PC装置。尽管这种方法速度快且响应迅速，但同时也带来了相当多不必要的功能。当然也可以构建小型的32位方案，但那需要去购买操作系统和TCP/IP栈授权。

包含TINI的DS80C400是介于两者之间的很好的方案。它具有经过长期完善的强大而成熟的TCP/IP栈。具有支持多进程、Java、线程和同步化的操作系统。处理器可以应付类似于数码相机这样的重型任务，但不需要同样重型和臃肿的操作系统。如果这种方案适用于普通的Joe，那么它也将适用于你。

类似文章发表于2003年2月的*Embedded Control Europe*。