Analog Devices公司DSP、处理器和开发工具的使用技术指南
联系我们的技术支持中心：dsp.support@analog.com 和 dsptools.support@analog.com
查看我们的在线资源：http://www.analog.com/ee-notes 和 http://www.analog.com/processors

ADSP-TS20x TigerSHARC R 处理器的启动加载内核运行

B. Lerner供稿

2004年3月4日，第一版

简介

本工程师对话解释了ADSP-TS20x TigerSHARC® 处理器系列的加电启动程序和启动加载程序内核的运行。

本工程师对话的重点部分是ADSP-TS201S和ADSP-TS202S处理器的内核。由于ADSP-TS203S处理器仅有两个连接端口和一个32位的外部总线，因此该处理器的内核形成一个所讨论功能的子集。除了这些限制，以下信息适用于所有的ADSP-TS20x处理器。

加载程序内核与启动模式

加载内核是由处理器执行的，通过VisualDSP++®开发工具的elfloader.exe实用程序可附加到用户应用代码。处理器在启动时间执行该加载程序内核处理器，允许处理器初始化其在应用代码里定义的内部和外部存储器空间。

加载程序内核是一段可以被传送到处理器内部存储器的自检程序。ADSP-TS20x系列处理器支持三种启动方法：EPROM启动（通过外部端口），主机启动（通过一个外部主机处理器或另一个ADSP-TS20x处理器）和连接启动（通过处理器的连接端口）。VisualDSP++ 包含三种独特加载程序内核，支持每一个处理器启动模式。另外，还有一些非启动模式，不需要内核。

启动程序

启动模式由处理器的/BMS管脚来选择。处理器被复位时，/BMS管脚是活性输入。如果复位之后，几个SCLK周期内，/BMS采样的数值为低电平，就会选择EPROM启动模式；经过一定的SCLK周期之后，/BMS管脚变成输出口，并且作为EPROM的选择芯片。如果/BMS的采样数值为高电平，ADSP-TS20x处理器将处于空闲状态，等待主机启动或者是连接端口的启动。处理器的数据手册[3]提供了采集/BMS启动条和运行/BMS的准确时间。

此外，在/BMS管脚上有一个安装在内部小下拉电阻。下拉电阻的值是否合适取决于安装在管脚上的外部线载。这样，选择EPROM启动模式就需要添加一个外部下拉电阻。如果要求使用主机启动和链接启动，而/BMS从未被用作芯片选择的话，/BMS必须在复位时和复位之后保持高电平，可直接和VDD__IO相连。

以下的章节详细描述了每一种启动方法。

EPROM启动

当选择EPROM启动方法时，ADSP-TS20x处理器初始化它的外部端口DMA的0通道以便将256个代码的32位字从启动EPROM传输到ADSP-TS20x处理器的0存储块中，地址为0x00-0xFF。相应的中断矢量（从DMA的0通道）都

已被初始化为0。一旦DMA完成，ADSP-TS20x处理器继续从地址0x00开始执行程序。执行代码的256个字作为启动加载程序去初始化ADSP-TS20x处理器的存储器的剩余部分。VisualDSP++开发工具提供了可以作为参考的缺省的启动加载程序内核源文件（TS201__prom.asm）。

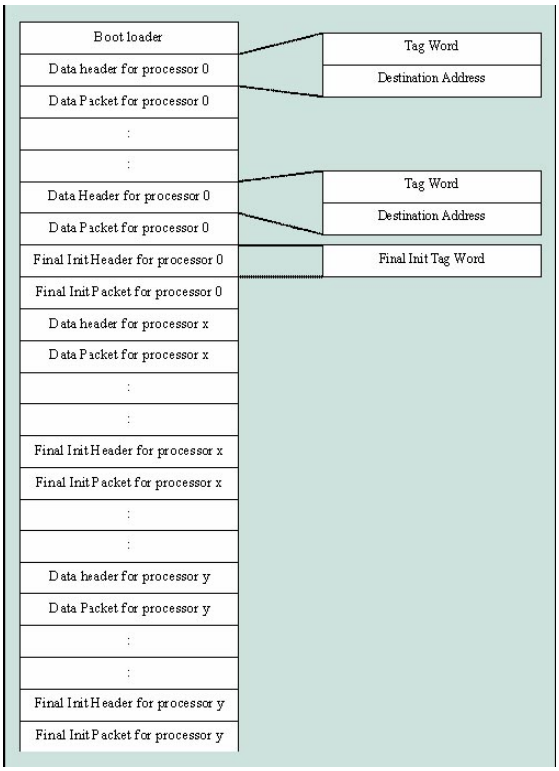
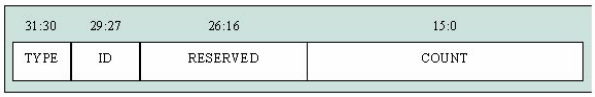


