单位代码 10006
学　　号 38060518
分 类 号 TP332

北京航空航天大學
BEIHANG UNIVERSITY

# 毕业设计(论文)

## MIPS 微处理器 MMU 的设计与实现

院（系）名称　计算机学院
专 业 名 称　计算机科学与技术
学 生 姓 名　杨沛人
指 导 教 师　高小鹏

2012 年 6 月

# MIPS 微处理器 MMU 的设计与实现

杨沛人

北京航空航天大学

# 北京航空航天大学

# 本科毕业设计（论文）任务书

Ⅰ、毕业设计（论文）题目：

MIPS 微处理器 MMU 的设计与实现

Ⅱ、毕业设计（论文）使用的原始资料（数据）及设计技术要求：

1、原始资料：MIPS 32 体系结构规范，包括寄存器命名、指令规范等

2、原始数据：COCOA-Cyclone3-用户编程手册以及 COCOA 之前各个历史版本的技术文档

3、设计技术要求：加入内存管理单元 MMU(Memory-Management-Unit)和地址转换 cache TLB(Translation Look aside Buffer) ，实现虚地址到实地址的转换

Ⅲ、毕业设计（论文）工作内容：

本论文的主要工作内容是设计实现内存管理单元 MMU，并将其加入原有的 MIPS CPU 的设计中。要实现的功能是：1.让新 CPU 能够实现虚拟地址到物理地址的转换；2.提供硬件机制的内存访问授权，并能够对越界访问进行报错。3.支持新的 TLB 指令，并在配套使用的操作系统中添加相应的异常处理程序，能够让 MMU 模块对用户完全透明。

Ⅳ、主要参考资料：

[1] Dominic Sweetman 著. 李鹏 等译.MIPS 体系结构透视[M].机械工业出版社,2008.

[2] David A. Patterson, John L. Hennessy 著 郑纬民 等译.计算机组成与设计 [M].机械工业出版社, 2008.

[3] MIPS Technologies, Inc. MIPS32 Architecture For Programmers [Z]. Revision 1.0, August 2002.

[4] 邓子君 32 位 MIPS 微处理器内存管理单元的设计和验证[D]，西安：西安电子科技大学,2008.

计算机 学院（系） 计算机科学与技术 专业类 380605 班

学生 杨沛人

毕业设计（论文）时间： 2012 年 3 月 1 日至 2012 年 5 月 26 日

答辩时间： 2010 年 6 月 5 日

成　　绩：

指导教师： 高小鹏

兼职教师或答疑教师（并指出所负责部分）：

系（教研室）　主任（签字）：

# 本人声明

我声明，本论文及其研究工作是由本人在导师指导下独立完成的，在完成论文时所利用的一切资料均已在参考文献中列出。

作者：杨沛人

签字：

时间：2012 年 6 月

# MIPS 微处理器 MMU 的设计与实现

学　　生：杨沛人
指导教师：高小鹏

## 摘　　要

MIPS 是采用 RISC 体系结构的一种处理器，在众多的 RISC 体系结构中，MIPS 微处理器总能跻身于每一代最有效的微处理器之列，并保证最简洁的设计。MIPS 是出现最早的商业 RISC 架构芯片之一，新的架构集成了所有原来 MIPS 指令集，并增加了许多更强大的功能。本文首先概述了 MIPS 处理器的具体特点以及国内外现状，介绍了本次使用的开发平台。

本文从理论角度出发，分析了页表项、TLB 和 MMU 的各种理论实现方法，比较了这几种方法的优势和劣势，结合本次设计实际所用的硬件环境以及实现复杂度，确定了设计所采用的实现方式，并对一些功能予以解读，对某些设计提出了简单的要求。

针对选定的理论解决方案，我们对 MMU 的设计方案进行编码实现，对整个设计过程和设计细节做了详细的说明和讨论。论文中设计了页表项和 TLB 表项，详细讨论了 MMU 的实现逻辑，修改了 CU 中对应的控制逻辑，加入了相应的异常入口点。修改了操作系统，增加了其对 TLB 缺失异常处理的处理。

最后本文对所有的设计进行测试验证。这部分综合运用了 Quartus II 7.2、ModelSim 和 Logic Analyzer 三个强大的工具。最终通过这一系列的仿真以及测试工作证明了整个系统的工作符合最初的设计要求，能够达到预期的目标。

**关键词：**COCOA，虚拟内存管理，内存管理单元，地址转换后备缓冲器

# The Design and Implementation of Memory Management Unit on MIPS CPU

Author: YANG Peiren

Tutor: GAO Xiaopeng

## Abstract

MIPS (Microprocessor without Interlocked Pipeline Stages) is a reduced instruction set computer (RISC) instruction set architecture (ISA) developed by MIPS Technologies. Among the RISC CPUs, MIPS CPU has a very brief and exquisite design, and it is the most efficient architecture, too. MIPS CPU is one of the earliest commercialized CPU.

At the beginning of this dissertation, we firstly overviewed the features of MIPS CPU and its current state on domestic and international. Then we gave a brief introduction of our COCOA platform.

In the second part, based on the theory, we took the different Implementation Methods of TLB (Translation Look-aside Buffers) and MMU (Memory Management Unit) into analysis; we try to get the advantages and disadvantages of different Implementation Methods. Integrate theory with our hardware environment and the complexity of implement, we proposed an original solution.

In the third part, we realized the solution which was proposed in the second part, we gave an elaborate introduction and discussion on the entire solution. We designed the Page Table Entry and TLB pages. We modified CU in corresponding control logic, to join the corresponding exception access points. We updated the Operating System, added codes of TLB page fault.

The fourth part we tested and verified the previous solution. Firstly we tested the independent part of the design, and then we built the blocks together to test the entire solution. In this part, we used three different powerful test tools, i.e. Quartus II 7.2, ModelSim and

Logic Analyzer. Finally, the result of the serial test and simulate shows that our design meets the original requirement; our solution can successfully reaches our goal.

**Key words**：COCOA, Virtual Memory, MMU, TLB

# 术语

表格 I 所列出的是部分本文中所用的技术的缩写，这里将对这些术语做出进一步解释。

**表格 I 本文所涉及术语**

| 术语 | 描述 |
| --- | --- |
| MIPS | MIPS（Microprocessor without Interlocked Pipeline Stages）架构，是一种采取精简指令集（RISC）的处理器架构，1981 年出现，由 MIPS 科技公司开发并授权，广泛被使用在许多电子产品、网络设备、个人娱乐装置与商业装置上。最早的 MIPS 架构是 32 位元，最新的版本已经变成 64 位元 |
| COCOA | COCOA 是一个 FPGA 硬件平台，支撑 5 门核心课程的实验体系：Digital LogiC, Computer Organization, Compiler，Operating System，Computer Architecture |
| MIPS-C | 面向 COCOA 设计的 MIPS 处理器 |
| MMU | 内存管理单元（memory management unit，缩写为 MMU）。它是一种负责处理中央处理器（CPU）的内存访问请求的计算机硬件。它的功能包括虚拟地址到物理地址的转换（即虚拟内存管理）、内存保护、中央处理器高速缓存的控制等 |
| TLB | 转译后备缓冲区（Translation Look-aside Buffer），为 CPU 的一种缓存，由内存管理单元用于改进虚拟地址到物理地址的翻译速度。TLB 具有固定数目的空间槽，用于存放将虚拟地址映射至物理地址的分页表条目 |
| CP0 | 协处理器 0（Coprocessor 0）是 MIPS 所谓的系统控制协处理器，主要做的工作有：CPU 的配置、缓存控制、异常/中断控制、存储管理单元控制、杂项 |
| CU | 控制单元（Control Unit），是 CPU 的五大部件之一，用于执行计算机指令，诸如寄存器，算术逻辑单元，指令寄存器，总线，甚至芯片外部的输入输出均由其控制 |

# 目 录

# 图 目 录



# 表 目 录

# 1 绪论

## 1.1 MIPS 微处理器概述

MIPS 是一种采用 RISC (Reduced Instruction-Set Computer)体系结构的处理器。在众多的 RISC 体系结构处理器中，MIPS 是最优雅的一种，其自身优雅的设计虽然不能在充满竞争的市场中保证长盛不衰，但其微处理器却总能跻身于每一代最有效的微处理器之列，并保证最简洁的设计[1,5]。MIPS 的意思是“无内部互锁流水级的微处理器”(Microprocessor without interlocked piped stages)，其机制是尽量利用软件办法避免流水线中的数据相关问题。最早在 80 年代初期，它由斯坦福(Stanford)大学 Hennessy 教授领导的研究小组研制出来的。

MIPS 是出现最早的商业 RISC 架构芯片之一，新的架构集成了所有原来 MIPS 指令集，并增加了许多更强大的功能。MIPS 公司的 R 系列处理器就是在斯坦福版本的架构基础上开发的 RISC 工业产品的微处理器。R 系列产品为很多计算机公司采用构成各种工作站和计算机系统。

## 1.2 MIPS 微处理器国内外现状

MIPS 是卖的最好的 RISC CPU，可以从任何地方看见 MIPS 产品在销售，如 Sony，Nintendo 的游戏机，Cisco 的路由器和 SGI 超级计算机。MIPS 有可能是起初 RISC CPU 设计中唯一的一个在本世纪盈利的，但目前 RISC 体系结构遭到 x86 芯片的竞争。和英特尔相比，MIPS 的授权费用比较低，因此其为除英特尔外的大多数芯片厂商所采用。

经过几年的努力，我国完成了从无芯片到有芯片的跨越，国内出现了龙芯、方舟、等多个自主研发的处理器，并应用到了各个领域。其中，龙芯处理器已经开始应用于服务器领域，这是我们值的骄傲的成果。各个研发单位都在研发的过程中积累了丰富的经验，并推动我国的处理器设计向更高的水平发展。但是处理器的设计只有在研发和市场之间形成良性循环才能得到更进一步的发展，面对国外公司成熟产品的竞争压力，我国的处理器在产业界站稳脚跟还有一段很长的路要走，所以我们的工作仍然任重道远。

## 1.3 FPGA 概述

FPGA（Field-Programmable Gate Array），即现场可编程门阵列，它是在 PAL、GAL、

CPLD 等可编程器件的基础上进一步发展的产物。作为专用集成电路（ASIC）领域中的一种半定制电路，它既解决了定制电路的不足，又克服了原有可编程器件门电路数有限的缺点[2]。

FPGA 产品的应用领域已经从原来的通信扩展到消费电子、汽车电子、工业控制、测试测量等广泛的领域。而应用的变化也使 FPGA 产品近几年的演进趋势越来越明显：一方面，FPGA 供应商致力于采用当前最先进的工艺来提升产品的性能，降低产品的成本；另一方面，越来越多的通用 IP（知识产权）或客户定制 IP 被引入 FPGA 中，以满足客户产品快速上市的要求。此外，FPGA 企业都在大力降低产品的功耗，满足业界越来越苛刻的低功耗需求。

FPGA 主要生产厂商有：Altera、Xilinx、Actel、Lattice 和 atmel。其中 Altera 作为世界老牌可编程逻辑器件的厂家，是可编程逻辑器件的发明者，开发软件 MAX+PLUSII 和 Quartus II。Xilinx 是 FPGA 的发明者，拥有世界一半以上的市场，提供 90%的高端 65nmFPGA 产品，开发软件为 ISE。Actel 主要提供非易失性 FPGA，产品主要基于反熔丝工艺和 FLASH 工艺，其产品主要用于军用和宇航。

本次设计所使用的 FPGA 是 Altera 公司生产的 EP3C120F780C7 系列芯片，工作电压 1.2V。配套的开发软件为 Altera 公司的 Quartus II 系列软件，本次开发使用的版本是 11.0 版。

## 1.4 COCOA 平台概述

COCOA（Digital LogiC, Computer Organization, Compiler, Operating System, Computer Architecture）项目于 2007 年开始实施,其中 COCOA 是数字逻辑、计算机组成原理、编译、操作系统和计算机体系结构这 5 门计算机专业核心课程的缩写。COCOA 的物理实现是一个以 FPGA 为核心的实验平台。该实验平台基本结构如图 1.1 所示。实验平台除不支持显示芯片外，其基本组成就是一台微型计算机[3]。

实验平台的核心是一片大规模的 FPGA。FPGA 外围是存储器（FLASH、SRAM、SDRAM）、常规输入输出部件（键盘、LED、并口、串口）以及以太网接口芯片。从配置角度出发，只要 FPGA 的规模足够大，该实验平台可以提供足够的实验支撑能力（包括与网络相关实验）。

