ANALOG DEVICES

AN-1322
应用笔记

One Technology Way • P.O. Box 9106 • Norwood, MA 02062-9106, U.S.A. • Tel: 781.329.4700 • Fax: 781.461.3113 • www.analog.com

ADuCM320代码执行速度

作者：Eckart Hartmann

简介

ADuCM320包含一个ARM® Cortex®-M3处理器以及用于代码和数据的集成闪存和RAM。为了提高中央处理器(CPU)的执行速度，ADuCM320还包括一个具有各种工作模式的缓存。本应用笔记介绍用户应用感兴趣的模式。本应用笔记并未涵盖ARM Cortex-M3的特定特性。请查阅ARM提供的相关文档了解这些特定特性。

基本的CPU时钟速度是80 MHz。在正常模式下，ADuCM320自64位宽的闪存开始执行，每四个时钟周期取回64位指令代码。若64位读取内容包含四个16位指令，则CPU可以在四个时钟周期内执行这四个指令，有效速度可达每秒8000万条CPU指令(原始MIPS)。然而，若64位读取内容包含两个32位指令，则CPU在四个时钟周期中只能执行这两个指令，因为它必须等待下一个读取完成才能继续。在这种情况下，有效执行速度是40 MIPS。对于16位和32位的混合指令，有效速度介于这两个数之间，而对于典型代码而言，大约是60 MIPS。

乘法和除法指令以及存储器访问指令需要占用一个以上的CPU时钟周期来执行，在这种情况下，ADuCM320中所包含的64位预判缓冲器有时间填充后续指令并使其加速。这种情况的细节非常复杂，取决于确切的指令序列和16位对齐。建议忽略这些细节，并把它们当作额外安全裕量来看待。在这种模式下与数据RAM的通信非常有效，因为RAM处于不同的总线，并且可以和闪存并行工作。

为了提高执行速度，还提供一个默认使能的缓存。执行代码并进入缓存之后，它以80 MIPS运行，与指令大小无关。执行新代码时，会覆写旧代码，而不再存在于缓存中的代码再次以较低速度运行。不过，对于典型代码而言，可实现的有效速度介于70 MIPS和80 MIPS之间。

在一些系统中，可以通过将关键代码置于RAM中并在那里执行来实现较高执行速度，因为RAM通常比闪存快得多。不过，在ADuCM320上，即使没有缓存，在RAM执行实际上也比在闪存执行要慢。

为了实现最快执行速度，提供了L1缓存模式。在L1缓存模式下，现有RAM的一半处在与CPU指令总线连接的模式下，因此代码能够以全速(80 MIPS)执行。不过L1缓存模式会使RAM中可用于数据的部分减少。应注意，在L1缓存中执行时，C语言数据一定不要位于RAM的L1区，以免总线之间有冲突。对于闪存和L1缓存中的代码，都要将RAM的非L1区用于C语言变量和数据。将RAM的L1区备用容量用于成批或很少使用的数据。使用L1区备用容量的细节必须具体情况具体分析。

本应用笔记说明上文所述方法的实施方案。不过，应注意，上述方法都是利用典型编译器工具来处理大多数复杂问题。其他方法也是可行的，但是采用编译器工具所产生的代码来决定函数和变量的位置，会显得有难度。因此，不推荐这样的方法。

缓存模式

缓存模式是默认模式，可用于大多数代码。编译器工具的基本设置采用这种模式，且无需进行特别设置。性能细节如示例项目部分所述。不过，应注意，已改善性能仅适用于执行时缓存中的特定代码。

非缓存模式

若必须具有稳定(尽管较慢)的执行速度，则禁用缓存模式。

采用以下代码来关闭指令缓存：

```
pADI_FEE_CACHEKEY = 0xf123f456;
pADI_FEE_CACHESETUP = 0x0;
```

采用以下代码来开启指令缓存：

```
pADI_FEE_CACHEKEY = 0xf123f456;
pADI_FEE_CACHESETUP = 0x2;
```

除非应用中需要非常严格的控制时序，否则不建议反复开启关闭缓存模式。

L1缓存模式

L1缓存模式不容易设置，需要仔细地注意细节。首先，编程人员必须能够识别出哪些函数对时间要求很高，必须将这些函数放置于单独文件中，与其他代码的文件分开。只包含必要的函数，因为置于RAM中的任何代码都会使得可用于变量和数据的RAM减少。写入代码之后，设置工具。

针对Keil µVision4的程序

如需设置Keil µVision4工具，请遵循以下步骤：

1. 点击**Project/Options for Target ‘Target 1’**，打开图1所示窗口。选择**IRAM1**，并将其开始和大小分别设置为0x20004000和0x4000。这是非L1 RAM的范围。选择**IRAM2**，并将其开始和大小分别设置为0x10000000和0x4000。这是L1 RAM的范围。**IROM1**保持不变，将大部分代码保存于闪存中。