图1. EPROM加载文件格式

缺省的EPROM启动加载程序与VisualDSP++开发工具提供的加载实用程序(elfloader.exe) 同时运行。该加载实用程序获取您的可执行文件（*.dxe）和启动加载程序可执行文件(缺省：TS201__prom.dxe),并且生成EPROM输出文件（*.ldr）。输出程序文件详细说明了ADSP-TS20x处理器的内部和外部存储器的各种存储块在启动过程中是如何被初始化的。图1描述了它的格式。图2描述了存储块的标签字符。



Type: 0=Final init, 1=Non-zero init, 2=Zero init

ID: ID of the processor to which the block belongsCOUNT: Number of 32-bit words in the block

图2. 存储块标签字格式

启动加载程序（TS201__prom.dxe）的具体操作如下：

1. 加载了启动加载程序后，DMA0中断唤醒ADSP-TS20x处理器并且开始执行地址0x00000000处的程序。在这一阶段，ADSP-TS20x处理器处于DMA0的中断级；禁用进一步的DMA0和全局（SQSTAT[20]）中断。

2. 注意标志符__init__debug__start和__init__debug__end所界定的代码。在程序完成了所有用户代码的启动之后，这些代码描述了处理器的系统寄存器的最终状态。在加载（无需执行程序）.DXE文件时，一些仿真器和模拟器就会利用这些代码将这些寄存器置于同一状态。RDS的功能对于程序的操作是非常重要的，因为它降低了DMA0的中断级，并允许发生进一步的DMA0中断。在RDS；；作用之前，它已将NMOD位置于SQCTL寄存器中以确保处理器始终处于监控状态，并且禁用所有链接端口DMAS。而程序代码从__init__debug__end标记处开始执行。

3. 禁用所有的缓冲器。这对于加载程序的正确操作来说是不必要的，但当用户代码控制时，程序已将缓冲器置于一个已知的净空状态。

4. 加载程序将NMOD，TRCBEN，和GIE位置于SQCTL寄存器中，确保监控模式和激活追踪缓冲器以及全局中断。

5. DAM0中断矢量被置于dma__int。设置DMA0可以将数据从初始地址为0x0400的启动PROM传输到初始地址为0x00000000的内部存储器。DMA路径通过编程TCB，提升PROM指针，置于空闲状态来开始DMA的操作直到DMA中断将其唤醒并把它送至dma__int。在那里，RTI返回到DMA路径，它反过来又返回到加载执行。

6. 寄存器xR7.0可以用来启动DMA0。xR1包含了源计数值，xR5包含了一些可以更改的（和修改一些通常为1和4的）目标计数值，因此在启动DMA0之前，xR1和xR5可以被单独设置。xR3被设置为源数据DP的值，该值为EPROM/Priority=Norm/Normal Word/Interrupt=On。XR7被设置为目标数据DP的值，该值为EPROM/Priority=Norm/Normal Word/Interrupt=On。XR0（i.e.，源地址）从地址0x00000400（必要的话可增加）处开始执行。XR4（i.e.，目标地址）设置为0x00000000.

7. 计算出处理器的ID之后，将该ID值储存在xR10.

8. 加载程序分析来自于PROM的数据块。两个字（下一个数据块的标签字）被转移到0x00000000和0x00000001两个地址中。第一个字中，31：30位属于数据块类型（0＝final init,1=ono-zero init,2=zero init），29：27位是处理器的ID号，26：16位为保留区域，15：0位是计数值。第二个标签字是指向目标数据的指针。