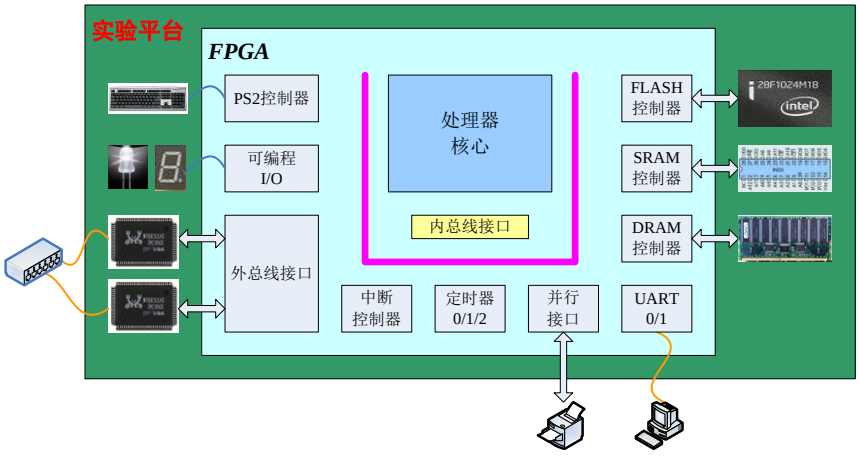


图 1.1 COCOA 实验平台

图 1.2 是实验平台的整体框架图。除外部存储器外，包括处理器核心在内的所有部件都集成在 FPGA 内。可以看出上述这些功能部件基本上涵盖了 COCOA 的机器结构。

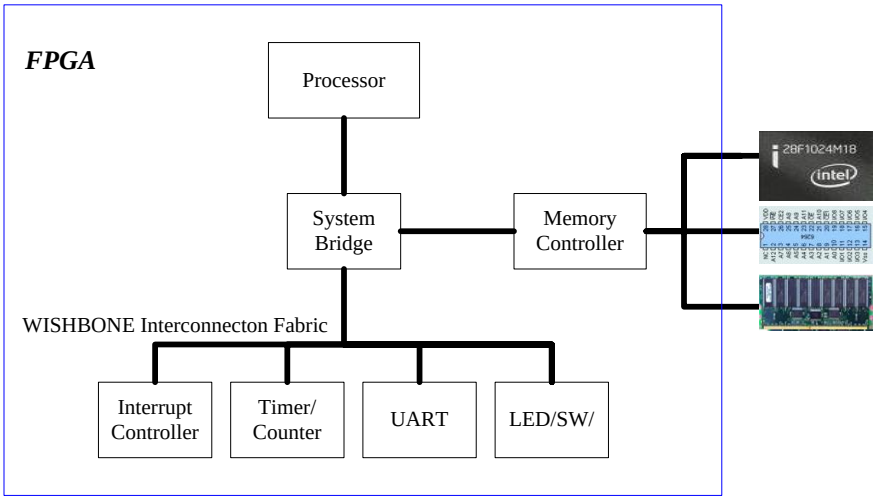


图 1.2 FPGA 系统框图

## 1.5 课题研究目的

此前版本的 MIPS 设计并不支持 MMU。由于没有 MMU 的管理，操作系统对内存空间没有保护机制，所有程序访问的地址都是实际物理地址。若一个用户程序访问了操作系统使用的内存空间，会造成不可预知的严重后果。加之 COCOA 配套的操作系统是支持多进程的，对于多进程操作系统，不同的进程的地址空间若不用虚拟映射，则同一段程序的两个进程在操作系统中运行，需要对运行空间重新编址，这将是非常繁琐的一项工作，无形中给程序员增加了不必要的工作量。使用虚拟地址只需要根据进程号来匹

配不同的进程便可，即使虚拟地址相同，进程也可以正常运行。

本次设计编写的 MIPS 处理器用于本科教学工作。硬件设计部分用于对计算机组成原理教学的支持。通过直观的硬件代码，能够让同学们对于 CPU 的底层工作有所掌握，进而透彻的理解处理器实际的工作逻辑，理解汇编代码怎样被变为逻辑电平，并正确执行。

此硬件系统也对操作系统（OS）的教学提供支持。在 MIPS 的硬件基础之上编写操作系统运行在 COCOA 平台之上，因此硬件对于存储管理部分提供支持是非常必要的。有了硬件中 MMU 的支持，操作系统中对于进程切换、地址空间的管理、页表映射的支持便能够实现，同学们对照着硬件支持和操作系统中相关的结构和代码，理解操作系统的存储管理会更加容易，教学工作的开展也会变得更加容易。

综上所述，此前版本的 MIPS 处理器有必要加入 MMU，来对操作系统的地址管理以及地址映射进行规范高效的管理。

### 1.6 MMU 设计概述

内存管理单元(MMU)是处理器支持操作系统高效运行的基础，它与软件内存管理模块相结合，完成了虚拟地址到物理地址的转换。同时 MMU 对处理器发出的地址进行合法性检验并在硬件上提供了内存访问授权控制。正是由于像 MIPS 这样的体系结构的迅速发展以及 SOC 研究的深入，带动了嵌入式处理器的设计向着更高层次发展。

早期许多嵌入式微处理器都由于没有 MMU 而不支持虚拟内存。没有内存管理单元所带来的好处是简化了芯片设计，降低了产品成本。由于大多数的嵌入式设备没有磁盘或者只有很有限的内存空间，所以无需复杂的内存管理机制。但是由于没有 MMU 的管理，操作系统对内存空间是没有保护的，所有程序访问的地址都是实际物理地址。

随着嵌入式处理器的设计水平的提高，处理器主频也不断升高，访问存储器的速度就越来越成为制约处理器性能提高的瓶颈。如何更加有效地管理更多的存储空间成了急待解决的问题。因此，现代嵌入式处理器广泛运用了虚拟存储技术，通过使用内存管理单元来弥补处理器时钟频率和存储器存取时间的差距，提高嵌入式系统的性能。另一方面，由于嵌入式系统任务复杂度的不断提高，现在流行的嵌入式处理器需要有操作系统的支持，以实现多个进程的协同工作和通信，这也要求嵌入式处理器提供相关的硬件支持来简化软件的工作复杂度，并提供保护机制来防止不安全的内存访问造成系统的崩

溃。由于 MMU 与处理器体系结构高度相关，因此在不同的处理器下内存管理机制区别很大。

根据现代 MIPS32 处理器对内存管理的规范，针对不同地址空间，采取了静态和动态映射相结合的方式。为了加快地址转换速度在 MMU 中加入了 TLB(Translation Look-aside Buffers)，TLB 页面为部分页表页面的缓存。MMU 与总线访问接口的连接采用了更加简洁更加高效的 Wishbone 协议。

### 1.7 本文所做工作以及内容安排

本文所作的工作主要围绕内存管理单元(MMU)的设计和验证展开，本文所作的工作和后面各章的内容安排如下：

第二章对 32 位 MIPS 处理器的体系结构进行介绍后，对内存管理单元的结构设计方案进行讨论，重点提出采用自底向上的层次型页表项设计、层次型的数据指令一体化地址转换后备缓冲器(TLB)来进行地址映射，以及采用地址空间标识符实现地址空间的保护机制的内存管理单元结构设计思想，从而加快 MIPS 地址映射过程。

第三章对上一章提出的设计方案的实现细节予以阐述，重点对 TLB 的设计、MMU 的设计以及 CP0 的修改做了全方位的论述，对 CPU 的控制单元做了必要的修改，让 MMU 的工作时序与 CPU 整体匹配，最后在操作系统中加入 MMU 错误处理代码。

第四章为设计的测试与验证工作。针对前几章的实现逻辑，分别对单个模块进行了测试，最后对 CPU 整体进行了测试，包含正常运行时的测试和非正常运行时报错功能的测试。

文章最后的部分是全文总结、参考文献、致谢以及附录。

# 2 总体设计概述

## 2.1 现版本 MIPS-C 处理器结构

要了解现版本MIPS-C处理器结构，则必须了解CPU的数据通路等一系列具体设计，下面列出这些设计，并作必要的说明。

### 2.1.1 MIPS-C 处理器现有数据通路

现有数据通路如图 2.1 所示，其中访存地址由 PC（Program Counter，程序计数器）或 ALU（Arithmetic Logical Unit，运算器）计算直接给出，不需要翻译。

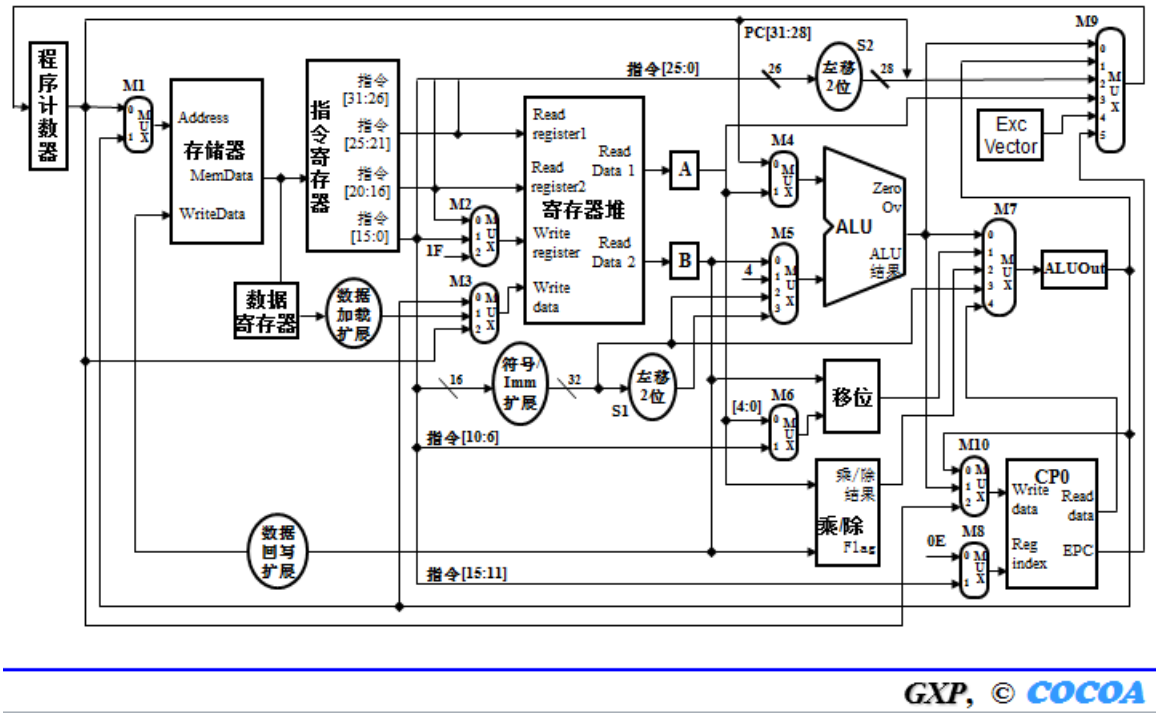


图 2.1 MIPS CPU 数据通路

### 2.1.2 现版本 CP0 中的寄存器

现版本的 CP0 中有四个寄存器，主要是为了辅助 CPU 完成异常和中断操作，具体寄存器名称以及描述见表 2.1。

要配合 MMU 工作，必须在 CP0 中新增加至少 9 个寄存器，辅之以这些寄存器的读写逻辑和它们与 MMU 交换数据的逻辑，才能支持操作系统进行进程切换等操作。

**表 2.1 现版本 CP0 的寄存器**

| Register mnemonic | CP0 register No. | Description | MIPS-C |
|---|---|---|---|
| SR | 12 | Status Register | 支持部分位域 |
| Cause | 13 | 中断或异常发生的原因 | 支持部分位域 |
| EPC | 14 | 异常程序计数器 | 支持 |
| PRId | 15 | 处理器 ID | 支持 |

## 2.2 页表项和 TLB 的设计方案

下面是对页表项和 TLB 的设计方案的讨论。

### 2.2.1 虚地址空间的管理方式讨论

虚拟地址空间可以分成具有固定大小的页或具有变长尺寸的段。针对虚地址空间的管理有页式管理，段式管理，段页式管理[4]。

页式管理的基本原理是将各个进程的虚拟空间划分成若干个长度相等的页(page)，把内存空间按页的大小划分成片或者页面（page frame），然后把页式虚拟地址与内存地址建立一一对应页表，并用相应的硬件地址变换机构，来解决离散地址变换问题。页式管理采用请求调页或预调页技术，实现了内外存存储器的统一管理。它分为静态页式管理和动态页式管理。页式管理的优点是没有外碎片，每个内碎片不超过页大小。一个程序不必连续存放，便于改变程序占用空间的大小（主要指随着程序运行而动态生成的数据增多，要求地址空间相应增长，通常由系统调用完成而不是操作系统自动完成），缺点是程序必须全部装入内存，而且要求有相应的硬件支持。例如地址变换机构，缺页中断的产生和选择淘汰页面等都要求有相应的硬件支持。这增加了机器成本，增加了系统开销，例如缺页中断处理机，请求调页的算法如选择不当，有可能产生抖动现象。 虽然消除了碎片，但每个作业或进程的最后一页内总有一部分空间得不到利用果页面较大，这一部分的损失仍然较大。