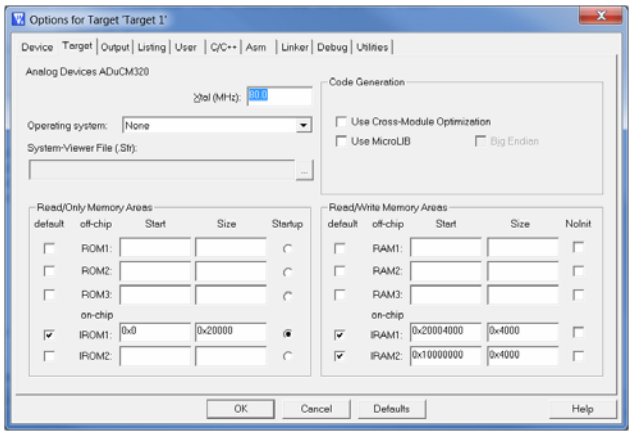

图1. 选择存储器范围

2. 将代码置于IRAM2中。在**Project**选项卡中，右击欲置于L1缓存中的文件。在打开的窗口中，点击顶端项目(**Options for File ‘RamCache.s’**)。在图2所示的窗口中，针对**Code / Const**选择**IRAM2 [0x10000000-0x10003FFF]**。针对**Zero Initialized Data**，选择**IRAM1 [0x20004000-0x20007FFF]**，或者保持**<default>**，让链接器来选择范围。在相同窗口中，针对**Other Data**选择范围，或者保持**<default>**，让链接器来选择范围。对于去往L1缓存的每一个附加文件，重复这个程序来定义**Code / Const**、**Zero Initialized Data**及**Other Data**，但是确保所选范围不会彼此重叠。

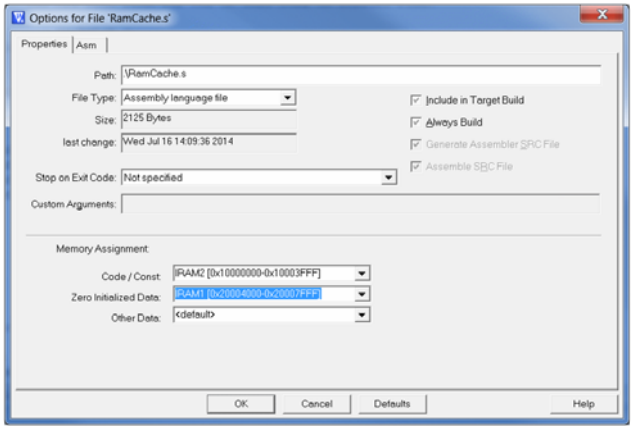

图2. 分配存储器范围

3. 将代码置于L1缓存中。在步骤2中，将代码置于IRAM2中。编译器首先将代码置于闪存中，然后在启动期间，到达主函数main( )之前，还有隐藏代码将代码复制到L1缓存区。遗憾的是，在这个阶段，L1缓存并未使能，写入0x10000000会造成不可接受的异常。因此，必须在启动代码开始处修改启动文件以选择L1缓存模式。通过添加以下缩进代码来修改启动文件。

```
Reset_Handler    PROC
EXPORT  Reset_Handler [WEAK]
IMPORT  __main
   movw   r1, #0x8104
   movt   r1, #0x4002
   movw   r0, #0x0000
   movt   r0, #0x5129
   str    r0, [r1]
LDRR0, = __main
BX        R0
ENDP
```

遵照步骤1到步骤3操作后，关键代码位于L1缓存中，可实现最快速度执行，所有代码都使用所有函数的正确进入点；所有变量以及其他数据和执行的位置均如同预期的那样。

注意不要将不兼容的启动代码和存储器映射设置组合在一起，因为这会造成器件锁定无法调试。如果确实出现锁定，使用应用程序MDIOWSD.exe来擦除出问题的代码，并下载其他合适的代码。

示例项目

QuickStart开发系统在C:\ADuCM320V1.0\code\ADuCM320\examples\Cache\Cache.uvproj下提供一个示例项目。这个示例的主代码从缓存模式切换到非缓存模式，然后切换到L1缓存模式。每个模式都运行一小段相同测试码两次，在每一个测试码开始处有一个窄标记，而在每一个测试码结束处有一个较宽标记。图3的示波器屏幕截图显示了不同模式下运行的短测试的执行时间是如何变化的。

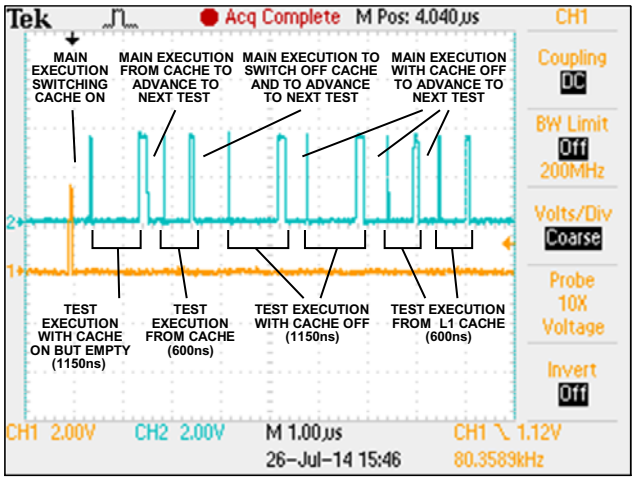

图3.示例示波器屏幕截图

AN12610sc-0-12/14(0)