9. 存储块的ID号和存储在xR10中的ID号是相当的。如果两个ID不相同PROM存储块会被跳过。

10. 如果ID号都相同，就要检查类型。

11. 如果类型是1，则字的计数值按每次一个字通过地址0x00000000被传输到目标数据中去。一旦结束，从步骤8开始重复执行。

12. 如果类型是2，COUNT个0字符将被传输到目标数据中去。一旦执行结束，同样的，从步骤8开始重复执行。

13. 如果类型是0，则加载程序执行最后的初始化程序（i.e.，该程序通过使用户代码可以在其原程序上复写自己）。0x00000000-0x000000ff中的256个字的DMA将会以唤醒的状态执行这个操作，但是会在DMA0的中断级下开始启动用户代码。为了避免这个执行操作的发生，使用如下算法：

a. 用户代码的前四个指令（目标定位在地址0x00000000-0x00000003）从PROM开始的DMA，并且存储在寄存器XR11：8中。

b. 下面的代码写入地址0x00000000-0x00000003

```
RETI = 0;;
SQCTL = yR0;;
RTI (ABS) (NP); Q[j31+=0] = xR11:8;;
```

c. DMA0中断矢量设置在地址0x00000000中。

d. yR0被事先调整为SQCTL__NMOD。SQCTL__TRCBEN（当使用代码开始操作时，SQCTL__TRCBEN停止全局中断，但是会确保NMOD位的（监控模式）和TRCBEN位（跟踪缓冲器）会被激活。这些数值将会以__last__patch__code形式写入SQCTL寄存器，该寄存器被重新设置地址到0x00000000-0x00000003。

e. 用分支目标缓冲器清空快速缓冲分支是无效的（BTBINV）。

f. 高速缓冲存储器被重新启用。

g. 设置DMA可以将目标用户代码的252个字符传输到地址0x00000004-0x000000FF上。

h. 通过写入TCB操作启动DMA，然后处理器处于空闲状态。

i. 当DMA执行完成后，中断操作唤醒ADSP-TS20x处理器并且跳过执行到达插入到0x00000000地址中的代码（参看步骤b）。

j. 该代码执行运算法则SQCTL=yR0;;该运算法则禁用所有的全局中断，然后执行RTI（ABS）（NP）；Q[j31+=0]=xR11:8;;,执行该操作可以降低中断级为0，将用户代码放置在地址0x00000000-0x00000003中，并且跳过执行到地址0x00000000上去（因为RETI已被置为0x00000000）。用户代码从地址0x00000000处开始执行，并且不会有中断。注意有必要选择NP，以阻止RTI从高速缓存进入BTB中。

如果外部存储器（例如SDRAM）需要特殊的初始化操作，启动加载程序内核必须对它进行配置。该存储器配置必须在其初始化之前在启动加载程序内核中进行配置。因此，你必须根据应用和系统的需求修改或重建启动加载程序。

主机启动

当选中主机或链接启动后，ADSP-TS20x处理器复位之后进入空闲状态，等待着外部主机处理器或链接端口来启动。主机启动使用ADSP-TS20x处理器的AUTODMA寄存器（两条通道的任意一个都行），这两个寄存器均被初始化以将代码的256个字传输到存储块0，位于ADSP-TS20x处理器的内部存储器的0x000xFF处。相应的中断矢量被初始化，并指向地址0x00。因此，一旦DMA一完成，ADSP-TS20x处理器继续在存储器地址0x00处执行。这些代码的256个字符充当启动加载程序来初始化剩余的ADSP-TS20x处理器的存储器。VisualDSP++开发工具提供了缺省的启动加载程序，名为TS201__host.asm。

缺省的主机启动加载程序与带有VisualDSP++开发工具的实用程序同时运行。实用程序获得了项目的可执行文件、可执行的启动加载程序（TS201__host.dxe），并生成一个带有.LDR扩展名的主机加载程序文件。主机加载程序文件详细说明了ADSP-TS20x处理器的内部和外部存储器部分如何被初始化的。图1描述了它的格式。图2描述了存储块标签字符的格式。

接下来的程序中，AUTODMA1寄存器可以代替AUTODMA0寄存器，AUTODMA0寄存器对启动加载程序代码做合适的修改以反映寄存器用法中的变化。

启动加载器使用AUTODMA0寄存器，并按如下方法运行：

1. 启动加载程序加载后，AUTODMA0中断唤醒ADSP-TS20x处理器，并在地址0x00000000处开始执行主机启动加载程序内核。在这个阶段，TigerSHARC处理器处于AUTODMA0的中断级。因此，禁用进一步的AUTODMA0和全局中断（SQSTAT[20]）。下面描述的加载程序分析了主机传送的后续字。

