毕业设计（论文）报告

题 目： Cache 中 TLB 的设计及优化

无线电工程 系 信息工程 专业

学　　号：04000717

学生姓名：孙　宏

指导教师：杨　军

起讫日期：2004.3.1－2004.6.5

设计地点：东南大学国家 ASIC 工程中心

目 录

摘要

在当今的 SOC 设计中，微处理器的频率已经越来越高，这对片外存储器也提出更高的要求，需要有一个大容量，高速度的存储器系统来支持。

不幸的是存储器容量越大其速度越慢，因此无法设计一个理想的足够大而且又足够快的单一存储器。目前采用的解决方案是使用一个小而快的 Cache 存储器和一个大而慢的虚拟存储器来构建复合存储器系统以满足处理器的需求。两者的使用就要求 Cache 与虚拟内存必须良好的连接方能正常工作。连接两者，其实质就是虚拟地址与物理地址之间的转换。

为了加快地址转换的速度，通常使用地址转换后备缓冲器 TLB(Translation Lookaside Buffer)。本论文基于逆向设计，提出了一种可行的 TLB 结构，不仅完成了地址转换的功能，还从硬件上支持了不同大小的页表格式。

此外，本文还引入了 DVS 技术以减少 TLB 的漏电功耗。DVS 技术诣在通过提供不同的电源电压以减少漏电功耗，是当今国外关于漏电功耗的最新研究方向。本文通过引入 DVS 技术，使 TLB 存储单元中的漏电功耗减少了 90％。

本论文分为三部分：第一部分将对 TLB 的功能进行描述；第二部分将给出 TLB 结构的实现；第三部分将提出一种 DVS 优化电路以减少功耗。

关键词

Cache，TLB，地址转化，虚拟地址，物理地址，功耗，DVS

Abstract

With the increasing frequency of nowadays' processor, there are higher requirements for the off-chip memory, a large and fast memory system is needed to support the processor.

Unfortunately, as the memory speed will slow down with the increase of its capacity, we can't design a single memory with both high speed and large capacity. The normally used solution is to employ a small and fast cache memory combined with a large and slow virtual memory to construct a combinational memory system to satisfy the desire of the processor. Since both are used in processors, caches must work in conjunction with virtual memory. The spirit of the conjunction is the translation between virtual address and physical address.

To speed up the translation speed, the Translation Lookaside Buffer (TLB) is commonly employed. Based on the adverse design, this paper present a applicable circuit structure of TLB, besides implementing the address translation function, it can also support the pages of different size in hardware level.

This paper also introduced the DVS technology to reduce the leakage power of the TLB. The DVS technology aimed at reducing the leakage power by providing different supply voltages, it is the latest research orientation of the leakage power reduction. The leakage power of the storage cell in the TLB is reduced by 90% with this DVS technology introduced.

This paper will be divided into three sections: Section one will describe the function of the TLB; Section two will give out the implement of its structure; section three will propose a DVS circuits to optimize its power consumption.

Key Words

Cache，TLB，physical address, virtual address, power consumption，DVS

第一章 绪论

1.1 问题的提出

在当前的 SOC 设计，面临的一个最大难题就是嵌入式微处理器的高主频速度与片外存储器的低读取速度极不相配，很大程度上限制了微处理器的性能效率。研究调查表明嵌入式微处理器的速度在最近几年以每年 55%的速度增长，而存储系统的存取速度增长相对缓慢，这已成为 SOC 处理速度的瓶颈。

对于如何减少这一差距，有很多解决方案，但是目前应用最为广泛的就是引入片上 Cache。Cache 存储器是一种从属存储器，它容量小，但非常快的，能够自动保存最近用到的存储器数据的拷贝。在处理器和存储器系统之间加入高速的 Cache 后，由于程序的局部性原理，Cache 有很大的命中率，从而能够显著的加快存取速度。

在 Cache 设计中，必须面临虚拟地址和实地址的转换问题。为了能使系统运行比主存更大的作业空间，在主存相对比较小的片上系统中都使用了虚拟地址。虚拟地址与实地址间的转换速度直接关系到了 Cache 的运行速度，是 Cache 设计中的重要环节。

目前为了提高地址转换速度，广泛使用的一个手段就是在 Cache 中加入 TLB(快表)，用一组高速的保存最常使用的页表，由于该组寄存器速度与处理器为同一个数量级，因此能够很大的加快地址转换速度。

此外，随着器件的速度和密度的提高，器件的静态功耗也大幅提高，据估计在当今的高速处理器中，静态功耗占到了芯片总功耗的 15%到 20%。而且随着处理器技术向 0.1 微米以下发展，静态功耗将大幅增加，从而使其成为 CPU 功耗中的主要部分。

由于 TLB 中的静态功耗是处理器中静态功耗的主要来源之一，降低 TLB 的静态功耗将对降低整个处理器功耗起到极大的推动作用。因此，本次设计在完成 TLB 的结构设计之后将着重考虑如何减少 TLB 中的静态功耗，以优化其性能。

综上可见 TLB 性能的优劣不仅对片上 SOC 系统的运行速度有很大影响，对其功耗也影响显著。对 TLB 进行解剖设计和优化具有很强的应用价值，而对其

进行的解剖分析设计的重点就在于观察如何实现地址间的快速转换以及如何对其静态功耗优化，而一旦 TLB 的性能得到某种提高，就能马上推动 Cache，乃至整个系统性能的大幅提高。

1.2 论文的工作

本论文通过分析 Cache 的组成，TLB 的具体配置，理解了 TLB 的工作原理，并由此进行 TLB 的电路设计及仿真，最后进行功耗优化。论文的工作主要由以下部分组成：

1、对 TLB 的工作原理进行分析，考查其工作机制，在地址变换中的具体实施，并对版图进行分析。

2、建立 TLB 模型，此模型将能完成信号的比对，存储信号的输出，当发生 TLB 未命中时，可以选中某个表项，将其内容更新为所要读的内容。此外，此模型还可以支持不同大小的页表。

3、对模型进行功能优化，加入动态电压缩放技术(DVS)，减少漏电功耗。其机理是适当缩小 Cell 的源电压，不仅能够保存内存单元中的信息。而且能使漏电流将显著下降。由于在这种 DVS 机制中电压和漏电流都显著下降了，因此可以大幅下降漏电功耗(事实上，由于源电压下降，动态功耗亦可得到显著抑制，但本文仅对静态功耗进行考察)。在 TLB 中的 Cell 使用 CAM 或 SRAM 结构实现，也为使用 DVS 技术提供了一定空间。

1.3 论文的结构

本论文分为以下部分：第一章为绪论；第二章介绍与论文相关的一些内容，包括 Cache 的作用及结构，地址转换的概念，TLB 的基本原理与功能；第三章将实现 TLB 的结构设计，建立一个符合要求的 TLB 模型；第四章将对模型进行性能优化，通过引入 DVS 技术减少静态功耗；第五章将进行总结和展望。

第二章 嵌入式处理器及 TLB 的组成原理

TLB 是一个用于页表匹配的缓存，它的存取时间与 Cache 的存取时间相仿，远远小于主存的存取时间。它的功能是在虚存的管理中，迅速将进程中的虚拟地址顺利的转换成主存中的实地址。本章主要将阐明虚拟存储的原理，嵌入式处理器的存取结构和 TLB 的功能。

2.1 虚拟存储器

虚拟存储器是指一种实际上并不(以物理形式)存在的虚假的存储器。在虚存的管理中，通常把一个运行进程可访问的地址称之为“虚拟地址”，而把实际的物理存储器中可用的地址称为“物理地址”。

2.1.1 虚拟存储器的提出

存储器是存放各种信息的主要场所，是系统中的关键资源之一。如何合理有效地使用这种资源，在很大程度上影响到整个计算机系统的性能，所以存储管理是操作系统的重要组成部分

在 SOC 设计中，由于要求运行的程序也越来越大，需要有足够大的存储器空间去支持程序的运行，对于片上 SOC 系统而言，其存储器容量通常有限，这就更需要使用软件的方法对存储器进行扩充。对此，通常采用的技术便是普通 PC 机中也广泛运用的虚拟存储技术。

所谓虚拟存储器(Virtual storage)，是指以透明方式提供给用户一个比实际内存大得多的作业地址空间。虚拟存储器管理技术首先由英国曼彻斯特大学提出，并在 1961 年在该校的 Atras 计算机上予以实现。70 年代以来，这一技术被广泛使用。它不是任何实际的物理存储器，而是一个容量非常大的存储器的逻辑模型。以处理机提供的逻辑地址访问虚拟存储器，用户可在非常大的地址空间中放心地安排自己的程序和数据，就仿佛真有这么大的内存空间一样。

虚拟存储的关键在于将进程访问的地址空间与物理存储器的实际地址空间分离开来。被进程访问的地址称为虚拟地址(Virtual Addresses)，而在物理存储器

中的可用地址称为物理地址(Real Addresses)。

虽然进程运行时访问的是虚拟地址，但它们实际上是运行于物理存储器中。因此，在进程运行过程中，虚拟地址必须能够被转化为物理地址。这种地址转换将由地址转换机制去完成，本论文所要描述的 TLB 便是一种加速地址转换的一种硬件结构。

2.1.2 页式结构

对于存储器的管理通常可分为连续分区管理和多重分区管理。连续分区存储管理，一般都建立在作业地址空间装入主存的一个连续分区中去。这样会造成零头，而为了“紧缩”存储空间，带来开销大、代价高。而且，正在进行 I/O 数据交换的程序不能搬家。多重分区不失为一种好办法，它解决了作业化整为零地装入主存问题。但由于是硬件实现的，零散的程度不够，只能分为主程序段，子程序段等。从而产生了分页思想。

所谓分页，是把主存存储空间按大小一定的块划分，称为物理块，或页框(page frame)。同时按同样的尺寸去划分作业的地址空间，形成一个个相等的页面，称为逻辑页或虚页。分页是为了系统管理存储器方便而设置的。而从系统外部，用户眼中是不可见的，或者说对用户是透明的。一个用户作业被划分成若干个页，不足一页的部分补齐为一页。因此，作业可以按页为单位，零散地放在主存的不连续的页框中，如何知道一道作业的各个页面与哪些页框对应，则是通过设置页表来实现的，而页的大小总是 2 的方幂。这样，允许把虚地址划分为页号和页内位移量两个部分。

在当今的嵌入式处理器设计中，大多采用了页式结构。虚拟页到物理页的映射由页表完成，页表是以映象方式实现动态再定位的重要工具，它可以用硬件实现，也可以在被保留的系统区内实现。因此，如何快速的读取页表便成了加快虚拟地址与物理地址转换中的重要一环，这也将是 TLB 所要完成的主要功能。

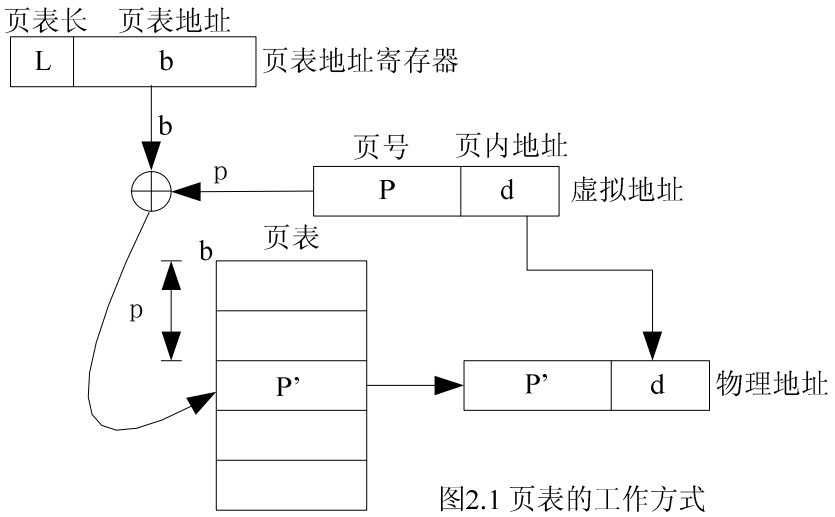

图2.1 页表的工作方式

虚拟存储器的提出极大的扩充了存储器的容量，但是，也带来了新的问题，那就是增加了读取数据的时间。嵌入式微处理器的速度在最近几年持续高速增长，而存储系统的存取速度增长相对缓慢，这也越来越成为 SOC 处理速度的瓶颈。而虚拟存储器的使用使得存取数据时，原本就已经显得漫长的存取时间还要加上地址转换的延时，从而变得更加冗长。对此，采取的策略是在设计中加入高速的 Cache 来加快存取数据的速度，加入 TLB 来加快地址转换的速度。

2.2 Cache 的组成

Cache 存储器是一个容量小但非常快的存储器，它保存最近用到的存储器数据的拷贝。对于程序员来说，Cache 是透明的。它自动地决定保存哪些数据，覆盖哪些数据。现在 Cache 通常与处理器在同一芯片上实现。Cache 能够发挥作用是因为程序具有局部性的特性。也就是说，在任何特定时间，微处理器趋于对相同区域地数据(如堆栈)多次执行相同的指令(如循环)。

一般而言，Cache 主要由以下几部分组成：TAG 部分与 DATA 部分。TAG 部分储存地址标签，DATA 部分储存数据。当地址标签与 TAG 部分中的某个标签相同时，可以从 DATA 部分快速给出数据。

2.3 地址转换的策略

使用虚拟存储技术需要将 CPU 中的虚拟地址转换成物理地址。关于地址的转换有几种策略。

方案一、在对 Cache 进行存取之前进行地址转换(完全内存实现方案)

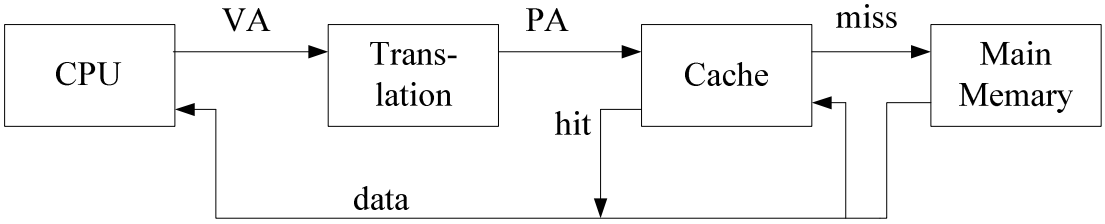

图2.2 Address Translation Before Cache Access

缺陷：

每次虚拟内存查询都将对物理内存进行两次访问：第一次获取所需的页表条目，第二次获取所需的数据。因此，一个最简单的虚拟内存机制将导致对内存的访问时间增加为两倍。这将使得对 Cache 的存取开销变大，而这恰恰是最最期望希望能够提高速度的环节。因此这不是一个可行的方案。