段式管理的基本原理是把程序按内容或过程（函数）关系分成段，每段有自己的名字。一个用户作业或进程所包含的段对应一个二维线形虚拟空间，也就是一个二维虚拟存储器。段式管理程序以段为单位分配内存，然后通过地址影射机构，将段式虚拟地址

转换为实际内存物理地址。程序通过分段(segmentation)划分为多个模块，如代码段、数据段、共享段。其优点是可以分别编写和编译，可以针对不同类型的段，采取不同的保护，可以按段为单位来进行共享，包括通过动态链接进行代码共享。

段页式管理的基本原理继承了段式管理和页式管理的诸多特点。

一个进程中所包含的具有独立逻辑功能的程序或数据仍被划分为段，并有各自的段号。段中的程序或数据则按照一定的大小将其划分为不同的页。和页式系统一样，最后不足一页的部分仍占一页。这反映了段页式管理中的页式特征。

段页式管理时的进程的虚拟地址空间中的虚拟地址由三部分组成：段号s，页号P和页内相对地址d。虚拟空间的最小单位是页而不是段，从而内存可用区域也就被划分成为若干个大小相等的页面，且每段所拥有的程序和数据在内存中可以分开存放。分段的大小也不再受内存可用区域的限制。

为了实现段页式管理，系统必须为每个作业或进程建立一张段表以管理内存分配与释放、缺段的处理、存储保护相地址变换等。另外，由于一个段又被划分成了若干页，每个段又必须建立一张页表来把段中的虚页变换成内存中的实际页面。显然，与页式管理时相同，页表中也要有相应的实现缺页中断处理和页面保护等功能的表项。另外，由于在段页式管理中，页表不再是属于进程而是属于某个段，因此段表中应有专项指出该段所对应页表的页表起始地址和页表长度。

在一般使用段页式存储管理方式的计算机系统中，内存辟出一块固定的区域存放进程的段表和页表。因此，在段页式管理系统中，要对内存中指令或数据进行一次存取的话，至少需要访问三次以上的内存：第一次是由段表地址寄存器得段表起始地址后访问段表，由此取出对应段的页表在内存中的地址；第二次则是访问页表得到所要访问的物理地址；第三次才能访问真正需要访问的物理单元。显然，这将使CPU的执行指令速度大大降低[6]。

为了提高地址转换速度，设置快速联想寄存器就显得比段式管理或页式管理时更加需要。在快速联想寄存器中，存放当前最常用的段号s、页号p和对应的内存页面与其它控制用栏目。当要访问内存空间某一单元时，可在通过段表、页表进行内存地址查找的同时，根据快速联想寄存器查找其段号和页号。如果所要访问的段或页在快速联想寄存器中，则系统不再访问内存中的段表、页表而直接把快速联想寄存器中的值与页内相对

地址d拼接起来得到内存地址。 因为段页式管理是段式管理的页式管理方案结合而成的，所以具有它们二者的优点。但反过来说，由于管理软件的增加，复杂性和开销也就随之增加了。另外，需要的硬件以及占用的内存也有所增加。更重要的是，如果不采用联想寄存器的方式提高CPU的访存速度，将会使得执行速度大大下降。

### 2.2.2 本设计中采用的虚地址管理方案

综合 2.2.1 中所述的三种管理方式的特点，以及设计的复杂度控制和硬件条件的限制，本次设计的虚拟存储系统采用页式管理，页式管理可以使物理存储器和磁盘的数据交换有更好的性能。具体页表结构见图 2.2。

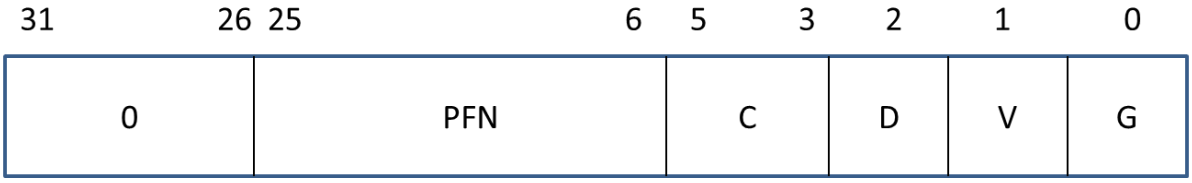


图 2.2 页表项

页表项其中的一些缩写释义如下：

PFN（Page Frame Number）：物理地址的高位部分，此域的有效宽度依赖于设计的 CPU 支持的物理内存空间。本设计中此域的宽度设置为 20 位，则一个实际的虚地址可以被认为是 20 位 PFN 与 12 位页内偏移地址的组合，之后翻译一个地址只需依据其高二十位的 VPN（Virtual Page Number），翻译得出相应的 PFN，再将低十二位的页内偏移拼接过来得到实际地址。

C：一个三位的域。操作系统通常知道某些页是不需要在多个缓存间自动跟踪它们的变化的，包括已知的仅被一个 CPU 使用的页或是已知只读的页，这个域的值，普遍代表“非缓存”或“非一致可缓存”。

D（脏）：这个位为“写允许”位。置为 1 表示允许写入，0 则任何使用本翻译进行的写入操作都会陷入。

V（有效）：如果置为 0，那么任何与该表项匹配的地址访问都会引发异常。

G（全局）：当一个页表项的 G 位被置为 1 时，表示这一页是全局都能访问到的页，而不是根据不同的进程来匹配访问的页，因此遇到这样的地址做匹配翻译时，硬件会忽

略从 CP0 送过来的进程号，仅用 VPN 去匹配做翻译。

### 2.2.3 TLB 的设计方案

拟定了页表结构的设计方案后，可以分析出这样的设计方案使程序员在设计应用程序时不需要关心目标代码的物理地址，但这是付出相当的性能代价的来得到的，每次存储器访问带来两次额外的存储器访问，即在访问所需数据之前进行 1 次页目录访问和页表访问[7]。

为了加速地址的转换过程，通常的做法是在 MMU 设计中采用地址转换后备缓冲器(Translation Look-aside Buffers)。TLB 是一种片上存储器结构，可以缓存最近完成转换过程的页表项。采用 TLB 设计，如果地址转换的信息存在于 TLB 上(TLB 命中或者 TLB 匹配)，系统可以通过 TLB 完成虚拟地址到物理地址的转换而无需访问页表。如果地址转换的信息不存在于 TLB 中，那么系统需要访问页表并把匹配的页表项填充到 TLB 中，这样就大大缩短存储器访问的时间。

由于 TLB 要匹配不同大小的页面，因此需要注意在 PageMask（页面掩码寄存器，控制 TLB 匹配页面大小）不为全 0 时，要能够根据其不同的值匹配不同大小的页面，页面大小从 4K 到 256M。TLB 也要能够更新页面，TLB 一个表项中包含两个实页面的内容，因此一次更新要更新一个虚页面和两个实页面。

TLB 最重要的最频繁的操作便是比对匹配，匹配主要有三处，分别是：

①根据 PFN 和进程号 ASID 来匹配翻译页；

②检查此次操作是读还是写；

③根据 D（Dirty）位，看看能否写，根据 V（Validate）位来确认此次匹配是否合法。

因此单个 TLB 的功能将不仅仅是存贮一个页面，还需要提供比对逻辑，更新逻辑。

## 2.3 MMU 的设计方案

下面分述 MMU 的两个功能：

第一是将虚拟地址映射为物理地址。现代的多用户多进程操作系统，需要 MMU，才能达到每个用户进程都拥有自己独立的地址空间的目标。使用 MMU，操作系统划分出一段地址区域，在这块地址区域中，每个进程看到的内容都不一定一样。例如

WINDOWS 操作系统将地址范围 4M-2G 划分为用户地址空间，进程 A 在地址 0x400000（4M）映射了可执行文件，进程 B 同样在地址 0x400000（4M）映射了可执行文件，如果 A 进程读取地址 0x400000, 读到的是 A 的可执行文件映射到 RAM 的内容，而 B 进程读取地址 0x400000 时，则读到的是 B 的可执行文件映射到 RAM 的内容。

第二是提供硬件机制的内存访问授权。多年以来，微处理器一直带有片上存储器管理单元(MMU)，MMU 能使单个软件线程工作于硬件保护地址空间。但是在许多商用实时操作系统中，即使系统中含有这些硬件也没采用 MMU。当应用程序的所有线程共享同一存储器空间时，任何一个线程将有意或无意地破坏其它线程的代码、数据或堆栈。异常线程甚至可能破坏内核代码或内部数据结构。例如线程中的指针错误就能轻易使整个系统崩溃，或至少导致系统工作异常。MMU 利用当前映射，将在指令调用或数据读写过程中使用的逻辑地址映射为存储器物理地址。MMU 还标记对非法逻辑地址进行的访问，这些非法逻辑地址并没有映射到任何物理地址。采用 MMU 还有利于选择性地将页面映射或解映射到逻辑地址空间。物理存储器页面映射至逻辑空间，以保持当前进程的代码，其余页面则用于数据映射。类似地，物理存储器页面通过映射可保持进程的线程堆栈。这样，如果任何线程分配的堆栈发生溢出，将产生硬件存储器保护故障，内核将挂起该线程，而不使其破坏位于该地址空间中的其它重要存储器区，如另一线程堆栈。这不仅在线程之间，还在同一地址空间之间增加了存储器保护。

考虑到硬件系统的资源，本次设计拟设置 8 个 TLB 页面，当这 8 个 TLB 页面装满时便有 64M 的空间被映射，而现在 COCOA 拥有的 Flash Memory 和 SRAM 加起来所用到的地址空间总共为 36M，因此 8 个 TLB 页面完全够用。MMU 不仅仅要能够完成对虚拟地址的翻译，还要能够完成对 TLBR、TLBWI、TLBWR、TLBP 四条指令的响应。要正确执行这四条指令，MMU 还必须与 CP0 完成配合，因此 MMU 需要加入很多与 CP0 通讯的接口。MMU 需要对翻译错误进行及时的响应，并发出异常，这一功能的实现需要在 CU（Control Unit，控制单元，见表格 I 本文所涉及术语）中修改一定的设计。

综上所述，MMU 中拟加入 8 个 TLB 页面，要有对于匹配错误的响应逻辑，能够发出异常。

## 2.4 本章小结

本章首先对此前版本的 MIPS-C 处理器从结构方面作了分析，阐述了在此前版本中

加入 MMU 的必要性。然后从理论角度出发，分析了页表项、TLB 和 MMU 的各种理论实现方法，比较了这几种方法的优势和劣势，结合本次设计实际所用的硬件环境以及实现复杂度，确定了设计的实现方式，并对一些功能予以解读，对某些设计细节提出了简单的要求。

# 3 详细设计以及实现

第二章提出了内存管理单元的设计方案，本章重点对上一章提出的方案进行详细的逻辑设计实现，并且对整个设计过程和设计细节做出详细的说明和讨论。

## 3.1 MIPS 32 位地址空间与总体设计

如图 3.1 所示，在 32 位 CPU 下，程序地址空间划分为 4 个大区域。每个区域有一个传统的名字。对于在这些区域的地址，各自有不同的属性：

kuseg: 虚拟空间 0x0000_0000 - 0x7FFF_FFFF (低端 2G)：这些地址是用户态可用的地址。在有 MMU 的机器里，这些地址将一概被 MMU 作转换，若 MMU 的设置没有被建立，这 2G 地址就不可用。对于没有 MMU 的机器，存取这 2G 地址的方法与具体机器相关。如果想要写的代码在有或没有 MMU 的 MIPS 处理器之间能偶兼容运行，则要尽量避免使用这块区域存取数据。

kseg0: 虚拟空间 0x8000_0000 - 0x9FFF_FFFF(512M)：这些地址映射到物理地址简单的通过把最高位清零，然后把它们映射到物理地址低段 512M(0x0000_0000 - 0x1FFF_FFFF)。因为这种映射是很简单的，通常称之为“非转换的”地址区域。几乎全部的对这段地址的存取都会通过快速缓存(cache)。因此在 cache 设置好之前，不能随便使用这段地址。通常一个没有 MMU 的系统会使用这段地址作为其绝大多数程序和数据的存放位置。对于有 MMU 的系统，操作系统核心会存放在这个区域。

kseg1: 虚拟空间 0xA000_0000 - 0xBFFF_FFFF(512M)：这些地址通过把最高 3 位清零的方法来映射到相应的物理地址上，与 kseg0 映射的物理地址一样。但 kseg1 是非 cache 存取的。kseg1 是唯一的在系统重启时能正常工作的地址空间。这也是为什么重新启动时的入口向量是 0xBFC0_0000。这个向量相应的物理地址是 0x1FC0_0000。我们将使用这段地址空间去存取初始化 ROM。大多数人在这段空间使用 I/O 寄存器。不能把这段地址空间映射到非低段 512M 空间。

kseg2: 虚拟空间 0xC000_0000 - 0xFFFF_FFFF (1G)：这段地址空间只能在核心态下使用并且要经过 MMU 的转换。在 MMU 设置好之前，不能存取这段区域。除非有一个真正的操作系统支持，一般来说不需要使用这段地址空间。