2. 当主机可以传送新数据到AUTODMA0时，加载程序的控制是非常重要的。虽然AUTODMA0缓冲器已满时，处理器可以将AUTODMA0写入所有的主机，但是当AUTODMA0禁用时，处理器忽略AUTODMA0的写入，因此，当AUTODMA0禁用时，任意写入数据均可非常容易地清空。正是由于该原因，在加载程序处理新数据时，直到数据处理完毕且AUTODMA0被重新启用时，加载程序使用BUSLK来对总线进行封定。为确保适当的启动操作，主机必须遵循的重要指导方针是符合加载程序算法的描述的。

3. 注意标志符__init__debug__start和__init__debug__end所界定的代码。加载程序完成了所有用户代码的启动之后，该代码详细说明了一些处理器系统寄存器的最终状态。加载.DXE文件时（该旁路必须运行该加载程序），仿真器和

模拟器就会利用该代码来把这些寄存器置于同一状态。RDS的功能对于加载程序的操作是非常重要的，因为它降低了DMA0的中断级，并且允许进一步的DMA0中断。在RDS；；作用之前，将NMOD位置于SQCTL寄存器中以确保处理器始终处于监控状态是非常重要的。并且禁用所有链接端口DMAS。真正的加载程序代码从__init__debug__end标志符处开始执行。

4. 禁用所有的缓冲器都。这对于加载程序的正确操作是不必要的，当用户代码控制时，程序已将缓冲器置于一个已知的净空状态。

5. 加载程序把NMOD，TRCBEN和GIE位设置在SQCTL寄存器中，确保监控模式，并且启用追踪缓冲器和全局中断。

6. AUTODMA0中断矢量设为_dma_int。

7. 计算出处理器ID号，并将其存入xR10。

8. 寄存器xR3:0将用于开启AUTODMA0.xR1，含有计数值，且在变化的（通常修改为1的）。这样，在开启AUTODMA0之前，将单独设置xR1。xR3设为DP的值，即Internal Memory/Priority=High/Normal Word/Interrupt=On。xR0的地址（如目标地址）设为0x00000000。

9. 加载程序分析来自主机的数据块。它转换两个字符（紧跟着的数据块的标志字符）。在第一个字符中，31:30位是数据块类型（0=final init, 1=non-zero init, 2=zero init）。29:27位是处理器ID，26:16位为预留空间，15:0位是数据块计数区。第二个标志符是指向DESTINATION（目标地址）的指针。

10. 区块的ID和存储在xR10种的ID相当。如果这些ID不相同，将跳过（如果类型为1或2，在没有任何存储的情况下通过拾取225或计数的字符）该区块，从步骤9开始重复运行。

11. 如果这些ID是相同的，就核查类型。

12. 如果是类型1，字的计数值将通过AUTODMA0转移到DESTINATION中。一旦结束，从步骤9开始重复运行。

13. 如果是类型2，COUNT个0字符将转移到DESTINATION中。一旦结束，从步骤9开始重复运行。

14. 如果是类型0，加载程序将执行最终初始化（即写入用户代码）。从IDLE中唤醒的位于地址0x00000000—0x000000ff的256个字符的AUTODMA0将会完成它，但是在AUTODMA0的中断级将会开始执行用户代码。为了避免这类情形，可使用下面的算法：

    a. 用户代码的前4条指令（地址为0x00000000—0x00000003）被并从主机AUTODMA，并被存入寄存器xR31:18中。

    b. 下面的代码写入了地址0x00000000—0x00000003：
    RETI=0；；
    SQCTL＝yRO；；
    RTI（ABS）（NP）；Q[j31+=0]=xR1：8；；

    c. AUTODMA0 中断矢量被设置在地址0x00000000处。

    d. yR0被预先放入SQCTL_NMOD | SQCTL_TRCBEN中（这样，用户代码启动时，禁用全局中断），以确保启用NMOD位（监控模式）和TRCBEN位（追踪缓冲器）。此值将被以重新分配到地址0x00000000—0x00000003 中的_last_patch_code形式写入SQCTL寄存器中。

    e. 分支目标缓冲器无法清除（BTBINV）被快速缓冲的分支。

f. 快速缓冲存储器被重新启用。

g. 设置AUTODMA0，将用户程序码的252个字转移到目标地址0x00000000—0x000000ff。

h. 通过写到TCB中，启动AUTODMA0。解除总线锁定，然后处理器进入空闲状态。

i. AUTODMA0结束后，中断唤醒TigerSHARC R处理器，并跳过执行到达插入地址0x00000000的代码（看步骤b）。

j. 代码执行SQCTL=yR0；；它禁用所有的全局中断。然后，RTI(ABS) (NP); Q[j31+=0]=xR11:8;;,将中断级降为0，并将用户代码放入地址0x00000000—0x00000003，同时跳过执行地址0x00000000（因为RETI的地址设为0x00000000）。用户代码在无中断的情况下在地址0x00000000中启动。注意选择NP来阻止RTI从缓冲器进入BTB是非常必要的。

