Blackfin®处理器与SDRAM技术

Fabian Plepp 撰稿

Rev 2 – December 11, 2008

引言

ADI公司的Blackfin®系列处理器提供了可与SDRAM接口的外部总线接口单元（EBIU）。

本EE-文件包括以下主题：

- 寄存器设置及其意义
- SDRAM初始化
- SDRAM硬件设计
- 使用容量小于16MB的SDRAM
- 性能优化
- 电源优化

本EE文件讨论SDR-SDRAM（不是DDR-SDRAM）。
本EE文件仅适用于ADSP-BF53x，ADSP-BF52x及ADSP-BF561处理器，不适用于ADSP-BF54x处理器。

虽然本文档涉及了SDRAM的基础功能，用户仍需参阅文档*The ABCs of SDRAM (EE-126)*[1]以获取更多信息。

本文档分为几个不同的主题，因此当需寻找特定信息时无需阅读整个文档。

目录

1 SDRAM简介

本节说明SDRAM（同步动态随机存取存储器）的参数以及相关的系统设置，为建立对EBIU（外部总线接口单元）寄存器变量的更好理解奠定了基础，另外，也为用户决策奠定了基础，当然，这不是对SDRAM的详细讨论，若需详细信息请参考*The ABCs of SDRAM (EE-126)*[1]。

1.1 SDRAM基本知识

SRAM利用晶体管存储二进制数据，而DRAM则利用电容器和晶体管存储数据：电容本身就是一个存储元件，它充电时，为逻辑1，否则为逻辑0。晶体管则像一扇门，控制对存储单元的访问以及放电，该技术的弊端在于电容器会随着时间逐渐放电。

为了防止数据丢失，DRAM必须定期刷新，修复存储单元上的电荷。因此，在刷新过程中，需要对存储数组中的每个存储单元至少执行一次读和写操作，这一过程通常是几毫秒。

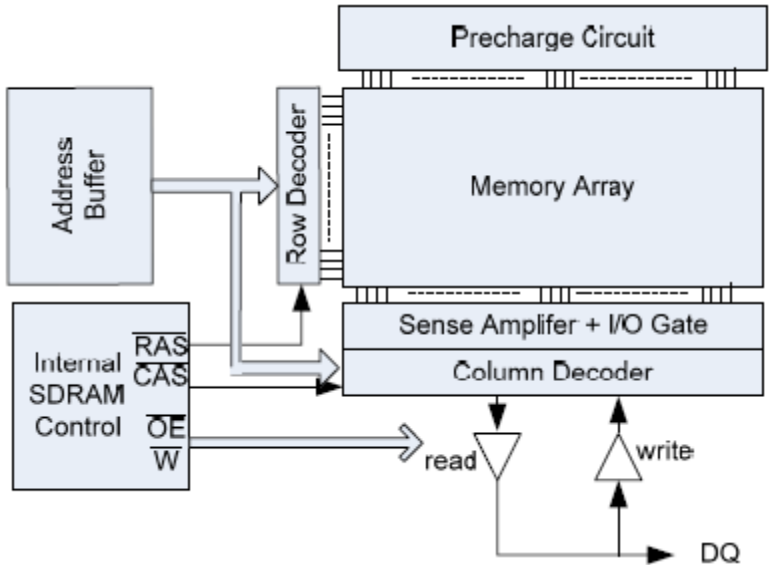


图1. SDRAM

DRAM单元组织为行和列的阵列形式，每个独立单元都可通过精确指定的行和列地址进行访问。通常把行称为*page（页）*，而列的数目则称为*page size（页大小）*。地址采用时分复用，先发送行地址，后发送列地址。

/RAS（行地址选通）及/CAS（列地址选通）信号控制行和列地址的时分复用。/RAS信号标志一个行地址有效，行地址将加载到地址缓存器中由内部行地址解码器进行解码，经过很短的延迟，标志列地址选通的/CAS信号载入地址缓存器中，并进一步由列地址解码器对其进行解码。

当/WE（写使能）为高时处理读命令，由行地址寻址的存储单元被完全读出、放大、并写回存储单元中，列地址寻址的单元则通过数据总线进行发送。

当/WE为低时处理写命令，对单个存储单元的写操作不可实现。因此，首先将一整行读入缓存器，在缓存器中，完成对要修改的元素的写入后，再将整行数据写回存储阵列中。

1.2 Blackfin寄存器中的SDRAM参数

1.2.1 EBCAW（SDRAM 外部存储块列地址宽度）

通常，把外部存储器块的列地址宽度称为SDRAM的*page size（页大小）*。Blackfin处理器可支持512字节（8 位），1 K字节（9 位），2 K字节（10 位），及4 K字节（11 位）的 *page size*。

Bank	Row Address	Column Address	Byte
2 MSB	[(EBCAW+[10,16]):(EBCAW+1)]	[EBCAW:1]	0

除字节地址位位于最低位（LSB）外，列地址是逻辑地址的最低位。根据列地址宽度的不同，行地址和存储块的位在逻辑地址中可能占有不同的位置。例如，当列地址宽度（CAW）为11位时，行地址的LSB处于逻辑32位地址[31:0]中的位12处。列地址宽度是连续访问SDRAM的一个非常重要参数。行地址数目取决于SDRAM的大小。要在数据手册中确定列寻址的宽度，以及查找有多少地址引脚用于列地址，请参考Blackfin处理器*硬件参考*中的EBIU部分内容。

1.2.2 EBSZ（SDRAM 外部存储块大小）

SDRAM外部存储块大小可由下面的公式来确定：

$$\text{存储器大小} = \frac{\text{存储块大小} \times \text{数据引脚数目} \times \text{存储块数目}}{8}$$

例如，MT48LC32M16A2有8 M字节×16引脚×4存储块，一共是536,870,912位，相当于64M字节。若使用的SDRAM容量小于16MB，请参考使用SDRAM小于16MB的Blackfin处理器。

1.2.3 SDRAM时序

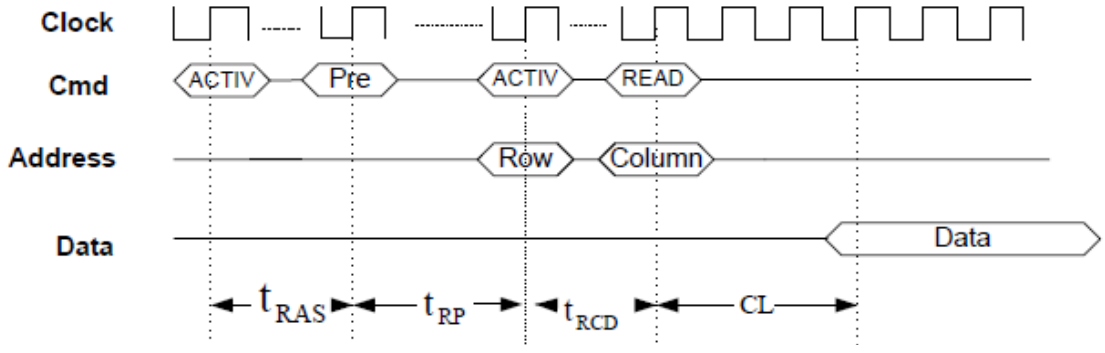


图2. SDRAM时序

SDRAM CAS等待时间（CL）、SDRAM存储块有效命令延迟（t_{RAS}）、SDRAM存储块预加电延迟（t_{RP}）、RAS到CAS延迟（t_{RCD}）以及对预加电延迟的写操作时间（t_{WR}）等SDRAM时序在用户处理器*硬件参考*中的EBIU部分有详细介绍。

1.3 多处理器环境选项

Blackfin处理器可用于多处理器环境中。多处理器连接的一种途径就是共享外部存储器，并通过外部存储器在处理器间传递信息，Blackfin处理器提供了专用的引脚用于实现仲裁功能。

在共享存储器的多处理器环境中使用Blackfin处理器，要实现访问控制，必须连接/BR，/BG和/BGH引脚。当外部设备要求访问总线时，就发出总线请求（/BR引脚）信号，若此时没有其他请求挂起，则允许该外部总线请求。处理器将使数据和地址总线保持三态，并确认总线准许（/BG引脚）信号。当总线授予给其他外部设备时，任何从外部存储器取数据或指令都会使处理器停止，直到释放总线，继续执行访问。外部设备释放/BR后，处理器撤销/BG信号，并从原先停止的地址继续执行。当处理器准备启动另一个外部端口访问时，将会触发/BGH信号，但此时由于总线已被授予，/BGH信号会无效。

1.3.1 BGSTAT（总线准许状态）

当总线允许后，SDSTAT寄存器中的BGSTAT位将被置位，处理器可用此位检查总线状态，避免初始化一个可能被外部总线准许延迟的处理。

1.3.2 PUPSD（上电启动延迟）

该选项为上电启动序列设置15个系统时钟周期的延迟。如果处理器将SDRAM转到另一处理器，必须使SDRAM处于自刷新模式。一旦启动自刷新模式，SDRAM为自身提供内部时钟，执行自身的自动刷新周期。SDRAM保持自刷新模式的时间不少于t_{RAS}。

退出自刷新模式的过程需要一系列命令。首先，CLK必须在CKE回到高电平前保持稳定（稳定时钟定义为在时钟引脚指定的时限内的时钟周期）。当CKE为高电平时，必须对SDRAM发出持续时间为t_{XSR} 的NOP命令，该时间长度是完成任何内部刷新过程所必需的。考虑到t_{XSR}的时钟周期，将出现15个周期的延迟，直到Blackfin处理器启动其上电过程。

1.3.3 CDDBG（准许期间的控制失效）

如果用户以共享存储器方式工作于多处理器环境，其他非Blackfin处理器的外部设备可以访问存储器，该特性将使能处理器的外部存储器接口，并使地址和数据引脚、存储器控制引脚（SRAS，SCAS，SWE，SA10和SCKE）及时钟引脚（CLKOUT）都处于三态。

1.4 可移动/低功率 SDRAM选项

Blackfin处理器支持可移动SDRAM芯片（也称为低功率SDRAM）。根据Blackfin处理器系列的不同，用户可选用3.3V、2.5V和1.8V SDRAM。选用2.5V和1.8V的芯片，要确认该处理器数据手册（电气特性）中处理器能输出SDRAM芯片数据手册要求的（最大）低电压的能力。可移动SDRAM能实现低功耗不仅因为其电压较低，其电流也很低。可移动SDRAM的另一优点是它通过设定温度，不激活用户不使用的存储区的自刷新功能，从而减小自刷新率。这些特性使用户能够为特定的应用优化功耗。如果Blackfin数据手册中没有其他声明，V_{DDEXT}为1.8V时，系统时钟限制在100MHz以内（系统时钟的最高频率）。

	ADSP-BF51x	ADSP-BF52x	ADSP-BF533/2/1	ADSP-BF537/6/4	ADSP-BF539/8	ADSP-BF561
1.8V	✓	✓	✓	✗	✗	✗
2.5 V	✓	✓	✓	✓/✗*	✓/✗*	✓/✗*
3.3 V	✓	✓	✓	✓	✓	✓

*: 与选择的SDRAM的芯片有关

Figure 3. 处理器系列支持的SDRAM电压

要使用可移动SDRAM的扩展寄存器，必须设置EMREN位。

1.4.1 PASR（部分阵列的自动刷新）

每次刷新都要消耗功率，如果应用程序在特定模式下不需要存储整个存储器，该特性可禁止SDRAM几个存储区的刷新，好处是降低了功耗。

例如，将数据（如一张图片）存储到SDRAM中，该应用只执行一些信号处理，并发送数据或将数据存储到非易失性存储器（如闪存）后，就返回到睡眠模式。大多数SDRAM都是在信号处理时使用，处理器处理完后就释放数据，因此保存数据的SDRAM存储块就不必再被刷新。

1.4.2 TCSR（有温度补偿的自刷新）

在标准SDRAM中，自刷新率是根据最坏情况设置的：高温将使电容器漏电更快，要保存数据就需要更高的刷新率，但高刷新率使功耗更高。因此，利用TCSR位，应用程序就可根据其测得的温度设置自刷新率。

1.5 满足SDRAM时序的选项

有时SDRAM的时序参数指标不满足EBIU的性能指标，因此，PLL控制寄存器提供了一个选项可以延迟输出信号并将输入锁存机制移动200ps。

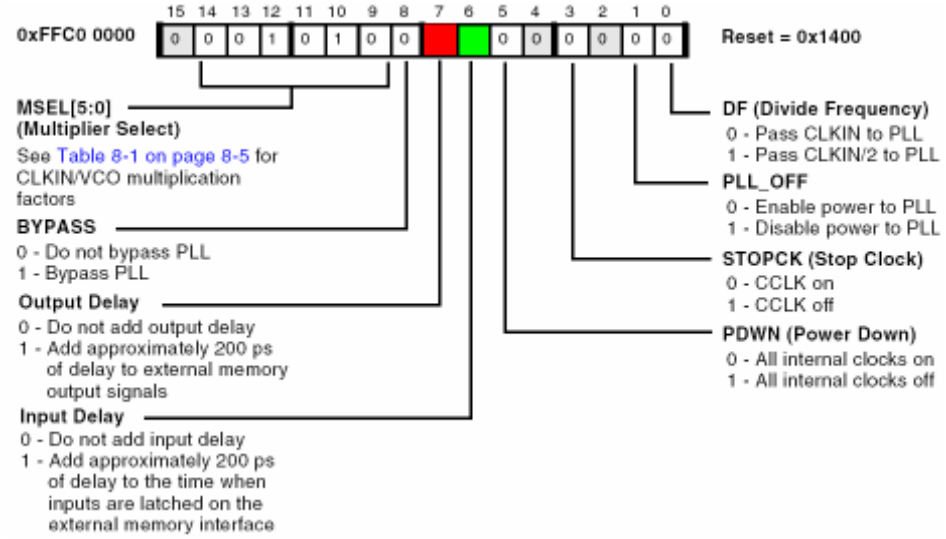


图4. PLL_CTL寄存器（从硬件参考复制）

位6和位7（图4）为SDRAM信号提供延迟，当违反写访问的时序指标时可应用输出延迟，这将使数据延迟保持200ps。当违反读访问的时序指标时需要应用输入延迟，将使锁存的输入信号延迟200ps。

t_{SCLK} CLKOUT周期

t_{SSDAT} CLKOUT前的数据建立时间（BF数据手册）

t_{AC} 从CLK开始的访问时间（SDRAM数据手册）

t_{DH} 输入保持时间（SDRAM数据手册）

t_{HSDAT} CLKOUT后的数据保持时间（BF数据手册）

t_{OH} 输出保持时间（SDRAM数据手册）

t_{OLZ} 从CLK开始的输出高阻延迟时间（SDRAM数据手册）

t_{OHZ} 从CLK开始，从高阻变为信号的延迟时间（SDRAM数据手册）

1.5.1 Blackfin输出/SDRAM输入等式（写）

对于SDRAM输入，下面的等式成立：

$t_{DH} < t_{HSDAT}$

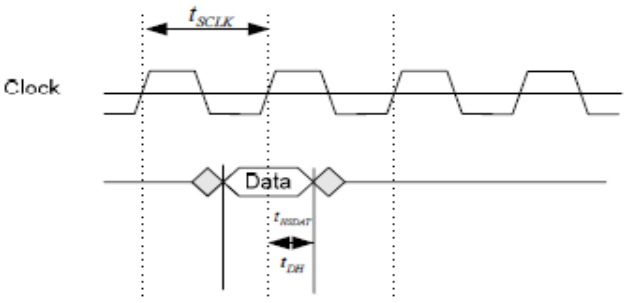


图5. 时序

1.5.2 Blackfin输入/SDRAM输出等式（读）

这种情况专门用于将SDRAM时钟设置为133MHz，此时时序参数达到了性能指标的极限。

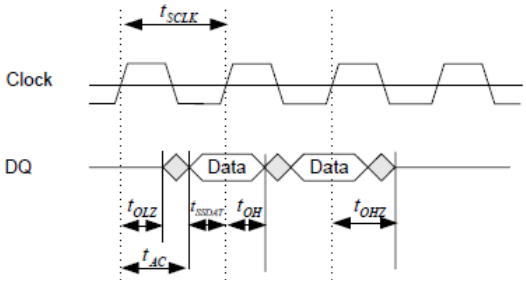


图6. 时序

要验证读操作处理的正确性，应保证$t_{SCLK}>t_{SSDAT}+t_{AC}$。

2 SDRAM初始化

SDRAM初始化将影响应用程序的性能和SDRAM的功耗，故期望对SDRAM设置有更好的理解。本节说明运行应用程序时如何设置EBIU（外部总线接口单元）寄存器。要初始化SDRAM，可用以下方法：

- 利用仿真器和VisualDSP++®.XML文件初始化SDRAM
- 通过在应用程序中的设置寄存器初始化SDRAM
- 由VisualDSP++系统服务模型初始化SDRAM
- 在加载实际应用程序前，利用初始化文件初始化SDRAM
- 利用OTP（单次可编程）存储器中的值初始化SDRAM

2.1 利用仿真器和VisualDSP++.XML文件初始化SDRAM

在软件开发的初级阶段，通过在线仿真器（ICE）加载应用软件，当将应用程序载入到处理器时，仿真器可自动设置EBIU寄存器。

要使能该功能，点击VisualDSP++的Setting菜单，选择Target Options。在弹出的Target Options对话框（图7），选择Use XML reset values选项。