方案二、在地址转换之前对 Cache 进行存取

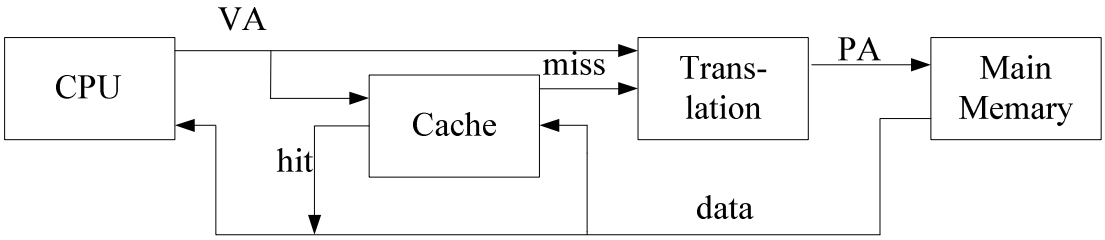

图2.3 Cache Access Before Address Translation

为什么一定要使用物理地址来访问 Cache 呢？我们可以考虑在使用虚拟地址来访问 Cache，而仅当未命中时再进行地址转换。

这一方案与方案看似可以显著提高 Cache 存取时间，然而遗憾的是这一方案也有其缺陷：

同义混淆现象(Synonym aliasing problem)：

两个不同的虚地址映射到了一个相同的物理地址上去⇒两个不同的 Cache 表项保存的数据有相同的物理地址。

出现同义项是因为地址转换机制通常允许重叠转换。如果处理器通过一条地址通道修改了数据，则 Cache 不可能更新第二个拷贝，从而导致 Cache 的不一致。解决这一问题需要相当数目的硬件支持：至少要对物理地址标签进行关联查找以确定是否有复命中。

但是这一方案已经接近我们所期望得到的性能了，那么能否通过操作系统，

通过软件的途径来解决上述问题呢？下面我们就要讨论使用 TLB 来实现地址转换。

解决方案、使用 TLB(地址转换后备缓冲器)

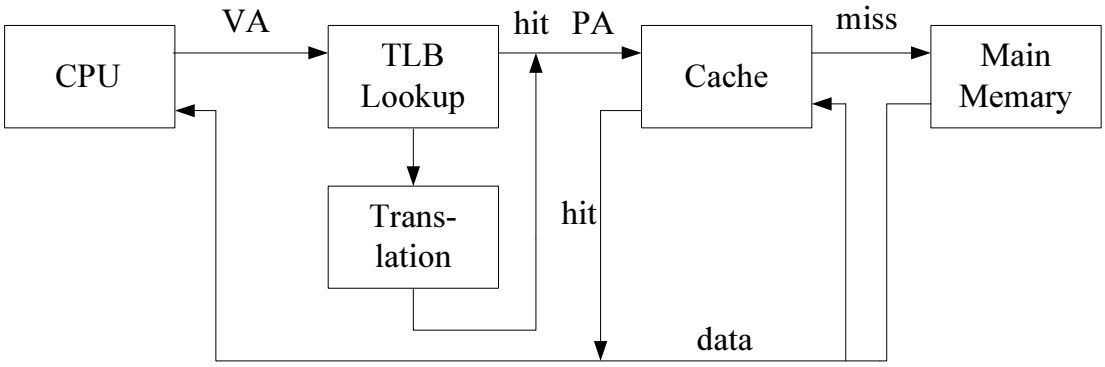

图2.4 Translation With a TLB

通过使用 TLB，以加速虚拟地址到物理地址的转换。使用 TLB 所提供的物理地址对 Cache 进行操作，从而可以有效的避免同义混淆现象，而因为 TLB 可以看作是是一个用于页表匹配的 Cache，它的存取时间与 Cache 的存取时间相仿，远远小于主存的存取时间。因此能快速的实现虚拟地址到物理地址的转换。而且使用 TLB 的机器进一步的减少了对缓存的存取时间。缓存的存取与 TLB 存取相重叠：虚拟地址的高位用于在 TLB 中查找，而低位地址则用作缓存中的变址。

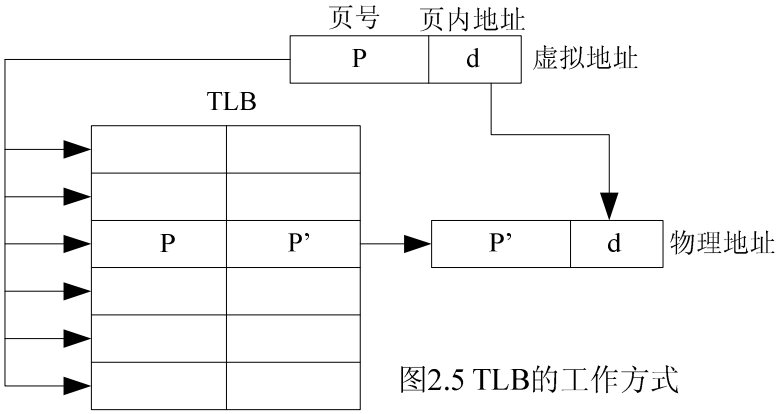

图2.5 TLB的工作方式

虚拟地址通过 TLB 与 Cache 系统的交互具体过程如图 2.5 所示。一个虚地址通常为页号，偏移量的形式，首先，存储器系统查看 TLB 中是否存在匹配的页表项，如果存在，通过帧号和偏移量组合起来形成实地址(物理地址)；如果不存在，则从页表中读取页表项。一旦形成实地址(形式为一个 Tag 和其余部分)，则查看 Cache 中是否存在包含这个字的块，如果有，把它返回给 CPU；如果没有，则从主存中检索这个字。

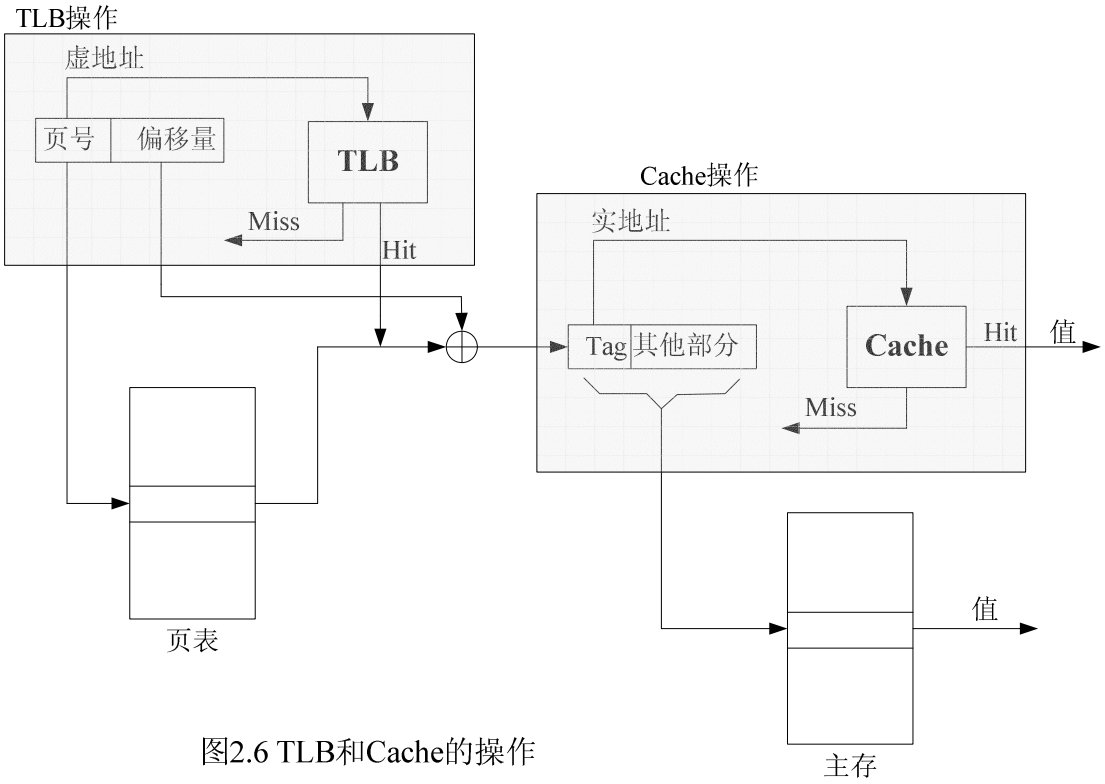

图2.6 TLB和Cache的操作

2.4 TLB 的结构配置

那么如何对 TLB 进行配置以使其能最好的发挥性能呢？

方案 a：进程页表完全放在 TLB 中(完全快表实现)

在本方案下，进程页表全部放在一组由高速寄存器组成的 TLB 中，此方案优缺点比较明显：

优点：由于进程页表全部放在 TLB 中，而进程页表本身的逻辑页号通常是连续顺序的，因此 TLB 中只需存储物理页号即可，不需存储逻辑页号，对其访问是直接存取。而且用户的访问页不会再因为访问进程页表而变成两次访存。

缺点：由于用于制造 TLB 表的寄存器成本较高，导致其容量受限制，不适用于大进程地址空间。以常见的 4GB 进程地址空间和 4KB 页长来说，其进程页表需要 1M 行(即该进程空间有 1M 页)，这意味着这组高速寄存器需要 1M 个高速寄存器需要由 1M 个高速寄存器组成。这样成本太高，不可能实现。因此目前普遍采用的是部分快表实现方案。

方案 b：将内存进程页表部分(正用)内容放在 TLB 中(部分快表实现方案)

在这种方案下，完整的进程页表放在内存中，但其正用的(或最近使用的)那

几行内容放在 TLB 中。对于该方案，可以从以下几点说明：

1、由于只是进程页表的部分内容在快表中，因此快表内容不再按逻辑页号连续顺序排列。这样快表中还要存储逻辑页号。为了加快查找速度，通常将 TLB 设计成按内容寻址 CAM(content addressable memory)。

2、用户程序执行访存时，如果 TLB 命中，则无需进行两次访存，如果 TLB 未命中，则需要给出未命中信号，并且必须访问内存中的进程页表，其访问过程如图 2.1 所示。由于程序的局部性原理，TLB 通常有超过 99％的命中率。

3、该方案是一种折衷的方案，既提高了地址转换的速率，又使硬件的开销不至于过大。在嵌入式处理器中使用的 TLB 通常很小，通常不大于 128－256 个表项，即使在高端机中也是如此。因此，本次设计的 TLB 大小定为 64 个表项。

第三章 TLB 结构的实现

3.1 TLB 的硬件结构描述

由上一章的叙述可知，TLB 所要完成的功能是与虚拟地址的页号进行比对，当匹配的时候输出所存储的地址信息(物理页号)，与虚地址中的偏移量一起构成实地址，交给 Cache，由 Cache 完成信息的输出。同时，还必须满足当 TLB 无法提供所需信息时，需要能通过译码电路将信息写入某个表项中。此外，为了配合操作系统对不同页长的需要，该 TLB 电路最好能够从硬件上实现不同页长的页变换。

因此，从结构上分析，要完成上述功能，需要完成的硬件部分主要有以下几部分：

1、 TAG 部分，此部分用于保存用于比对的标签信号，使用 CAM 单元实现。

2、 DATA 部分，此部分用于保存用于 match 后输出的地址信息，使用 SRAM 单元实现。

3、 连接部分，用于处理将 TAG 部分和 DATA 部分的连接，当 TAG 中有表项 match 时，DATA 部分输出相应的地址，具体包含以下几部分：

a、 位长变换电路，此电路用于改变用于比对的 TAG 的长度，从而改变相应的实地址的长度，实现大小页的变换。

b、 判决电路，当外部数据与内部信息进行比对之后，需要能够判断 TLB 是发生了 hit 还是 miss。

c、 输入电路，控制 TAG 及 DATA 中的位线输入。

d、 输出电路，控制 DATA 中物理地址的输出。

e、 译码电路，当需要更新表项时，需要由译码电路来选中所需要置换的表项。

此外，从整体上讲，本次设计所完成的 TLB 将包含 64 个表项，用于比对的 TAG 长度最多支持 20 位，也可以通过控制，使其支持的位数为 16 位，从而可以从硬件上支持不同大小的页。

3.2 TAG 部分设计

在 TAG 部分的设计中，需要考虑的是速度问题。因为使用 TLB 的目的就是为了要能够加快地址转换的速率，因此，在 TLB 的设计中，设计的重点之一便是如何能够提高转换的速率。而 TAG 部分用于将内部存储的虚拟页号与外部虚地址的 TAG 进行比对，如何提高比对速率至关重要。如果采用逐次比较，则 64 个表项的比较就需要至少 64 个时钟周期，这无疑会使转换速度。因此，我们所使用的 TAG 单元必须使用并行比较。因此本此设计考虑使用 CAM 单元进行 TAG 部分的设计，以实现并行比较。

3.2.1 CAM 单元的结构

CAM 是一种特殊的存储器件，可以有效地并行比较存储的数据与输入数据。图 3.1 所示 CAM 单元由一个 6 管的 RAM 存储单元和一些附加电路组成一位数据比较(M_1-M_3)。当单元被写入时，互补的数据输到位线上，而字线则像标准的 SRAM 单元一样使能。所不同的是 SRAM 单元在输出数据时打开字线，而 CAM 单元则并不输出内部数据，其内部数据仅用作比较。在比较模式下，其字线处于关闭状态，存储的数据(S 以及与其互补的 $\overline{S}$)与位线(B 和与其互补的 $\overline{B}$)输入的数据进行比较。Match 线连接了每一行中所有的 CAM 单元，而且预充至 V_{DD}。如果 S 和 B 匹配，则内部结点 int 放电，M_1 关断，保持 Match 线为高电平。而如果存储数据与输入数据不同，int 将被充电至 $V_{DD}-V_T$，从而导致 Match 线放电。例如如果 $B=V_{DD}$，$S=0$，则 int 通过 M_3 上拉，显而易见这一电路完成了 XNOR 逻辑(即比较器)。