如果外部存储器（例如SDRAM）需要特殊的初始化，启动加载程序内核就*必须*配置它。这样，启动加载程序内核必须要修改和重建，并使其满足你的特殊应用与系统的需求。

重要的主机指导方针

查看以下指导方针：

1. 处理器的AUTODMA0寄存器收到数据后，将延迟多达12CCLK个周期在完成AUTODMA0寄存器的ISR和启用总线锁定被激活之前。此时送入AUTODMA0的数据将丢失。因此，主机必须在每次写到从DSP的AUTODMA0寄存器之间，等待至少12*SCLKRAT个外部周期以确保合适的启动。

2. 如果主机有外部的端口输出FIFO（例如，当主机是另一个ADSP-TS20x处理器时），当这些交换在外部端口上发生时，推迟写入外部端口可能也不能保证其间有相同的延迟。如果该输出端口FIFO有几条记录在同时排队进入，当从属DSP解除总线锁定时，所有排队的交换都将会接连出现，这样将违背了上面的指令。处理这个问题需要专门的方法。例如如果主机是另一个ADSP-TS20x处理器，就可在外部总线上将具有任意虚拟地址的插入数据读入拥有独立读写功能的ADSP-TS20x处理器中。直到数据读完后，读取才会自动停止。这样，直到第一个读写完成，紧接着哑读取入的写入才会顺序进入先进先出端口。

下面是ADSP-TS20x处理器主机代码的一个例子：

```
// Write to AUTODMA0 of processor ID=2
[j31+(P2_OFFSET_LOC+AUTODMA0_LOC)]=xR0;;
// delay the required number of cycles
call delay_12_times_sclkrat_cycles;;
// read external dummy memory location
xR1=[j31+dummy_external_memory];;
// insert dependency before next write
xR1=xR1;;
.....
// Write to AUTODMA0 of processor ID=2

[j31+(P2_OFFSET_LOC+AUTODMA0_LOC)]=xR0;;
```

当ADSP-TS20x处理器可以作为主机使用时，向启动另一个ADSP-TS20x处理器的主机提供针对ADSP-TS20x EZ-KIT Lite™开发系统的实例

链接启动

当选中主机或者链接启动时，复位后，ADSP-TS20x处理器进入空闲状态，等待主机或链接端口去启动它。链接启动可使用任何ADSP—TS20x处理器的链接端口，所有这些链接端口

的DMA均被初始化以将256字代码传给ADSP—TS20x处理器的存储区块0，其地址为0x00到0xFF。与之相对应的DMA中断矢量将被初始化为零。因此，链接DMA一旦完成，ADSP-TS20x处理器将继续在地址0x00处运行。这些256字代码将作为启动加载程序来初始化剩下的ADSP-TS20x处理器的存储器。Analog Devices提供缺省启动加载程序TS201_link.asm，并使用了VisualDSP++工具包。

缺省链接启动加载程序与VisualDSP++提供的加载实用程序同时运行。加载实用程序获得您的项目可执行文件（*.DXE）和启动加载可执行文件（缺省为：TS201-link.dxe），并能够生成加载程序输出文件（*.LDR）。.LDR文件详细说明了在ADSP-TS20x处理器的内部和外部存储器块是如何被初始化的。图1描述其的格式。图2中描述区块标签字的格式。

所提供的启动加载程序按下述方式运行：

1. 加载了启动加载程序以后，链接端口DMA中断唤醒TigerSHARC处理器，并在地址0x00000000处开始运行加载程序。在此阶段，TigerSHARC处理器位于链接DMA的中断级，然后进入链结端口DMA。此时，禁用全局（SQSTAT[20]）中断。