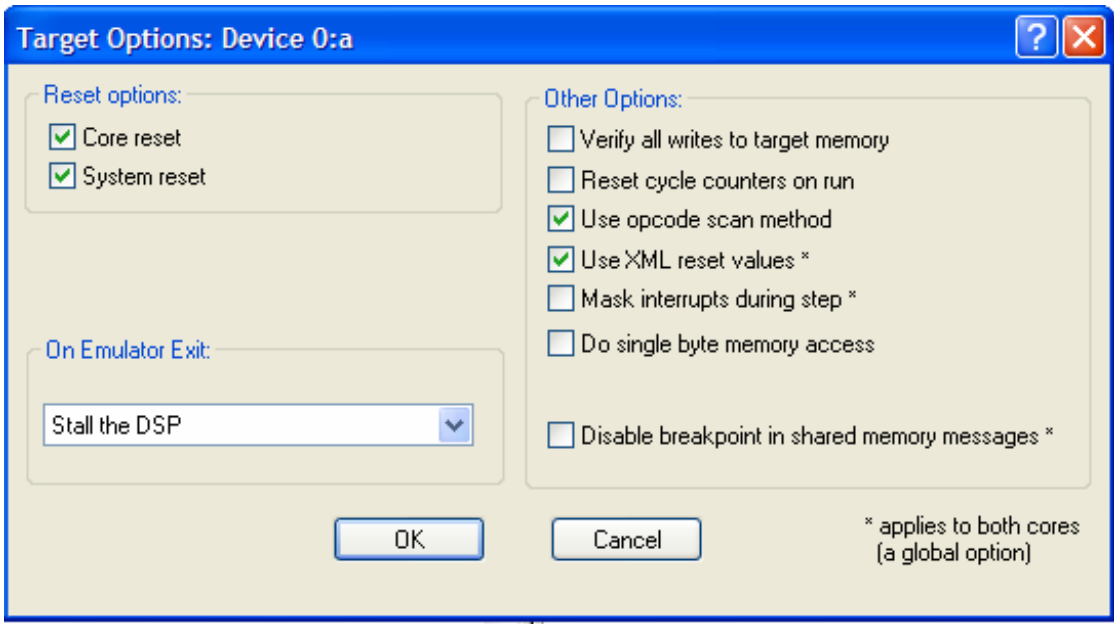


图7. Target Options对话框

VisualDSP++4.5版或更低版本，初始化值可由名为ADSP-BF5XX-proc.xml（其中XX代表处理器结构；如：ADSP-BF537-proc.xml或ADSP-BF561-proc.xml）的.xml（可扩展标记语言）文件得到。这些文件保存在<install-path>\Analog Devices\VisualDSP X.X\System\ArchDef目录中。配置EBIU的设置时用户可改变这些参数值。图8给出了处理器.XML文件的一部分。更多信息参见Visual++帮助中的"定制板支持"。

```
<!--  ********************************************  -->
<!--  Register resets used by emulator              -->
<!--  ********************************************  -->
- <register-reset-definitions>
    <register name="EBIU_SDRRC" reset-value="0x03A0" core="Common" />
    <register name="EBIU_SDBCTL" reset-value="0x25" core="Common" />
    <register name="EBIU_SDGCTL" reset-value="0x0091998d" core="Common" />
    <register name="EBIU_AMGCTL" reset-value="0xff" core="Common" />
  </register-reset-definitions>
  <!--  ************************************************************  -->
  <!--  ******* Customizations: If you make changes to this file, make a  -->
  <!--  ******* copy of your customized file to facilitate future upgrades.  -->
  <!--  ************************************************************  -->
</visualdsp-proc-xml>
```

图8. ADSP-BF5xx-proc.xml文件的一部分（Visual++4.5）

只在启动时才读取处理器.xml文件，故VisualDSP++ IDDE运行过程中对其任何编辑都无效。当然，应确定在使用VisualDSP++开发工具前已设置好参数值。当运行VisualDSP++工具时，.xml文件用于设置ADI公司EZ-KIT Lite®开发板相应SDRAM的参数值。

对于VisualDSP++5.0版本或更高版本，不建议直接修改ArchDef文件夹中的.xml文件，而应使用以下章节说明的定制板支持。

利用VisualDSP++5.0版本或更高版本中的定制板支持，可在应用程序中很简单的设置参数的复位值：开发者可以有多组复位值选择，并可变换使用这些组，而不需重新启动VisualDSP++工具。打开文本编辑器，并将如下代码放入文件并保存。在本实例中（列表1），文件名为 My_custom_board_reset_settings.xml:

```
<?xml version="1.0" standalone="yes"?>

<custom-visualdsp-proc-xml
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="\Program Files\Analog Devices\VisualDSP
5.0\System\ArchDef\ADSP-custom-board.xsd"
     processor-family="BLACKFIN"
     file="My custom board reset settings.xml">

<!-- ************************************************************************ -->
<!-- ******* My custom board reset settings.xml                               -->
<!-- ************************************************************************ -->

<custom-register-reset-definitions>

     <register name="EBIU_SDRRC" reset-value="0x03A0" core="Common" />
     <register name="EBIU_SDBCTL" reset-value="0x25" core="Common" />
     <register name="EBIU_SDGCTL" reset-value="0x0091998D" core="Common" />

</custom-register-reset-definitions>
</custom-visualdsp-proc-xml>
```

列表1. 定制板复位设置值实例（VisualDSP++ 5.0）

要使能VisualDSP++开发工具中的这一特性，只需执行以下步骤：

1. 在Setting菜单中选择Session。
2. 在Setion Setting对话框中选择Enable customizations。
3. 在Custom board support file name中定位到名为“My_custom_board_reset_settings.xml”的文件，该文件包含了SDRAM的复位值。

每当处理器由VisualDSP++工具进行复位时，这些参数复位值将设置到寄存器中。

2.2 使用存储器映射寄存器初始化SDRAM

初始化SDRAM控制器的经典方法是直接将复位值分配到存储器映射寄存器（MMR）中。

比如，以可移动SDRAM数据手册[1]为例。

[1] 三星 K4M56163 R(B)N/G/L/F 可移动SDRAM

• 64ms refresh period (8K cycle).

Part No.	Max Freq.	Interface	Package
K4M56163LG-R(B)N/G/L/F75	133MHz(CL3), 111MHz(CL2)	LVCMOS	54 FBGA Pb (Pb Free)
K4M56163LG-R(B)N/G/L//F1H	111MHz(CL2)		
K4M56163LG-R(B)N/G/L/F1L	111MHz(CL=3)*1, 83MHz(CL2)		

Parameter	Symbol	Version			Unit	Note
		-75	-1H	-1L		
Row active to row active delay	tRRD(min)	15	18	18	ns	1
$\overline{RAS}$ to $\overline{CAS}$ delay	tRCD(min)	18	18	24	ns	1
Row precharge time	tRP(min)	18	18	24	ns	1
Row active time	tRAS(min)	45	50	60	ns	1
	tRAS(max)	100			us	
Row cycle time	tRC(min)	63	68	84	ns	1,2
Last data in to row precharge	tRDL(min)	2			CLK	3
Last data in to Active delay	tDAL(min)	tRDL + tRP			-	4

图9. 实例：部分可移动SDRAM的数据手册

Address configuration

Organization	Bank	Row	Column Address
16Mx16	BA0,BA1	A0 - A12	A0 - A8

图10. 数据手册中的地址配置说明

在设置SDRAM寄存器前，必须先配置PLL。本实例的系统时钟为133MHz，因此，可首先计算得刷新率：

$$\mathrm{RDIV} = \frac{f_{\mathrm{SCLK}} \cdot t_{\mathrm{REF}}}{\mathrm{NRA}} - \left(t_{\mathrm{RAS}} + t_{RP}\right)$$

由数据手册（-75），可得到以下信息：
f_{SCLK}=133MHz
t_{REF}=64ms
NRA=8192
在133MHz时t_{RAS}=45ns：t_{RAS}=6个周期
在133MHz时t_{RP}=18ns：t_{RP}=3个周期

$$\mathrm{RDIV} = \frac{133 \cdot 10^{6} \cdot 64 \cdot 10^{-3}}{8192} - (6+3) = 1030.0625 \approx 1030, \text{十六进制为:0x406}$$

进一步，可计算出以下值：
在133MHz时t_{RCD} =18ns：t_{RCD}=3个周期
三星公司文档中，将t_{WR}称为t_{RDL}。
t_{WR}=t_{RDL}=2个周期

然后，我们必须保证所有上面算出来的时间参数必须在SDRAM手册的规格范围此外，还必须使能扩展寄存器，设置温度和部分自刷新。PASR可以设置为如下值：

$t_{XSR} = t_{RAS} + t_{PR}$ -> 6 cycles + 3 cycles = 9 cycles at 133 MHz : 67.5 ns

$t_{RRD} = t_{RCD} + 1$ -> 3 cycles + 1 cycle = 4 cycles at 133 MHz : 30 ns

$t_{RC} = t_{RAS} + t_{PR}$ -> 6 cycles + 3 cycles = 9 cycles at 133 MHz : 67.5 ns

$t_{RFC} = t_{RAS} + t_{PR}$ -> 6 cycles + 3 cycles = 9 cycles at 133 MHz : 67.5 ns

If we compare these values with the SDRAM's data sheet, we notice we are within the specifications.

Additionally, we have to enable the extended register to set the temperature and the partial self-refresh. The PASR can be set to following settings:

- **PASR_ALL** – 所有四个SDRAM存储块都以自刷新方式刷新
- **PASR_B0_B1** – SDRAM存储块0和1以自刷新方式刷新
- **PASR_B0** – 仅SDRAM存储块0以自刷新方式刷新

列表2说明C语言初始化程序。

```
//SDRAM Refresh Rate Setting
    *pEBIU_SDRRC = 0x406;
//SDRAM Memory Bank Control Register
    *pEBIU_SDBCTL =          EBCAW_9    | //Page size 512
                             EBSZ_64    | //64 MB of SDRAM
                             EBE;         //SDRAM enable
//SDRAM Memory Global Control Register
      *pEBIU_SDGCTL    = ~CDDBG      & // Control disable during bus grant off
                         ~FBBRW      & // Fast back to back read to write off
                         ~EBUFE      & // External buffering enabled off
                         ~SRFS       & // Self-refresh setting off
                         ~PSM        & // Powerup sequence mode (PSM) first
                         ~PUPSD      & // Powerup start delay (PUPSD) off
                         TCSR        | // Temperature compensated self-refresh at 85
                         EMREN       | // Extended mode register enabled on
                         PSS         | // Powerup sequence start enable (PSSE) on
                         TWR_2       | // Write to precharge delay TWR = 2 (14-15 ns)
                         TRCD_3      | // RAS to CAS delay TRCD =3 (15-20ns)
                         TRP_3       | // Bank precharge delay TRP = 2 (15-20ns)
                         TRAS_6      | // Bank activate command delay TRAS = 4
                         PASR_B0     | // Partial array self refresh Only SDRAM Bank0
                         CL_3        | // CAS latency
                         SCTLE       ; // SDRAM clock enable
```

列表2. 通过寄存器初始化SDRAM

汇编语言的程序代码可在初始化程序代码（第2章）中找到。

如需要在后面的应用中改变PLL的设置该怎么办呢？为了达到最佳的存储器性能，必须同时修改设置。如果不介意存储器性能，应计算出最坏情况下的值。

本EE文件附有ADSP-BF537 EZ-KIT Lite (汇编和C语言) 以及ADSP-BF561 EZ-KIT Lite (C语言)上的SDRAM初始化示例。

2.3 利用系统服务初始化

另一种初始化SDRAM的方法是在应用程序内处理设置问题，而改变EBIU设置最合适的方式就是利用系统服务。利用EBIU系统服务的主要优点在于不用计算就可完成EBIU设置。第二个优点是EBIU的改变与PLL的改变是一致的，因此不需要考虑实际的系统时钟频率。

初始化由函数adi_ebiu_init完成，仍以上面相同的SDRAM为例说明。

要利用系统服务实现初始化，必须将SDRAM参数传递给系统，故使用名为ADI_EBIU_COMMAND_PAIR的命令结构体（表1）。

为了在本例中初始化该结构体，将变量（对于-75）设置为列表3所示。

```
// set the sdram to 64 MB
ADI_EBIU_SDRAM_BANK_VALUE bank_size = {0, ADI_EBIU_SDRAM_BANK_64MB};
//set the sdram to 9 bit Column Address Width (like we see in the Address
//Configuration)
// A0-A8 -> 9bits Column Address
ADI_EBIU_SDRAM_BANK_VALUE bank_caw = {0,
(ADI_EBIU_SDRAM_BANK_SIZE)ADI_EBIU_SDRAM_BANK_COL_9BIT}; // 9bit CAW
//set the twr to 2 sclk +  no time
ADI_EBIU_TIMING_VALUE MyTWR = {          2,          // 2 cycle
{0, ADI_EBIU_TIMING_UNIT_PICOSEC}};  // 0 ns
//set refresh rate to 64ms at 8192 rows as seen in the data sheet
ADI_EBIU_TIMING_VALUE Refresh = {                    // SDRAM Refresh rate:
                                8192,                // 8192 cycles
{64, ADI_EBIU_TIMING_UNIT_MILLISEC }}; // 64ms
//set TRAS to 45ns
ADI_EBIU_TIME MyTRAS = {45, ADI_EBIU_TIMING_UNIT_NANOSEC};
//set TRP to 18ns
ADI_EBIU_TIME MyTRP  = {18, ADI_EBIU_TIMING_UNIT_NANOSEC};
//set TRCD to 18ns
ADI_EBIU_TIME MyTRCD = {18, ADI_EBIU_TIMING_UNIT_NANOSEC};
//set CAS threshold frequency
u32 MyCAS = 133;
//Enable Extended Mode Register because we are using Mobile SDRAM
ADI_EBIU_SDRAM_EMREN MyEMREN = ADI_EBIU_SDRAM_EMREN_ENABLE;
//Refresh only the first bank
ADI_EBIU_PASR  MyPASR = ADI_EBIU_PASR_INT0_ONLY;
//Temperature Compensation at 85°C
ADI_EBIU_SDRAM_TCSR  MyTCSR = ADI_EBIU_SDRAM_TCSR_85DEG;
//we don't have any registered buffer
ADI_EBIU_SDRAM_EBUFE MyEBUFE = ADI_EBIU_SDRAM_EBUFE_DISABLE;
//no fast-back-to-back read/write
ADI_EBIU_SDRAM_FBBRW MyFBBRW = ADI_EBIU_SDRAM_FBBRW_DISABLE;
//Do not disable the control during bus grant
ADI_EBIU_SDRAM_CDDBG MyCDDBG = ADI_EBIU_SDRAM_CDDBG_DISABLE;
//We don't need any delay at Power Up
ADI_EBIU_SDRAM_PUPSD MyPUPSD = ADI_EBIU_SDRAM_PUPSD_NODELAY;
//Do first the refresh
ADI_EBIU_SDRAM_PSM MyPSM = ADI_EBIU_SDRAM_PSM_REFRESH_FIRST;
```

列表3. EBIU结构体的初始化

Description	Command	Value Type
Bank Size	ADI_EBIU_CMD_SET_SDRAM_SIZE	ADI_EBIU_SDRAM_BANK_VALUE
Bank col. address width	ADI_EBIU_CMD_SET_SDRAM_BANK_COLUMN_WIDTH	ADI_EBIU_SDRAM_BANK_VALUE
CAS latency	ADI_EBIU_CMD_SET_SDRAM_CL_THRESHOLD	u32
TRAS	ADI_EBIU_CMD_SET_SDRAM_TRASMIN	ADI_EBIU_TIME
TRP	ADI_EBIU_CMD_SET_SDRAM_TRPMIN	ADI_EBIU_TIME
TRCP	ADI_EBIU_CMD_SET_SDRAM_TRCDMIN	ADI_EBIU_TIME
TWR	ADI_EBIU_CMD_SET_SDRAM_TWRMIN	ADI_EBIU_TIMING_VALUE
Refresh period	ADI_EBIU_CMD_SET_SDRAM_REFRESH	ADI_EBIU_TIMING_VALUE
Extended Mode Enable	ADI_EBIU_CMD_SET_SDRAM_EMREN	ADI_EBIU_SDRAM_EMREN
PASR*	ADI_EBIU_CMD_SET_SDRAM_PASR	ADI_EBIU_SDRAM_PASR
TCSR**	ADI_EBIU_CMD_SET_SDRAM_TCSR	ADI_EBIU_SDRAM_TCSR
External Buffer are used	ADI_EBIU_CMD_SET_SDRAM_EBUFE	ADI_EBIU_SDRAM_EBUFE
Fast Back-to-Back-Read/Write	ADI_EBIU_CMD_SET_SDRAM_FBBRW	ADI_EBIU_SDRAM_FBBRW
Control disable during bus grant	ADI_EBIU_CMD_SET_SDRAM_CDDBG	ADI_EBIU_SDRAM_CDDBG
Power Up Start Delay	ADI_EBIU_CMD_SET_SDRAM_PUPSD	ADI_EBIU_SDRAM_PUPSD
SDRAM powerup sequence	ADI_EBIU_CMD_SET_SDRAM_PSM	ADI_EBIU_SDRAM_PSM

表1. EBIU命令概述

参数设置完成后，需要初始化服务。为此，将参数捆绑到命令表中，随后再调用函数adi_EBIU_init。

```
ADI_EBIU_COMMAND_PAIR Sdram_Values[] = {

        { ADI_EBIU_CMD_SET_SDRAM_BANK_SIZE,          (void*)&bank_size  },
        { ADI_EBIU_CMD_SET_SDRAM_BANK_COL_WIDTH,     (void*)&bank_caw   },
        { ADI_EBIU_CMD_SET_SDRAM_CL_THRESHOLD,       (void*)&MyCAS      },
        { ADI_EBIU_CMD_SET_SDRAM_TRASMIN,            (void*)&MyTRAS     },
        { ADI_EBIU_CMD_SET_SDRAM_TRPMIN,             (void*)&MyTRP      },
        { ADI_EBIU_CMD_SET_SDRAM_TRCDMIN,            (void*)&MyTRCD     },
        { ADI_EBIU_CMD_SET_SDRAM_TWRMIN,             (void*)&MyTWR      },
        { ADI_EBIU_CMD_SET_SDRAM_REFRESH,            (void*)&Refresh    },
        { ADI_EBIU_CMD_SET_SDRAM_FBBRW,              (void*)&MyFBBRW    },
        { ADI_EBIU_CMD_SET_SDRAM_EMREN,              (void*)&MyEMREN    },
        { ADI_EBIU_CMD_SET_SDRAM_PASR,               (void*)&MyPASR     },
        { ADI_EBIU_CMD_SET_SDRAM_TCSR,               (void*)&MyTCSR     },
        { ADI_EBIU_CMD_SET_SDRAM_EBUFE,              (void*)&MyEBUFE    },
        { ADI_EBIU_CMD_SET_SDRAM_CDDBG,              (void*)&MyCDDBG    },
        { ADI_EBIU_CMD_SET_SDRAM_PUPSD,              (void*)&MyPUPSD    },
        { ADI_EBIU_CMD_SET_SDRAM_PSM,                (void*)&MyPSM      },
        { ADI_EBIU_CMD_END, 0}
    };
//Init the service and ensure that the Refresh rate is reset if fsclk is changing
Result = adi_ebiu_Init( Sdram_Values, true);    // true enables automatic adjustment
```