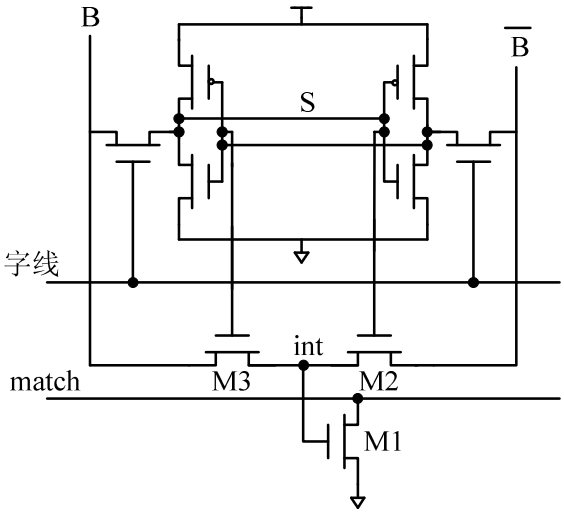

图3.1 CAM单元的结构

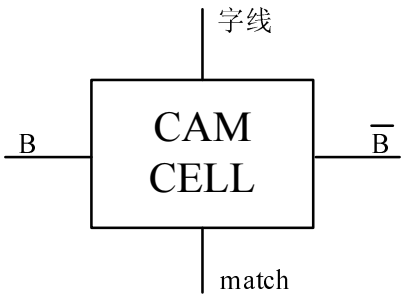

图3.2 CAM单元抽象图

必须注意的是比较器中下拉器件以线或的方式连接到每一列的 CAM 单元中。也就是即使在所在列中只要有一位不同，整个 Match 线都将被下拉到 0。对于一个有 N 列的 CAM Block，大多数(不匹配)将在一个给定的周期内下拉到 0，最多仅有一列的 Match 线可以保持高电平。每一列中有多少个 CAM 单元连接在一起就表示了 TAG 有多少位，每一列就代表了一个表项，有多少列就代表了有多少个表项。而每一行中的 B 和与其互补的 $\overline{B}$ 分别对应的以线或的方式连接起来，当进行比较时所有的位同时进行比较。

3.2.2 CAM Block 的实现

因为本次设计所需要设计的 TLB 有 64 个表项，支持的用于比对的 TAG 位有 16、20 两种，因此在设计 CAM Block 中，需要有 64 列×20 行的 CAM 单元。其中每一行的位线相连，每一列的字线相连，每一列前四位的 Match 线连在一起，将其输出为 Match1，后 16 位的 Match 线连在一起，将其输出为 Match2，其中 Match2 将始终参与对比对结果的输出，而通过选择 Match1 的有效与否可以选择用于比对的 TAG 的位数是 20 位还是 16 位。其具体实现将在 3.5 连接部分中详细叙述。

CAM Block 的结构图如图 3.3 所示：

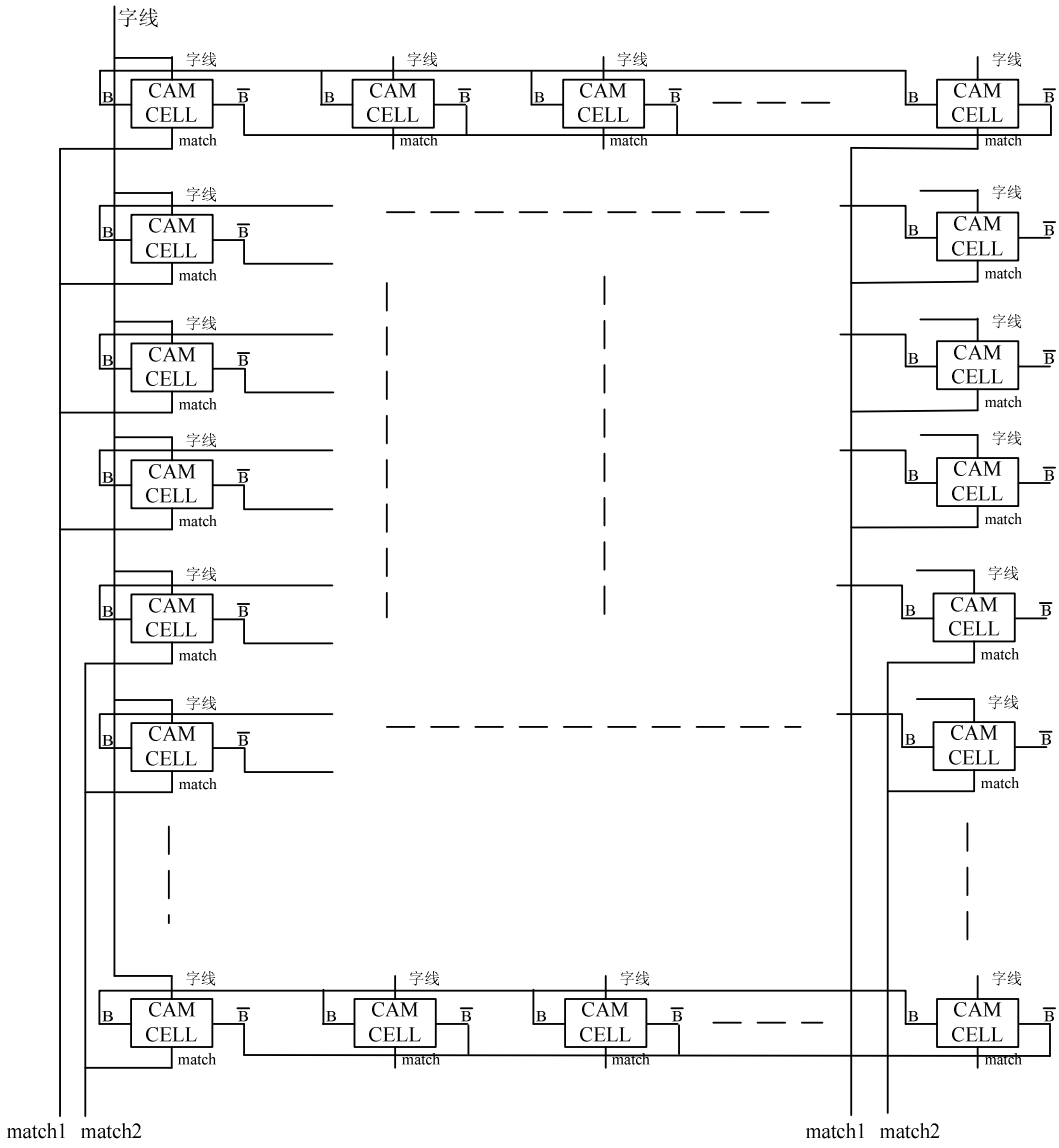

图3.3 CAM Blocks

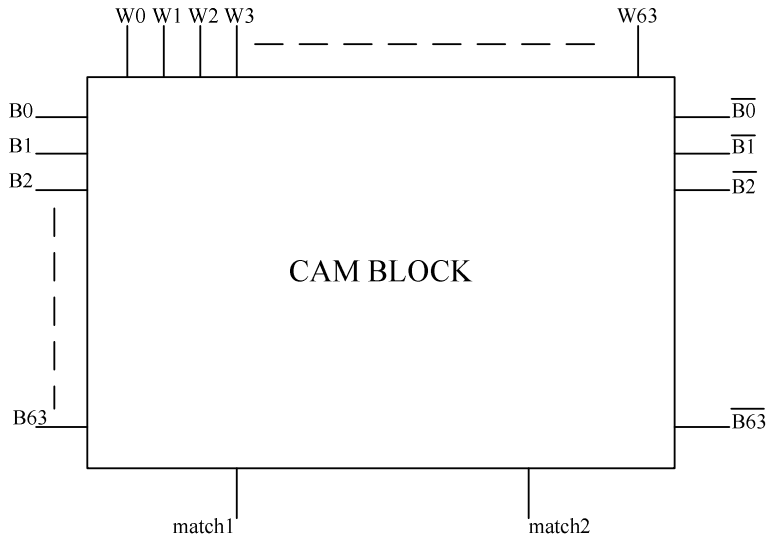

图3.4 CAM BLOCK抽象图

3.3 DATA 部分设计

DATA 部分用来储存物理页号，当 TLB hit 时，DATA 部分输出其存储的物理地址；当 TLB miss 时，DATA 中的某个表项需要与对应的 CAM 一起更新其中的内容。因此，本次设计中采用 SRAM 单元实现 DATA 部分的设计。在 SRAM 的设计中，要求 TLB miss 时，要求某个表项的 SRAM 能与 CAM 一起更新为所要转换的地址，因此，通常把 CAM 的字线与 SRAM 的字线直接相连，当选中了 CAM 中的某个表项的字线，开始更新其中的内容时，相应的 SRAM 中的表项也将字线选中。但是，如果 CAM hit，也要求对应的 SRAM 表项将字线打开，读出存储的地址，但是此时 CAM 的字线应该处于关断状态。因此，如果采用的是标准的单字线的六管单元，则如何控制其字线以及与 CAM 单元的连接将变得十分复杂。为此，在本次设计中，对 SRAM 单元采用双港口设计，将对 SRAM 的读和写由两根字线分别控制，从而为简化 CAM 和 SRAM 的连接打下了基础。

3.3.1 双港口 SRAM 单元的设计

双港口 SRAM 单元的显著优点是能够方便的控制数据的流向。如图所示，双港口 SRAM 单元有两根字线 Wa，Wb，当其中任何一根导通时，都可以向单元内部写入信息或者从单元内读出信息，从而可以双向的控制数据流。在本次设计中，并不需要利用 SRAM 双向控制数据流，只需要利用其两根字线的特点，将字线 Wb 与 CAM Block 的字线相连，控制对 SRAM 的写入；而字线 Wb 则受 CAM Block 中 match 线的控制，当比对 match 时，将 SRAM 中的物理页号读出。

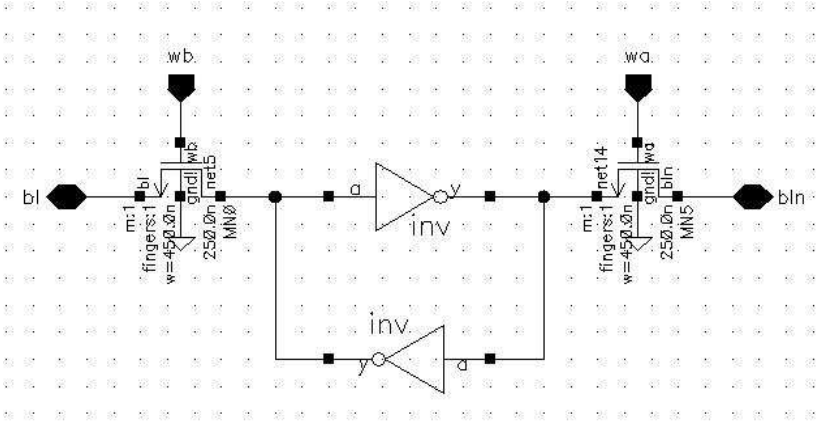

图 3.5 双港口寄存器单元

3.3.2 SRAM Block 的实现

与 CAM Block 的设计相类似，因为本次设计所需要设计的 TLB 有 64 个表项，所以用于存储物理地址的 SRAM 有 64 列。因为可以支持的页面大小有 16 位、20 位两种，为了在大小页变换时能够迅速切换，而不必更新 SRAM 中的表项，因此在设计 SRAM Block 时，考虑使用分别为 16 位的 SRAM 单元 SRAM1 和 20 位的 SRAM 单元 SRAM2 去分别支持大页和小页。每一块中每一行的位线相连，每一列的字线分别相连 Wb 控制写，Wa 控制读。两块 SRAM 的字线并不直接连接，而是通过一个简单的连接控制电路，其具体实现将在 3.5 连接部分中详细叙述。

3.4 TLB 的实现(CAM 与 SRAM 的连接)

TLB 的内部结构结构模型如图 3.6 所示：

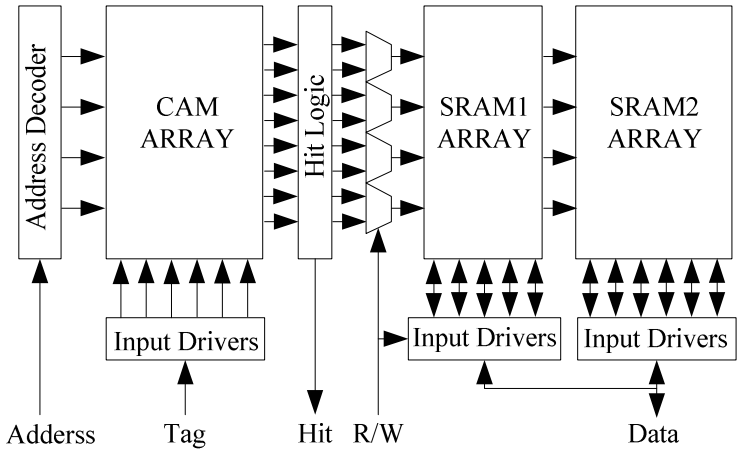

图3.6 TLB的内部结构

图中具体电路的实现如下所述：

3.4.1 判决电路及位长变换的实现

位长变换电路的原理非常简单，只需控制 CAM 单元中 Match1 与 Match2 是否有效就可以决定页号为多少位。当只有 Match2 有效时，页号为 16 位，当 Match1， Match2 都有效时，页号为 20 位。因为 match 线的使能与判决电路密切相关，因此，将位长变换电路与判决电路一起描述。重点在于 match 线的使能。

判决电路就是当 CAM 中 match 时，要能输出信号表示已经 match，并输出

物理页号；当 miss 时，输出信号表示 miss。

设计时，需要必须注意的一个问题是 match 线在比对前必须全部上拉为高电平，而此时不能将其用于判决，否则将会导致 SRAM 中多根字线同时导通，导致逻辑出错。因此，必须在 CAM 比对完后才能对 SRAM 中的字线进行操作。

因此，对于上述电路考虑采用以下步骤实现：

1、当接收到需要进行比对的信号后将 match 线上拉，同时将信号线 y 下拉为 0；

2、关闭对 y 和 match 的驱动，进行比对结果，并用比对结果控制 y；

3、使用 y 对 SRAM 的字线进行控制。

4、根据字线是否被选通输出 hit 或 miss 信号。

实现上述步骤的电路如图 3.7 所示，其原理是：

a、当 MMU 收到需要比对的信号时，使信号 a 给出一个低电平脉冲，平时 a 为高电平。当 a 为低电平时，驱动左上方的上拉电路将两根 match 线 m_1，m_2