2. 链接常数详细说明了将被用来启动的链接端口。在这个程序码中，它被设成3（即链接端口3）。如果使用了不同的端口，就要修改值，重写程序码。

3. 注意由标志符_init_debug_start和_init_debug_end所界定的代码。加载程序完成了所有用户码的启动后，该代码详细说明了一些处理器的系统寄存器的最终状态。加载了.DXE文件时（旁路必须运行加载程序），仿真器和模拟器利用该代码将这些寄存器设置到相同的状态。RDS功能对于加载程序的操作是非常重要的，因为它能降低AUTODMA0的中断级，并允许进一步的AUTODMA0中断。在RDS；；之前，设置SQCTL寄存器中的NMOD位对于确保处理器处于监控模式是非常重要的。此外，所有链接端口DMA都被禁用。实际的加载程序代码将在标志符_init_debug_end初开始运行。

4. 所有的缓冲器都被禁用。当然，这对于加载程序的正确操作是不必要的，但是当用户代码控制时，必须将缓冲器放放入已知净空状态。

5. 加载程序在SQCTL寄存器中可设置NMOD，TRCBEN和GIE位，以确保监控模式，并启用追踪缓冲器和全局中断。

6. 链接端口收到来自_dam_int文件的DMA中断矢量。未使用的链接端口将被清除，然后被禁用。链接端口DMA也被禁用。

7. 链接断口控制寄存器被初始化。

8. 从链接端口提取数据的DMA一次可完成一个四字字。xR3:0被事先预置为TCB所需的值。

9. 来自链接的数据由_read_word程序读入。由于来自链接的数据是四字格式，且处理器按单个32字去分析它，就需要有一个内部的FIFO缓冲器。在存储区地址为0x00—0x03处，其可作为可循环缓冲器来实现。J2是专用的，因为指向缓冲器，J2、JB2和JL2的读取指针将被相应地初始化。_read_word的执行流是：

   a. 检查J2是否已返回零（即缓冲器中的所有数据已经读取）。如果没有，进入“步骤d”来读取缓冲器的下一条数据。

   b. 通过链接DMA，将另一个四个字从链接端口读入寄存器。通过将XR3:0写到TCB来启动链接端口DMA，且程序等

待IDLE中的DMA中断。

c. 从链接端口收到新的四个字时，DMA中断将处理器从IDLE中唤醒，并进入_dma_int中执行。在那， RTI；；紧跟其后的NOP越过IDLE返回到指令。注意这组NOP在这是必要的，但不允许RTI；；出现在第一个四字ISR中。

d. 由J2指示的来自缓冲器的数据被读到地址xR4处，且J2将循环增加。

10. 与其他的启动模式不同，这里不使用处理器ID。加载程序不支持MP启动。

11. 加载程序分析来自链接端口的数据块。两个字（数据块的标志字符紧跟其后）将被移到地址yR8和J10。在第一个字中，31:30位是数据块类型（0=final init, 1=non-zero init, 2=zero init）。29:16位是预留的，15:0位是数据块计数区。第二个标志字是指向DESTINATION的指针。

12. 如果是类型1，字的计数值将以每次一个字通过read_word转移到DESTINATION中。一旦结束，算法进入步骤11。

13. 如果是类型2，COUNT个0字符将移至DESTINATION。一旦结束，算法进入步骤11。

14. 如果是类型0，加载程序执行最终初始化（即写入用户代码覆盖原启动程序）。**将用到下面的运算法则**，通过_read_word将用户代码（地址为0x00000000—0x0000001B）的前28条指令从链接端口转移并存入寄存器xR31:18和yR31:28中。

a. _dma_int处的中断服务程序被移到地址0x04—0x08中。

b. _last_patch_code的19条指令被移到地址0x09—0x1B中。

c. 分支目标缓冲器无法清除（BTBINV）被快速缓冲的分支。

d. 重新启用快速缓冲存储器。

e. 链接端口DMA中断矢量被设置到地址0x04（由于步骤b，此地址中现在包含中断服务程序）

f. yR0被预先设成SQCTL_NMOD | SQCTL_TRCBEN（这样，当用户代码启动时禁用全局中断），以确保启用NMOD位（监控模式）和TRCBEN位（追踪缓冲器）。此值将以被移到地址0x00000000 — 0x0000001B 中的_last_patch_code写入SQCTL寄存器中。