列表4. EBIU初始化

在本EE文件附带的.ZIP文件中存有ADSP-BF537和ADSP-BF561处理器的系统服务用法的程序代码实例。

2.4 利用OTP存储器中的值初始化SDRAM

有单次可编程（OTP）存储器的设备（ADSP-BF52x和ADSP-BF54x处理器）提供了另外一种初始化EBIU寄存器设置的方法。在加载过程中，通过OTP中编程的值，加载ROM程序可以初始化EBIU设置。OTP包括一个称为预加载设置（PBS）的模块，该模块通过OTP_write函数进行编程。

PBS02L页面包含了EBIU设置值，更多信息概述参见用户处理器*硬件手册*。

在设置完EBIU控制寄存器后，加载过程访问SDRAM的0x00000000地址将初始化SDRAM。默认情况下，设置将执行一次读访问。由于读访问耗费的时间比写访问要多，因此该访问可由OTP_EBIU_POWERON_DUMMY_WRITE位（图10）进行自定义，这样就可将读访问更换为写访问，节省时间。使用该选项，必须保证在复位或进入休眠状态之前在该地址没有保存重要数据。

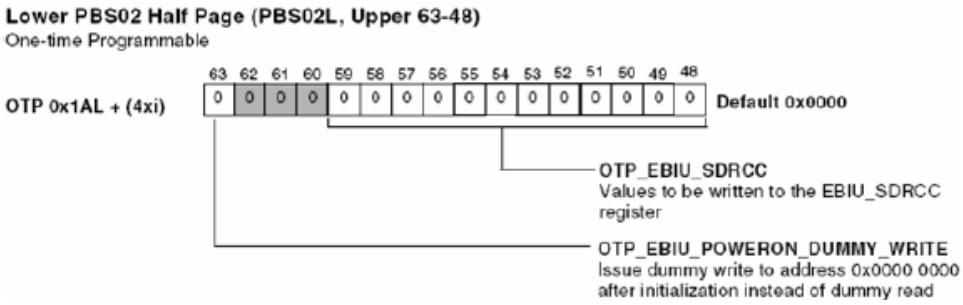


图11. 虚拟写选项

```
/* Declare local variable */
DU64 Data;
.
.
/* Enable the EBIU Init settings */
Data.h = 0x04000000     | //use the content of PBS02L for the EBIU
         ...            | //other settings
Data.l = ...

otp_write(0x18, OTP_LOWER_HALF, &Data);

/* Assign SDRAM values */

/* The low data byte is the EBIU_SDGCTL register */
Data.l =      ~CDDBG    & // Control disable during bus grant off
              ~FBBRW    & // Fast back to back read to write off
              ~EBUFE    & // External buffering enabled off
              ~SRFS     & // Self-refresh setting off
              ~PSM      & // Powerup sequence mode (PSM) first
              ~PUPSD    & // Powerup start delay (PUPSD) off
               TCSR     | // Temperature compensated self-refresh at 85
               EMREN    | // Extended mode register enabled on
               PSS      | // Powerup sequence start enable (PSSE) on
               TWR_2    | // Write to precharge delay TWR = 2 (14-15 ns)
               TRCD_3   | // RAS to CAS delay TRCD =3 (15-20ns)
               TRP_3    | // Bank precharge delay TRP = 2 (15-20ns)
               TRAS_6   | // Bank activate command delay TRAS = 4
               PASR_B0  | // Partial array self refresh Only SDRAM Bank0
               CL_3     | // CAS latency
               SCTLE;

/* The upper 32 bit are EBIU_SDBCTL(0-15) and EBIU_SDRRC (16-27). Further we will
   Set the dummy write bit (32), which will speed up the initialization          */

Data.h =      0x80000000      | // dummy write bit
              (0x406)<<16     | // Refresh rate
              EBCAW_9         | //Page size 512
              EBSZ_64         | //64 MB of SDRAM
              EBE;              //SDRAM enable

;
otp_write(0x1A, OTP_LOWER_HALF, &Data);
```

列表5. 通过PBS初始化

2.5 在加载应用程序前利用初始化代码初始化存储器

如果初始化时需要将指令和数据段放置到SDRAM，在载入应用程序之前，必须使用初始化文件初始化SDRAM。为此，需要打开工程并编写初始化文件代码，并将该工程编译成.DXE文件。初始化.DXE文件可通过Project Options对话框（Project：Load：Options页面）包含到用户实际应用程序的加载文件中。

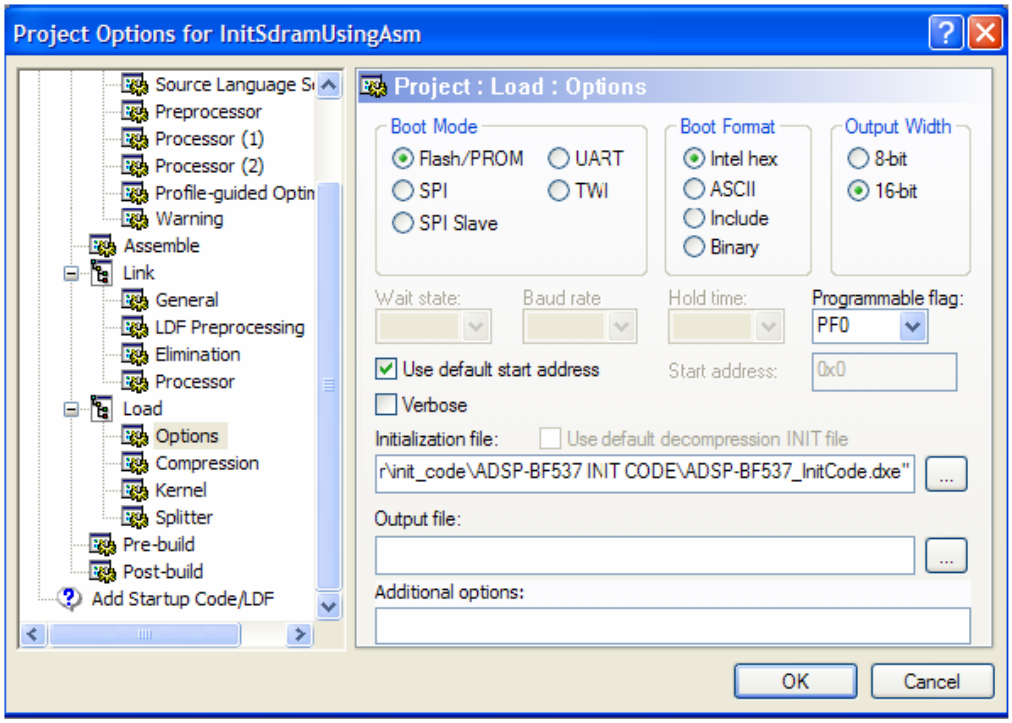


图12. 指定工程加载选项

列表6给出了如何实现初始化代码的实例。

```
#include <defBF537.h>
.section program;
    //save all registers on the stack
    [--SP] = ASTAT;
    ...
    [--SP] = L3;

//Setup the PLL Settings
    //Ensure no other interrupt will disturb us
    CLI R1;
    //Setup the voltage regulator
    P0.L = lo(VR_CTL);
    P0.H = hi(VR_CTL);
    R0 = 0x40DB (z);
    w[P0] = R0;

    //Wait the VR settings are done
    idle;
    //Setup the DIV of ssclk and cclk
    P0.L = lo(PLL_DIV);
    R0 = 0x0003 (z);
    w[P0] = R0;

    //Setup wait cycles until PLL is set
    P0.L = lo(PLL_LOCKCNT);
    R0 = 0x200 (z);
    w[P0] = R0;

    //Setup the PLL
    P0.L = lo(PLL_CTL);
    R0 = 0x2000 (z);
    w[P0] = R0;

    //Wait the PLL settings are done
    idle;
    //restore interrupts
    STI R1;

//Our program needs 4 MBs of RAM
    P0.L = lo(EBIU_AMGCTL);
    P0.H = hi(EBIU_AMGCTL); //Asynchronous Memory Global Control Register
//Uncomment your setting
    R0 = 0x00F0
//    |AMBEN_NONE(Z);          //No Asynchronous Memory
//    |AMBEN_B0(Z);            //1MB Asynchronous Memory
//    |AMBEN_B0_B1(Z);         //2MB Asynchronous Memory
//    |AMBEN_B0_B1_B2(Z);      //3MB Asynchronous Memory
     |AMBEN_ALL(Z);           //4MB Asynchronous Memory
W[P0] = R0;

//Setup the SDRAM
//SDRAM Refresh Rate Setting
    P0.H = hi(EBIU_SDRRC);
    P0.L = lo(EBIU_SDRRC);
    R0 = 0x406 (z);
    w[P0] = R0;
```

```
//SDRAM Memory Bank Control Register
    P0.H = hi(EBIU_SDBCTL);
    P0.L = lo(EBIU_SDBCTL);
    R0 =          EBCAW_9    | //Page size 512
                  EBSZ_64    | //64 MB of SDRAM
                  EBE;         //SDRAM enable
    w[P0] = R0;

//SDRAM Memory Global Control Register
    P0.H = hi(EBIU_SDGCTL);
    P0.L = lo(EBIU_SDGCTL);
    R0.H=         hi(     ~CDDBG      & // Control disable during bus grant off
                          ~FBBRW      & // Fast back to back read to write off
                          ~EBUFE      & // External buffering enabled off
                          ~SRFS       & // Self-refresh setting off
                          ~PSM        & // Powerup sequence mode (PSM) first
                          ~PUPSD      & // Powerup start delay (PUPSD) off
                          TCSR        | // Temperature compensated self-refresh at 85
                          EMREN       | // Extended mode register enabled on
                          PSS         | // Powerup sequence start enable (PSSE) on
                          TWR_2       | // Write to precharge delay TWR = 2 (14-15 ns)
                          TRCD_3      | // RAS to CAS delay TRCD =3 (15-20ns)
                          TRP_3       | // Bank precharge delay TRP = 2 (15-20ns)
                          TRAS_6      | // Bank activate command delay TRAS = 4
                          PASR_B0     | // Partial array self refresh Only SDRAM Bank0
                          CL_3        | // CAS latency
                          SCTLE       ); // SDRAM clock enable

    R0.L=     lo(         ~CDDBG      & // Control disable during bus grant off
                          ~FBBRW      & // Fast back to back read to write off
                          ~EBUFE      & // External buffering enabled off
                          ~SRFS       & // Self-refresh setting off
                          ~PSM        & // Powerup sequence mode (PSM) first
                          ~PUPSD      & // Powerup start delay (PUPSD) off
                          TCSR        | // Temperature compensated self-refresh at 85
                          EMREN       | // Extended mode register enabled on
                          PSS         | // Powerup sequence start enable (PSSE) on
                          TWR_2       | // Write to precharge delay TWR = 2 (14-15 ns)
                          TRCD_3      | // RAS to CAS delay TRCD =3 (15-20ns)
                          TRP_3       | // Bank precharge delay TRP = 2 (15-20ns)
                          TRAS_6      | // Bank activate command delay TRAS = 4
                          PASR_B0     | // Partial array self refresh Only SDRAM Bank0
                          CL_3        | // CAS latency
                          SCTLE     ) ; // SDRAM clock enable
    [P0] = R0;
    ssync;
    //Restore registers from the stack
    L3 = [SP++];
    ...
    ASTAT = [SP++];
    RTS;
```

列表6. 通过初始化文件初始化SDRAM

3 使用.LDF文件将数据和程序代码放入存储器

若在加载应用程序前已经初始化SDRAM（见利用OTP存储器中的值初始化SDRAM），则用户可以将数据和程序代码放入存储器段。为此，用户需在链接描述文件（.LDF）中定义一个存储区，并通知链接器有数据或程序代码将存入存储器。

假定需要将数据存入SDRAM存储块1，仔细阅读由VisualDSP++专家连接器向导（本例中的SDRAM为512MB）生成的标准链接描述文件（.LDF），就会发现内部SDRAM存储块是各自命名的：

```
MEMORY
{  MEM_SYS_MMRS        { TYPE(RAM) START(0xFFC00000) END(0xFFDFFFFF) WIDTH(8) }
   […]
   MEM_SDRAM0_BANK0        { TYPE(RAM) START(0x00000004) END(0x07ffffff) WIDTH(8) }
   MEM_SDRAM0_BANK1        { TYPE(RAM) START(0x08000000) END(0x0fffffff) WIDTH(8) }
   MEM_SDRAM0_BANK2        { TYPE(RAM) START(0x10000000) END(0x17ffffff) WIDTH(8) }
   MEM_SDRAM0_BANK3        { TYPE(RAM) START(0x18000000) END(0x1fffffff) WIDTH(8) }
} /* MEMORY */
```

列表7. 分配存储区

还会在“SECTIONS”区中再次发现标记为“MEM_SDRAM0_BANK1”的存储区。

```
PROCESSOR p0
{
   OUTPUT($COMMAND_LINE_OUTPUT_FILE)
   RESOLVE(start, 0xFFA00000)
   KEEP(start, _main)

   SECTIONS
   {     [...]

      sdram0_bank1
      {
         INPUT_SECTION_ALIGN(4)
         INPUT_SECTIONS($OBJECTS(sdram0) $LIBRARIES(sdram0))
         INPUT_SECTIONS($OBJECTS(sdram0_bank1) $LIBRARIES(sdram0_bank1))
         INPUT_SECTIONS($OBJECTS(sdram0_data) $LIBRARIES(sdram0_data))
         INPUT_SECTIONS($OBJECTS(cplb) $LIBRARIES(cplb))
         INPUT_SECTIONS($OBJECTS(data1) $LIBRARIES(data1))
         INPUT_SECTIONS($OBJECTS(voldata) $LIBRARIES(voldata))
         INPUT_SECTIONS($OBJECTS(constdata) $LIBRARIES(constdata))
         INPUT_SECTIONS($OBJECTS(cplb_data) $LIBRARIES(cplb_data))
         INPUT_SECTIONS($OBJECTS(.edt) $LIBRARIES(.edt))
         INPUT_SECTIONS($OBJECTS(.cht) $LIBRARIES(.cht))
     } > MEM_SDRAM0_BANK1

         [...]

   } /* SECTIONS */
} /* p0 */
```

列表8. 指定存储器段

可看到有几个库文件和目标文件映射到了该存储空间。用户也可以将目标文件或库映射到多个存储区中。此时，链接器将决定每个存储段分别放置哪一部分数据或程序代码。

如果用C/C++将数据明确地放入一个存储器段，可使用编译指示SECTION指令。本例中，考虑将全局变量x放入内部SDRAM存储块，则只将标号为sdram0_bank1的段映射到内部SDRAM存储块1，没有其他段。因此可以使用该标号映射变量。

```
//define a variable which lies in SDRAM Bank 1
#pragma
section ("sdram0_bank1") long int x = 0;
```

通过使用相同的程序代码和操作，也可以将一个函数原型分配到一个存储器段中。

```
//define a function prototype of a function which lies in SDRAM Bank 1
#pragma
section ("sdram0_bank1") void foo();
```

如在“提高系统的SDRAM性能”中所描述的那样，有时手动映射一个数据段也是有用的，这可以防止由于打开或关闭一个页面造成的延迟。因此，可以在.LDF文件中定义属于自己的存储器段：

```
   MEM_SDRAM0_BANK2        { TYPE(RAM) START(0x02000000) END(0x02ffffff) WIDTH(8) }
   MEM_SDRAM0_BANK3        { TYPE(RAM) START(0x03000800) END(0x03ffffff) WIDTH(8) }
   //define my own page
   MEM_SDRAM0_BANK3_PAGE0  { TYPE(RAM) START(0x03000000) END(0x030007FF) WIDTH(8) }
```

在链接描述文件中的SECTIONS部分，将把所有标号为MyDefinedMemory的目标文件链接到该存储空间。

```
sdram0_bank3_page_0
 {
   INPUT_SECTION_ALIGN(4)
   INPUT_SECTIONS($OBJECTS(MyDefinedMemory) )
 } > MEM_SDRAM0_BANK3_PAGE0
```

然后，在C/C++文件中，将数组放入该存储器段。

```
//define an array which lies in my own defined memory space (Bank 3 Page 0)
#pragma
section ("MyDefinedMemory") long int MyArray[100];
```

4 SDRAM硬件设计

由于SDRAM驱动频率高于50MHz，所以硬件布局必须满足高速设计的要求。目前，硬件设计已能满足多数EMI和EMC的相关标准。这些标准必须保证高频时的信号完整性，大部分标准是在低功率环境下设定的。因此，正确的印制电路板（PCB）设计就是一个关键因素。本节说明在PCB板上如何设计Blackfin处理器和SDRAM之间的连接。

4.1 连接SDRAM到Blackfin处理器（原理图）

Blackfin处理器提供了SDRAM无缝接口。Blackfin处理器支持电源在1.8V到3.3V之间的SDRAM。

Blackfin处理器的地址引脚没有正确连接到SDRAM上，这是经常出现的一种错误。

地址线必须按照以下说明进行正确连接。

4.1.1 SDAP-BF53x 系列处理器

- 连接Blackfin处理器的ADDR1和SDRAM的A0，ADDR2连A1，依此类推。
- 不用Blackfin的ADDR11；用SA10连接A10
- ADDR18连接到SDRAM的BA0
- ADDR19连接到SDRAM的BA1
- /ABE0连接到DQML引脚（对于16位SDRAM）或芯片的DQM（与D0-D7连接）
- /ABE1连接到DQMH引脚（对于16位SDRAM）或芯片的DQM（与D8-D15连接）

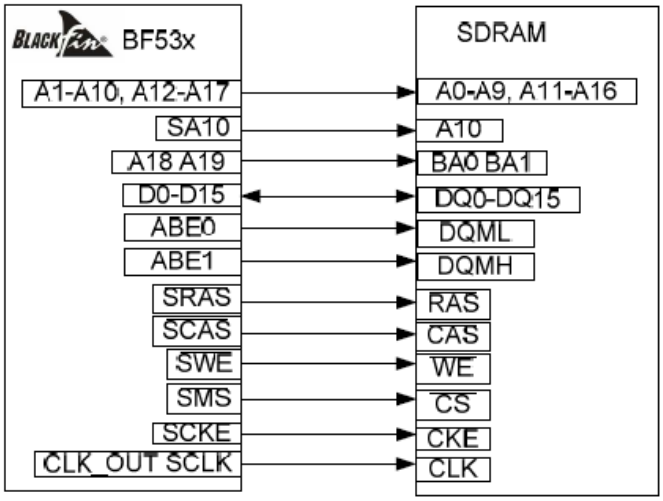


图13. ADSP-BF53x处理器：Blackfin处理器与SDRAM的连接

4.1.2 ADSP-BF561处理器（16位SDRAM）

- Blackfin处理器的ADDR2连接SDRAM的A1，ADDR3连A2，依此类推。
- Blackfin处理器的SDQM3连接SDRAM的A0
- 不用Blackfin的ADDR11； SA10连接A10
- ADDR18连接到SDRAM的BA0