综上所述，MIPS32 CPU 不经过 MMU 转换的内存窗口只有 kseg0 和 kseg1 的 512M

的大小，而且这两个内存窗口映射到同一 512M 的物理地址空间。其余的 3G 虚拟地址空间需要经过 MMU 转换成物理地址，这个转换规则是由 CPU 厂商实现的。换句话说，在 MIPS32 CPU 下面访问高于 512M 的物理地址空间，必须通过 MMU 地址转换。

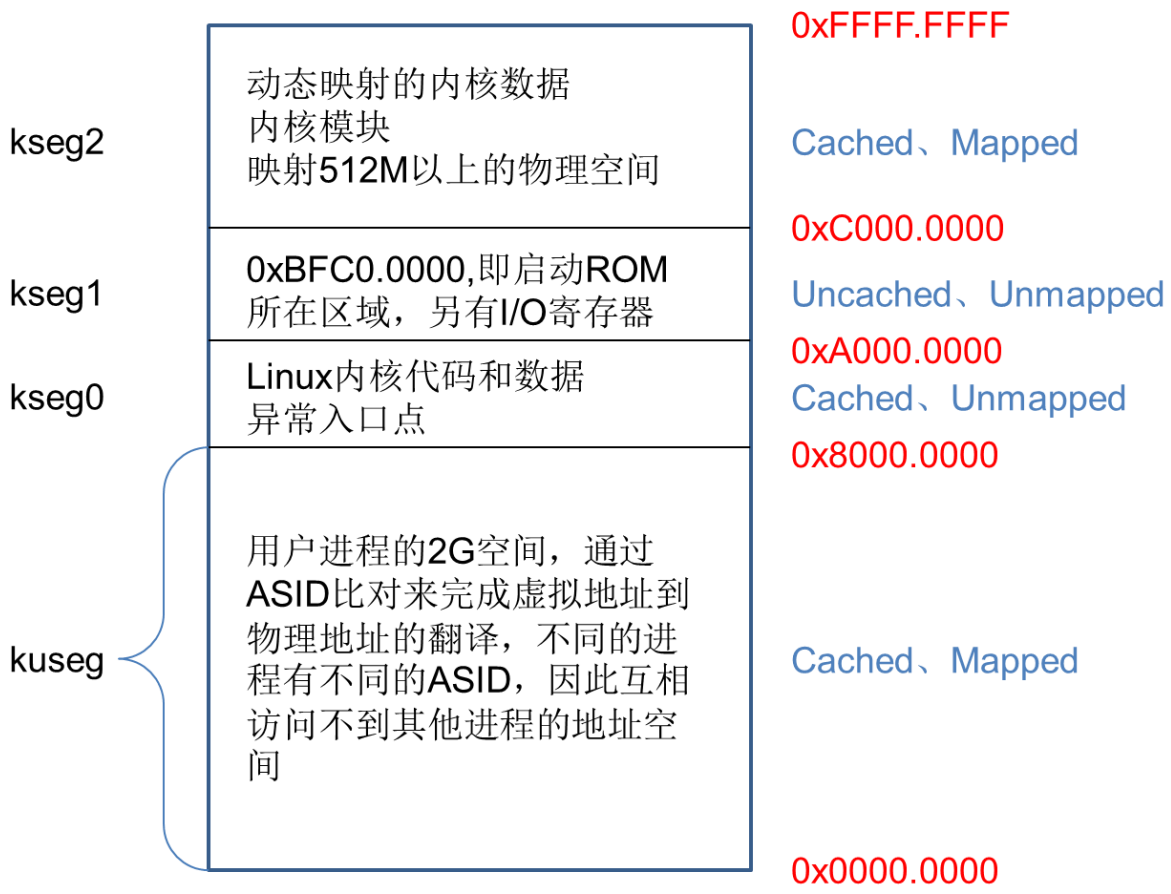


图 3.1 MIPS 32 位地址空间

一个 MIPS CPU 可以运行在两种优先级别上，用户态和核心态。MIPS CPU 从核心态到用户态的变化并不是 CPU 工作不一样，而是对于有些操作认为是非法的。在用户态，任何一个程序地址的首位是 1 的话，这个地址是非法的，对其存取将会导致 CPU 异常。另外，在用户态下，一些特殊的指令将会导致 CPU 进入异常状态。在核心态下(CPU 启动时)，CPU 可以作任何事情。在用户态下，2G 之上的地址空间是非法的，任何存取将会导致系统异常处理。如果一个 CPU 有 MMU，则意味着所有的用户地址在真正访问到物理地址之前必须经过 MMU 的转换，从而使得操作系统可以阻止用户程序随便访问。在核心态下，CPU 可以存取低端 2G 地址空间，这个存取也是通过 MMU 的转换。

上述逻辑的硬件代码实现如图 3.2 描述：

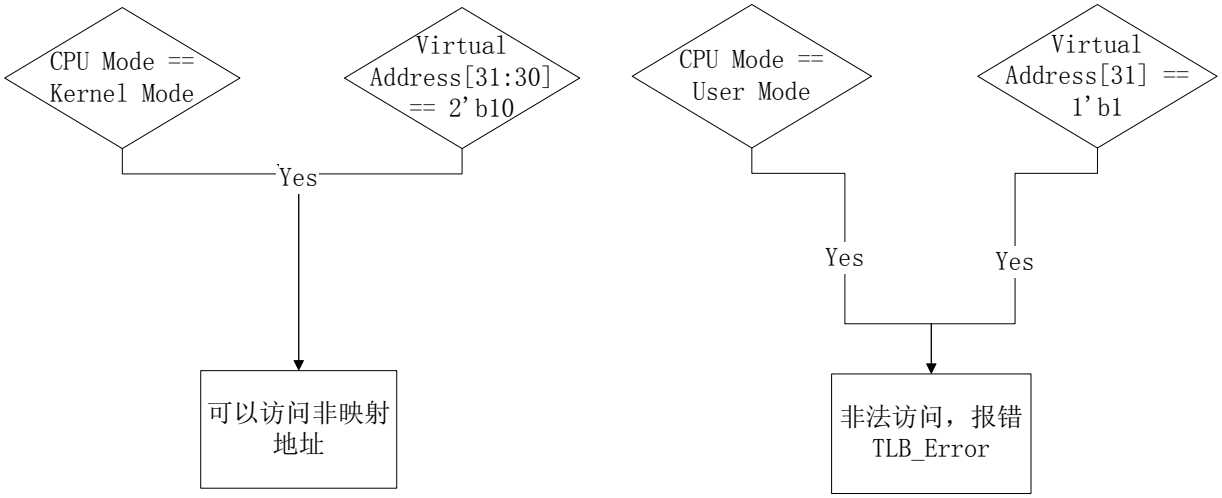


图 3.2 CPU 运行模式与地址保护

内存管理单元采用哪种地址转换方式，取决于处理器处于哪种操作模式。根据操作模式的不同，虚拟地址分成不同的段。32 位虚拟地址构成了 4GB 虚拟地址空间。在设计中设定处理器通常处于用户模式(User Mode)；当系统处于复位状态或检测到异常状态时，处理器进入核心模式(Kernel Mode)。在执行例外返回指令之前，一直保持在核心模式。

用户模式的地址空间是核心地址模式地址空间的一个子集，在用户模式下，设定处理器只允许访问 0x0000_0000 - 0x7FFF_FFFF 的地址空间，而 0x8000_0000 - 0xFFFF_FFFF 这个地址段是不可访问的，如果访问就会引起 CPU 异常。用户模式下默认只可以使用单个均匀的 2GB 虚拟地址空间。在此模式下，地址的最高位 A(31)为 0。在虚拟地址向物理地址转换期间，用 8 位的地址空间标识符(ASID)字段扩充虚拟地址，以形成供多达 256 个用户进程使用的唯一的虚拟地址，通过为每一个 32 位 MIPS 微处理器内存管理单元分配一个 ASID，系统就能跨越上下文切换管理 TLB 内容，对 kuseg 的所有引用都通过 TLB 被映射。

## 3.2 TLB 的设计

根据第二章对 TLB 的设计，每个 TLB 表项对应两个在虚拟空间连续的页，每个虚拟页面可以映射到不同的物理页，并有相关的页面存取控制信息（Flags）[8,9-14]。正如

图 3.3 TLB 表项所示.

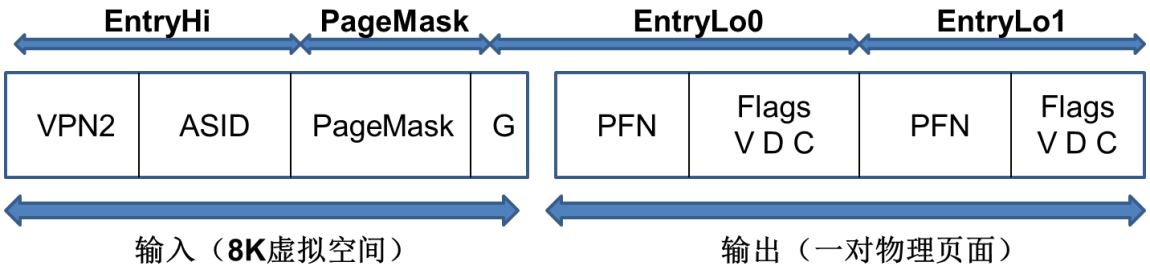


图 3.3 TLB 表项

### 3.2.1 页表项结构定义

对照上图 3.3 TLB 表项，代码设计中对 VPN2（虚页号）、ASID（地址空间标识符）、PageMask、G、两个 PFN（实页号）和两个 Flags（与实页面相关的标记）分别进行定义。其中取 VPN2 长度为 19 位（20 位 VPN，去掉最低位，这一位将来用来自动选择两个可能的输出中的一个），取 ASID 长度为 8 位，取 PageMask 长度为 16 位，取 G 长度为 1 位，取两个 PFN 的长度分别为 20 位，两个 Flags 分别包含 3 位长度的 C 位，1 位长度的 D（Dirty）位和 1 位长度的 V（Valid）位。最后将这些项目按照图中的顺序组合在一起，拼成一个完整的 TLB 页面。

### 3.2.2 页表项更新逻辑

页表的更新信号是 CP0_Update，在时钟上升沿检测到 CP0_Update 为 1 时，从 CP0 中送来的数据写入相应的位里面，完成更新[15,16-18]。

```
if (Reset)
    TLB_Page = 96'b0; //清空 TLB 页面
else if (CP0_Update)//若此时检测到更新信号
begin
    更新 TLB 表项中的每个条目。
    页表中的 Global（全局）位 =  EntryLo0[0] |EntryLo1[0];
end
```

要更新的 TLB 页面数据来自 EntryHi、EntryLo0、EntryLo1、PagemMask 四个寄存

器。特殊说明的是两个实地址页面的 G 位必须一致，这里做它们的或运算，防止在两个位不一样时发生意想不到的错误。

### 3.2.3 不同页面大小的匹配逻辑

PageMask 寄存器允许通过设置 TLB 域，来映射更大的页。PageMask 是 TLB 表项的一部分，值为 1 的各位表示虚地址中相应的位在匹配 TLB 表项时被忽略掉（并且那一位会原封不动地传给最终的物理地址），这样就有效的匹配更大容量的页面。

绝大多数 MIPS CPU 把页面的大小限制在 4KB 到 16MB 之间，且页面大小是以 4 的倍数递增的，为了编程方便，这里将页面大小范围更进一步增大，变为 4KB 到 256MB 之间，表 3.1 显示了页面可支持的范围：