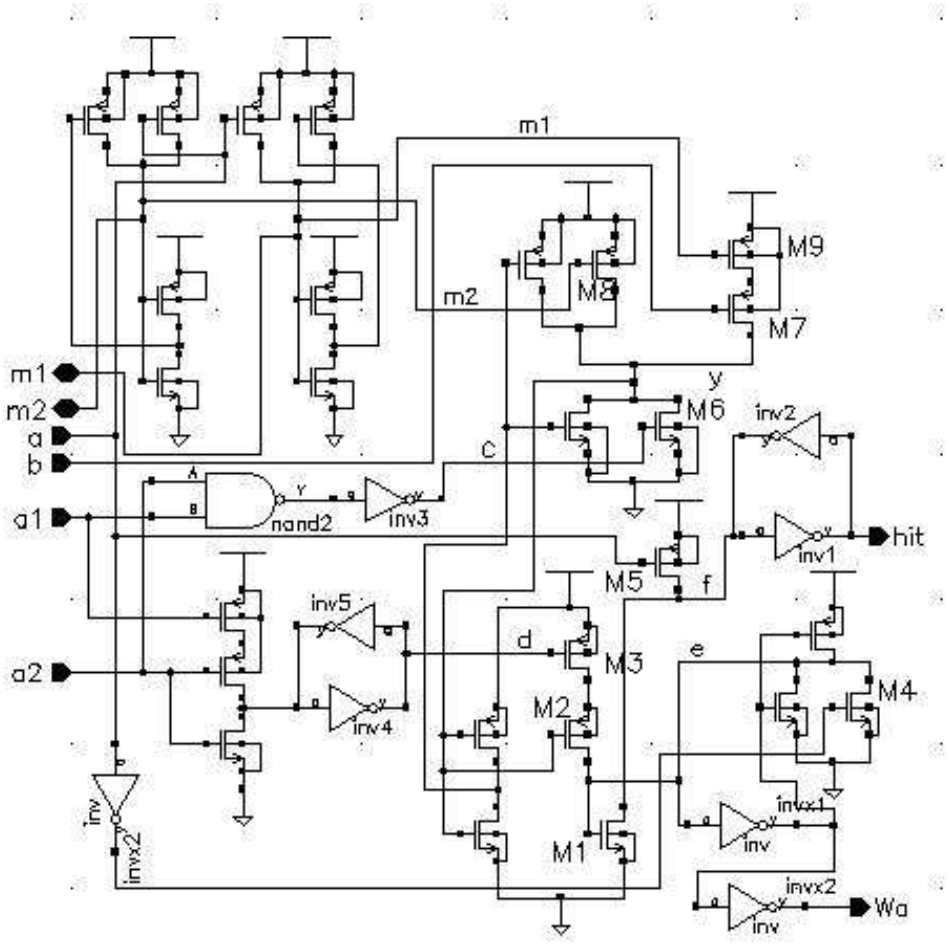

图 3.7 TLB 主电路(判决及位长变换)

均上拉为高电平。 a 经过一个导向器后的高电平脉冲将导通 M_4 管，将字线 Wa 设为低电平，而且比对输出 hit 也将在 M_5 管及 M_1 管的作用下被置为低电平(表示 miss)。同时，a_1，a_2 信号也均置为高电平，这两个信号经过与门后的输出 C 将 M_6 管导通，把中间信号 y 下拉为低电平(中间信号 y 将控制 Wa 的输出，只有当 y 和 d 为低电平，Wa 才可能输出高电平)。而当 a_2 信号为高电平时，中间信号 d 也被上拉至高电平，从而禁止 Wa 输出高电平。

b、将 a_1 置为低电平，则信号 C 将变为 0，从而不再控制中间信号 y。

c、m_1，m_2 根据 CAM 中的比对结果而变化，其结果将通过 M_7，M_8，M_9 管决定是否将 y 上拉至高电平(当发生 miss 时，m 信号将为 0，从而导通 PMOS 管将 y 上拉至高电平，若 hit，则对 y 无影响，y 将通过两个导向器的锁存保持原有状态)。其中 m_1 信号的有效与否取决于输入信号 b，当 b 为高电平时，M_7 管截至，无论 m_1 信号如何变化，都无法影响中间信号 y，此时只有 m_1 有效，则此时 TAG 为 16 位；当 b 为低电平时，m_1，m_2 均有效，此时 TAG 为 20 位。

d、将 a_2 置为低电平，则信号 d 将被下拉至低电平，则输出 Wa 取决于比对结果，若 match，则 y 为 0，将信号 e 上拉至高电平，e 经过两个导向器之后的输出 Wa 将为高电平，从而打开 SRAM 输出物理页号；如果不能 match，则 y 将为高电平，从而使 Wa 保持原有的低电平，无法导通相连的 SRAM 单元。

e、中间信号 e 还将用于控制右下方 M_1 管的导通与否，每一个表项经过 M_1 管后的输出均连在一起，接到中间信号 f 与 M_5 的输出连在一起。a 信号为低电平脉冲时，f 被上拉至高电平。只要有一根字线被选中，则 M_1 管将被打开，f 将被下拉至低电平，其反相输出 hit 将为高电平，表示命中，否则 hit 信号将保持为低电平，表示未命中。

在导出 Cadence 中电路图 3.7 的网表文件后，在 Hspice 中对其进行仿真，获得如下所示时序图：

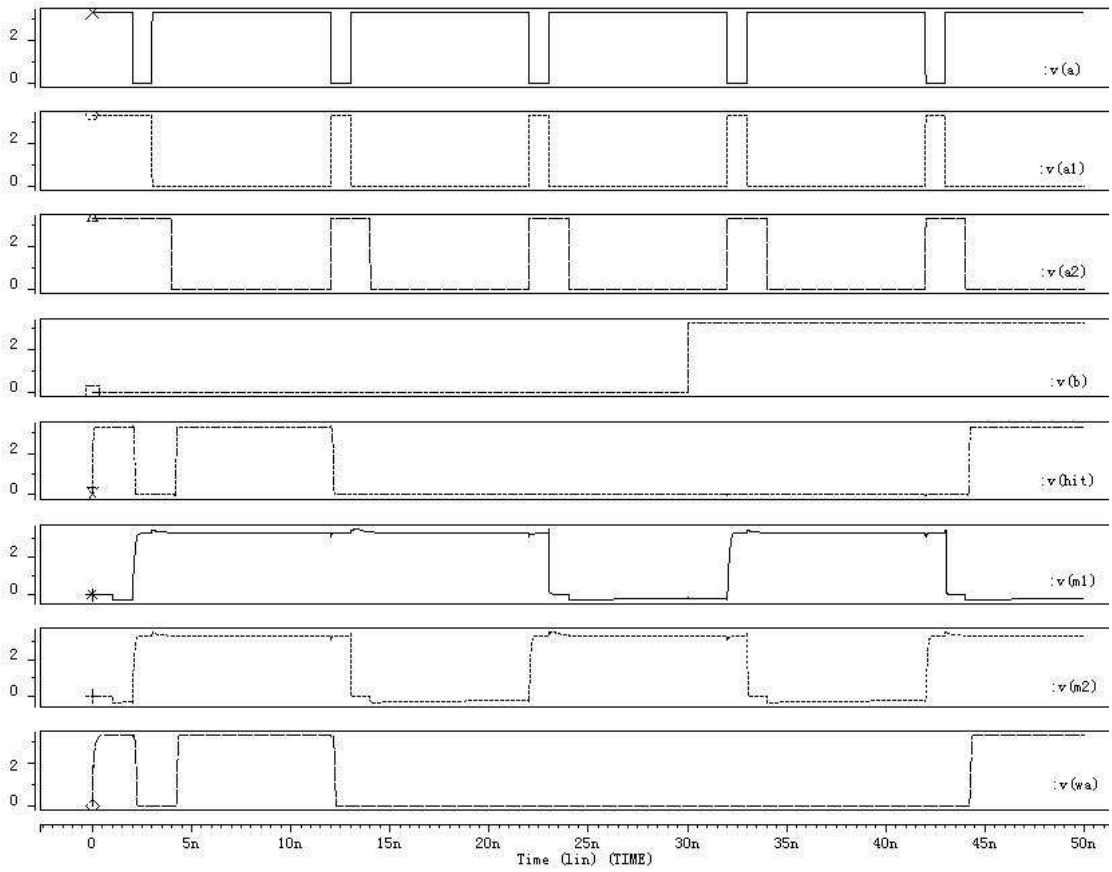

图 3.8 TLB 时序仿真

上图以 10ns 为间隔，在第 3 个 ns 时信号 a 有一个 1ns 的低电平脉冲，信号 a_1，a_2 被置为高电平 a_1 在 1ns 后降为低电平，a_2 在 2ns 后降为低电平。0-10ns 对应的是 b=0，m_1=1，m_2=1，结果是 match，字线被选通，输出 hit 信号；10ns-20ns 对应的是 b=0，m_1=1，m_2=0，结果是 miss，字线未选通，输出 miss 信号；20ns-30ns 对应的是 b=0， m_1=0，m_2=1，结果是 miss，字线未选通，输出 miss 信号；30ns-40ns 对应的是 b=1， m_1=1， m_2=0，结果是 miss，字线未选通，输出 miss 信号；40ns-50ns 对应的是 b=1， m_1=0， m_2=1，结果是 match，字线被选通，输出 hit 信号。可见仿真结果完全符合设计电路时所作分析，hit 与 m_1， m_2，b 中间满足以下关系：

$$hit = m_2 \cdot (b + m_1)$$

3.4.2 输入电路的实现

CAM 中位线的输入可以考虑将来自虚拟地址中的信号经过一个导向器对锁存后连接到位线 $\overline{b}$，经过导向器对后再经过一个导向器连接到 b。这样设计对于 CAM 的写操作没有问题，但是，对于比对操作存在几个问题。首先是导向器存

在延时，会使得比对不精确；其次，最主要的是无法控制比对的进行，当需要进行比对，对 match 线进行上拉时，位线上保留的无论是上次比对时输入的比对信号还是此次比对输入的比对信号，至少有 63 个表项将显示为不匹配，导通图 3.1 中所示的 M_1 管。此时对 match 线进行上拉，不仅无法将 match 上拉为高电平，还将导致短路。因此，在对 match 进行上拉时，必须要将图 3.1 中所示的 M_1 管关断。通过对图 3.1 的分析可知，除了位线信号与存储的内容相匹配外，当位线上的信号 b 与 $\bar{b}$ 均为低电平时，传送到 int 的信号始终为低电平，M_1 始终关断。因此可以考虑使用电路使位线上的信号对在比对以外的时间内均为低电平，仅当进行比对或写入时才输入高电平。根据以上分析所设计的输入电路如图 3.9 所示：

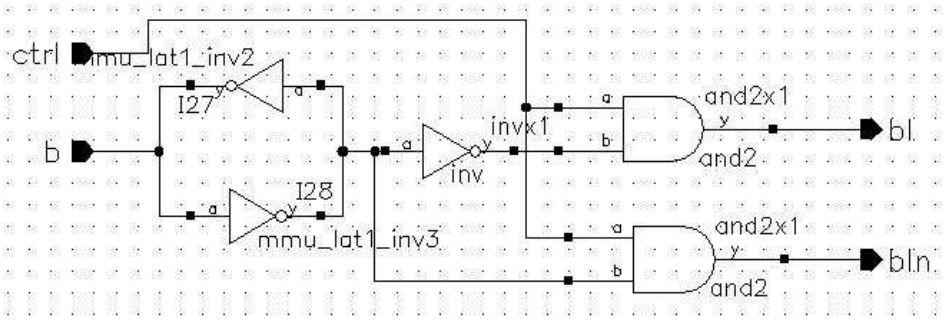

图 3.9 CAM 的位线输入

在上图所示电路中，当对 CAM 进行写或比对操作结束后，控制信号 ctrl 将为 0，从而使 bl 和 bln 均为低电平，可以用于比对前对 match 线的上拉，当需要进行比对或写入时，再将 ctrl 信号置为高电平，则 bl 和 bln 将输出一对互补信号，而且互补信号将同时输出，bl 信号不会受中间导向器的作用而延时。

对于 SRAM 的输入设计，所要考虑的一个问题就是进入 SRAM 的位线是输入输出共用的，因此在向外读数据前必须将输入信号断开。具体电路实现如下

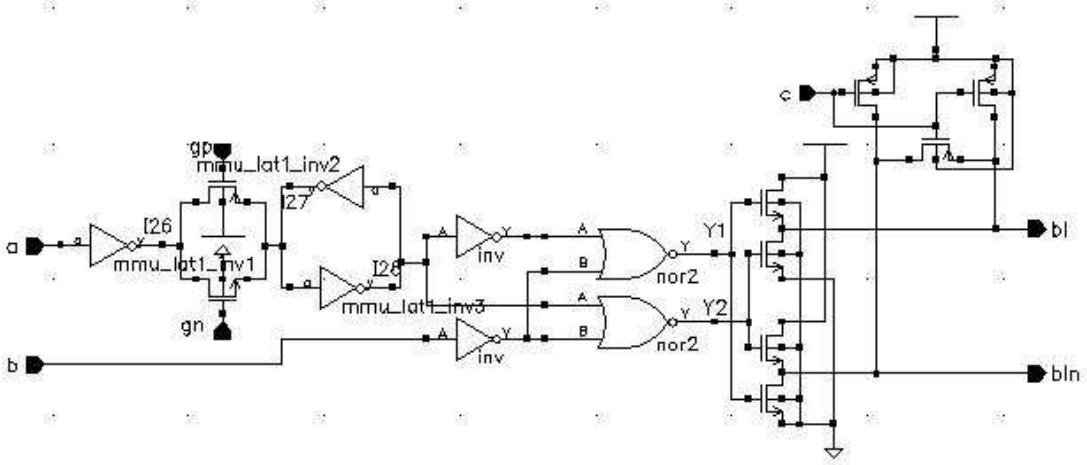

图 3.10 SRAM 的位线输入

在上图所示的电路中，输入信号 a 首先经过一个锁存器进行锁存，然后经过一个组合电路输出 Y_1，Y_2，Y_1 和 Y_2 的功能为：

$$Y_1 = ab \qquad Y_2 = \bar{a}b$$

当信号 b 为 0 时，输出 Y_1，Y_2 均为 0，bl 和 bln 保持。

当信号 b 为 1 时，输出 Y_1，Y_2 为一对互补信号，输出 bl 和 bln 分别为：

$$bl = Y_1\overline{Y_2} \qquad \overline{bl} = \overline{Y_1}Y_2$$

再将 $Y_1 = ab$；$Y_2 = \bar{a}b$ 代入上式，得：

$$bl = a \qquad \overline{bl} = \bar{a}$$

可见信号 b 实现的是控制选通的功能。当信号 b 为 1 时，输入选通，来自地址线的信号 a 输入给位线；当信号 b 为 0 时，输入通路断开，此时可以保存输入的信号并且向外读出信号。