- ADDR19连接到SDRAM的BA1
- SDQM0连接到DQML或DQM（见上述）
- SDQM1连接到DQMH或DQM（见上述）
  （在使用x16并有多个SDRAM时：每个芯片连一条/SMS线路）
  （在使用x8并有多于两个SDRAM时：每成对的两个使用一条/SMS线路）

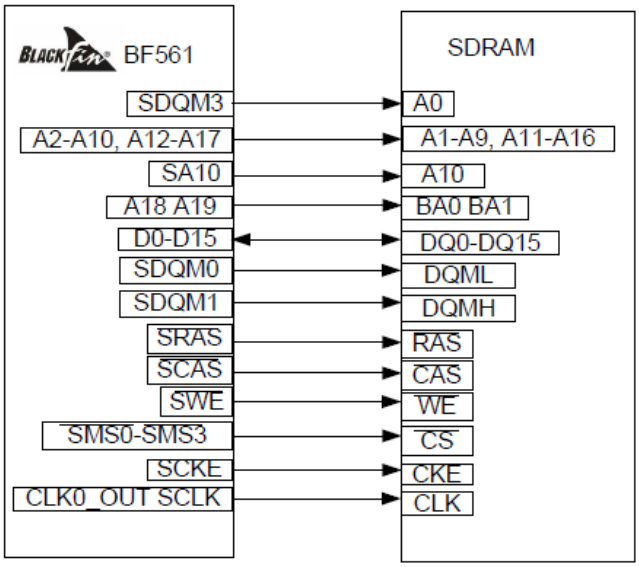


图14. ADSP-BF561处理器：Blackfin处理器与16位SDRAM的连接

4.1.3 ADSP-BF561处理器（32位SDRAM）

- Blackfin处理器的ADDR2连接到SDRAM的A0，ADDR3连接到A1，依此类推。
- 不用Blackfin的ADDR12； SA10连接到A10
- ADDR18连接到SDRAM的BA0
- ADDR19和连接到SDRAM的BA1
- SDQM0连接到SDRAM1（D0-D15）的DQML
- SDQM1连接到SDRAM1的DQMH
- SDQM2连接到SDRAM2（D16-D31）的DQML
- SDQM3连接到SDRAM2的DQMH

如果使用的是8位SDRAM

- SDQM0连接到SDRAM1的DQM（D0-D7）
- SDQM0连接到SDRAM2的DQM （D8-D15）
- SDQM0连接到SDRAM3的DQM（D16-D23）
- SDQM0连接到SDRAM4的DQM（D24-D31）（原文如此）

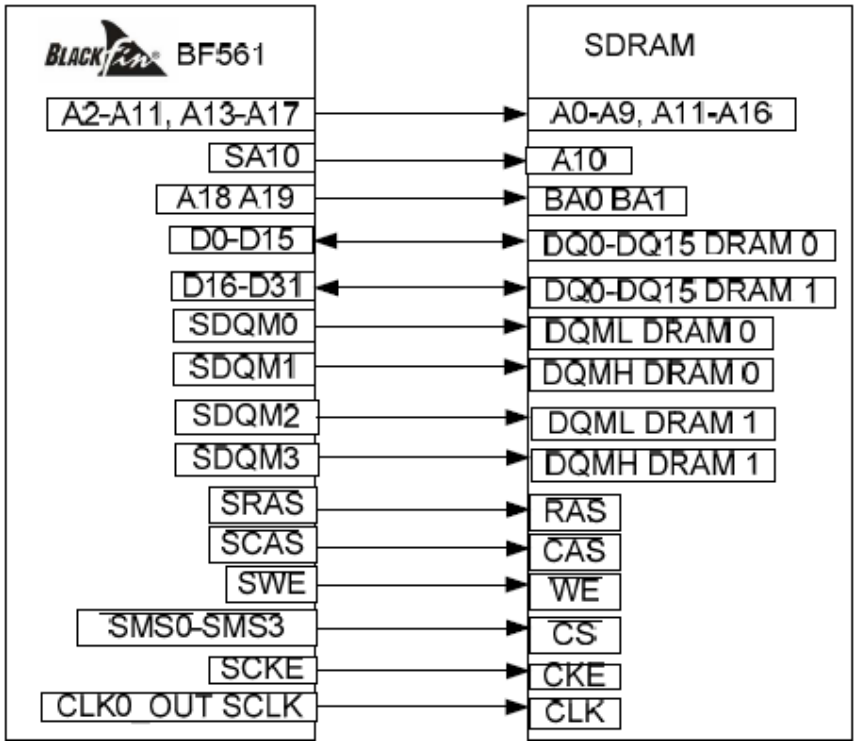


图15. ADSP-BF561处理器：Blackfin处理器与32位SDRAM的连接

当/BR引脚悬空时会出现另一个问题，Blackfin处理器将该信号解析为总线请求并用/BG信号应答，这会使并行总线出现无时限阻塞，因此总是在/BR引脚上放置一个上拉电阻器。

使用去耦电容器为SDRAM供电电源去耦。

靠近SDRAM的每个数据引脚都加一个串行电阻，下一部分将解释如何确定该阻值。

4.2 高速设计

SDRAM连接布局是一个关键因素，尤其在低功率设计中。本节将说明如何优化SDRAM布局设计以满足应用要求。最关键的连接是时钟、低位地址线、DQM和数据线。

4.2.1影响信号质量的因素

这一部分说明硬件设计影响的因素。

反射

如果连接线的阻抗值与接收机的输入电阻相当，则能量会被电阻器完全吸收。否则，传输的能量将被反射回来，通过叠加会干扰所期望的信号。

耦合

不同传输线中的电流会彼此影响。如果流经布线A的电流发生变化，则会引起磁场变化，而该磁场会耦合到布线B中，在布线B中生成一个电流，大小取决于耦合系数。感应电流的方向与布线A中的电流方向相反，两个信号走线相互影响就会加剧串扰）。但是，当该效应发生在信号线路和其回馈线路（GND）间时会产生增强效应。耦合的信号增强了回馈信号，而回馈信号又增强了原初始信号。

4.2.2 避免反射

计算特性阻抗，并增加正确的终端电阻。

有以下几种方法给传输线路加终端电阻：

- 串联终端
- 并联终端
- 戴维宁等效终端
- 二极管终端
- AC终端

下面将讲述SDRAM设备最重要的技术。

串联终端

SDRAM使用串联终端（图16）。将串联电阻器放在靠近发送器的输出引脚，这样做的好处是不会出现任何DC电流吸收（就像用户在使用一个并联终端），这对低功耗设计是十分必要的。但这样做的弊端是在接收端将会产生100%的反射，并送回到发送端。但因为SDRAM走线都比较短，且信号从Blackfin处理器到每个SDRAM芯片的路径长度基本相同，因此该效应就不会影响到SDRAM的功能。

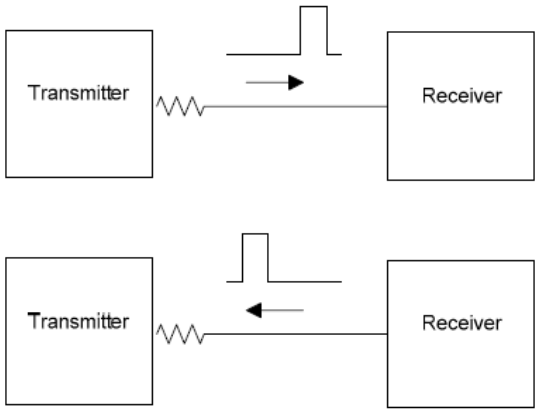


图16. 利用串联终端

为了避免驱动器（发送机）的二次反射，电阻器的值必须合适。必须对串联终端的阻值进行设置，使其与驱动器的输出阻抗之和，与走线的阻抗相当，可得到以下等式：

$R_S=Z_0 - Z_{OUT}$，其中 Z_{OUT}为发送器的输出阻抗

读命令比写命令更要求更严格，因此，应将数据线的串联电阻尽可能放置在接近SDRAM数据引脚的位置。

并联终端

另一种选择就是利用并联终端电阻（图17）。如前所述，对于标准SDRAM，如果遵照本章最后部分的设计指导，就不需要这种方法了。

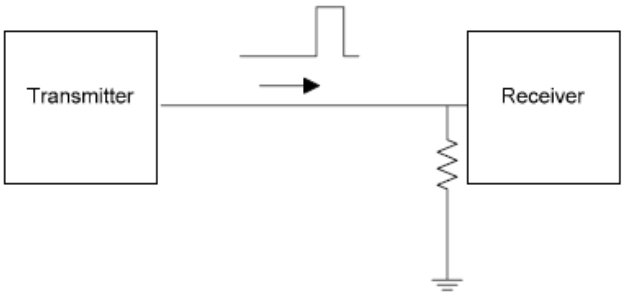


图17. 利用并联终端

反射系数 $\rho = \frac{Z_L - Z_0}{Z_L + Z_0}$

为了使反射尽可能低，并联终端电阻（Z_L）应该等于Z_0。

- 当然需要考虑是否需要用加终端电阻，因附加电阻将发射更多EMI。

Note that for ADSP-BF51x and ADSP-BF52x processors termination is mandatory. Consult the datasheet for more details.

4.3 SDRAM连接设计指导

关于EMC和信号完整性，推荐使用以下设计指导。在开始SDRAM PCB布局时，不要将所有信号都采取同样的处理，要考虑每个信号的重要性并将最关键信号的布线优先布局。推荐顺序如下：

1. 时钟分配
2. 数据线和DQM线，命令线包括SA10
3. 地址线
4. 其他信号（如CKE）

4.3.1 元件放置考虑

在设计PCB布局时应考虑以下几点：

- 将SDRAM芯片放在Blackfin处理器旁边
- 线路越短越好
- 当分配信号时，到所有设备的布线长度（图18）应（尽可能）相同。避免出现图19中的回环。

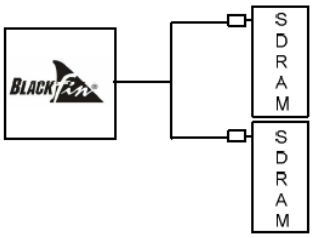


图18. 正确：分布线路并使各线路长度相同

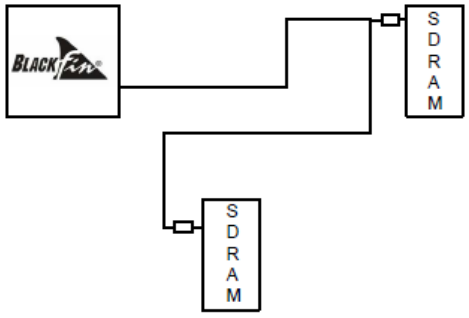


图19. 错误：线路回环

4.3.2 在线路边缘利用布线工具的圆滑功能

图20给出一个不圆滑的PCB线路边缘。图21给出了相同的线路边缘圆滑后的结果。

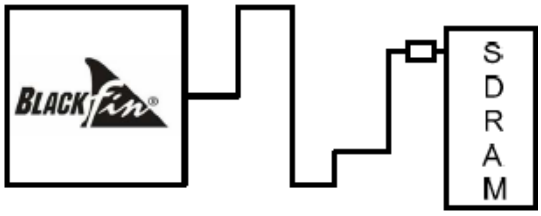


图20. 错误：线路边缘不圆滑

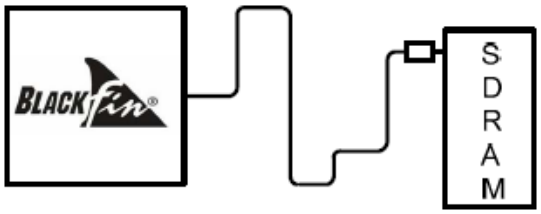


图21. 正确：线路边缘圆滑

4.3.3 将VCC和GND层之间的距离应尽可能小

图22给出了一个4层PCB，未对关键信号进行隔离。图23给出了4层PCB的适当隔离。

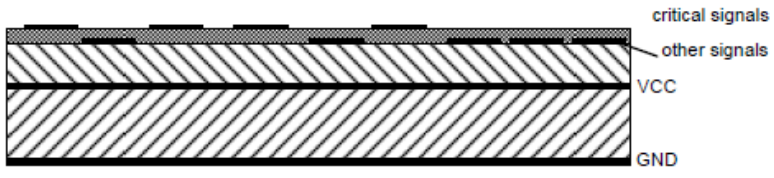


图22. 错误：VCC和GND层离得太远

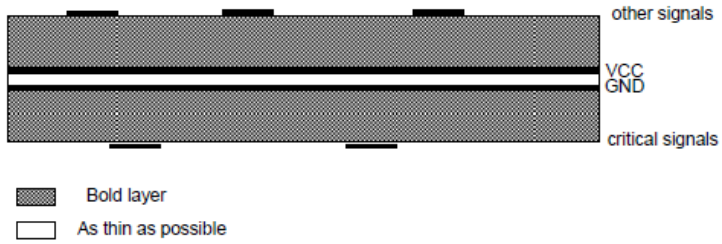


图23. 正确：VCC和GND层靠在一起

4.3.4 通过将关键信号放在内部层使其隔离

图24给出了一个6层PCB，其中的关键信号没有隔离。图25给出了适当隔离后的结果。

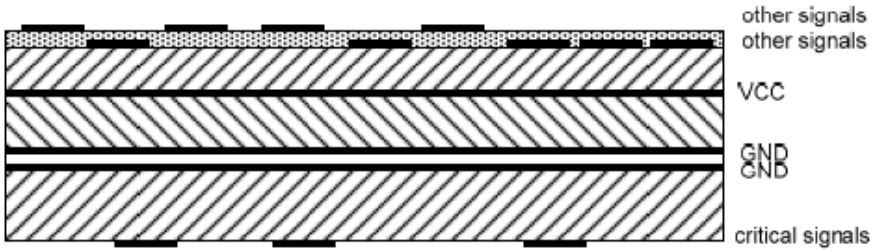


图24. 错误：关键信号没有进行适当隔离

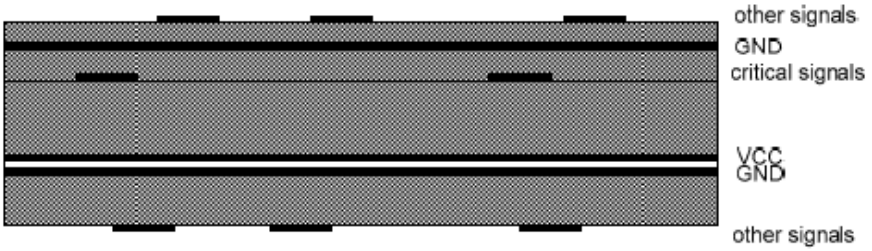


图25. 正确：关键信号与地层之间的距离最小化

4.3.5 串联电阻靠近SDRAM放置

图26给出了一个过孔太多的信号路径；该串联电阻离SDRAM太远。图27给出了没有过孔的短信号路径，经过一个紧靠SDRAM的串联电阻。

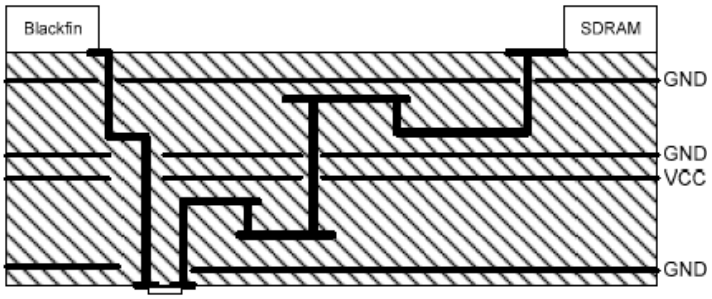


图26. 错误：关键信号路径中的过孔过多且串联电阻离得太远

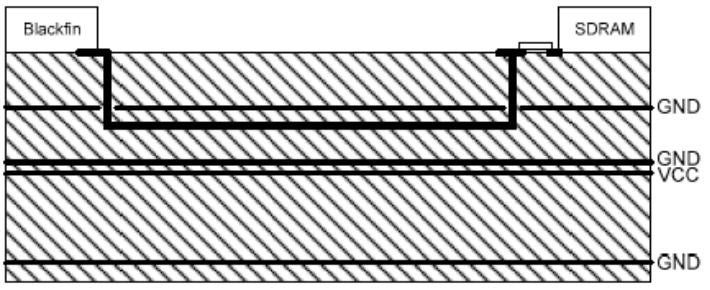


图27. 正确：该串联电阻靠近SDRAM

4.3.6 避免在GND层中出现沟道

图28给出了有沟道的GND层。图29中的GND层则避免了沟道。

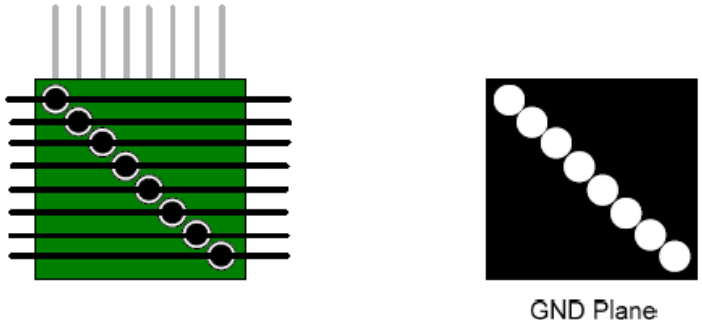


图28. 错误：地层有沟道

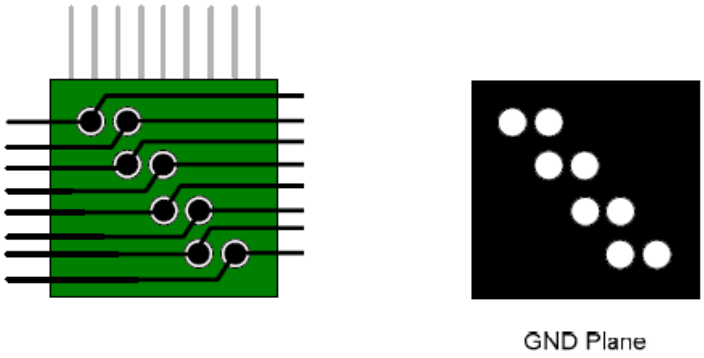


图29. 正确：地层无沟道

4.3.7 最小化来自过孔的逆流路径

如果不能避免从一层到另一层的方向改变，应最小化逆流路径。图30给出了一个信号路径从过孔向两个方向延伸。图31中的方向改变更好一些。

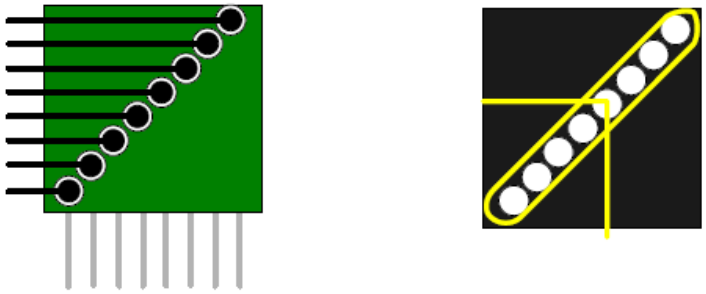


图30. 错误：层与层之间的方向改变

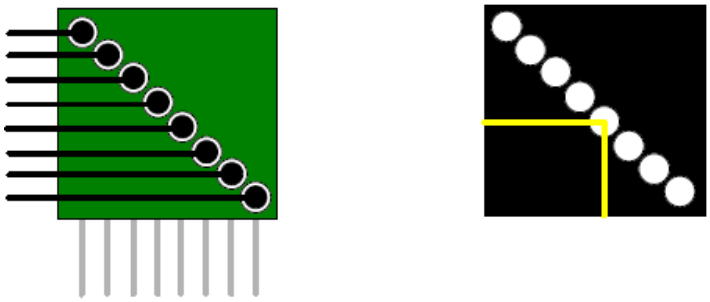


图31. 正确：避免直接改变

4.3.8 充分利用驱动能力配置功能

为了减少电磁辐射，可以降低EBIU管脚的驱动能力。但是必须保证在在一定时间要求内，EBIU管脚能够提供足够的电流来驱动。位字段00b表示低驱动能力，01b表示高驱动能力。具体信息可以参照芯片数据手册。