**表 3.1 PageMask 值与页面大小关系**

| 28-25 | 24-21 | 20-17 | 16-13 | 页面大小 |
|---|---|---|---|---|
| 0000 | 0000 | 0000 | 0000 | 4KB |
| 0000 | 0000 | 0000 | 0011 | 16KB |
| 0000 | 0000 | 0000 | 1111 | 64KB |
| 0000 | 0000 | 0011 | 1111 | 256KB |
| 0000 | 0000 | 1111 | 1111 | 1MB |
| 0000 | 0011 | 1111 | 1111 | 4MB |
| 0000 | 1111 | 1111 | 1111 | 16MB |
| 0011 | 1111 | 1111 | 1111 | 64MB |
| 1111 | 1111 | 1111 | 1111 | 256MB |

### 3.2.4 页表项匹配逻辑

页表匹配是指检查页表的三个相关项与送入的虚地址的相关项是否相同，三个相关项都匹配成功就是页表命中，有一项不相同就是匹配失败。第一项是匹配有效 VPN2，有效的 VPN2 是指虚拟地址的高 19 位。如果 PageMask 全零，那么有效 VPN2 就是虚拟地址的[31:13]，否则根据 PageMask 屏蔽相应低位的比较。第二项是匹配有效 ASID。如果页表的 G=1，则表示该页面是一个公共页面，此次匹配会自动忽略送过来的 ASID 值，第二项匹配恒为真。否则如果 CP0 送过来的 ASID 地址与页表中的 ASID 域一致，称为

匹配有效 VPN2。匹配的第三项是匹配状态位（Flags）。如果页表的 V=0，那么此页表无效。如果当前 D=0，那么此页表地址不允许写访问。

#### 3.2.4.1 VPN 匹配

下面这段伪代码是 TLB 匹配逻辑，其中 VPN 从第 32 位开始到第 13 位逐位匹配，第 28 位到第 13 位之间的匹配，必须考虑相应的 PageMask 值是否为 1，如果为 1，则自动忽略这些位的匹配。

```
32 到 29 位匹配  = (VPN[31:29] == TLB_VPN2[18:16]);
29-28 位匹配  =(VPN[28:27]==TLB_VPN2[15:14]) | ( PageMask[14]& PageMask[15]);
…
15-14 位匹配 =(VPN[14:13]==TLB_VPN2[1:0]) | ( PageMask[0]& PageMask[1]);
TLB_Match  = 以上所有位全部匹配 & ASID 匹配 & Flags 匹配。
```

#### 3.2.4.2 ASID 匹配

下面是匹配有效的 ASID 的代码，如果送入的 ASID 和 TLB 页面中存储的页面 ASID 相同，或者 TLB Global 位置位，那么 ASID 匹配成功。

```
ASID 匹配 = (TLB 页面中存储的 ASID == 送入的 ASID)| Global 位 == 1’b1;
```

#### 3.2.4.3 Flags 匹配

Flags 包括 Dirty 位和 Valid 位的匹配，如下所示：

Dirty（脏）位匹配逻辑

```
TLB_MD0 = write & !TLB_D0;
TLB_MD1 = write & !TLB_D1;
//其中 V1 和 V0 为 TLB 页面中存储的页面脏数据信息。
TLB_Modified = 查询的是第一个页面 ? TLB_MD1 : TLB_MD0;
```

Valid（页面有效）位匹配

```
TLB_Valid = 查询的是第一个页面? V1 : V0; // V1 和 V0 为页面有效性信息
```

#### 3.2.4.4 匹配成功后的输出

如果匹配成功，那么输出相应的 PFN（实页面），这里的 PFN 在 PageMask 为全 0 的时候，由送入的 VPN 的第十二位来选择输出页面，若 VPN 不为全 0，会根据 PageMask 掩码掉的最低位的下一位得出输出页面。

```
PFN（实页面） = 查询的是第一个页面? PFN1 : PFN0;
```

## 3.3 MMU 的设计

### 3.3.1 MMU 的整体结构

前面阐述了 TLB 的页面结构、页面匹配逻辑的设计、页面更新逻辑的设计。TLB 表项设计好之后，应当将其组合在 MMU 中用来存储内存页表页面，如图 3.4 所示，MMU 中设置了八个 TLB 页面，这八个 TLB 页面的所有逻辑都与上述 3.2 中 TLB 的逻辑相同。从图中可以看出，与 MMU 通讯的主要部件是 CP0，另外还有外部的数据通路。CU 控制着 MMU 的正确工作时序。

正常工作时，MMU 接收到 CU 的请求信号，将从数据选择器中送过来虚拟地址和在 CP0 中的 EntryHi 寄存器中存储的进程号 ASID 一起送入八个 TLB，进行比对，若比对成功，则输出物理地址，最终物理地址会传送至存储机构的 Flash Memory 或是 SRAM，进而得到数据。

若匹配失败，即八个 TLB 查询之后，均报出缺失，或者匹配的时候发生了 TLB 页面异常，此时 MMU 会向 CU 报告错误，CU 在接受到错误后，进过简单逻辑比对，发出相应的 CPU 异常。

### 3.3.2 虚地址到实地址的翻译

虚地址来源于数据通路中的一个二选一数据选择器，此数据选择器的数据来源为 PC（Program Counter，程序计数器）和 ALU（Arithmetic Logical Unit，运算器），因此虚地址的来源与之前版本 CPU 的实地址的来源相同，即来自 PC 或 ALU。翻译时，由于 8 个 TLB 需要完成匹配工作，因此与原来没有 MMU 时相比，一条普通的访存指令的执行周期将会增加两个时钟周期（MMU 进行一次正常的翻译需要两个时钟周期）。每次翻译时，需要 CU 向 MMU 发出请求信号，设计中在 CU 的状态机中的取址状态加入

一个输出的请求信号，并置为高，当状态机到达读内存或者写内存状态时，再将其复位。这样在每次做访存操作时，CU 都会给 MMU 发出一个周期的请求，使 MMU 完成相应的翻译工作。

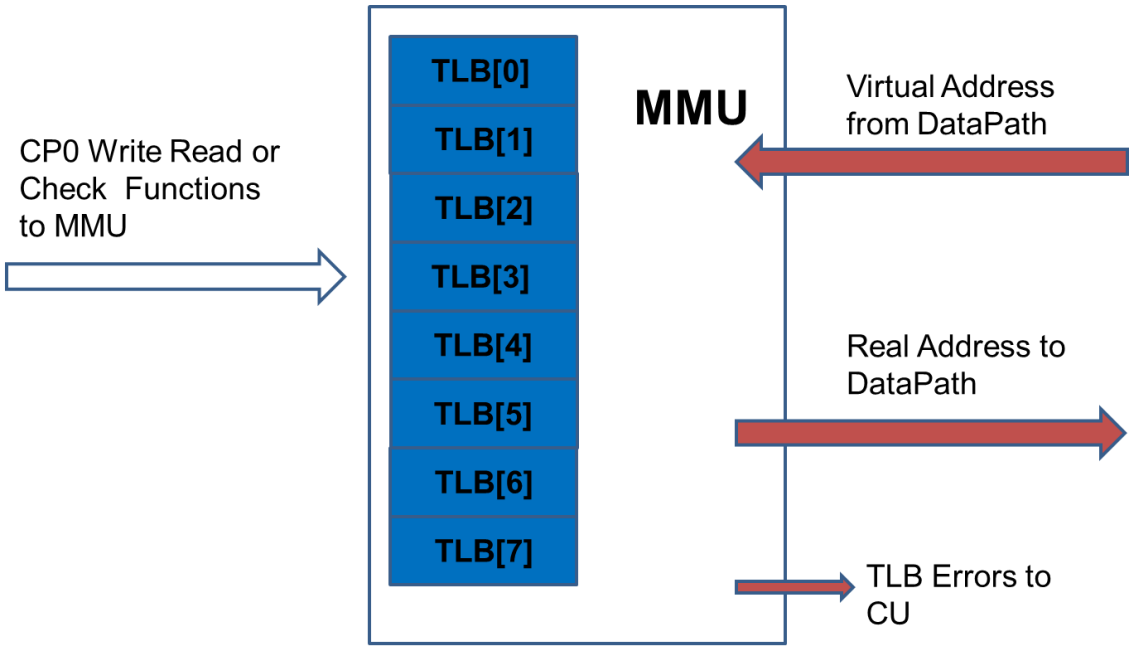


图 3.4 MMU 逻辑结构

### 3.3.3 错误处理与错误优先级

MMU 的定义的错误有 4 种，分别是：

①在用户模式下访问高 2G 的地址空间，引起异常，定义为 TLB_Error；

②匹配失败引起异常，定义为 TLB_Miss；

③执行写入操作时，D 位为 0，引起异常，定义为 TLB_Modified；

④检测页面 Valid 位时，Valid 位为 0，引起异常，定义为 TLB_InValid。

其中后三种异常 TLB_Miss、TLB_InValid 和 TLB_Modified，设计中将其组合成为一个组信号，称为 TLB_Fault。

TLB_Error、TLB_Miss、TLB_InValid 和 TLB_Modified 四中异常的优先级由高到低，即优先级高的一种异常发生之后优先级低的异常即使发生，也会被屏蔽掉，具体实现的逻辑如图 3.5 所示。

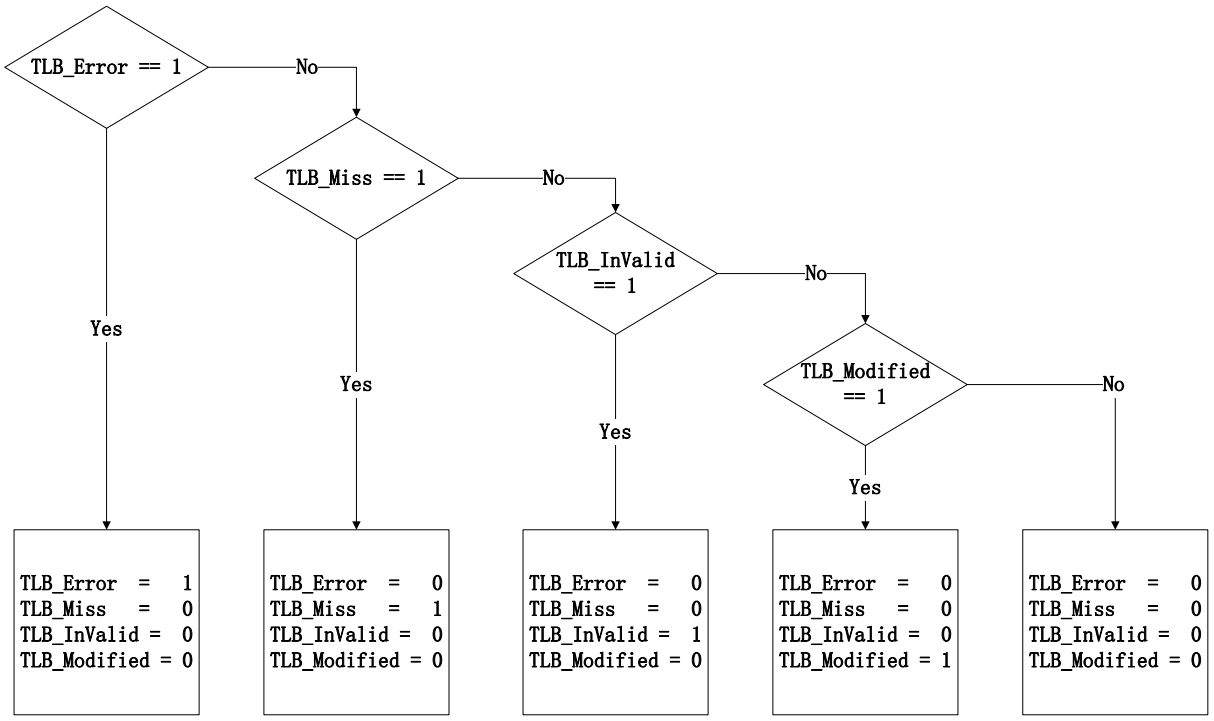


图 3.5 MMU 报错优先级示意图

### 3.3.4 TLB 相关指令的支持

#### 3.3.4.1 TLBR 的支持

即 TLB Read，执行读取 TLB 页面。该指令根据 CP0 中 Index 寄存器所存储的值所指定的 TLB 页面分解，并将相应的内容装入 CP0 中的 EntryHi、EntryLo0、EntryLo1 和 PageMask 寄存器（此处的这些寄存器及其作用可参考 3.4 中的表 3.2 新 CP0 中的寄存器）。

#### 3.3.4.2 TLBWI 和 TLBWR 的支持

TLBWI：即为 TLB Write Index，根据 CP0 中的 Index 寄存器内容写 TLB。该指令将 EntryHi、EntryLo0、EntryLo1、PageMask 寄存器的内容写入指定的 TLB 项[19,20-24]。