此外，在输入电路中还加入了一个预充电电路，此电路与 CAM 中对 match 线进行预充电的电路原理相同，利用 P 管的导通对线路进行上拉，但是在 SRAM 输入电路的设计中，预充电电路所要完成的功能是在写入数据前把 bl 和 bln 上拉至 $1/2V_{DD}$，从而加快写入数据的速度。

3.4.3 输出电路的实现

输出电路所需完成的功能是当 CAM 中 match 时，将 SRAM 中所存储的物理地址正确地输出。因此，输出电路的设计可以视为是 SRAM 部分的设计完善。在 3.3.2 节中已经说明了为了加快大小页变化，将采用分别为 16 位和 20 位的两块 SRAM，因此，如何处理好两块 SRAM 的输出，并与虚拟地址中的偏移量一起合并成物理地址便成为了输出电路所要完成的功能。

当使用大页时，页号为 16 位，所以 CAM 中 M1 信号无效，仅有 16 位比对有效，而 SRAM 中的输出来自 16 位的 SRAM1，与 16 位的偏移量一起组成所要输出的物理地址；当使用小页时，页号为 20 位，CAM 中 M_1，M_2 信号均有效，SRAM 中的输出来自 20 位的 SRAM2，与 12 位的偏移量一起组成物理地址。信号的具体流向见图 3.11 所示：

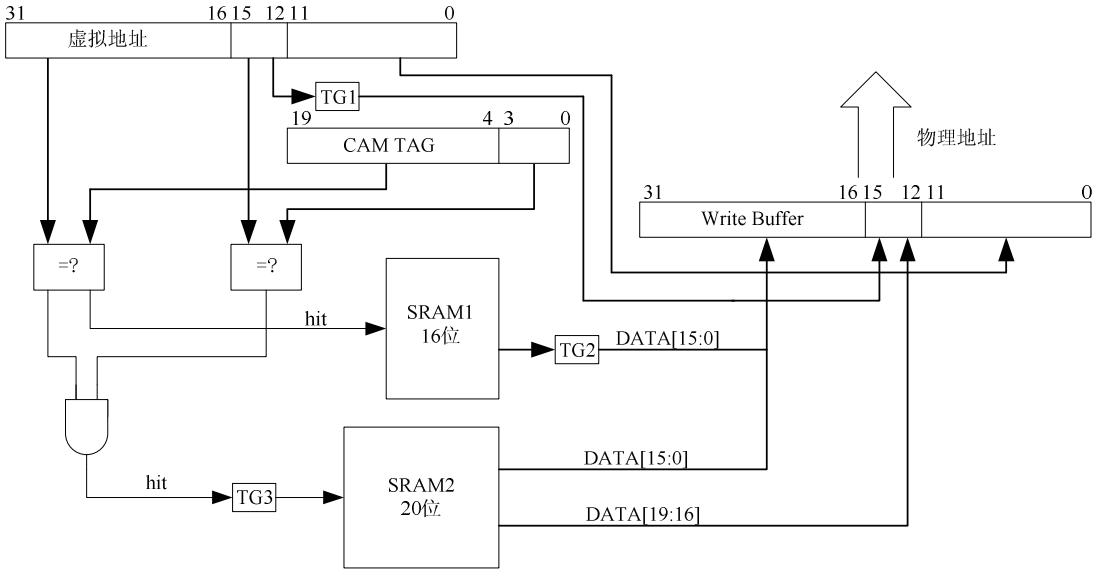

图 3.11 TLB 数据具体流向

当处于小页模式时，传输门 TG_1，TG_2 关闭，TG_3 导通，SRAM2 的 20 位输出输入到 write buffer 的高 20 位，作为物理页号，而虚拟地址的低 12 位输入到 write buffer 的低 12 位，作为偏移量，两者一起组成物理地址。

当处于大页模式时，传输门 TG_1，TG_2 导通，TG_3 关闭，SRAM 的 16 位输出输入到 write buffer 的高 16 位，作为物理页号，虚拟地址的[15:12]通过 TG_1 后与虚拟地址的低 12 位一起输入到 write buffer 的低 16 位作为 16 位的偏移量。

在处理信号流向时，虚拟地址的低 12 位与 write buffer 的低位直接相连，VA[15:12]与 DATA[19:16]则对应输入到 write buffer 的[15:12]，由于 TG_1 和 TG_3 的控制使得两个输入只有一个被导通，此逻辑结构简单，仅为两个传输门。关键部分在于 SRAM1 与 SRAM2 的输出处理。其输出共用 16 位的数据通道，而且其选通又涉及到 match 电路的输出，因此输出电路的设计重点就在于对 SRAM1 和 SRAM2 的选通以及对共用通道 DATA[15:0]的控制。

为此，本次设计中所采用的方法是将 SRAM1 与 SRAM2 的字线通过一个开关电路联系在一起，而 SRAM1 的输出也通过一个开关电路接到 SRAM2 的 DATA[15:0]输出上去。具体的电路实现见下图所示：

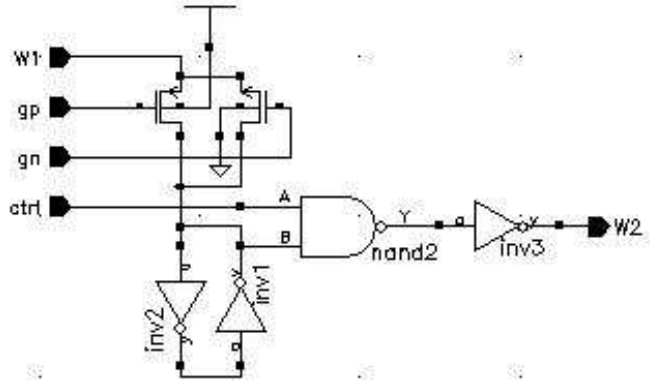

图 3.12 字线开关电路

在上图所示开关电路中，W_1 为 SRAM1 的字线，W_2 为 SRAM2 的字线，当 ctrl 信号为 0 时，处于大页模式，SRAM2 字线关断，没有输出；当 ctrl 信号为 1 时，处于小页模式，SRAM2 中的字线与 SRAM1 中的字线直接相连，SRAM2 正常运作。

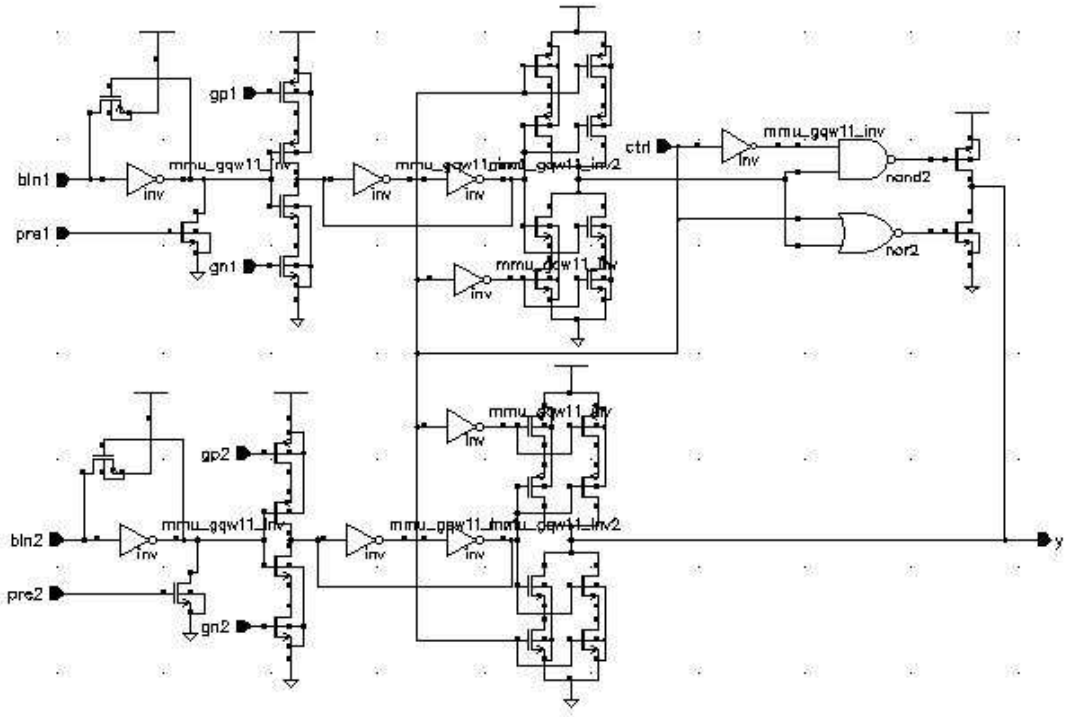

图 3.13 SRAM 输出电路

在 SRAM 输出电路中，SRAM1 和 SRAM2 经过相似的电路输出，但是 SRAM1 的输出经过一个由 ctrl 信号控制的电路控制开断。当 ctrl=0 时，SRAM1 的输出能通过该电路，当 ctrl=1 时，SRAM1 的输出被该电路关断。因此，当采用大页模式时，ctrl=0，SRAM2 的传输控制将其输出关断， y 为 SRAM1 中的输出；当采用小页模式时，ctrl＝1，SRAM1 的输出被关断，y 为 SRAM2 中的输出。

3.4.3 译码电路的实现

当 TLB 中比对结果为 miss 时，需要往 CAM 及 SRAM 内写入信息，这就需要使用译码电路来对字线进行选通。这就需要实现译码逻辑，本次设计中的译码电路如图 3.14 所示：

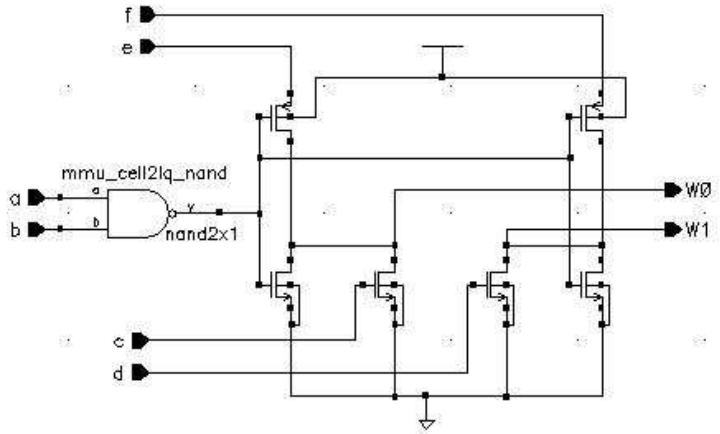

图 3.14a 译码子电路

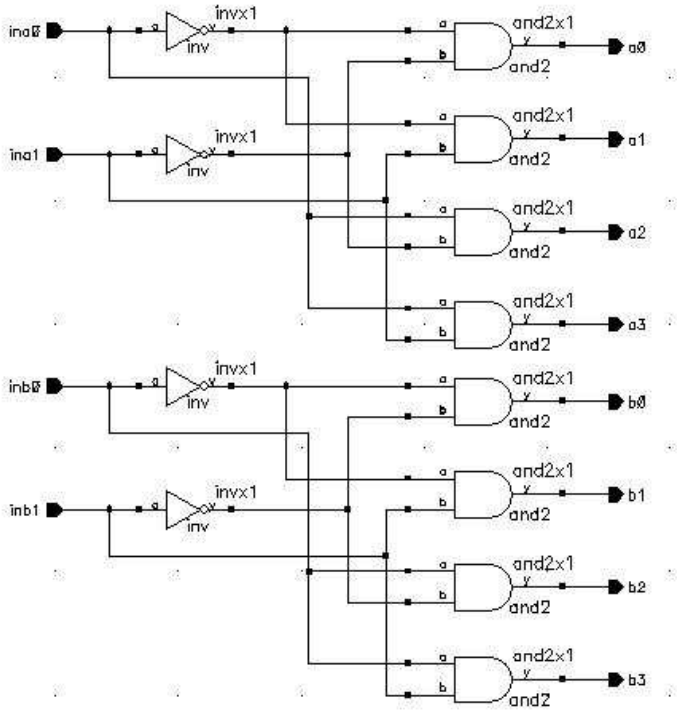

图 3.14b 2-4 译码电路

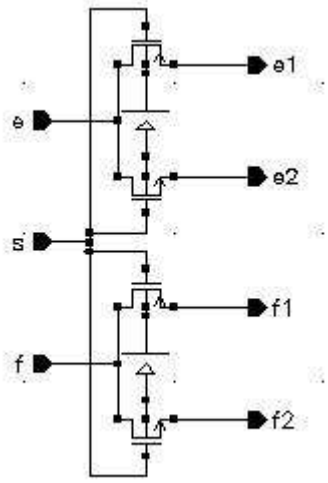

图 3.14c 块选择电路

译码电路由 32 个译码子电路以及 2-4 译码电路，块选择电路组成。32 个译码子电路分成两块，由块选择电路选择哪块工作。在译码子电路中，可以选通字线 W_1 或 W_2，a 和 b 为译码部分，只有当 a 和 b 均为高时才有可能将字线上拉。

在译码子电路中，当被选通时(即 a 和 b 均为高电平)，输出 W_1 或 W_2 的值即分别为输入 e 和 f。c 和 d 为下拉信号，当写操作完成后，需要将输入信号 e 和 f 关闭，同时下拉信号 c 和 d 将为高电平，将字线下拉为低电平，从而结束写操作。原始的输入信号 ina_0，ina_1 和 inb_0，inb_1 经过 2-4 译码构成了一个 4-16 译码器，可以对 16 个译码子电路进行选择。因此，在设计时，将 16 个子电路设置成一块，每块中的子电路的 c，d，e，f 信号均相同，由 a，b 控制选通，而不同的块的选择则通过图 3.14c 的块选择电路完成。每块可以对 32 个表项的字线进行控制，两块一共可以对本论文所设计的 64 个表项的 TLB 进行译码控制。

第四章 性能优化－低功耗设计

在 TLB 的设计中，由于 TLB 通常与嵌入式处理器做在同一块芯片中，其功耗占了芯片的相当大的一部分，因此如何减少其功耗也成了 TLB 设计中的一个重要部分。虽然目前 CMOS 的功耗主要还是由动态功耗组成，但是随着工艺尺寸的缩小，漏电功耗已经受到越来越多的关注，在低功耗，高性能设计中开始扮演了越来越重要的角色。本论文将低功耗设计的重点放在了减少漏电功耗方面去进行考虑，结合当今对减少漏电功耗的研究，其目的在于对针对减少漏电功耗的低功耗设计有一定了解，对未来的研究方向进行一些先期的研究。