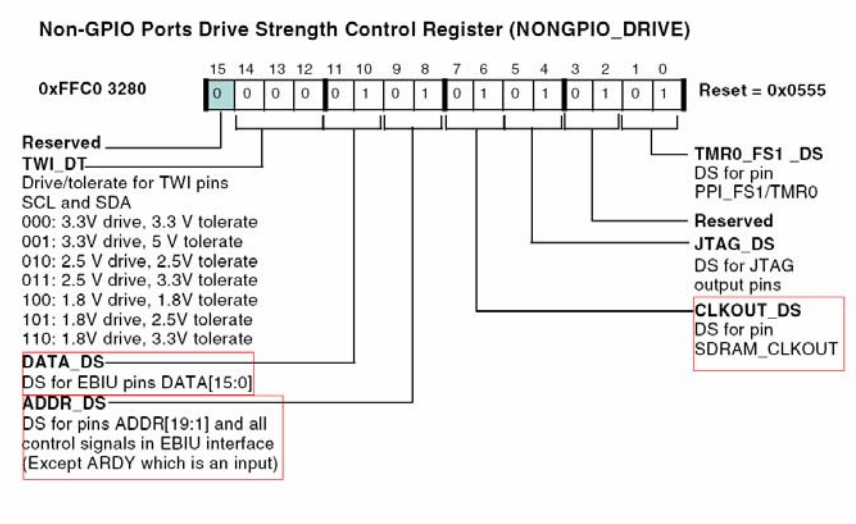


图32：BF52x的驱动控制能力配置寄存器

4.3.9 利用转换速率控制功能

减少电磁辐射一个更加平滑的方法。当改变到更慢的转换速率模式下时，再次确认是否还满足时间要求。00b表示比较快的转换速率，01b表述比较慢的转换速率。

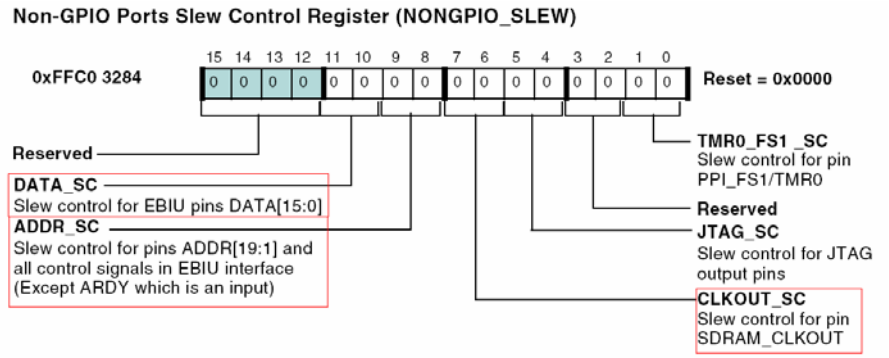
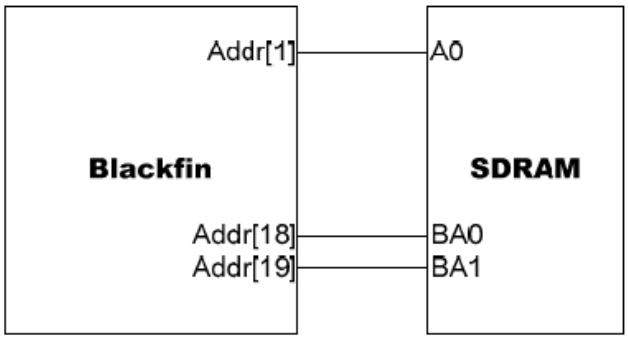


图33：BF52x的转换速率配置寄存器

5 使用容量小于16MB的SDRAM与Blackfin处理器连接

使用容量小于16MB（128Mbit）的SDRAM对于低功耗的应用尤为重要。本节为使用小于16MB的应用提供指导。

5.1 系统设置



硬件部分没有特别的设置，只要将地址线按照SDRAM硬件设计中的说明连接即可。

使用小于16MB的第一步就是设置EBIU_SDBCTL（SDRAM存储区控制）寄存器中的外部存储区的大小位，将其设为16M字节。这样就将Blackfin处理器的内部地址配置为16M字节，而地址空间将是不连续的，如图34所示。

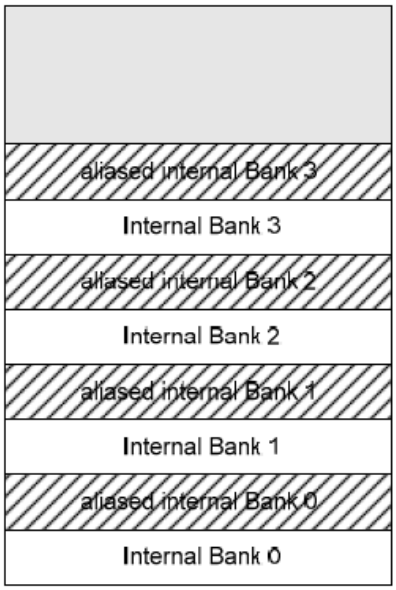


图34. 地址空间

“别名”的地址空间的内容为相应存储区的复件，对该存储区的每次写访问就是对“实际”存储区的一次写访问。链接描述文件（.LDF）中应考虑到这一情况，否则Blackfin处理器就会将指令和数据放入不存在的地址中。

Blackfin处理器不会检测到这种类型的地址错误，也没有能够检测一个地址是否可用的功能。在后面的应用程序中，处理器核试图从一个不存在的地址空间中读会出现错误，并得到随机数值，而将其解析的为有效的代码或数据。

5.2 修改.LDF文件

首先，要确保.LDF文件不会被专家链接器向导改变。因此，打开Project Options对话框（Project->Project Options）到Remove Startup Code/LDF页面并选择Leave the files in the project，but stop regenerating them选项（图35）。

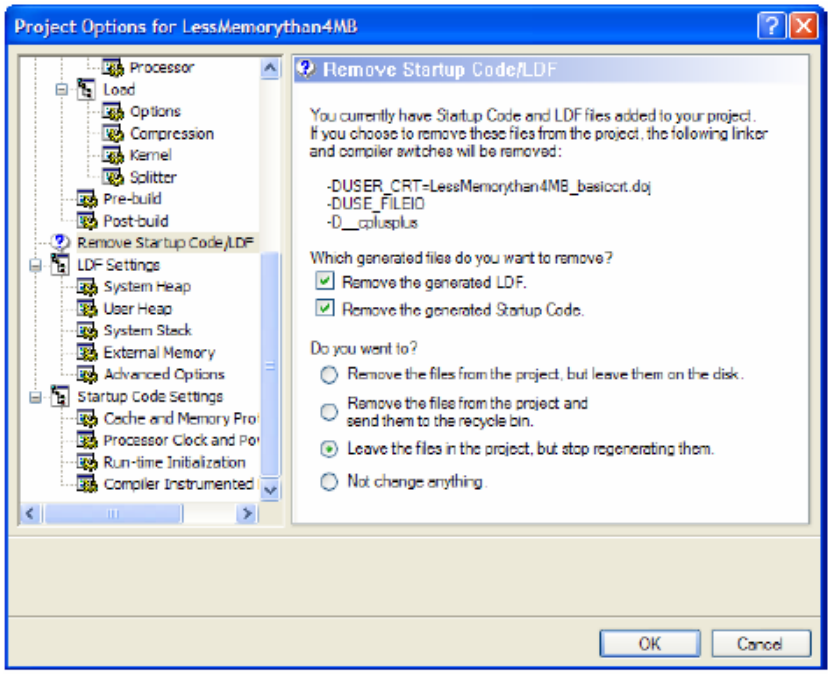


图35. 确保.LDF文件不会被改变

这样做就允许用户手动修改.LDF文件。

例如，如果用64Mbit的SDRAM（8MB），则链接描述文件的地址空间有：

```
MEM_SDRAM0_BANK0 { TYPE(RAM) START(0x00000000) END(0x001FFFFF) WIDTH(8) }
MEM_SDRAM0_BANK1 { TYPE(RAM) START(0x00400000) END(0x005FFFFF) WIDTH(8) }
MEM_SDRAM0_BANK2 { TYPE(RAM) START(0x00800000) END(0x009FFFFF) WIDTH(8) }
MEM_SDRAM0_BANK3 { TYPE(RAM) START(0x00C00000) END(0x00DFFFFF) WIDTH(8) }
```

图36. 确保.LDF文件没有改变

如图36所示，地址空间中有2MB的缝隙。

5.2.1 附注：背景

.LDF文件指定了代码和数据在存储器空间中的存放位置。EBIU设置对Blackfin处理器的SDRAM控制器进行配置，并指定其大小，时序和SDRAM特性。由于EBIU设置不能设置8MB的SDRAM，所以必须将SDRAM的大小设为16MB。这对寻址有什么影响呢？用一个列地址宽度（CAW）为10位的SDRAM，这就是说可寻址2^10 = 1024列。对每个一列和行地址，可寻址2个字节（对于x16 SDRAM）。现在，已将RAM设置为16MB（=16777216字节），即有宽度为13位的行地址（存储器大小/（数据宽度*2^地址）=16777216/（2字节*2^10）=8192=2^13），但是只用其8MB RAM，其行地址宽度为12，而控制器计算的行地址宽度为13，且行地址和列地址以时分复用方式发送到SDRAM。查看SDRAM，会发现有12条地址线和2条存储区地址线。但控制器计算出了13条并使用了13条地址线，因为第13条地址线没有连接，将对相同物理地址进行寻址，而该地址独立于行地址中位13的状态。这就是存储空间为什么被镜像的原因。

例如，行地址0x1000与对行地址0x0000访问的数据相同。当向“别名”存储空间放置信息时这就很关键，因为覆盖了其他地址空间中的信息。

因此，以上述方式定义.LDF文件，将不会在镜像RAM中放置任何信息：

```
MEM_SDRAM0_BANK0 { TYPE(RAM) START(0x00000000) END(0x001FFFFF) WIDTH(8) }
MEM_SDRAM0_BANK1 { TYPE(RAM) START(0x00400000) END(0x005FFFFF) WIDTH(8) }
MEM_SDRAM0_BANK2 { TYPE(RAM) START(0x00800000) END(0x009FFFFF) WIDTH(8) }
MEM_SDRAM0_BANK3 { TYPE(RAM) START(0x00C00000) END(0x00DFFFFF) WIDTH(8) }
```

5.3 有2个存储块的SDRAM

几种SDRAM只有两个存储块，硬件连接如图34所示。

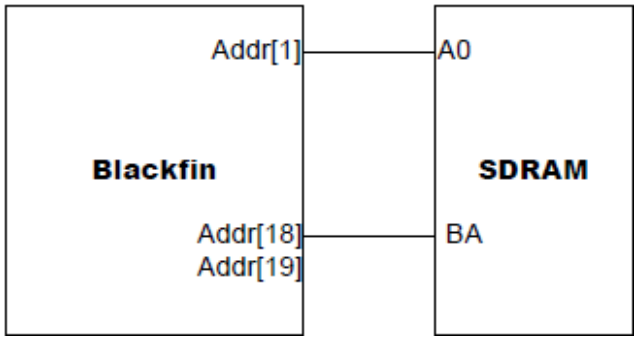


图37. 只有两个存储块的SDRAM的硬件连接

BA为SDRAM的存储块片选引脚，它应与Blackfin处理器的地址[18]连接。将地址[19]悬空，其它地址按照SDRAM硬件设计中的说明相连。将EBIU_SDBCTL（SDRAM存储区控制）寄存器设置为16M字节。逻辑地址空间将按图38进行分段。

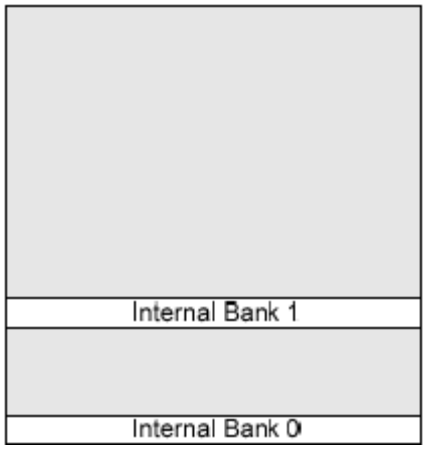


图38. 分段后的逻辑地址空间

如此，必须重新调整.LDF文件建立存储空间。步骤与上述相同。

图39给出了一个16Mbit（2MB）的SDRAM的存储空间。

```
MEM_SDRAM0_BANK0 { TYPE(RAM) START(0x00000000) END(0x000FFFFF) WIDTH(8) }
MEM_SDRAM0_BANK1 { TYPE(RAM) START(0x00400000) END(0x004FFFFF) WIDTH(8) }
```

图39. 例：16Mbit（2MB）的SDRAM的存储空间

如图所示，地址空间中有3MB的缝隙。

6 提高系统中SDRAM的性能

许多应用中，程序执行时间是个关键因素，数据和指令的存放对应用程序的处理速度有重要影响。本节内容将概述提高SDRAM性能的各种方法。关于系统方法，请参见*Blackfin处理器系统优化技术（EE-324）*[2]。

6.1 优化多存储块访问

通过激活命令打开页面或通过预充电命令关闭页面都是很耗时的，因此，减少页面转换就能获得更好的SDRAM性能。

多数应用程序中都有通过DMA方式将数据从一个数组复制到另一数组，但若两个数组都位于同一个存储块内时问题就出现了。但若这种情况发生在同一页面就不存在问题，而两个数组的大小若超出页面大小，DMA就要访问至少两个页面。

图40说明了DMA如何在一个存储器块中工作（单存储块访问）。

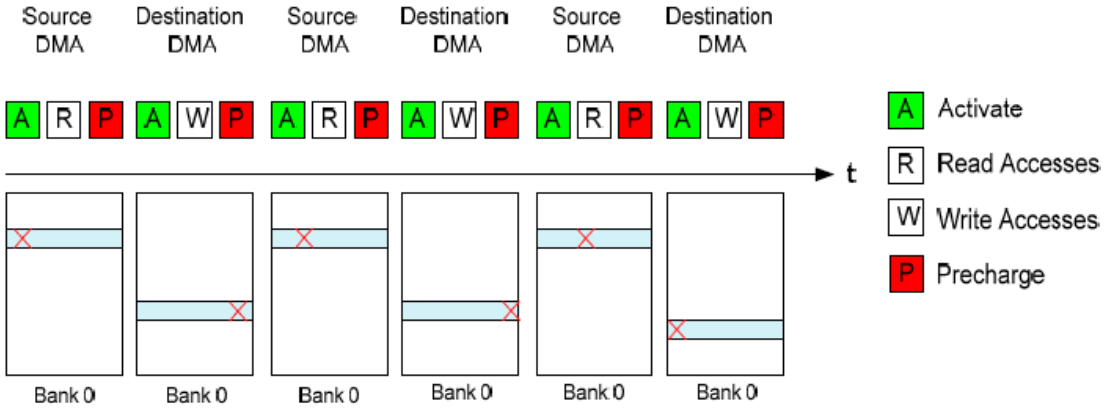


图40. 对单存储器块的访问

数据存放在同一存储器块的不同页面时，在源和目的DMA间的每次转换后都应执行一次激活和预充电命令。此外，还要考虑到处理器内核（或cache）也有可能访问存储器块。因此，每次读操作和写操作间都存在延时，且该延时通过内部DMA结构可放大：设计的DMA是一种反馈控制状态机，在特定的环境中会引入额外的等待状态。