TLBWR：即为 TLB Write Random，随机写 TLB。该指令将 EntryHi、EntryLo0、EntryLo1、PageMask 寄存器的内容装入 CP0 中的伪随机地址寄存器（Random 寄存器）指定的 TLB 项。

TLBWI 和 TLBWR 在 MMU 中可以被合成为一条指令。CP0_MMU_Func（CP0 发

给 MMU 的功能编号） 指定操作为 TLB 更新操作，具体的写入哪个 TLB 是由 CP0 中的 Index 寄存器给出的还是 Random 寄存器给出的，MMU 并不需要关心这个问题。

写入操作即为 TLB 的 Update 操作，更新 TLB 页面，这里需要 CP0 发出写请求，然后再根据发过来的 TLB 编号，确定要更新哪个 TLB 页面。

#### 3.3.4.3 TLBP 的支持

TLBP 即为 TLB Probe，探测 TLB。具体是用 EntryHi 中的 VPN2 和 ASID 对 TLB 进行查找。如果没有一个 TLB 项与之相匹配，则 CP0 的 Index 寄存器最高位置 1，这样可以让 Index 寄存器的整个值看起来像个负数，便于 CPU 进行进一步的响应和处理。否则，将匹配项的序号写入 Index 寄存器。实现逻辑比较简单，此处不赘述。

### 3.4 CP0 的修改

从 3.3 节中可以看出，要完成 MMU 的匹配工作，需要多个寄存器的支持，这些寄存器放在 CP0 中，用 CP0 指令 mfc0 和 mtc0 来读写这些寄存器，这样便可以在操作系统中编写针对类似于进程切换或者 TLB 异常处理操作的代码。表 3.2 列出了 CP0 全部的寄存器，其中新加入的寄存器已被标记出来。这些寄存器均是 MIPS 32 体系结构定义的标准寄存器。

**表 3.2 新 CP0 中的寄存器**

| 寄存器助记符 | 寄存器编号 | 描述 | 是否为新加入 |
|---|---|---|---|
| Index | 0 | 在使用适当的指令时决定读或写哪个 TLB 表项 | 是 |
| Radom | 1 | 这个伪随机值用来让 TLBWR 写入新的 TLB 表项到一个随机选择的位置。为使用随机替换算法的软件在陷入 TLB 重填异常时的处理节省了时间。 | 是 |
| EntryLo0 | 2 | 描述两个独立页面的 PFN 和访问允许标志分别由 EntryLo0 和 EntryLo1 这两个寄存器指出 | 是 |

| | | | |
|---|---|---|---|
| EntryLo1 | 3 | — | 是 |
| Context | 4 | 用来加速 TLB 重填的过程，它的高位可读写，低位来自不被翻译的 VPN 地址，如果内存翻译记录的内存副本使用了合适的安排方式，那么 Context 寄存器域的这样布置会使在发生 TLB 重填陷阱后，Context 会装有一个指针，指向用来映射触发异常地址的页表记录 | 是 |
| PageMask | 5 | 可以用来创建映射超过 4KB 的表项 | 是 |
| Wired | 6 | 从中制定基值，之后 Random 寄存器会在此值到最大值之间随机取数字，此值之下的 TLB 页面不会被随机替换，成为永久表项 | 是 |
| BadVaddr | 8 | 导致最近的地址相关异常（address-related exception）的程序地址。各种地址错误或异常都会设置它。 | 是 |
| EntryHi | 10 | 存储 VPN（虚页号）和 ASID（地址空间标识符） | 是 |
| SR | 12 | 状态寄存器，大部分是可写控制位。包括决定 CPU 的特权等级，中断引脚使能，以及其他的 CPU 模式等的位域 | 否 |
| Cause | 13 | 什么导致异常或中断 | 否 |
| EPC | 14 | 异常程序计数器：处理完异常/中断之后从哪里重新开始执行 | 否 |
| PRID | 15 | CPU 类型和版本号信息。这个类型号由 MIPS 公司管理，指令集或者 CP0 寄存器集发生了改变的时候必须变化。 | 否 |

## 3.5 CU 的修改

设计中加入了 MMU 之后， CU（Control Unit）的控制逻辑必须加以修改，让 MMU 的工作逻辑与整体时序相匹配。

### 3.5.1 状态机中增加 MMU 翻译状态

在 MMU 翻译状态下，如果执行 TLBR 或 TLBP 指令，MMU 会在执行完毕时发出应答脉冲，CU 接收到此脉冲时，状态机转到取址（Fetch）状态，如果执行 TLBWI 或 TLBWR，则不需要应答信号，状态机直接进入取址（Fetch）状态。

### 3.5.2 增加异常跳转逻辑

设计中将 MMU 发出的 TLB_Error 和 TLB_Fault（见 3.3.3）接入 CU，在 CU 检测到这两个信号时， CPU 进入异常处理状态，CU 设置 ExcCode 并跳入设置的异常入口。

这里根据 TLB_Error 和 TLB_Fault 分别设置了三种 ExcCode，具体见表 3.3。

**表 3.3 ExCode 值：不同异常类型**

| ExcCode 值 | 助记符 | 进入条件 | 描述 |
|---|---|---|---|
| 2 | TLBL | TLB_InValid == 1'b1 \| TLB_Modified == 1'b1 | 没有 TLB 转换 |
| 3 | TLBS | TLB 匹配失败 | 根本没有匹配的页面且 CPU 没有处于异常模式，这是一个 TLB 缺失，将由一个特殊的指向能流畅处理这种常见事件的异常入口点来处理 |
| 4 | AdEL | TLB_Error == 1'b1 | 地址错误，是在用户态下试图访问 kuseg 以外的段，或是试图读一个双字、字或半字而地址没有相应的对齐 |

## 3.6 新数据通路

完成上述工作之后，之后的工作是将 MMU、新 CP0 和新 CU 加入整体设计，形成新的数据通路。

如图 3.6 所示，MMU 被添加在原访存地址的出口处，此时它将接受数据选择器 M1

（程序地址或数据地址的选择器）给出的虚地址，将其翻译为实地址输出给存储器。MMU 工作时要与 CP0 密切配合，因此数据通路中的 MMU 与 CP0 交换信号比较多，但是由于数据通路图的大小的限制，不能完全画出，这里用一根线代替所有的交换信号，并在图的最下面注释为 MMU 和 CP0 之间的数据交换。

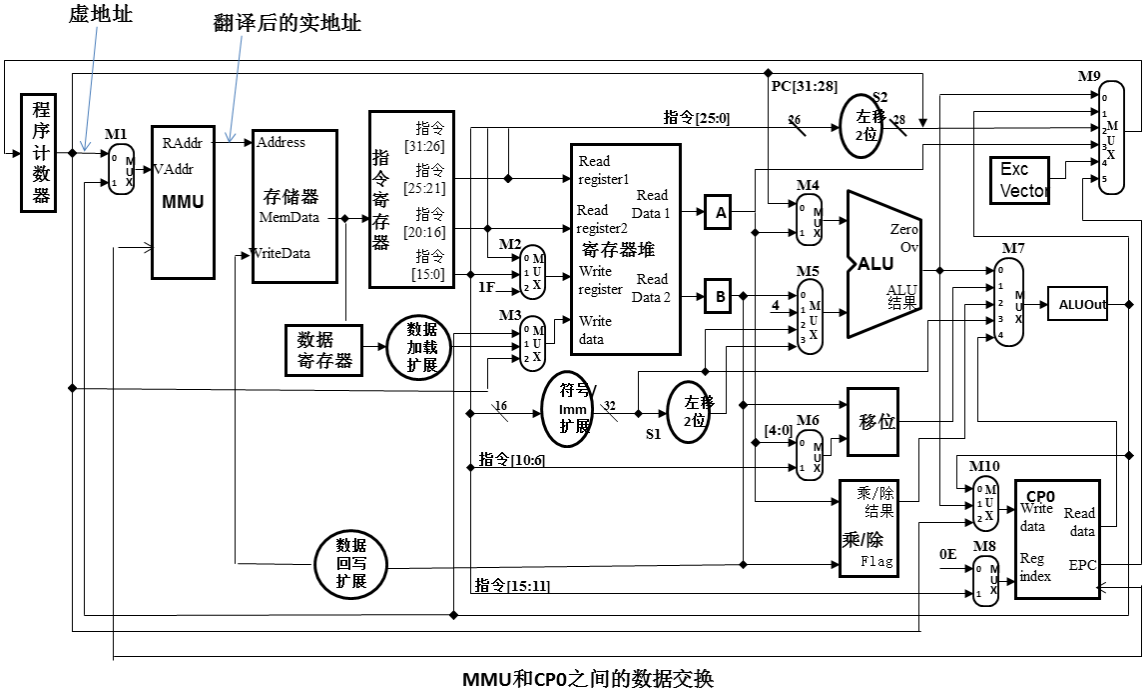


图 3.6 加入 MMU 之后的数据通路

## 3.7 操作系统的修改

一个 TLB 缺失异常总使用一个专用的入口点（即为 0xBFC0_0200），除非 CPU 已经在处理异常中，即 SR（EXL）位被置位。附录一中的代码是通用的 MIPS32 CPU 的 TLB 缺失异常处理代码，列出来供参考。

## 3.8 本章小结

本章对上一章提出的 MMU 的设计方案进行设计实现，对整个设计过程和设计细节做了详细的说明和讨论。具体设计了页表项和 TLB 表项，将 8 个 TLB 页面加入 MMU，详细讨论了 MMU 的实现逻辑。为了配合 MMU 工作，在 CP0 中新加入了 9 个寄存器，并编写了它们的读写逻辑。

完成 MMU 的设计之后将 MMU 加入了整个 CPU 中，在 CU 中加入其控制逻辑，

使地址翻译时的时序与整个系统的时序相匹配，这样MMU遇到错误时能够正确的报出，并能够跳入正确的异常入口点。为了配合 TLB 缺失的异常处理，操作系统中也添加了相应的处理代码，这样 TLB 缺失异常能够自动化处理。

# 4 测试和验证

测试工作分为两部分来进行，第一部分是单个模块的功能测试，以及几个简单模块的集成测试。每个功能模块都需要进行测试，将局部错误减到最少甚至消除。第二部分是 MMU 接入 CPU 之后的整体测试。

## 4.1 测试环境

单个模块测试的测试环境与整体测试的测试环境有所不同，因此这里分开说明测试环境。

### 4.1.1 模块测试环境

单个模块功能的测试是将代码文件放入 Quartus II 7.2 版中，并建立与之相匹配的波形文件，然后用这个版本的 Simulate Tool（仿真工具）实现的。由于时序仿真波形中毛刺比较多，因此测试时使用功能仿真。

### 4.1.2 整体测试环境

整体测试时，有以下几个工具要使用：

①Quartus II 11.0 版：Quartus II 是 Altera 公司的综合性 PLD/FPGA 开发软件，支持原理图、VHDL、Verilog HDL 以及 AHDL（Altera Hardware Description Language）等多种设计输入形式，内嵌有综合器以及仿真器，可以完成从设计输入到硬件配置的完整 PLD 设计流程。在本次测试中主要用于综合整体设计并下载。

②NIOS II：Nios II 嵌入式处理器是 ALTERA 公司推出的采用哈佛结构、具有 32 位指令集的第二代片上可编程的软核处理器，其最大优势和特点是模块化的硬件结构，以及由此带来的灵活性和可裁减性。相对于传统的处理器，NIOS II 系统可以在设计阶段根据实际的需求来增减外设的数量和种类。设计者可以使用 ALTERA 提供的开发工具 SOPC Builder, 在 PLD 器件上创建软硬件开发的基础平台, 也即用 SOPC Builder 创建软核 CPU 和参数化的接口总线 Avalon。在此基础上, 可以很快地将硬件系统(包括处理器、存储器、外设接口和用户逻辑电路)与常规软件集成在单一可编程芯片中。而且, SOPC Builder 还提供了标准的接口方式,以便用户将自己的外围电路做成 NIOS II 软核可添加

的外设模块。在本次测试中主要用于将编译好的操作系统和用户程序写入 Flash Memory。

③USB Blaster：Altera 的 FPGA/CPLD 程序下载电缆，通过计算机的 USB 接口可对 Altera 的 FPGA/CPLD 以及配置芯片进行编程、调试等操作。

④Logic Analyzer：Logic Analyzer 全称 SignalTap II Logic Analyzer，是第二代系统级调试工具，可以捕获和显示实时信号，观察在系统设计中的硬件和软件之间的互相作用。Quartus II 软件可以选择要捕获的信号、开始捕获的时间，以及要捕获多少数据样本。还可以选择时间数据从器件的存储器通过 JTAG 端口传送至 SignalTap II Logic Analyzer，还是至 I/O 引脚以供外部逻辑分析仪或示波器使用。能够将实时数据提供给工程师，帮助 debug。