4.1 几种 Cache 中常用的低功耗技术

目前有很多电路研究通过减少漏电流以达到降低功耗的目的，这些技术或者通过在通向地线的通道中制造高阻以实现电路的完全关断；或者折衷地增加运算时间以减少运算功耗。在某些情况下，这些技术可以完全在电路级实现，而无需改变结构或者只需对结构稍作改动。因此，本文将主要考虑如何在保存 TLB 原有结构的基础上，如何通过一些简单的电路改进来达到减少漏电功耗的目的。下面是几种 Cache 中常用的低功耗设计。

4.1.1 Gated—V_{DD} 技术

以 Gated－V_{DD} 为代表的技术通过有选择的关断一些不大可能被重新用到的 Cache 的线可以减少漏电功耗。

其核心思想是当 Cell 设置成低功耗模式下时，通过高门限(V_T)晶体管来关断内存单元的功耗以减少漏电功耗。高门限器件可以显著降低电路中的漏电流，因为漏电流的大小与门限电压的指数有关。这种方法能够显著的减少漏电流，他的主要缺点是当跳转到低漏电流模式时，Cell 中的信息会丢失。这就意味着当对 Cell 中的信息进行存取时会有明显的性能衰减，而将信息重新载入又将带来很大的性能损耗，从而抵消其所节省的功耗。为了避免这一缺陷，需要使用相当复杂的适应算法，并且相当慎重的决定将关闭哪根线。

4.1.2 ABBMT—CMOS 技术

关断 Cache 线并不是减少漏电功耗的唯一方法，通过把 Cache 线设置成低功耗的睡眠模式也可以显著的降低漏电功耗。当处于睡眠模式下时，Cache 线中的信息得以保存，然而在存取里面的数据前必须把它设置成高功耗模式。一种多门限 CMOS 自偏置适应电路(ABBMT－CMOS)可以实现 Cache 的睡眠。

在这种方法下，当 Cell 被设置为睡眠模式时，通过增加晶体管中源和衬底间的电压使晶体管的门限电压动态增加。增高 V_T 可以减少漏电流，同时也能在睡眠模式下保持 Cell 中的信息。然而，为了避免使用双阱工艺，睡眠模式下 V_T 的动态缩放在 NMOS 器件中通过增加源电压实现，而在 PMOS 器件中则通过增加阱的衬底电压实现。在这种机制下，虽然通过内存单元中的漏电流显著减少，但是电路的供电压增加了，这将抵消一些节约的漏电功耗。

而且这种机制在每次电路进入睡眠状态时都要改变 N 阱电压，而 PMOS 器件中的 N 阱电容相当大，这增加了将 Cache 中的 Cell 转换到高功耗模式时所需要的能量。而且明显增加了转入和转出睡眠状态的时间。与 Gated—V_{DD} 技术相同，ABBMT—CMOS 技术也需要特殊的高 V_T 器件用于控制逻辑。

4.1.3 动态 V_{DD} 缩放技术(DVS)

除了增加门限电压外，降低电源电压也可以降低漏电流。动态 V_{DD} 缩放技术(DVS)就是基于这一原理提出的。

在 DVS 技术中，通过将 Cell 的电压缩放到 V_T 的大约 1.5 倍时，内存单元中的信息能够保存。对于一个目前典型的嵌入式处理器而言，这个睡眠电压通常不足 1V。由于在这种 DVS 机制中电压和漏电流都显著下降了，因此可以大幅下降漏电功耗。而且由于功率通道的电容比 N 阱中的电容要小的多，因此在 DVS 机制下两种功率间的转换时间要比在 ABBMT—CMOS 机制下小的多。在下面一节中将对这种 DVS 技术进行概述。

4.2 DVS 技术概述及其电路结构

DVS 技术的理想实施对象是嵌入式 Cache，其核心思想是为每根 Cache 线设

置两个不同的供电压，从中选择一个作为正常的供电压，而另一个则作为睡眠电压。在经典的六管 SRAM 单元中，只要睡眠状态时的供电压大于组成导向器的 MOS 管 V_T 的 1.5 倍时，则导向器所锁存的信息将得到保存，而当写入或读出数据时则将供电压恢复到正常模式下。

恢复睡眠电压至正常供电压只需要一两个时钟周期。在 SRAM 中应用时需要决定哪根线将被设置成睡眠模式，一个简单的算法就是每个一段时钟周期把所有的 Cache 线都设置成睡眠模式，然后等读入时再唤醒。由于程序的局部性原理，这一简单的算法就可以有效的将大部分未使用的 Cache 线设置成睡眠状态而取得减少漏电功耗的目的。由漏电流公式可知，电压缩小将显著减少漏电流大小。漏电流减小再加上电压的减少，其综合效果将显著减少漏电功耗。

图 4.1 阐述了使用 DVS 技术的内存单元的电路结构。

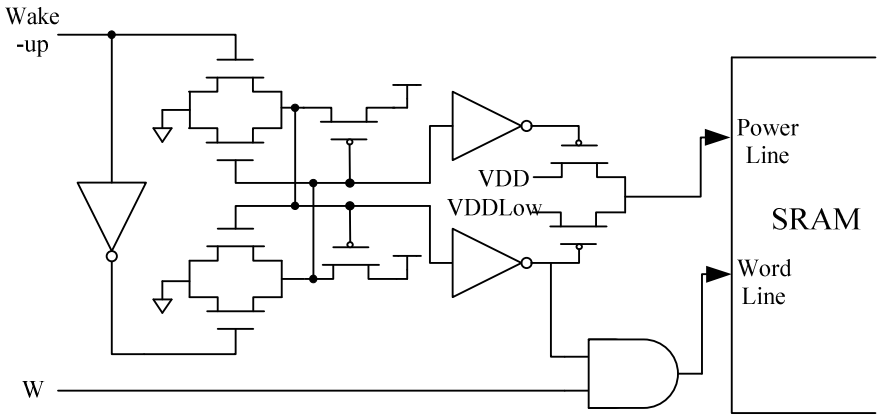

图4.1 DVS电路

当 Wake-up 信号为高电平时，通过组合电路，将把电压 V_{DD} 传送给 SRAM 的电源线，处于正常工作模式；Wake-up 信号为低电平时，该电路把电压 V_{DD}Low 传给 SRAM 的电源线，SRAM 将处于低功耗状态。该 Wake-up 信号为边沿触发。当处于低功耗状态时，字线将无法选通，以低功耗时防止字线选通后对内部逻辑的破坏。

4.3 DVS 技术在 TLB 设计中的使用

通过上一章的描述，已经可以了解到 TLB 是由 CAM 单元和 SRAM 单元所组成的，因此在本论文中，对于 TLB 的功耗优化也主要从 CAM 和 SRAM 部分分别进行分析。

4.3.1 关于 CAM 单元的优化设计

由 3.2 对 CAM 的描述可知，对于整个 CAM 块而言，因为是并行比较，输入的信号同时与整个块中的所有表项同时比较。因此，若是采用 DVS 技术简单的使其中的某些表项设置为睡眠模式，则相当于减少了 TLB 的表项。而当 TLB 未命中时，需要从内存中把所需信息写入到 TLB，而从内存写入 TLB 需要相当长的时间，这将带来相当大的性能损耗。而在 Cache 的 SRAM 中，如果要对睡眠状态中的 Cache 线进行操作的话，只需要耗费一两个时钟周期的时间把该线唤醒，而不会造成其他损失。因此，在对 CAM 单元进行功耗优化时不能简单的采用 DVS 技术。

不过通过对 CAM 结构的分析，可知当其存储了所需用于比对的信息之后，可以降低用于锁存信息的两个导向器的供电压，从而达到减少漏电功耗的目的。而当需要对 CAM 写入信息时，可以通过字线 W 把导向器的供电压设置为正常状态。

用于 Spice 仿真的电路模型及各节点如图 4.2 所示：

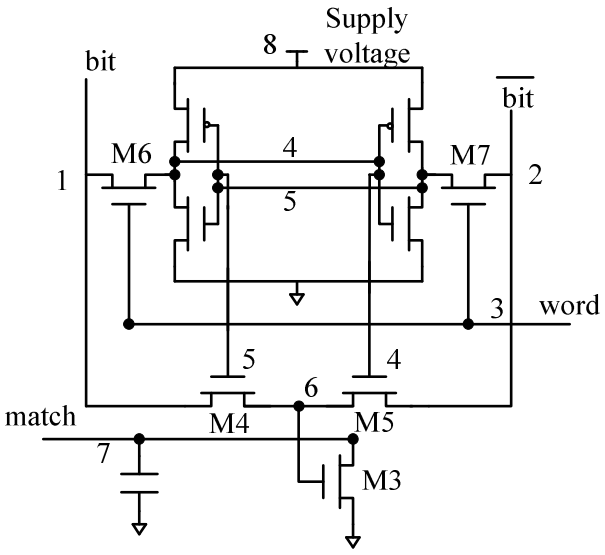

图 4.2 电路模型

事实上由于所存储的用于比对的信息并不与外部 Match 线发生接触，只是用于导通 MOS 管 M_4，M_5，而且在字线恢复为 0 后，导向器也将恢复为睡眠状态，其仿真结果如下图所示(此处假设睡眠状态的供电压为 0.6V)：

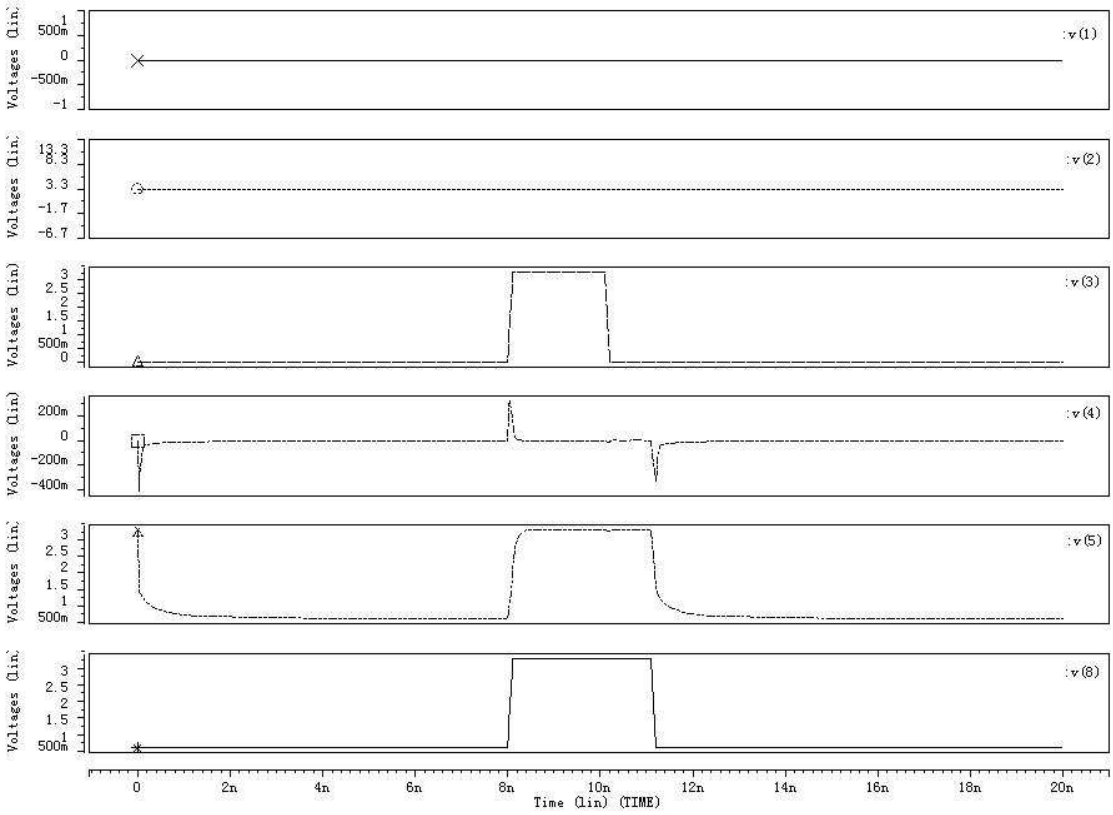

图 4.3 导向器的电压分析

可见对导向器单元设置正常状态仅仅使导向器单元在字线为 1 的短暂时间内以一个较高的电压工作，而对整个电路而言并没有任何改善，毫无意义。因此只须设置一个睡眠电压，将 CAM 单元中导向器的供电压设置为睡眠电压既可达到降低漏电功耗的目的。

根据导向器的结构可知当供电压缩小到 $1.5V_T$ 仍可以保存所存储的信息，需要注意的是使 Match 线降为低电平所需要的时间，如果这一时间过长，将严重影响系统速度，甚至造成功能错误。而 Match 线在内容匹配的时候，也需要能保持高电平。根据一般嵌入式 Cache 的特征，假设其 Match 线的正常供电压为 3.3V，频率为 60MHz，则我们所设置的睡眠电压必须满足当所存储的信息与外部信息不匹配时，必须使 Match 线在 15ns 内下降至 1.5V 以下方可视为合格。

Match 线下拉的速度与实际电路中 Match 线的寄生电容有关，寄生电容越大，下拉速度越慢，而寄生电容越小，则 Match 线为高电平时的漏电会增加。一般而言，Match 线上通常只有 0.001PF 数量级的寄生电容，因此可以认为其为 0.01PF，为了能确保电路在最坏情况下工作正常，在仿真比对结果为不匹配，考查 Match 线下拉速度时使用的寄生电容定义为 0.1PF；而在仿真比对结果为匹配，考查

Match 线保持高电平能力时，使用寄生电容为 0.001PF。仿真结果表明在睡眠电压为 0.6V 的情况下，电路性能依然良好。下图为睡眠电压为 0.6V 时的仿真结果。所使用的工艺库是新加坡 Chartered Semiconductor Manufacturing 的 0.25um 的工艺库。