为了避免这种耗时情况的发生，应以允许存储器块内间DMA复制的方式来组织存储器。图41说明了这种方法。

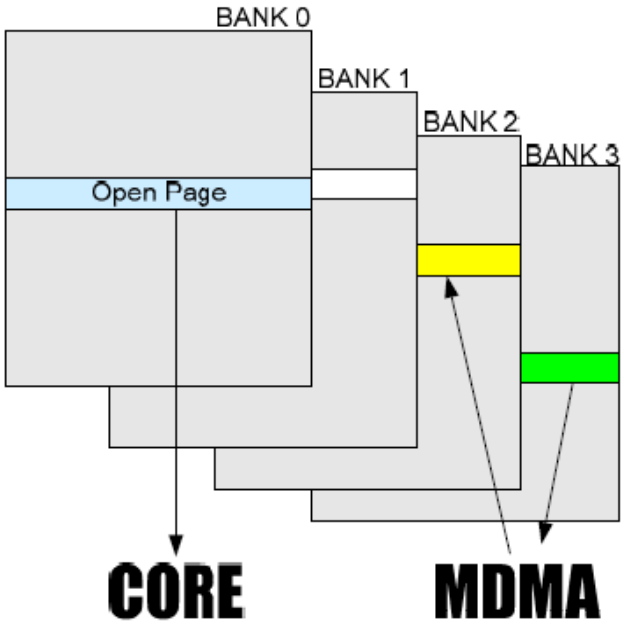


图41. 通过DMA和处理器核的多存储块访问方法

处理器核从存储器块0取程序代码，而在存储器块3和存储器块2之间进行MDMA传输。图42说明了两个存储器块间的数据传输过程，如图所示，预充电命令和激活命令的数目明显减少。如前所述，由于预充电和激活命令是耗时的处理过程，因此该技术节省了大量的时间。

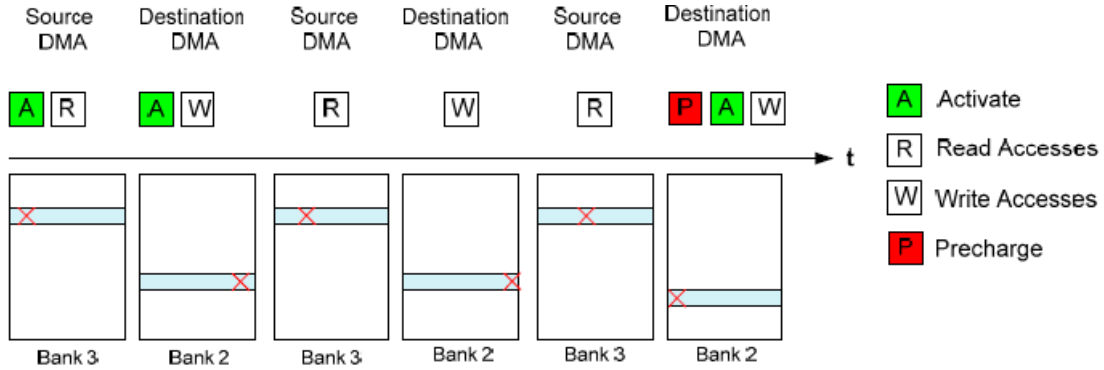


图42. 智能存储器块访问

6.2 优化的页面访问

本节说明如何改善页面访问。如果所有访问都维持在一个页面中，此时就需要更多的激活和预充电命令（除了那些发布刷新的命令）。以下说明了如何独立地访问页面。

打开Project Options对话框的Remove Startup Code/LDF页面（Project -> Project Options），选择Leave the files in the project, but stop regenerating them选项。同样，选择Remove the generated LDF复选框。要通过点击OK按钮确认所选的项。

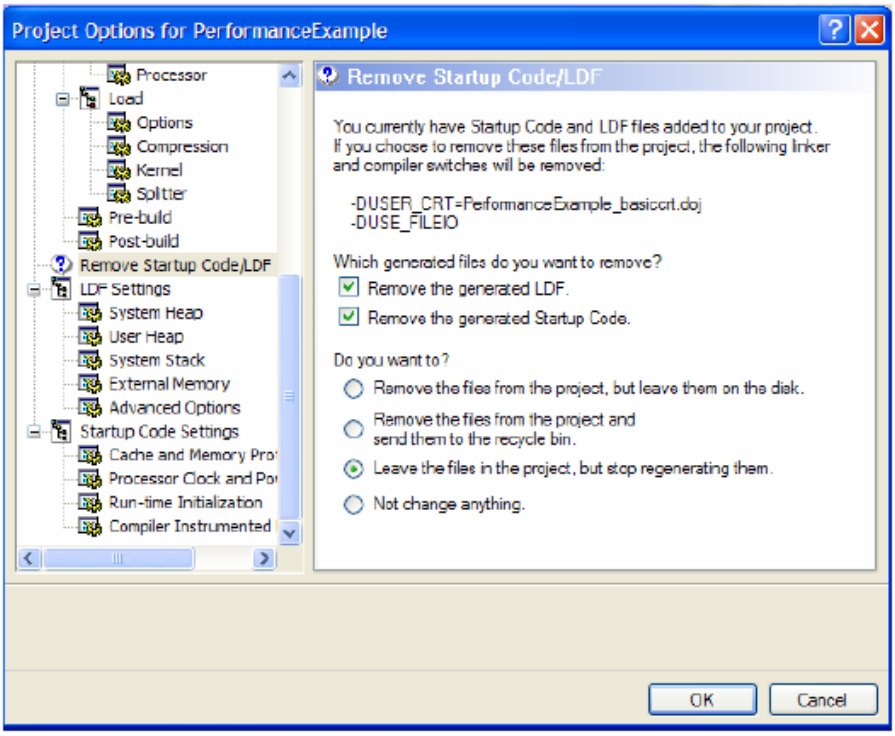


图43. Project Options对话框设置

现在就可以修改链接描述文件（.LDF）了。

ADSP-BF53x处理器页面

ADSP-BF53x处理器有16位的存储器接口，其地址映射方案如图44所示。

Bank	Row Address	Column Address	Byte
2 MSB	[(EBCAW+[10,16]):(EBCAW+1)]	[EBCAW:1]	0

图44. ADSP-BF53x存储器映射方案

若使用的SDRAM不同，则页面大小也不同，表2说明了根据EBCAW位设置的16位EBIU页面大小。

EBCAW	页面大小（16进制）
8位	0x200
9位	0x400
10位	0x800
11位	0x1000

表2. 16位EBIU的页面大小

考虑地址宽度为10位的列地址。每个页面都有0x800字节，可在.LDF文件中定义自己的存储器映射（列表9）。

```
MEMORY
{...
/* We define 10 pages in bank 0 of a 10-bit memory of a BF53x
SDRAM_BANK_0_PAGE_0       { TYPE(RAM) START(0x00000000) END(0x000007FF) WIDTH(8) }
SDRAM_BANK_0_PAGE_1       { TYPE(RAM) START(0x00000800) END(0x00000FFF) WIDTH(8) }
SDRAM_BANK_0_PAGE_2       { TYPE(RAM) START(0x00001000) END(0x000017FF) WIDTH(8) }
SDRAM_BANK_0_PAGE_3       { TYPE(RAM) START(0x00001800) END(0x00001FFF) WIDTH(8) }
SDRAM_BANK_0_PAGE_4       { TYPE(RAM) START(0x00002000) END(0x000027FF) WIDTH(8) }
SDRAM_BANK_0_PAGE_5       { TYPE(RAM) START(0x00002800) END(0x00002FFF) WIDTH(8) }
SDRAM_BANK_0_PAGE_6       { TYPE(RAM) START(0x00003000) END(0x000037FF) WIDTH(8) }
SDRAM_BANK_0_PAGE_7       { TYPE(RAM) START(0x00003800) END(0x00003FFF) WIDTH(8) }
SDRAM_BANK_0_PAGE_8       { TYPE(RAM) START(0x00004000) END(0x000047FF) WIDTH(8) }
SDRAM_BANK_0_PAGE_9       { TYPE(RAM) START(0x00004800) END(0x00004FFF) WIDTH(8) }

//if we would define a section for each page we have to define 8192...
//so we define sections only for the amount of pages which are performance
//relevant
SDRAM_BANK_0_OTHER{ TYPE(RAM) START(0x00005000) END(0x00FFFFFF) WIDTH(8) }
/* the pages on the second bank… */
SDRAM_BANK_1_PAGE_0       { TYPE(RAM) START(0x01000000) END(0x010007FF) WIDTH(8) }
SDRAM_BANK_1_PAGE_1       { TYPE(RAM) START(0x01000800) END(0x01000FFF) WIDTH(8) }
SDRAM_BANK_1_PAGE_2       { TYPE(RAM) START(0x01001000) END(0x010017FF) WIDTH(8) }
SDRAM_BANK_1_PAGE_3       { TYPE(RAM) START(0x01001800) END(0x01001FFF) WIDTH(8) }
SDRAM_BANK_1_PAGE_4       { TYPE(RAM) START(0x01002000) END(0x010027FF) WIDTH(8) }
SDRAM_BANK_1_PAGE_5       { TYPE(RAM) START(0x01002800) END(0x01002FFF) WIDTH(8) }
SDRAM_BANK_1_PAGE_6       { TYPE(RAM) START(0x01003000) END(0x010037FF) WIDTH(8) }
...
}
PROCESSOR p0
{
SECTIONS
    {    ...
         sdram0_page_0
         {
             INPUT_SECTION_ALIGN(4)
              INPUT_SECTIONS($OBJECTS(sdram0page0) )
         } > SDRAM_BANK_0_PAGE_0
    ...
}
}
```

列表9. 将存储器划分为页面

ADSP-BF561处理器的页面

ADSP-BF561处理器有32位存储器接口，当使用16位接口时，寻址过程与ADSP-BF53x相同。ADSP-BF561处理器的32位地址映射方案如图45所定义。

Bank	Row Address	Column Address	Byte
2 MSB	[(EBCAW+[11,17]):(EBCAW+2)]	[(EBCAW+1):2]	1-0

图45. ADSP-BF561存储器映射方案

表3说明了与EBCAW位相关的32位EBIU页面大小。

EBCAW	页面大小（16进制）
8位	0x400
9位	0x800
10位	0x1000
11位	0x2000

表3. 32位EBIU的页面大小

独立访问页面有什么优势呢？如果能确保所有访问都停留在一个页面内，就不需要额外的预充电和激活命令，从而可以节省时间。

图46说明了可以避免页面转换的外设DMA传输方法，DMA将输入的数据写入到位于不同存储器块的页面。

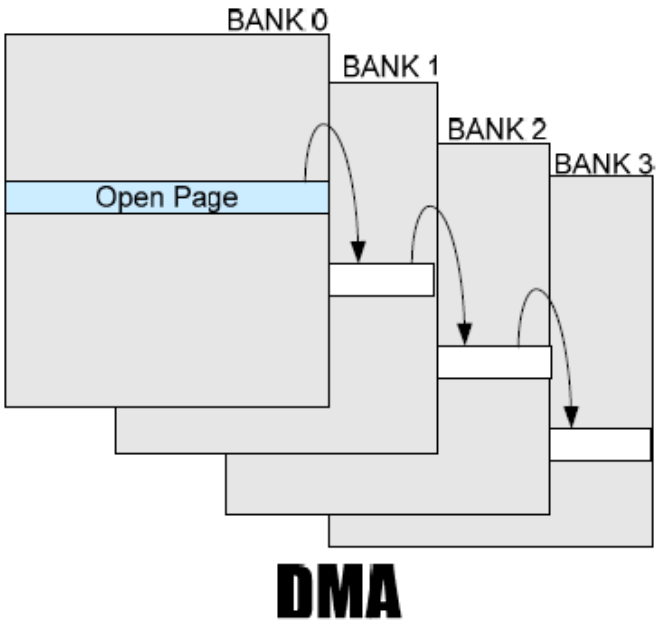


图46. DMA方法打开页面

6.3 处理器核访问的SDRAM性能指标

处理器核访问是SDRAM最耗性能的访问。因此，如果用户不使用数据cache，就必须组织对SDRAM进行访问的策略。本节说明引起这一瓶颈的原因和解决方法。

缓存用于处理系统时钟域和处理器核时钟域间的数据传输，这些缓存组织为状态机，可以修改处理器核和系统时钟而不需考虑内部处理过程。根据处理器核/系统时钟比，引入了等待状态来组织处理器核与系统域之间的读写操作。

解决问题的一种方法就是使用DMA传输。用DMA来传输需要在内部存储器进行处理的数据。DMA有内部FIFO缓存结构，可从SDRAM读取数据，而没有额外的时间损失。软件规划和设计也很关键，尤其是使用非常大（数组大小远远大于内部数据存储器）的多维数组时。此时，编写的算法必须以不同的方式访问该数组，可在算法中使用DMA来整理数组实现算法优化。

6.3.1 程序代码覆盖

使用cache的一个不足就是cache要求应用程序具有顺序流，而多数应用不满足这一点，因此常常导致高速缓存未命中。另一种方法就是利用DMA传输将必要的代码加载到内部存储器，智能覆盖管理器组织管理程序代码，并可预测接下来需要用哪一部分程序。当然，覆盖管理器能否改善性能在很大程度上取决于系统设计和程序的执行顺序。对于大多数应用，使用cache来优化是较好且简单的提升系统性能的方法。最大的挑战就是如何识别应用程序中相互独立且彼此无需直接调用的模块。从程序中分离出复盖程序段，并将其机器码放入更大的存储器中。开发覆盖管理器，用于组织管理将复盖程序段从SDRAM到内部存储器的智能DMA传输。

6.4 使用cache时的SDRAM性能指标

程序代码

cache存储器访问优化就意味着减少cache未命中，因此必须用高速缓存最少未命中的方式组织程序代码和数据。当为cache访问优化程序代码时，应尽可能保持程序顺序执行，并将程序代码中不常用的函数声明为内联函数，这样就使程序代码保持紧凑并能最小化cache未命中的数目。若有可能，应将经常调用的函数应放在内部存储器中。

数据

查看以下算法并尝试找到一种方法来实现顺序数据存取。顺序访问数据的快速傅里叶变换（FFT）实例可参见“为*ADSP-TS201 TigerSHARC® 处理器编写高效的浮点FFT（EE-218）*”[8]。在C语言中，多维数组在存储器中的顺序如下（这里为3维数组）：

$A_{0,0,0}$, $A_{0,0,1}$, $A_{0,0,2}$, $A_{0,1,0}$, $A_{0,1,1}$, $A_{0,1,2}$, $A_{0,2,0}$, $A_{0,2,1}$, $A_{0,2,'2}$, $A_{1,0,0}$, $A_{1,0,1}$, $A_{1,0,2}$, $A_{1,1,0}$, $A_{1,1,1}$, $A_{1,1,2}$, $A_{1,2,0}$, $A_{1,2,1}$, $A_{1,2,'2}$, $A_{2,0,0}$, $A_{2,0,1}$, $A_{2,0,2}$, $A_{2,1,0}$, $A_{2,1,1}$, $A_{2,1,2}$, $A_{2,2,0}$, $A_{2,2,1}$, $A_{2,2,'2}$

列表10给出了通过顺序存取填充数组的程序代码。

```
for (i=0;i<3; i++)
    {
    for (j=0;j<3; j++)
    {
         for (k=0;k<3; k++)
            { MyArray[i][j][k] = getValue();
            }
    }
    };
```

列表10. 优化访问数组

7 功耗优化

Blackfin处理器常在便携式或低功耗应用中使用。在这类应用中，尽可能低的功耗就十分关键。而SDRAM在功率消耗中占有很高的比例。因此，正确选择、使用和配置SDRAM是低功耗设计的基础。本节概述了降低功耗的方法。

7.1 简介：功耗指标

器件生产商用各种不同的标准符号（表4）来定义功耗，测量这些指标的方法也各有不同，因此，对这些值的解释也有差别。

标记1	含义
I_{CC1}	活动模式下的工作电流
I_{CC2}	预充电静态电流
I_{CC3}	无操作/静态电流
I_{CC4}	突发模式/所有存储器块都激活下的操作电流
I_{CC5}	自动刷新电流
I_{CC6}	自刷新电流

表4. 功耗测量符号

7.2 降低SDRAM功耗的技巧

以下是降低SDRAM功耗的技巧。

- 尽可能少的用SDRAM。
- 使用1.8V或2.5V移动SDRAM（不是所有Blackfin处理器都可以，参见SDRAM基本说明）。
- 在低温环境下降低刷新率。最坏情况（高温）下的刷新率是64ms，因此，在标准环境下的操作仍有进一步降低刷新率的空间。
- 尽量在存储器块之间传输数据（不是在一个存储器块内）。
- 使能EBIU_SDGCTL寄存器中的自刷新位（SRFS），该模式下SDRAM的功率处于最低点。

7.3 移动SDRAM

有较高功耗要求的嵌入式应用可选用移动SDRAM或低功耗SDRAM。在这些应用中，SDRAM使用不频繁，大部分时间都处于空闲状态。移动SDRAM提供了一种特殊模式，可在SDRAM空闲时降低功耗，而Blackfin处理器支持这种特性就是温度补偿自刷新（TCSR）和部分阵列自刷新。温度补偿自刷新特性允许在温度低于45°C时，减小空闲状态下的自刷新频率。存储单元的漏电与温度密切相关，温度较高时，漏电比低温时快。标准SDRAM的自刷新率是根据高温最坏情况设定的，且SDRAM也是特定的。在Blackfin处理器中，可通过EBIU_SDGCTL寄存器中的TCSR位设置温度，TCSR的值代表温度上限（例如，45°C意味着工作环境温度低于45°C）。

许多应用程序仅使用SDRAM设备的一部分来缓存数据数组。而数据存储在数据处理后，因此，此时采用部分阵列自刷新特性是节省功率的一种好方法。该特性可使应用程序选择空闲模式下要刷新的存储器块。如果SDRAM上保存有程序代码，应将其放入SDRAM的存储器块0和1中，否则程序代码会丢失。

对于该应用，应使用低电压（1.8V或2.5V）Blackfin处理器。

7.4 进入休眠及恢复

ADSP-BF537，ADSP-BF54x，和ADSP-BF52x处理器在进入休眠模式时应保护SDRAM的内容。因此，VR_CTL寄存器中的CKELOW位必须设置为1以在休眠状态下保持CKE信号为低电平，这样可以保护SDRAM不会丢失数据。内部存储器和所有寄存器（除VR_CTL寄存器外）的内容将会丢失。因此，程序必须将所有寄存器的设置和内部存储器的内容写入SDRAM。