⑤串口 232（RS-232）：个人计算机上的通讯接口之一，由电子工业协会(Electronic Industries Association，EIA) 所制定的异步传输标准接口。通常 RS-232 接口以 9 个引脚 (DB-9) 或是 25 个引脚 (DB-25) 的型态出现。在本次测试中应用 RS-232 来传输 COCOA 平台打印出的字符至计算机终端。

⑥终端：终端是一个用于硬件串口通讯的软件，能够将串口发送的数据接收为字符串，并显示在电脑上。

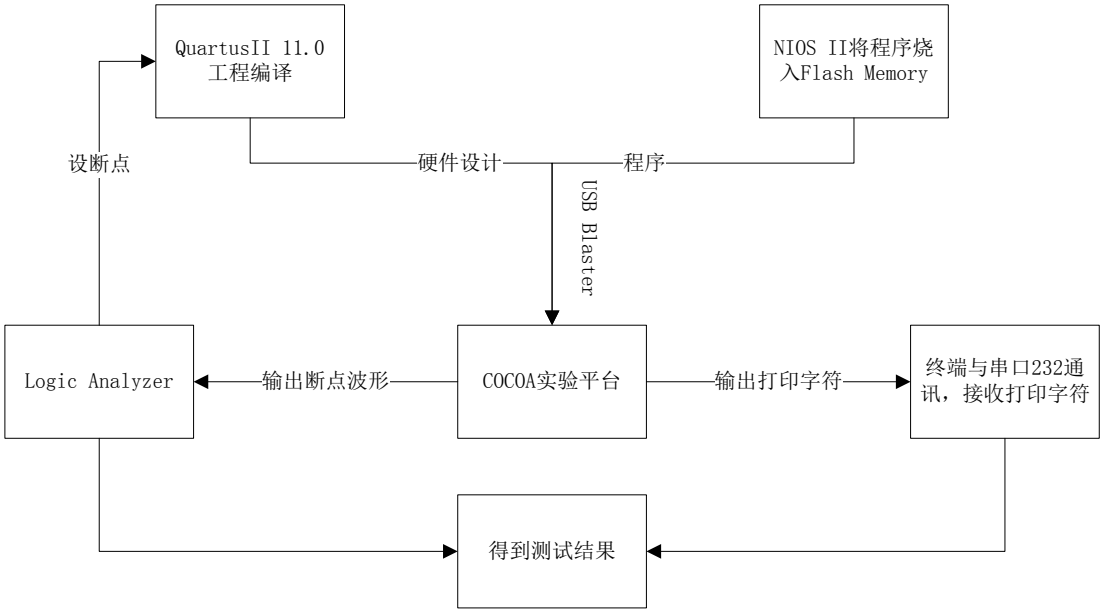


图 4.1 整体测试流程图

图 4.1 是整体测试的测试流程。硬件方面，设置硬件断点并在 Quartus 中编译整个

工程，下载至 COCOA 平台。软件方面，将写好的操作系统代码以及用户程序代码用 GCC 编译，并用 NIOS 将其下载至 COCOA 平台的 Flash Memory 中。在 Logic Analyzer 和终端的监控下，得到最终的测试结果。

## 4.2 模块测试

这部分包括 TLB、MMU、CP0 新加入的寄存器以及整体测试，还有 MMU 和 CP0 这两个模块协同工作的整体测试。

### 4.2.1 TLB 模块测试

TLB 模块的功能主要有两个，一个是 TLB 页面的更新，在每次操作系统切换进程之后，TLB 需要配合更新相应的页面，或者是所有的 TLB 页面全部未命中时，操作系统需要随机替换掉一个页面，这时 TLB 也需要更新其页面。另外一个是 TLB 的匹配功能，外部给出一个 VPN（虚页面）和一个 ASID（进程号），TLB 需要在匹配的时候给出匹配信号，在不匹配的时候，相应的 TLB_Miss 、TLB_InValid 和 TLB_Modified（见 3.3.3）三个信号需要指出不匹配的原因。

#### 4.2.1.1 TLB 更新测试

在测试的时候，由波形送入要写入 TLB 的数据，分别是 EntryHi、EntryLo0、EntryLo1 和 PageMask 四个 32 位数据，他们的值分别为 0x6000_0001，0x0110_0006，0x2300_0002，0x0000_0000，将 TLB_Update（TLB 更新，见 3.2.2）信号长度以及这四个输入的长度持续一个时钟周期，测试结果如图 4.2 所示。

由图 3.3 可以看出在 Posedge Clk 之后，TLB_Page 变为一个非 0 的值，PFN 也变为一个非 0 的值。所得的结果为预期结果，根据 TLB 的写入逻辑进一步分析可得，写入的数据正确，是四个寄存器值的组合，写入逻辑这方面符合最初的设计预期，能够满足设想的功能。

#### 4.2.1.2 TLB 匹配测试

需要测试两种情况，匹配成功和匹配无效页面，首先用 0x6000 这个虚页号和值为 0x01 的 ASID（进程号）来测试匹配成功的情况，如图 4.3 所示。

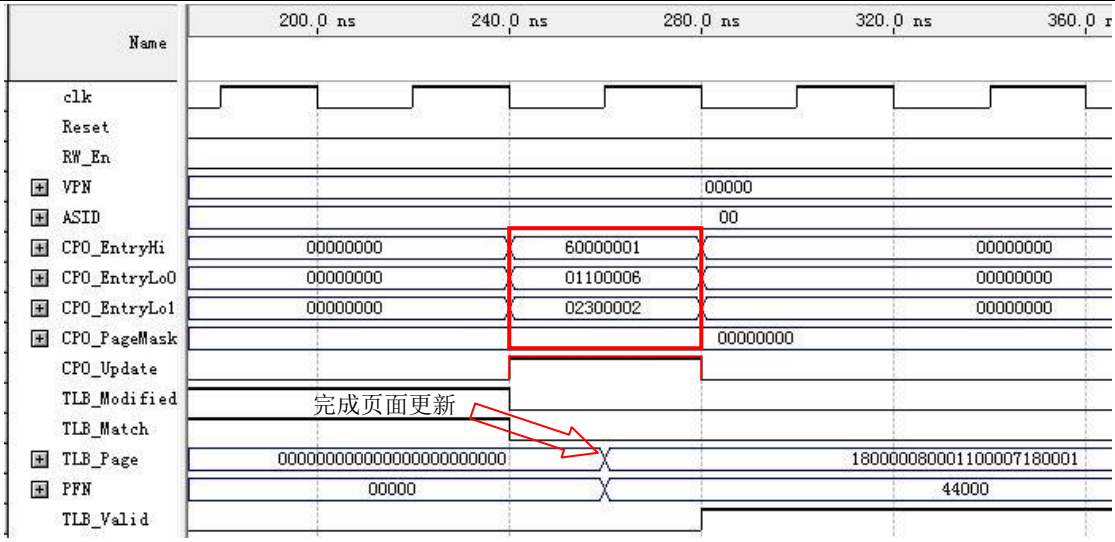


图 4.2 TLB 更新测试

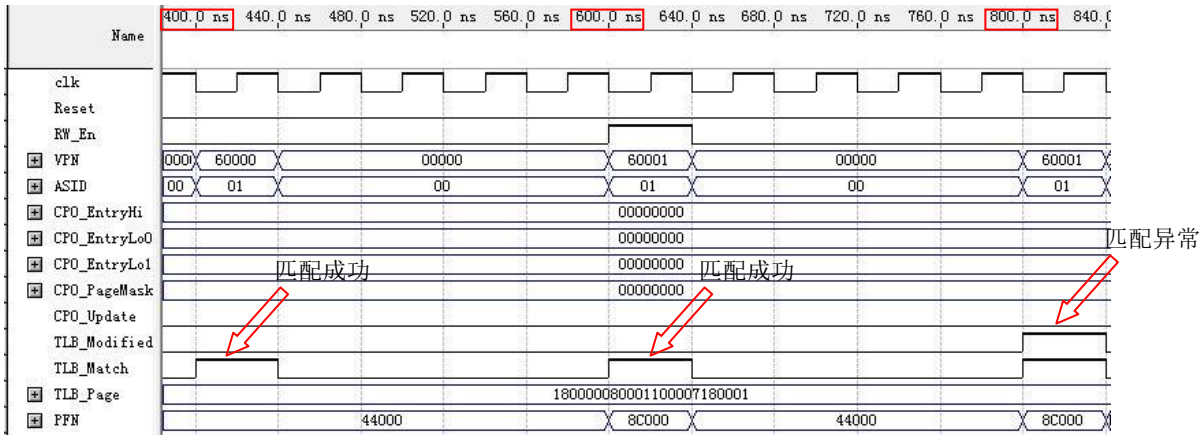


图 4.3 TLB 匹配测试

由图 4.3 的 400ns 处可以看出，将 VPN（虚页号）、ASID 这两个值分别给一个时钟周期的宽度，TLB_Match 为高，证明 TLB 匹配成功。

后面 600ns 处和 800ns 处的两次对比匹配都是匹配 TLB 中的第二页面，此时的 VPN 值为 0x60001，第二个页面的值为 0x2300_0002，即 D 位为 0，所以如果是一次写入操作，TLB_Modified 会为高，此次匹配失败。600ns 处的 RW_En（指出此次访存操作为读内存还是写内存）为高，这是一次模拟读的操作，因此 TLB_Modified 未报出错误；而在 800ns 处，RW_En 为低，这是一次模拟写的操作，此次操作 TLB_Modified 为高，报出了错误，测试成功。

综合这两个测试结果，可知 TLB 的测试结果符合预期，完全能够满足设计要求。

### 4.2.2 MMU 模块测试

MMU 模块的测试主要有以下 3 点：

①测试 MMU 的写入功能，指定 MMU 中的 TLB 号码，将页面写入其中；

②测试 MMU 的查询功能，即写入的页面是否能被正确的翻译；

③测试 MMU 的报错功能。

由于 MMU 模块信号过多，测试时间也过于长，因此不能再一个完整的图上面标示出来，这里用语言叙述来描述测试结果。

首先分别设置 TLB 页面，并写入编号为 0、1、2、5 的 4 个 TLB，接下来测试查询功能。给出一个虚地址和一个进程号和一个请求信号，信号长度持续一个时钟周期，两个时钟周期之后，匹配结果正确。

测试 MMU 的报错功能，首先将 CP0_MMU_Mode 置为 0，即处理器权限处于用户模式，此时的 MMU 遇到高 2G 空间的访问请求会报出错误，这里用 0x9000_0000 这个地址来测试，经测试 MMU 能够正确报出 TLB_Error 错误。

综合以上测试结果，可知 MMU 的测试结果符合预期，完全能够满足设计要求。

### 4.2.3 CP0 模块测试

从第三章的设计可知 CP0 中加入了 9 个与 MMU 操作有关的寄存器，CP0 测试主要是测试这些寄存器具体的读写逻辑的完整性，以及一些特殊用途的对外透明的寄存器的操作逻辑。

首先测试新加入的 EntryLo0、EntryLo1、EntryHi 和 PageMask 四个寄存器的写入功能，从 DataIn_I（见图 4.4 CP0 整体测试(1)中的信号名 name，下面几个信号缩写也在图中标出）送入数据，RegIdx_I 为相应的寄存器号码的输入，We_I 为写使能的输入，然后观察一个周期之后的结果，从图 4.4 中可以看出，这几个寄存器的写入逻辑均正常。

接下来模拟 MMU 输出数据给 CP0， MMU 与 CP0 之间的协同工作与 TLBR，TLBWI，TLBWR，TLBP 四条 TLB 指令的执行密不可分，具体测试结果见图 4.5。

首先是 TLBP 指令的执行测试，此指令是用 EntryHi 中的 VPN2（虚页号 VPN 去掉最低 1 位）和 ASID 对 TLB 进行查找，若没有一个 TLB 项与之相匹配，则 CP0 的 Index 寄存器最高位置 1，这样可以让 Index 寄存器的整个值看起来像个负数，否则，将匹配