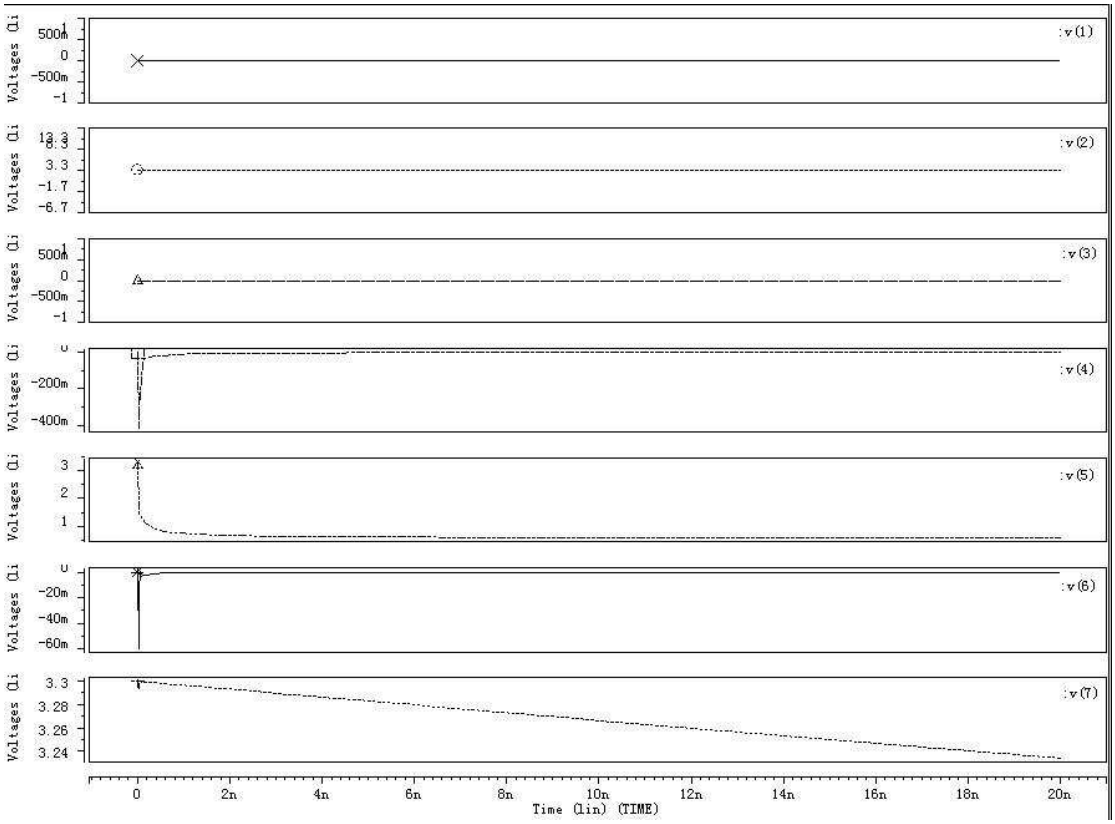

图 4.4 在睡眠电压为 0.6V 下的性能(match)

由上图可见，当存储的信息与外部比对的信息相同时，match 信号在 20ns 的时间内仍然保持了 2.24V 以上的高电压，不会造成逻辑错误。

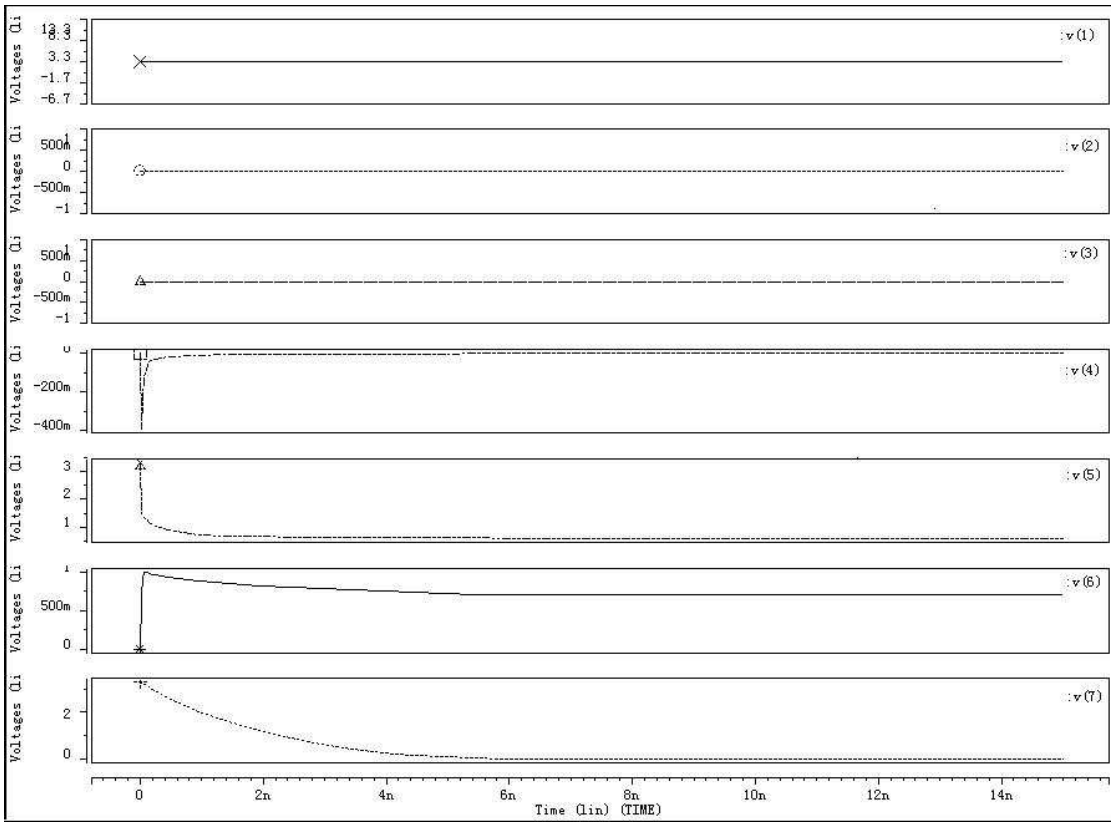

图 4.5 在睡眠电压为 0.6V 下的性能(miss)

由上图可见，当存储的信息与外部比对信息不相同时，Match 线在 2ns 的时间内降到了 1.1V，在 6ns 的时间内降到了接近于 0。

对导向器对管进行了优化前后的静态功耗测定，在优化前使用 3.3V 的供电压时，漏电功耗为：2.19E-11W

在使用优化，将供电压降低到 1.1V 后，漏电功耗为：7.25E-13W

可见优化后的漏电功耗仅为优化前的 3.3%。

4.3.2 关于 SRAM 单元的优化设计

由 3.3.1 的描述可知，在本论文所描述的 SRAM Block 的设计过程中，我们所采用的是双港口的 SRAM 设计，其中的 Wb 字线与 CAM Block 的字线相连接，仅在改变表项的内容，往 SRAM 中写入时才导通其控制的 NMOS 管，而字线 Wa 则受到 CAM Block 所输出的 match 信号的控制，仅当 CAM 中的某个 TAG 与所比对的 TAG 匹配时才导通其控制的 NMOS 管。在大部分的时间内，SRAM Block 的字线 Wa 和 Wb 都处于低电平状态，其控制的 NMOS 管都处于关断状态，内部所存储的信息与外部不发生联系。因此，可以考虑在平时将 SRAM Cell 的

内部供电压置为睡眠状态，而仅当需要将存储的信息读出，或者是需要更新表项内容，将信息写入 SRAM Cell 时才将内部供电压上拉至正常模式并且保持在此高电平模式。每隔若干个时钟周期(一般为 2000)，将所有的 SRAM 的供电压均置为睡眠状态。这样，根据程序的局部性原理，在大部分时间内只有少数表项工作在正常状态，而大多数表项则在大部分时间内工作在睡眠状态。

在本设计中，考虑使用字线 Wa 和字线 Wb 对 SRAM 的供电压进行选择，当比对 hit，需要读出信息；或者是字线 Wb 为高电平，需要写入信息时才将 SRAM 的供电压置为正常模式，否则就将其置为睡眠状态。

为实现上述功能，所作电路图如图 4.6 及图 4.7 所示：

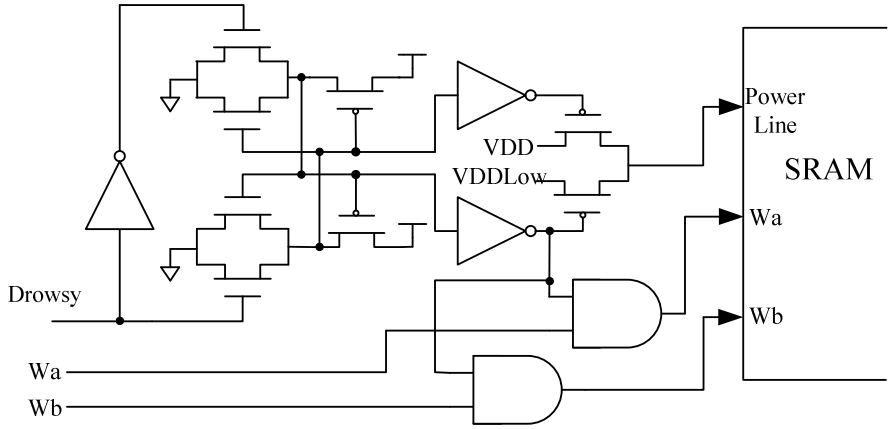

图4.6 双港口SRAM所用DVS电路

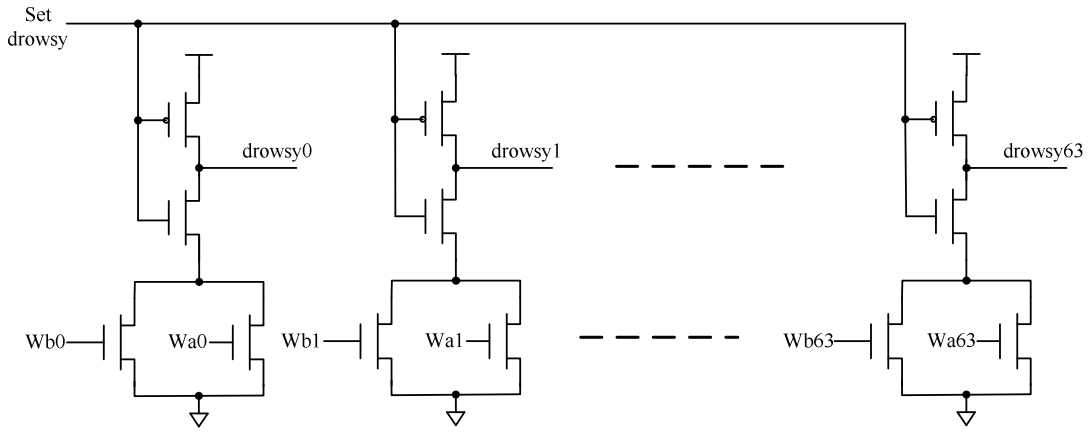

图4.7 控制电路

由于进行优化的 SRAM 模块为双港口 SRAM，因此对使用的 DVS 电路做了如上图所示的修改，使其可以满足两根字线的操作。同时，由于唤醒电路是 Wa 与 Wb 的或逻辑，因此将唤醒电路的控制电路设计成如图 4.7 所示，当输入 Wa 或 Wb 为高电平时，将把输出 drowsy 下拉至 0。为了与控制电路的输出相协调，

DVS 电路也改成如图 4.6 所示，当输入的 drowsy 信号为 1 时，DVS 电路将把睡眠电压输给 SRAM 的电源线；而当 drowsy 信号为 0 时，则把正常电压 V_{DD} 输给 SRAM 的电源线。

在图 4.7 所示的控制电路中，当 set-drowsy 信号为低电平时，其控制的 N 管关断，从而使 Wa，Wb 信号无效，防止短路；其控制的 P 管导通，将每个表项的 drowsy 信号均置为高电平，从而使每个表项均进入低功耗的睡眠状态。当 set-drowsy 信号为高电平时，其控制的 P 管关断，N 管导通，每个表项的 drowsy 均由该表项的 Wa，Wb 信号控制，根据 Wa 和 Wb 的值决定是继续保持睡眠状态还是将其转到正常工作模式。

在睡眠电压的设置方面，一般把睡眠电压设置成 $1.5V_T$ 即可。对于采用 DVS 技术后 SRAM 的功耗评估比较复杂，因为 SRAM 单元中处于睡眠状态和正常状态的表项的时间无法预测，但是由于程序的局部性原理，在大部分时间内，大部分表项将处于睡眠状态，因此可以预测 DVS 技术在 SRAM 中的运用也将取得良好的优化效果，但是对漏电功耗的减少会略少于 CAM 单元。

第五章 总结及展望

TLB 在地址转换的过程中发挥着极其重要的作用，通过本次设计，对 TLB 的原理及结构有了深入的了解。在设计中，给出了实现 TLB 的电路结构，并且用 Hspice 对导出的网表进行了仿真分析，验证了其结果的正确性。在下一步的设计当中，可以考虑进一步考查 TLB 与 MMU 的关系，减少 TLB 对操作系统的依赖，而通过其内部的硬件结构实现逻辑功能，从而进一步加快地址转换速度。

在完成了 TLB 的电路结构设计后，本论文又对低功耗优化方面做了一定的研究分析。所采用的 DVS 技术是当今减少漏电功耗方面的最新技术，而 DVS 技术在 TLB 中的运用也减少了 TLB 的漏电功耗---在 CAM 中，将漏电功耗降为了原来的 3.3%，在 SRAM 中，也可预测将取得良好的优化效果。但是，因为未优化前的漏电功耗就已经相当小了，因此，本次设计中对漏电功耗的优化虽然将漏电功耗降低到一个很低的比例，但是实际所节约的功耗数值很小。不过，本次设计对功耗优化采用从优化漏电功耗着手，目的就在于对这一新兴的技术有一定的了解，对未来基于减少漏电功耗的低功耗研究做一些先期的工作。随着处理器工艺的进一步发展，工艺线宽呈现了不断减少的趋势，下图所示为通过导向器的漏电功耗随着线宽的减少和温度的升高大幅的增加。本论文关于漏电功耗的研究虽然对于本次设计的 TLB 并无重大意义，但是在今后低线宽的设计中有其应用价值。而本次毕设也达到了这一目的。

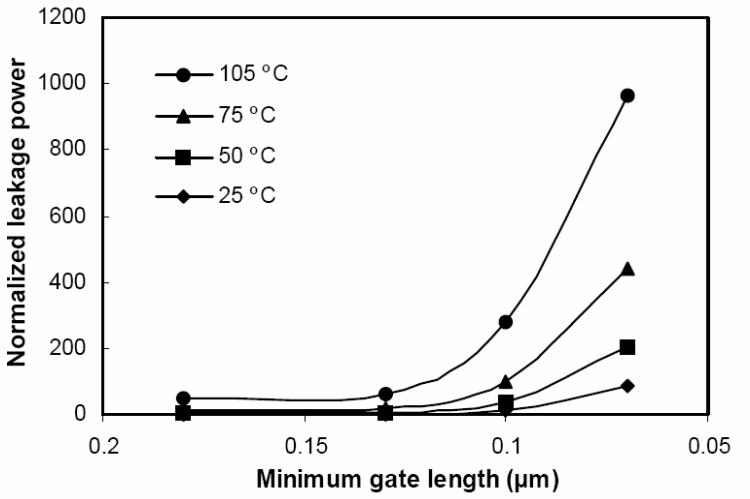

图 5.1 通过导向器的漏电功耗

参考文献：

[1] Krisztian Flautner 等. Drowsy Caches: Simple Techniques for Reducing Leakage Power

[2] 孟静. 操作系统教程－原理和实例分析. 北京：高等教育出版社. 2001

[3] 屠立德 屠祁. 操作系统基础. 北京：清华大学出版社. 1995

[4] William Stallings. 计算机组织与结构——性能设计[M]. 北京：电子工业出版社.2001.