以下步骤用于进入休眠模式及恢复数据。

步骤1

将所有重要寄存器存入SDRAM。

将内部存储器的重要数据存入SDRAM。

确保VR_CTL寄存器中的CKELOW位和SDGCTL寄存器中的自刷新位都置位。

让处理器进入休眠模式（图47和列表11）。

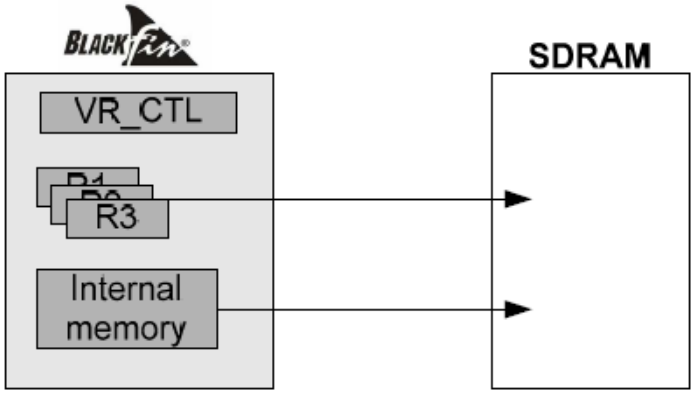


图47. 进入休眠模式

```
    SaveTheMemory();
    SaveTheRegister();
    //Let's setup the RTC to wake up
    SetupRTC();
    IntMask = cli();
    *pVR_CTL = (        WAKE    | //wake by
                        CKELOW  | //keeps the content of the SDRAM
                        CANWE   | //ensures that the CAN RX can wake up the BF
                        (*pVR_CTL & ~FREQ)); // Send to hibernate
```

列表11. 进入休眠模式

步骤2

处理器处于休眠模式（图48），除VR_CTL寄存器外，所有寄存器的内容都丢失，数据存储在SDRAM。

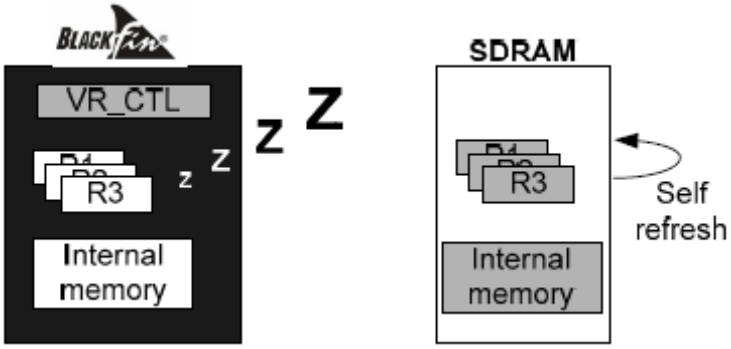


图48. 休眠模式

步骤3

当处理器从休眠模式唤醒后（图49），处理器从初始化文件加载，并查看CKELOW位，判定唤醒是来自休眠还是复位。若Blackfin处理器唤醒来自复位，则处理器继续加载过程，否则，它调用子程序恢复内部存储器和寄存器值，并跳转到要执行的程序代码处。

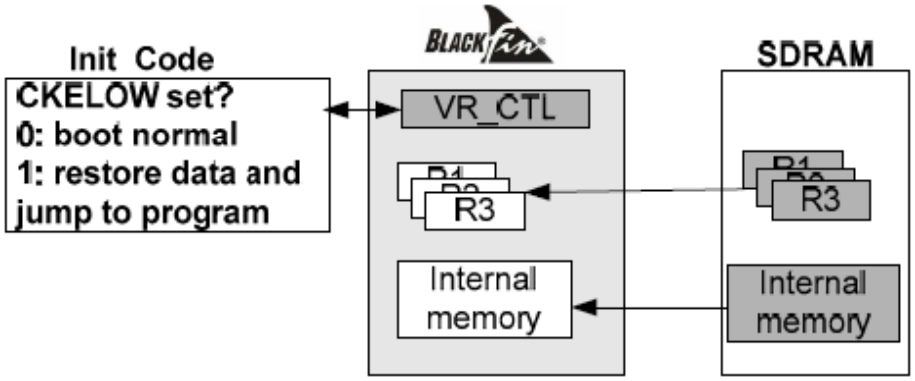


图49. 从休眠模式恢复

7.5低功耗结构数据

预充电和激活命令不仅耗时，且对功耗有很大的影响，因此应尽量减少这些操作的数目。

应用中的预充电操作将在分页处和SDRAM刷新时执行。因此，应尽可能以减少页面分页数目的方式组织数据。具体实例请参考提高用户系统的SDRAM性能。

附录

附录A：词汇表

访问时间	从开始设备访问到下次访问之间的时间
阵列	保存数据的存储器区，由行和列组成，每个存储单元位于一个产生交叉的地址处。存储器中的每一位通过行和列坐标进行查找
异步	操作是独立进行的过程
自动预充电	一种SDRAM功能，在突发操作后关闭页面
自动刷新	一种模式，内部振荡器设置刷新率，内部计数器记录要刷新的地址
存储器块	SDRAM模块上物理存储块（与行相同）数。也可以指单独SDRAM设备中内部逻辑存储区（现在常为4个存储区）的数目
突发模式	突发就是将数据快速传送到系列存储器单元地址
旁路电容	基本功能是稳定电源供电，尤其是为邻近的设备或电路
总线周期	存储设备和Blackfin处理器系统域间的单次交换
CAS	列地址选通。将列地址锁存到SDRAM控制寄存器的控制信号
CAS-before-RAS（CBR）	行地址选通前的列地址选通。CBR是一种快数刷新功能，并跟踪记录要刷新的下一行
列	部分存储器阵列。每一位存储于行和列的交叉地址
串扰	由一条导线或通路中的电流在另一条导线或通路中感应的信号
DDR	双数据率。在时钟上升沿或下降沿都传输数据。由于地址线路保持不变，所以数据按地址顺序传输
DQM	数据屏蔽信号，用于写过程中的屏蔽。每8个I/O一个DQM
DRAM	动态随机访问存储器。一类存储设备，常用于计算机系统中的大容量存储。动态指要保存数据必须维持持续存储器刷新
EBIU	外部总线接口单元。主要为SDRAM提供同步外部存储器接口，与PC100和PC133标准兼容，还为SRAM，ROM，FIFO，闪存器，和FPGA/ASIC设计提供异步接口

EMC	符合控制EMI的规则和条例
EMI	电磁辐射引起的干扰
外部缓存	如果引脚负载大于50pF，外部缓存用于驱动SDRAM命令、时钟、时钟使能和地址引线。最终容抗为输入引脚数乘以SDRAM输入引脚的容抗（一个SDRAM引脚的容抗约为4.5pF[查询SDRAM数据手册]）再加上PCB线路容抗
FBBRW	快速循环读-写。在标准应用中，写命令在读命令执行之后要延迟一个时钟周期。快速循环读-写使能读命令执行后直接写入，而没有1周期延迟。鉴于数据总线必须在读和写数据间进行快速交换，该特性依赖数据总线的容量。这包含SDRAM芯片数目和数据总线电路路径的设计
FPM	快速页面模式。通用的SDRAM数据访问方案
休眠	Blackfin处理器的一种特殊电源模式，提供最低功耗。I/O供电稳定，而处理器核掉电。Blackfin处理器可由几个外部事件唤醒
交叉存取	对SDRAM的两个或多个页面交换读/写数据的过程
JEDEC	电子工程设计发展联合会议
延迟	时间的长度，通常以时钟周期表示，从请求存储器读到数据准备好这段时间
存储器周期时间	指发生一个完整的存储器操作的时间（如一次读或写操作）
微波传输带	一种线路配置，仅信号线路的一边有参考面
页面	一行地址可以访问的字节数
页面模式	当RAS为逻辑低电平且列地址被选通时的一种操作。SDRAM记录最后一个行地址并从该行移到新的列地址
PCB	印刷电路板

RAS	行地址选通。将行地址锁存到SDRAM的控制信号。与列地址组合使用选择一个独立的存储单元
RAS到CAS延迟	行访问选通到列地址选通之间的时间
读时间	行和列地址有效后，数据出现在输出端所需的时间。读时间与访问时间相关
刷新	为保存数据，SDRAM单元电荷所需的周期性恢复
刷新周期	SDRAM中一行刷新的时间周期
刷新时间	SDRAM中每一行必须刷新的最短时间
行	存储器阵列的一部分。每一位存储在行和列的交叉处
SDR	单数据率。一个时钟周期传输一次数据。引入该定义是为了区分SDRAM和DDR
SDRAM自刷新	在正常操作模式，Blackfin处理器通过发送自动刷新命令来控制数据单元的自刷新。但是，如果应用程序需要控制SDRAM（例如处理器进入休眠模式或在多处理器应用中），SDRAM就必须负责数据的一致性。另一种使SDRAM进入自刷新的情况是降低功耗，当Blackfin处理器发送自刷新命令时，SDRAM自己开始计时并自刷新周期。自刷新模式的缺点在于访问SDRAM时会出现延时。
选通	将数据同步锁存到SDRAM的输入控制信号，
带状线	一种电路配置，其中信号线路放置在两个参考层之间
同步存储器	一种存储设备，其信号与参考时钟信号同步
终端	一个或多个元件，与传输线路结合，用于控制信号反射
写时间	从数据锁存到SDRAM到其真正存入存储位置间的时间

附录B：代码实例，图表，及附注

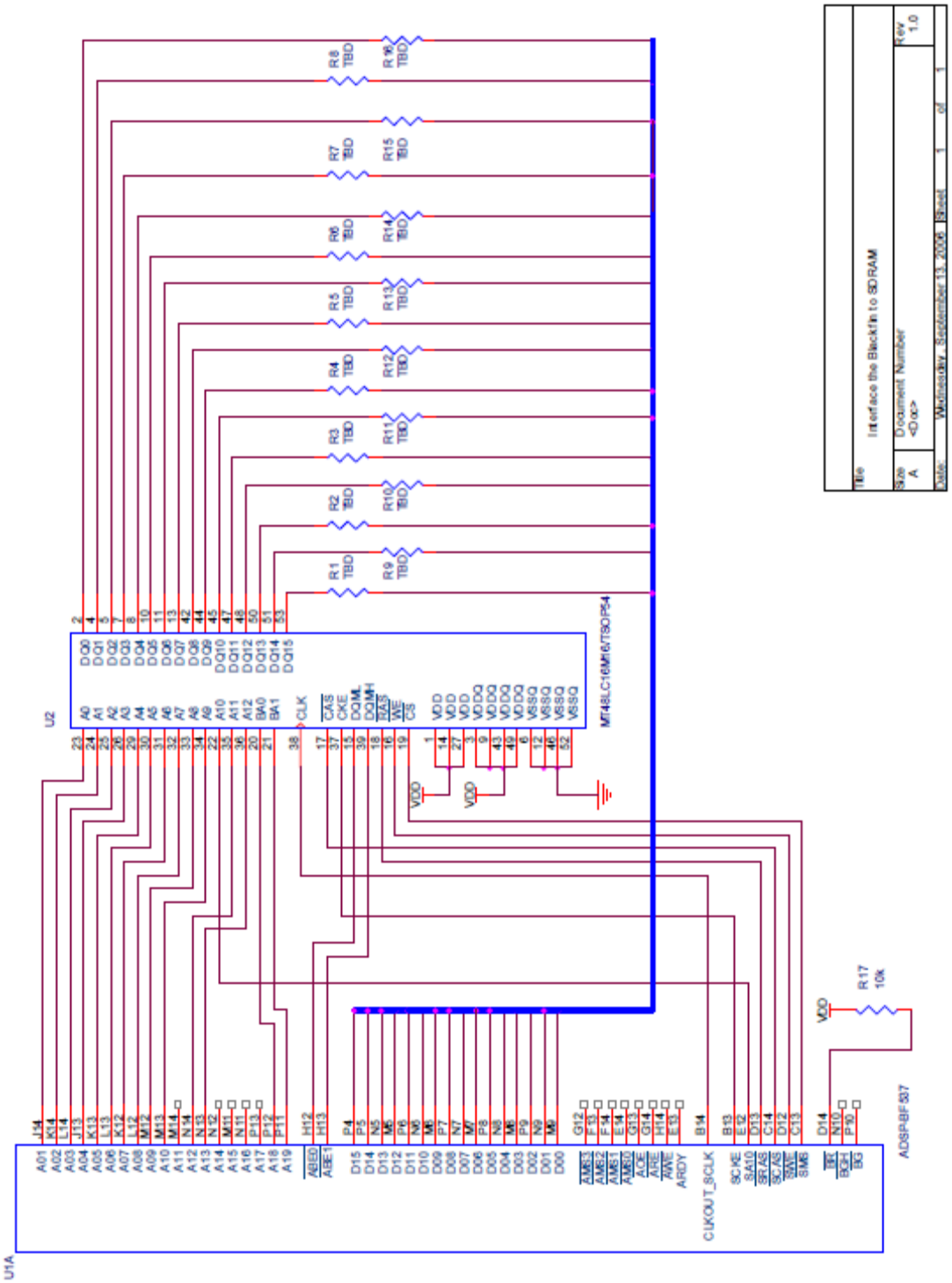
初始化代码（第2章）

列表12给出了一个汇编语言初始化实例。

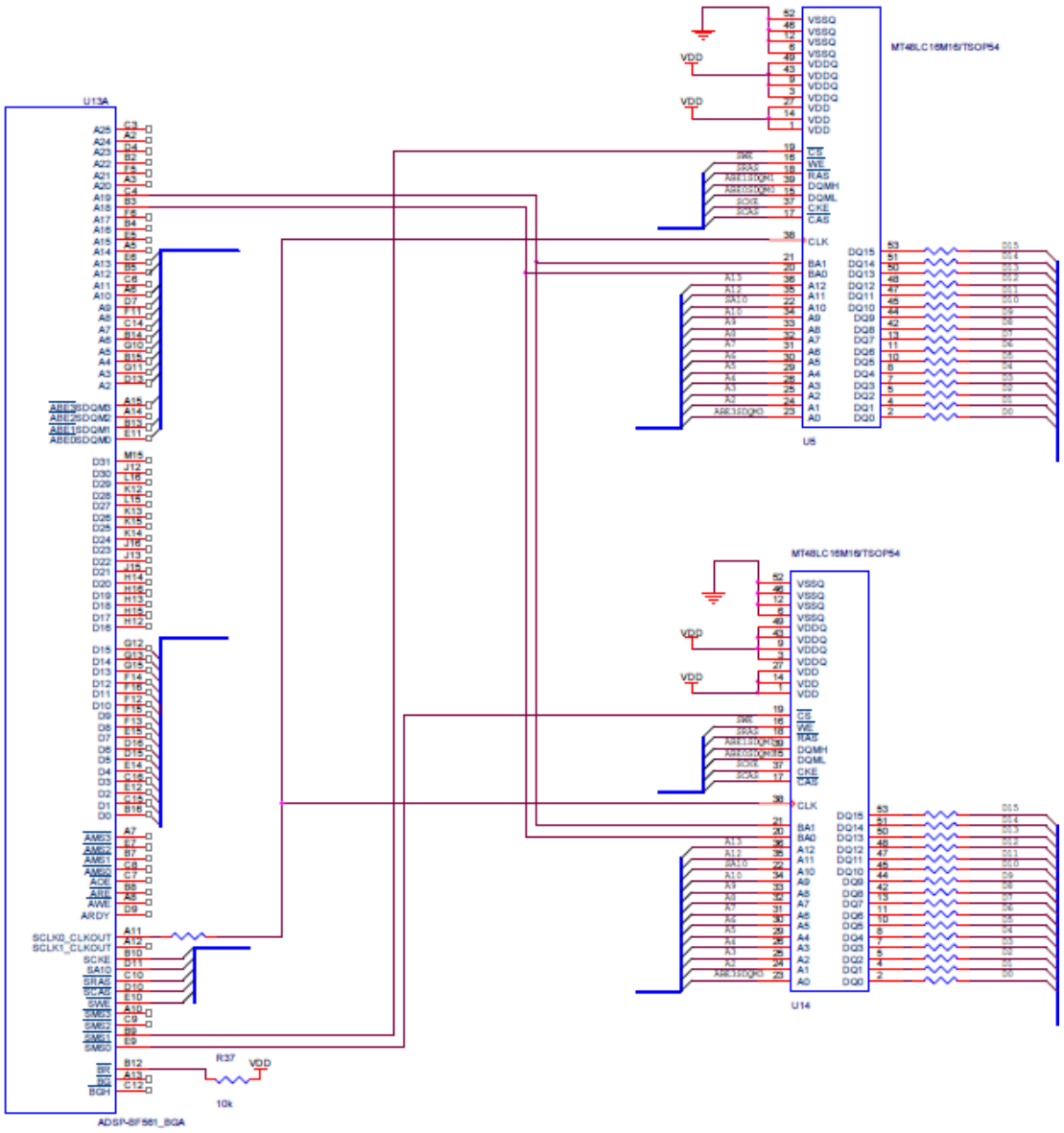
```
//SDRAM Refresh Rate Setting
    P0.H = hi(EBIU_SDRRC);
    P0.L = lo(EBIU_SDRRC);
    R0 = 0x406 (z);
    w[P0] = R0;
//SDRAM Memory Bank Control Register
    P0.H = hi(EBIU_SDBCTL);
    P0.L = lo(EBIU_SDBCTL);
    R0 =          EBCAW_9    | //Page size 512
                  EBSZ_64    | //64 MB of SDRAM
                  EBE;         //SDRAM enable
    w[P0] = R0;
//SDRAM Memory Global Control Register
    P0.H = hi(EBIU_SDGCTL);
    P0.L = lo(EBIU_SDGCTL);
    R0.H=         hi(     ~CDDBG      & // Control disable during bus grant off
                          ~FBBRW      & // Fast back to back read to write off
                          ~EBUFE      & // External buffering enabled off
                          ~SRFS       & // Self-refresh setting off
                          ~PSM        & // Powerup sequence mode (PSM) first
                          ~PUPSD      & // Powerup start delay (PUPSD) off
                          TCSR        | // Temperature compensated self-refresh at 85
                          EMREN       | // Extended mode register enabled on
                          PSS         | // Powerup sequence start enable (PSSE) on
                          TWR_2       | // Write to precharge delay TWR = 2 (14-15 ns)
                          TRCD_3      | // RAS to CAS delay TRCD =3 (15-20ns)
                          TRP_3       | // Bank precharge delay TRP = 2 (15-20ns)
                          TRAS_6      | // Bank activate command delay TRAS = 4
                          PASR_B0     | // Partial array self refresh Only SDRAM Bank0
                          CL_3        | // CAS latency
                          SCTLE       ); // SDRAM clock enable

    R0.L=         lo(     ~CDDBG      & // Control disable during bus grant off
                          ~FBBRW      & // Fast back to back read to write off
                          ~EBUFE      & // External buffering enabled off
                          ~SRFS       & // Self-refresh setting off
                          ~PSM        & // Powerup sequence mode (PSM) first
                          ~PUPSD      & // Powerup start delay (PUPSD) off
                          TCSR        | // Temperature compensated self-refresh at 85
                          EMREN       | // Extended mode register enabled on
                          PSS         | // Powerup sequence start enable (PSSE) on
                          TWR_2       | // Write to precharge delay TWR = 2 (14-15 ns)
                          TRCD_3      | // RAS to CAS delay TRCD =3 (15-20ns)
                          TRP_3       | // Bank precharge delay TRP = 2 (15-20ns)
                          TRAS_6      | // Bank activate command delay TRAS = 4
                          PASR_B0     | // Partial array self refresh Only SDRAM Bank0
                          CL_3        | // CAS latency
                          SCTLE     ) ; // SDRAM clock enable
    [P0] = R0;
```