g. J0被初始化为0x1C（通过重新放置_last_patch_code的第一个地址），且LC0被初始化为0xE4（留在被读取的最终初始化的字数）。

h. 在这个阶段，地址0x04—0x1B被如下初始化：

```
0x04: _relocated_dma_int:
      nop; nop; nop; nop;;
      rti(NP);;


0x09: _relocated_read_word:
      // if J2 -> start of the buffer...

      comp(j2,0);;
       // ...bring in more data
       if           njeq,           jump
       _relocated_read_buffer (NP);;
       // start the DMA
       DCx = xR3:0;;
       // wait till DMA interrupts
       idle;;
       _relocated_read_buffer:
```

```
        // read the word from the buffer
        xR4 = cb[j2+=1];;
        // and return
        cjmp (ABS) (NP);;

 0x0F: _relocated_final_init1:
        // read word
        call _read_word (NP);;
        // write it
        [j0 += 1] = xR4;;
        if           NLC0E,           jump
        _relocated_final_init1 (NP);;
        // disable interrupts
        SQCTL = yr0;;

nop;;
        // overwrite 0x00-0x03
        Q[j31 + 0] = xR11:8;;
        // overwrite 0x04-0x07
        Q[j31 + 4] = xR15:12;;
        // overwrite 0x08-0x0b
        Q[j31 + 8] = xR19:16;;
        // overwrite 0x0c-0x0f
        Q[j31 + 0xc] = xR23:20;;
        // overwrite 0x10-0x13
        Q[j31 + 0x10] = xR27:24;;
        // overwrite 0x14-0x17
        Q[j31 + 0x14] = xR31:28;;
        // overwrite 0x18-0x1b, start at
        0
        jump
        0(ABS)(NP);Q[j31+0x18]=yr31:28;;
```

i. 代码执行时将跳到地址0x0F（即，上述的_relocated_final_init1）；

j. 地址0x1C-0xFF充斥着来自链接端口的数据；注意调用_relocated_final_initl处的_read_word(NP)；；是相对调用。因此，它实际上是调用_relocated_read_word，并且重写原来的_read_word不会造成任何问题。

k. 现在完成了链接接收。正确的数据放在地址0x1C-0xFF中，并且应该放在地址0x00-0x1B中的数据在寄存器xR31:8和yR31:28中。剩余的代码用地址XR31:8中的数据重写存储器地址0x00-0x17。最后，执行绝对跳转到地址0x00时，代码最后一行利用来自yR31:28的数据重写地址0x18-0x1B(包括它自身)。

l. 用户代码从地址0x00全新启动。

如果需要特殊设置的外部存储器（例如SDRAM）需要由加载程序来初始化，那么在启动加载程序中，存储器的设置将必须先于其初始化过程。这样，启动加载程序必须由使用者来修改和重建。

无启动程序

当选中主机或者链接启动时，复位后，ADSP-TS20x处理器进入了空闲状态，等待主机或者链接端口来启动它。它并不是必须得由主机或者链接端口来启动。如果启用外部中断IRQ3:0被激活（复位后，IRQEN带状管脚选择了），它可依据缺省中断矢量强迫代码执行，如表1所示。

在这个环节中，另一个装置，例如主机处理器，用适当的代码来初始化适当的存储器，然后强迫执行相应的IRQx。

中 断	地 址
IRQ0	0x30000000(MS0)
IRQ1	0x30000000(MS1)
IRQ2	0x30000000(MSH)
IRQ3	0x30000000(内部存储器)

表1 IRQ3:0缺省中断矢量

对于主机来说，另外一个启动ADSP-TS20x处理

器的无启动方法是用代码来初始化存储器缓冲器，之后通过中断矢量来强迫执行代码。

无启动方法对于系统调试十分有用，但是对于生产不推荐使用。启动芯片的时候，一些复杂细节必须说明。支持的标准加载程序管理那些细节，因此极力推荐基于以上加载程序的系统启动。

参考文献

[1] *ADSP-TS201 TigerSHARC Processor Hardware Reference.* Revision 1.0, May 2003. Analog Devices, Inc. [2] *ADSP-TS201 TigerSHARC Processor Programming Reference.* Revision 0.1, June 2003. Analog Devices, Inc. [3] *ADSP-TS201S TigerSHARC Embedded Processor Preliminary Data Sheet.* Rev PrH, January 2004. Analog Devices, Inc.

文件历史

版本	描述
版本1－2004.03.04　B.Lerber提供	第一版