[5] 李伟华. VLSI 设计基础. 北京：电子工业出版社. 2002

[6] Jan M.Rabaey. 数字集成电路设计透视. 北京：清华大学出版社. 1998

[7] Steve Furber. ARM SoC 体系结构.北京：北京航空航天大学出版社. 2002

[8] 高文焕 汪蕙. 模拟电路的计算机分析与设计. 北京：清华大学出版社. 2002

[9] J.Bhasker. Verilog HDL 硬件描述语言. 北京：机械工业出版社. 2000

[10] R.Jocob Baker CMOS 电路设计、布局与仿真. 北京：机械工业出版社. 2003

[11] 黄正瑾. 计算机结构与逻辑设计. 北京：高等教育出版社. 2001

[12] 谢嘉奎. 电子线路—线性部分. 北京：高等教育出版社. 1999

[13] Nam Sung Kim 等. Drowsy Instruction Caches: Leakage Power Reduction using Dynamic Voltage Scaling and Cache Sub-bank Prediction

致谢：

本人毕业设计从选题，到具体实施及论文写作，自始至终是在ASIC中心的老师悉心指导和大力帮助下完成的。在此向ASIC中心所有指导和帮助我的老师致以由衷的感谢！

除此之外，在课题的实施过程中，我收到了来自多方面的帮助和指导。杨军老师和薛骏师兄作为整个项目负责人，帮助我理解了TLB在整个项目中的作用，并且在我遇到问题时给予了及时的支持。在工作中，毕宗军师兄，高沁伟师姐关于宏单元建立的工作加快了毕设工作的进展；邬贵明，曹冕同学的工作使我对Cache中其他部分的工作有了清晰的认识；王怡同学对写缓冲的工作使我对数据的流向有了更深的了解；而凌青师姐在我对TLB电路的分析中给了我很大的帮助，在此向他们表示感谢！

此外还要感谢吴金老师，李冰老师对我学业上的帮助；邱振清老师，赵霞老师对我生活上的帮助，在此对他们致以最衷心的谢意！

谨以此文献给我的父母以及所有关心和帮助过我的人们！

附录

附录一、用于图 3.7 所示 TLB 电路仿真的源文件，由 Cadence 导出

```
match5
********************************************************************************
* auCdl Netlist:
*
* Library Name: cache
* Top Cell Name: sun_match3
* View Name: schematic
* Netlisted on: May 30 12:12:09 2004
********************************************************************************
*.EQUATION
*.SCALE METER
*.MEGA
.PARAM
.GLOBAL VDD
+ GND
*.PIN VDD
*+ VDD
*+ GND
*+ GND
********************************************************************************
* Library Name: cache
* Cell Name: sun_g8_inv2
* View Name: schematic
********************************************************************************
.SUBCKT sun_g8_inv2 a y
MMN65 y a GND GND NMOS_TK W=1.4u L=250.0n M=1.0
MMP61 y a VDD VDD PMOS_TK W=1.6u L=250.0n M=1.0
MMP62 y a VDD VDD PMOS_TK W=1.6u L=250.0n M=1.0
.ENDS
********************************************************************************
* Library Name: cache
* Cell Name: sun_g8_inv1
* View Name: schematic
********************************************************************************
.SUBCKT sun_g8_inv1 a y
MMP58 y a VDD VDD PMOS_TK W=820.0n L=250.0n M=1.0
MMN62 y a GND GND NMOS_TK W=650.0n L=250.0n M=1.0
.ENDS
```

```
********************************************************************************
* Library Name: cache
* Cell Name: mmu_gqw19_inv3
* View Name: schematic
********************************************************************************
.SUBCKT mmu_gqw19_inv3 a y
MMN0 y a GND GND NMOS_TK W=3u L=250.0n M=1.0
MMP0 y a VDD VDD PMOS_TK W=5.4u L=250.0n M=1.0
.ENDS
********************************************************************************
* Library Name: cache
* Cell Name: mmu_gqw19_inv4
* View Name: schematic
********************************************************************************
.SUBCKT mmu_gqw19_inv4 a y
MMN0 y a GND GND NMOS_TK W=6u L=250.0n M=1.0
MMP0 y a VDD VDD PMOS_TK W=10.6u L=250.0n M=1.0
.ENDS
********************************************************************************
* Library Name: cache
* Cell Name: mmu_gqw19_inv5
* View Name: schematic
********************************************************************************
.SUBCKT mmu_gqw19_inv5 a y
MMN0 y a GND GND NMOS_TK W=610.0n L=1.2u M=1.0
MMP0 y a VDD VDD PMOS_TK W=610.0n L=2u M=1.0
.ENDS
********************************************************************************
* Library Name: cache
* Cell Name: mmu_gqw19_nand2
* View Name: schematic
********************************************************************************
.SUBCKT mmu_gqw19_nand2 A B Y
MMP4 Y B VDD VDD PMOS_TK W=2u L=250.0n M=1.0
MMP3 Y A VDD VDD PMOS_TK W=2u L=250.0n M=1.0
MMN4 net0105 B GND GND NMOS_TK W=1.6u L=250.0n M=1.0
MMN3 Y A net0105 GND NMOS_TK W=1.6u L=250.0n M=1.0
.ENDS
********************************************************************************
* Library Name: cache
* Cell Name: mmu_gqw20_inv2
* View Name: schematic
********************************************************************************
.SUBCKT mmu_gqw20_inv2 a y
```

```
MMN0 y a GND GND NMOS_TK W=600n L=2u M=1.0
MMP0 y a VDD VDD PMOS_TK W=600n L=1.2u M=1.0
.ENDS
********************************************************************************
* Library Name: cache
* Cell Name: mmu_gqw20_inv1
* View Name: schematic
********************************************************************************

.SUBCKT mmu_gqw20_inv1 a y
MMN0 y a GND GND NMOS_TK W=2.2u L=250.0n M=1.0
MMP0 y a VDD VDD PMOS_TK W=4.2u L=250.0n M=2.0
.ENDS
********************************************************************************
* Library Name: cache
* Cell Name: sun_match3
* View Name: schematic
********************************************************************************
.SUBCKT sun_match3 Wa a a1 a2 b hit m1 m2
XI11 a net9 / sun_g8_inv2
XI8 net12 Wa / sun_g8_inv2
XI10 e net12 / sun_g8_inv1
XI2 net22 c / mmu_gqw19_inv3
XI3 net108 d / mmu_gqw19_inv4
XI4 d net108 / mmu_gqw19_inv5
XI5 a2 a1 net22 / mmu_gqw19_nand2
MMP3 e net12 VDD VDD PMOS_TK W=700n L=1.1u M=1.0
MMP46 m2 a VDD VDD PMOS_TK W=1.1u L=250.0n M=1.0
MMP51 net42 m1 VDD VDD PMOS_TK W=630.0n L=250.0n M=1.0
MMP49 net58 m2 VDD VDD PMOS_TK W=630.0n L=250.0n M=1.0
MMP53 m1 net42 VDD VDD PMOS_TK W=460.0n L=1.6u M=1.0
MMP52 m1 a VDD VDD PMOS_TK W=1.1u L=250.0n M=1.0
MMP1 net108 a2 net52 VDD PMOS_TK W=4.4u L=250.0n M=1.0
MMP2 net52 a1 VDD VDD PMOS_TK W=4.4u L=250.0n M=1.0
MMP47 m2 net58 VDD VDD PMOS_TK W=460.0n L=1.6u M=1.0
MMP62 net112 y VDD VDD PMOS_TK W=630.0n L=250.0n M=1.0
MMP61 e y net65 net65 PMOS_TK W=3.3u L=250.0n M=1.0
MMP60 net65 d VDD VDD PMOS_TK W=3.3u L=250.0n M=1.0
MMP57 y b net76 VDD PMOS_TK W=4.2u L=250.0n M=1.0
MMP56 net76 m1 VDD VDD PMOS_TK W=4.2u L=250.0n M=1.0
MMP55 y net112 VDD VDD PMOS_TK W=570.0n L=1u M=1.0
MMP54 y m2 VDD VDD PMOS_TK W=2u L=250.0n M=1.0
MMP0 net116 a VDD VDD PMOS_TK W=3.7u L=250.0n M=1.0
MMN63 e net12 GND GND NMOS_TK W=650.0n L=1.8u M=1.0
```

```
MMN64 e net9 GND GND NMOS_TK W=1.4u L=250.0n M=1.0
MMP48 net58 m2 GND GND NMOS_TK W=630.0n L=250.0n M=1.0
MMP50 net42 m1 GND GND NMOS_TK W=630.0n L=250.0n M=1.0
MMN4 net108 a2 GND GND NMOS_TK W=1u L=250.0n M=2.0
MMN1 net112 y GND GND NMOS_TK W=630.0n L=250.0n M=1.0
MMN0 net116 e GND GND NMOS_TK W=2.8u L=250.0n M=1.0
MMP59 y c GND GND NMOS_TK W=790.0n L=250.0n M=1.0
MMP58 y net112 GND GND NMOS_TK W=630.0n L=2u M=1.0
XI1 hit net116 / mmu_gqw20_inv2
XI6 net116 hit / mmu_gqw20_inv1
.ENDS

x1 Wa a a1 a2 b hit m1 m2 sun_match3
VA a 0 pwl(0 3.3 2ns 3.3 2.01ns 0 3ns 0 3.01ns 3.3 12ns 3.3 12.01ns 0 13ns 0 13.01ns 3.3 22ns 3.3 22.01ns 0 23ns 0 23.01ns 3.3
                +32ns 3.3 32.01ns 0 33ns 0 33.01ns 3.3 42ns 3.3 42.01ns 0 43ns 0 43.01ns 3.3)
*Vm1 m1 0 pwl(0 3.3 23ns 3.3 23.01ns 0 43ns 0)
*Vm2 m2 0 pwl(0 3.3 13ns 3.3 13.01ns 0 33ns 0)
Va1 a1 0 pwl(0 3.3 3ns 3.3 3.01ns 0 12ns 0 12.01ns 3.3 13ns 3.3 13.01ns 0 22ns 0 22.01ns 3.3 23ns 3.3 23.01ns 0
                +32ns 0 32.01ns 3.3 33ns 3.3 33.01ns 0 42ns 0 42.01ns 3.3 43ns 3.3 43.01ns 0)
Va2 a2 0 pwl(0 3.3 4ns 3.3 4.01ns 0 12ns 0 12.01ns 3.3 14ns 3.3 14.01ns 0 22ns 0 22.01ns 3.3 24ns 3.3 24.01ns 0
                +32ns 0 32.01ns 3.3 34ns 3.3 34.01ns 0 42ns 0 42.01ns 3.3 44ns 3.3 44.01ns 0)
MN1 M1 A3 GND GND NMOS_TK W=3.7u L=250.0n M=1.0
MN2 M2 A4 GND GND NMOS_TK W=3.7u L=250.0n M=1.0
VA3 A3 0 PWL(0 3.3 1NS 3.3 1.01NS 0 23.02NS 0 23.03NS 3.3 24NS 3.3 24.01NS 0 43.02NS 0 43.03NS 3.3 44NS 3.3 44.01NS 0)
VA4 A4 0 PWL(0 3.3 1NS 3.3 1.01NS 0 13.02NS 0 13.03NS 3.3 14NS 3.3 14.01NS 0 33.02NS 0 33.03NS 3.3 34NS 3.3 34.01NS 0)

VI1 vdd dc 3.3
VI2 gnd dc 0
Vb b 0 pwl(0 0 30ns 0 30.01ns 3.3)
.IC V(Wa)=0
.lib    sm079003-1h.hspice typical
.tran 0.1ns 50ns uic
.END
```

附录二：用于 CAM 模块仿真及功耗分析的源文件

CAM 模块仿真：

```
CAM
.SUBCKT INVERTER 1 2 3 4
*1:INPUT 2:OUTPUT 3:VDD 4:GND
```

```
M1 2 1 4 4 NMOS_TK W=1.6U L=0.5U
M2 2 1 3 3 PMOS_TK W=1.6U L=0.5U
.ENDS
M3 7 6 0 0 NMOS_TK W=0.9U L=0.25U
M4 1 5 6 0 NMOS_TK W=0.9U L=0.25U
M5 2 4 6 0 NMOS_TK W=0.9U L=0.25U
M6 1 3 4 0 NMOS_TK W=0.9U L=0.25U
M7 2 3 5 0 NMOS_TK W=0.9U L=0.25U
.lib    sm079003-1h.hspice typical
XINV1 4 5 8 0 INVERTER
XINV2 5 4 8 0 INVERTER
C 7 9 0.1PF
*VIN 8 0 DC 0.6
VIN 8 0 PULSE (0.6 3.3 8nS 0 0 3nS 30nS)
VIN1 1 0 DC 0
VIN2 2 0 DC 3.3
*VIN3 3 0 DC 0
VIN3 3 0 PULSE (0 3.3 8nS 0 0 2nS 30nS)
VIN5 9 0 DC 0
.IC V(4)=0
.IC V(5)=3.3
.IC V(7)=3.3
.TRAN 0.1nS    20nS UIC
.MEAS PWR AVG P(VIN)
.END
```

导向器对的功耗分析：

```
INVERTERS
.SUBCKT INVERTER 1 2 3 4
*1:INPUT 2:OUTPUT 3:VDD 4:GND
M1 2 1 4 4 NMOS_TK W=0.8U L=0.25U
M2 2 1 3 3 PMOS_TK W=0.8U L=0.25U
.ENDS
.lib    sm079003-1h.hspice typical
XINV1 4 5 8 0 INVERTER
XINV2 5 4 8 0 INVERTER
VIN 8 0 DC 3.3
VIN2 4 0 DC 3.3
VIN3 5 0 DC 0
.TRAN 0.0001US    1US UIC
.MEAS PWR AVG P(VIN)
.END
```