列表12. 汇编语言初始化代码实例

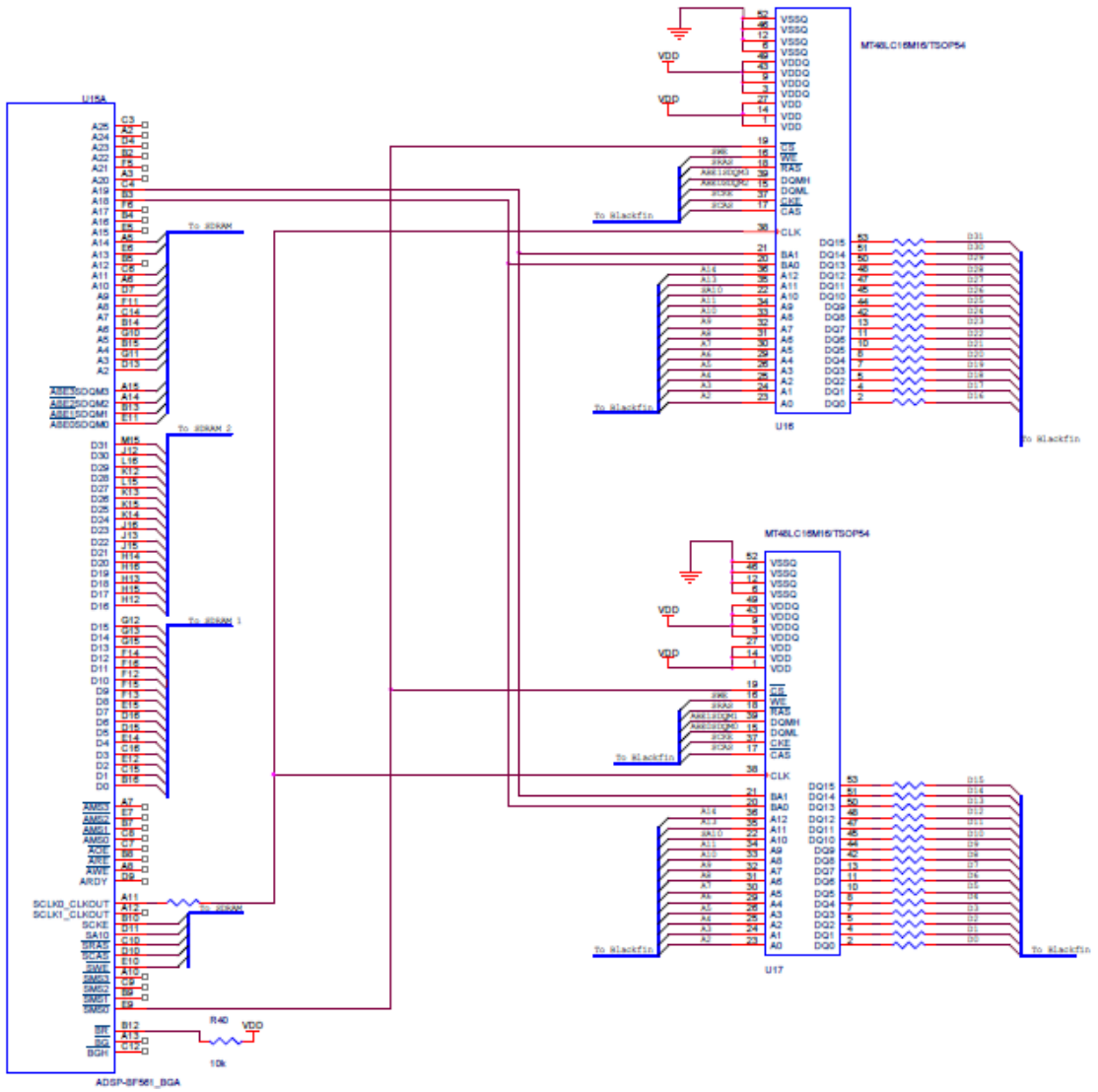
下面几页给出了实现实例，给出的原理图用于说明SDRAM和Blackfin处理器间的正确连接（而不是一种信号完整性方法）。



16位模式的ADSP-BF561



32位模式的ADSP-BF561



附注：计算Z_0（第4章）

本节说明电报等效方法。

SDRAM连接本身是一个传输线路，这就是可以用电报等效精确模拟电流和电压沿设备传播的原理。传输线路不是理想导体，因此必须利用包含影响线路本身的等效电路图表示（图50）。

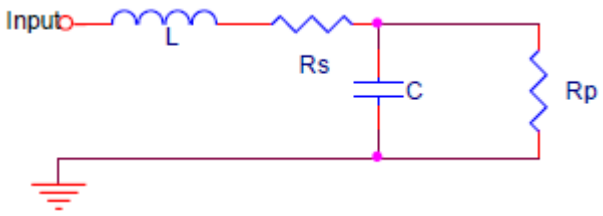


图50. 包括线路自身影响的等效电路

但是一个线路不仅包括一个这样的结构，可以将连接线路（图51）想象成由无穷级构成。

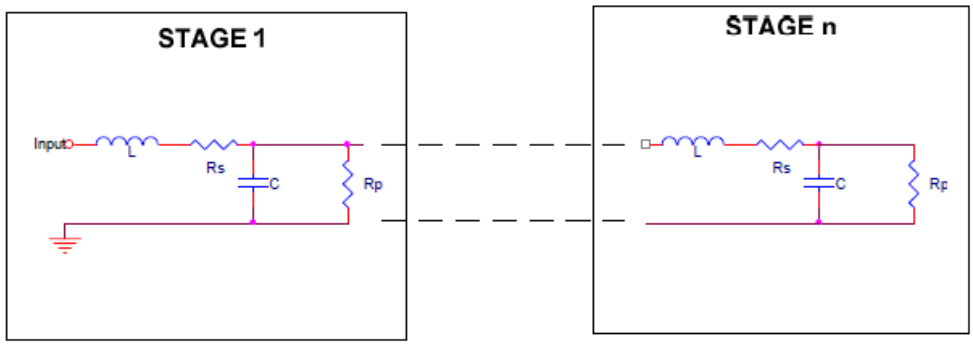


图51 由无穷多布线组成的连接线

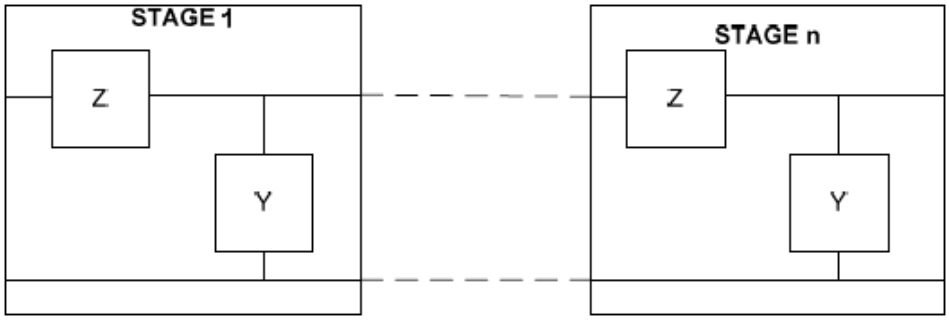
变量Z_C是阻抗的函数，决定于频率。如果从数学上正确表示，应写为$Z_C(\omega)$。

变量Z_0是单值常量，表示特定频率ω_0下特性阻抗的值。

通过定义阻抗Y和Z来简化模型：

$Z = j\omega L+Rs$

$Y = j\omega C+1/Rp$



最终得到的阻抗是阻抗Z和阻抗Y以及其他所有并联阻抗的和。假定有n个元素，且每个元素有相同的Rp，Rs，C和L。

$$\tilde{Z}_c = \frac{Z}{n} + \frac{1}{\frac{1}{\tilde{Z}_c} + \frac{Y}{n}}$$

两边同乘以 $(1+\frac{Y}{n}\tilde{Z}_c)$

$$\tilde{Z}_c(1+\frac{Y}{n}\tilde{Z}_c) = \frac{Z}{n}(1+\frac{Y}{n}\tilde{Z}_c) + \tilde{Z}_c \Leftrightarrow$$

$$\tilde{Z}_c^{\,2} = \frac{Z}{Y} + \frac{Z}{n}\tilde{Z}_c \Leftrightarrow$$

$$\tilde{Z}_c = \sqrt{\frac{Z}{Y} + \frac{Z}{n}\tilde{Z}_c}$$

假定一条传输线由无限多个长度近似为0的元素组成：

$$Z_c = \lim_{n\to\infty}\sqrt{\frac{Z}{Y} + \frac{Z}{n}\tilde{Z}_c} = \sqrt{\frac{Z}{Y}} = \sqrt{\frac{jwL + Rs}{jwX + 1/Rp}}$$

频率增加，R和G两项最后可以忽略，因为他们分别被jωL和jωC项淹没，导致稳定的高阻抗产生。感抗jωL和容抗jωC间的很好平衡将使阻抗在高频下保持稳定。这种稳定高阻抗为高速数字电路的设计提供了很大帮助，因为这使用单个电阻作为传输线路终端匹配成为可能。特性阻抗的值称为Z_0。

$$Z_0 = \lim_{w\to\infty}(Z_C(w)) = \sqrt{\frac{L}{C}}$$

计算微波传输带线路的感应系数

强烈推荐测量该参数值，可利用下面的方程估计该值：

$$L \cong 5 \cdot \ln\left(\frac{2\pi \cdot h}{w}\right)$$

其中：

L是感应系数，单位为nH/英寸

h是平面以上的高度（毫米mils）

w是线宽（毫米mils）

计算电容

$$C = \varepsilon_0 * \varepsilon_r * \frac{A}{d}$$

其中：

A是线长乘以线宽

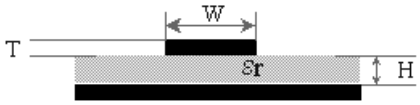
d是地和线间的距离

必须为每个地平面都计算

Z0的IPC和Douglas Brooks方法

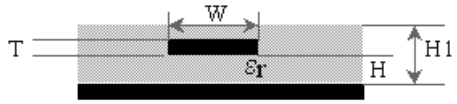
IPC和Douglas Brooks为PCB提供了一些便于应用的方程。

微波传输带线



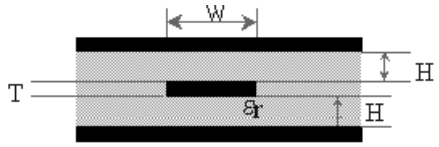
$$Z_0 = \frac{87}{\sqrt{\varepsilon_r + 1.41}} \ln\left(\frac{5.98 \cdot H}{0.8 \cdot W + T}\right)$$

嵌入式微波传输带线



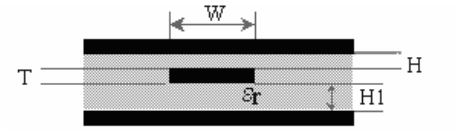
$$Z_0 = \frac{60}{\sqrt{\varepsilon_r \left[1 - e^{\frac{(-1.55 \cdot H1)}{H}}\right]}} \ln\left(\frac{5.98 \cdot H}{0.8 \cdot W + T}\right)$$

带状布线



$$Z_0 = \frac{60}{\sqrt{\varepsilon_r}} \ln\left(\frac{1.9 \cdot (2H + T)}{0.8 \cdot W + T}\right)$$

非均匀带状布线



$$Z_0 = \frac{80}{\sqrt{\varepsilon_r}} \ln\left(\frac{1.9 \cdot (2H + T)}{0.8 \cdot W + T}\right)\left(1 - \frac{H}{4 \cdot H1}\right)$$

其中 H1>H

印制电路板（PCB）的介电常数（第4章）

以下表格来自“印制*电路板制造中所用的绝缘体材料的测量与教程*”[7]。

几种常见的增强安全玻璃纤维

Material	Tg	e_{r*}	Tan (f)	DBV (V/mil)	WA, %
Standard FR-4 Epoxy Glass	125C	4.1	0.02	1100	0.14
Multifunctional FR-4	145C	4.1	0.022	1050	0.13
Tetra Functional FR-4	150C	4.1	0.022	1050	0.13
Nelco N4000-6	170C	4	0.012	1300	0.10
GETEK	180C	3.9	0.008	1100	0.12
BT Epoxy Glass	185C	4.1	0.023	1350	0.20
Cyanate Ester	245C	3.8	0.005	800	0.70
Polyimide Glass	285C	4.1	0.015	1200	0.43
Teflon	N/A	2.2	0.0002	450	0.01
		* Measured with a TDR using velocity method. Resin content 55%			

Tg = glass transition temperature
er = relative dielectric constant
Tan (f) = loss tangent
DBV = dielectric breakdown voltage
WA = water absorption

All materials with woven glass reinforcement except teflon.

非纤维或含量极低的安全玻璃材料列表

Material	Tg	e_{r*}	Tan (f)	DBV (V/mil)	WA, %
Speedboard N	140C	3	0.02	N/A	N/A
Speedboard C	220C	2.7	0.004	N/A	N/A
Rogers Ultralam C	280C	2.5	0.0019	N/A	N/A
Rogers 5000	280C	2.3	0.001	N/A	N/A
Rogers 6002	350C	3	0.0012	N/A	N/A
Rogers 6006	325C	6 to 10	0.002	N/A	N/A
Rogers RO3003	350C	3	0.0013	N/A	N/A
Rogers RO3006	325C	6 to 10	0.003	N/A	N/A
Teflon	N/A	2.2	0.0002	450	0.01
		Information from manufacturer's data sheets.			

Tg = glass transition temperature
er = relative dielectric constant
DBV = dielectric breakdown voltage
WA = water absorption

参考文献

[1] *The ABCs of SDRAM (EE-126)*, Rev. 1, March 2002. Analog Devices, Inc.

[2] *System Optimization Techniques for Blackfin Processors (EE-324)*, Rev. 1, July 2007. Analog Devices, Inc.

[3] *ADSP-BF52x Blackfin Processor Hardware Reference (Volume 1 of 2) Revision 0.31 (Preliminary).* May, 2008. Analog Devices, Inc.

[4] *ADSP-BF52x Blackfin Processor Hardware Reference (Volume 2 of 2) Revision 0.3 (Preliminary).* September, 2007. Analog Devices, Inc.

[5] *ADSP-BF533 Blackfin Processor Hardware Reference.* Rev 3.2, July 2006. Analog Devices, Inc.

[6] *ADSP-BF537 Blackfin Processor Hardware Reference.* Rev 3.0, December 2007. Analog Devices, Inc.

[7] *ADSP-BF561 Blackfin Processor Hardware Reference.* Rev 1.1, February 2007. Analog Devices, Inc.

[8] *ADSP-BF522/523/524/525/526/527 Blackfin Embedded Processor Preliminary Data Sheet.* Rev PrE, August 2008. Analog Devices, Inc.

[9] *ADSP-BF512/ADSP-BF514/ADSP-BF516/ADSP-BF518 Blackfin Embedded Processor Preliminary Data Sheet.* Rev PrC, October 2008. Analog Devices, Inc.

[10] *ADSP-BF531/ADSP-BF532/ADSP-BF533 Blackfin Embedded Processor Data Sheet.* Rev F, 2008. Analog Devices, Inc.

[11] *ADSP-BF534/ADSP-BF536/ADSP-BF537 Blackfin Embedded Processor Data Sheet.* Rev E, March 2008. Analog Devices, Inc.

[12] *ADSP-BF538/ADSP-BF539 Blackfin Embedded Processor Data Sheet.* Rev B, 2008. Analog Devices, Inc.

[13] *ADSP-BF561 Blackfin Embedded Processor Data Sheet.* Rev C, 2007. Analog Devices, Inc.

[14] *A Survey and Tutorial of Dielectric Materials Used in the Manufacture of Printed Circuit Boards* - By Lee W. Ritchey, Speeding Edge, for publication in November 1999 issue of Circuitree magazine. Copyright held by Lee Ritchey of Speeding Edge, September 1999.

[15] *Writing Efficient Floating-Point FFTs for ADSP-TS201 TigerSHARC® Processors (EE-218)*, Rev. 2, March 2004

[16] *Micron Technical Note 48-09 LVTTL DERATING FOR SDRAM SLEW RATE VIOLATIONS*

[17] *High Speed Digital Design – A Handbook of Black Magic* by Howard W. Johnson and Martin Graham, 1993 PTR Prentice Hall, ISBN 0-13-395724-1

[18] *High-Speed Signal Propagation – Advanced Black Magic* by Howard W. Johnson and Martin Graham, 2003, PTR Prentice Hal, ISBN 0-13-084408-X

[19] *EMV-Design Richtlinien* by Bernd Föste and Stefan Öing, 2003 Franzis' Verlag GmbH, ISBN 3-7723-5499-8

阅读材料

[20] *ADSP-BF53x/ADSP-BF56x Blackfin Processor Programming Reference.* Rev 1.2, February 2007. Analog Devices, Inc.

文档记录

Revision	Description
Rev 2 – December 11, 2008 *by Fabian Plepp*	Provides a more detailed description of the SDRAM initialization. Also, adds information on drive strength control for ADSP-BF52x devices. Incorporates support for ADSP-BF51x processors.
Rev 1 – May 12, 2008 *by Fabian Plepp*	Initial public release. Adds Low Power section, OTP and ADSP-BF52x processors.
Rev 0 – August 21, 2006 *by Fabian Plepp*	Internal version (draft).