项的序号写入 Index 寄存器。

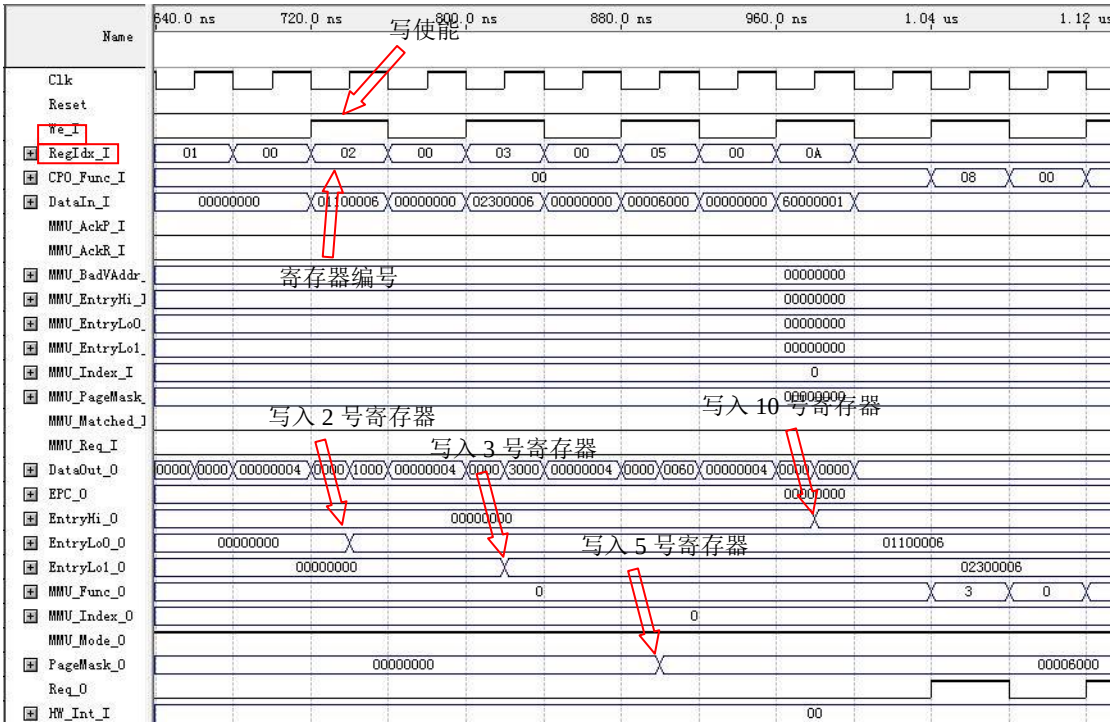


图 4.4 CP0 整体测试(1)

1.36us 处为 TLBP 指令执行完毕，并且有匹配的 TLB，因此此处的 MMU_Matched_I 为高。由于 CP0 中的 Index 寄存器的寄存器号为 0 号，因此在未做任何操作时，DataOut_O 所示的值就是 Index 寄存器的实际值。从图中可以看出 6 被写入了 Index 寄存器，TLBP 指令执行正确。1.44us 处测试的也是 TLBP 指令，但是此时的 MMU_Matched_I 为低，即匹配失败，Index 的第 32 位被置位，从图中可以看出这时的 DataOut_O 变为 0x8000_0006，最高位被置位，测试结果符合预期。

TLBR 指令会根据 CP0 中 Index 寄存器所存储的值将指定的 TLB 页面分解，并将相应的分解内容装入 CP0 中的 EntryHi、EntryLo0、EntryLo1 和 PageMask 寄存器。1.52us 处是 TLBR 指令的测试，图 4.5 中可以看出当 MMU_AckR_I（MMU 应答 TLBR 指令，表示 TLBR 执行完毕，返回值送到了相应位置）为高，EntryHi、EntryLo0、EntryLo1 和 PageMask 寄存器送过来数据之后，CP0 中这四个寄存器的值被成功改写，测试成功。

TLBWI 和 TLBWR 指令测试只是输出，此处不再做特殊说明，具体结果可见测试文件。

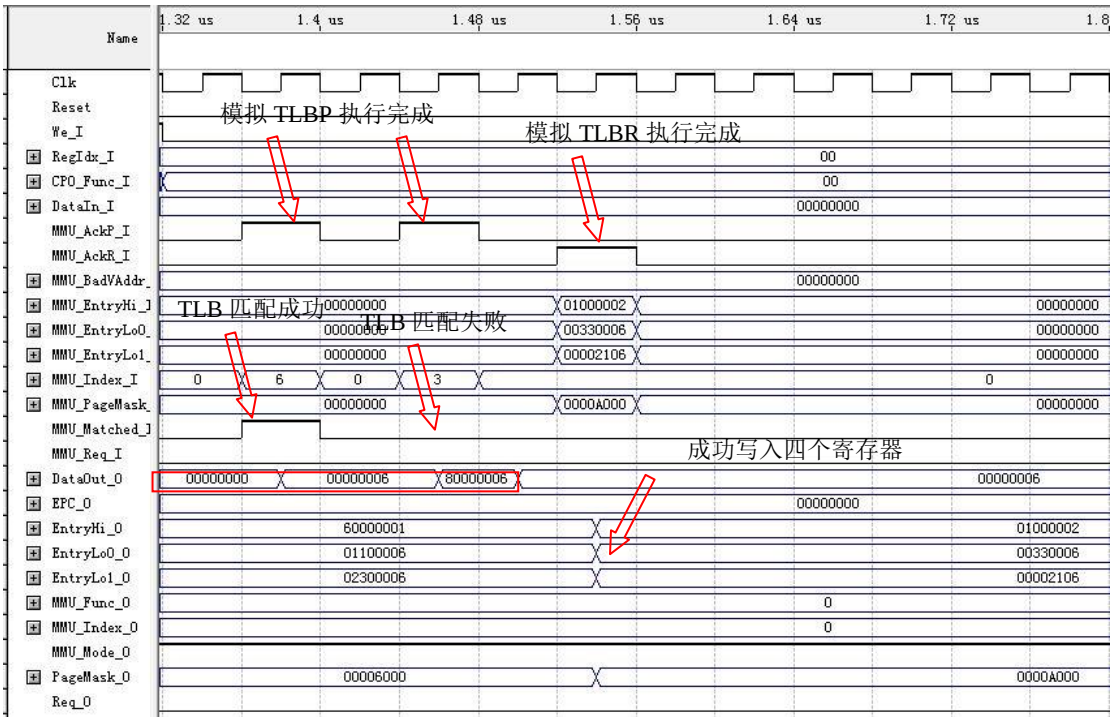


图 4.5 CP0 整体测试(2)

## 4.3 整体测试

整体测试的工作包括两部分，一部分是整体接入之后的测试，这部分的工作主要在 ModelSim 这个软件下进行的[25]，主要包括整体时序的匹配，TLBR，TLBWI，TLBWR，TLBP 四条 TLB 指令的执行，以及最终下载之前的模拟。这部分的工作耗时比较长，但是由于这部分工作的软件 ModelSim 的仿真界面是黑色的，截出图片效果很差，故如果想得到这部分的测试数据，可以从测试工程中查看，论文中将不再列出。

另一部分工作是下载之后的测试，这部分工作的工具有主要两个：Logic Analyzer（逻辑分析仪）和串口通讯软件。串口通讯软件打印出来的是最后的结果，这里主要应用 Logic Analyzer 来查看具体的指令的执行。下面将主要说明使用这个工具调试的一些结果。

### 4.3.1 测试用例说明

如附录二中的三段代码所示，整体测试主要工作围绕操作系统展开。

首先启动时在非映射区设置页表，并把页表映射关系建立起来，这里取 0x9040_xxxx 的地址存储页表，低 12 位为页面内偏移，所以页表地址的低 12 位并不代表页面地址，

因此低 5 位存储页面的 Flags 信息，包括 C 位，D 位和 G 位（具体见图 2.2）。

当启动程序执行完毕之后，操作系统转而执行用户程序，由于用户程序测试用例是这时唯一在操作系统中运行的用户程序，故操作系统转而执行测试程序。操作系统在进行进程调度之前，首先要设置 CP0 中与存储管理密切相关的寄存器的值，这是附录二中的第二段代码完成的工作。

执行测试程序时，首先操作系统会从 0xBFC0_xxxx 的地址跳转到之前设置的用户程序地址 0x4000_0000 开始执行，测试程序中有一次非法访问高 2G 空间的操作，即访问 0xF000_0000，这里操作系统应该调用错误处理程序，并让系统进入死循环。

### 4.3.2 测试用例期望结果

首先在用户程序刚切换过来，执行到 0x4000_0000 处时，应该引发 TLB 缺失异常，这时候 CPU 进入异常状态，并跳转到相应的异常处理入口 0xBFC0_0200，处理完毕之后返回 0x4000_0000 处继续执行；处理异常之后的地址翻译都是成功的，没有再一次引发异常，对照着实地址，可以预测到翻译后的结果为 0x9040_xxxx。

非法访问时，操作系统会调用异常处理函数，打印非法访问的字符串至串口，并且整个操作系统陷入死循环。

### 4.3.3 正常运行结果

系统启动之后首条指令入口地址将是 0xBFC0_0000，这个地址是一个非映射的地址，因此会被 MMU 直接输出，之后的几条指令亦是如此，直到用户程序之前的所有启动指令都不会被 MMU 翻译。因此翻译测试有两点，一方面是非映射地址是否能够被正确输出，另一方面是用户程序地址（映射的地址）能否被正确翻译。

首先测试非映射地址是否能够被正确输出，据下图 4.6 可知，Pr_Vaddr（虚地址）送入源地址，翻译完的地址经由 MMU_Pr_RAddr 送出，翻译后的地址输出比输入稍有滞后延迟。

用户的第一条指令执行时会造成 TLB 缺失（TLB_Miss），这时候应该跳转到 TLB 重填页面处理的异常入口处，TLB 重填页面的异常入口为 0xBFC0_0200。

从图 4.7 中可以看出，用户程序的第一条指令 0x4000_0000 引起了 TLB 缺失异常，TLB_Miss 为高，之后 PC 跳转进入 TLB 异常处理入口 0xBFC0_0200，测试成功，符合

设计构想。

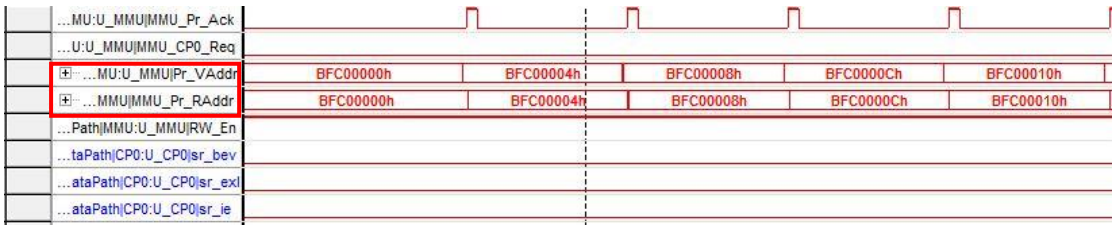


图 4.6 Unmapped 类型的地址翻译测试

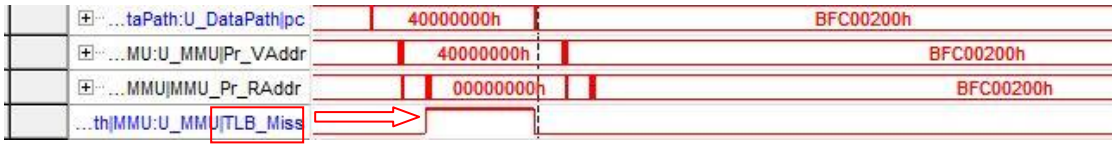


图 4.7 pc 跳转入 TLB 异常处理入口

在异常处理程序中，除了计算得出错误地址，从页表中载入正确的数据并加载到 CP0 相应的寄存器中等一系列操作之外，还有一条指令非常值得关心，即 TLBWR，此指令为按照 Random 寄存器所指的 Index 写入 TLB，在编译后的操作系统代码中找到此条指令之后在 PC 值中将其设为断点，查看 0xBFC0_0200 附近的代码，可以找到此代码的位置，如下面代码所示：

```
…
bfc0027c:   409b1800    mtc0    k1,$3
bfc00280:   00000000    nop
bfc00284:   42000006    tlbwr           ；此处设置断点，观察 tlbwr 的执行
bfc00288:   00000000    nop
bfc0028c:   42000018    eret
…
```

此条指令的地址为 0xBFC0_0284，在 Logic Analyzer 中将 PC 的断点设置为此地址之后看执行此指令执行时 TLB 页面有无变化以验证设计。

由于 Logic Analyzer 的资源有限，因此不能将 MMU 中的 8 个 TLB 页面监视，但在图 4.8 中可以通过特定信号看出 TLB 页面是否被更新。图中可以看到 EntryHi、EntryLo0、EntryLo1 的值都被分配好了，MMU_Func（MMU 从 CP0 得到的功能号）为 2，查宏定

义（为了方便读者理解起见，将这里的宏定义列出来），可知当 MMU_Func 值为 2 时，MMU 会执行 TLB 更新的相应逻辑。

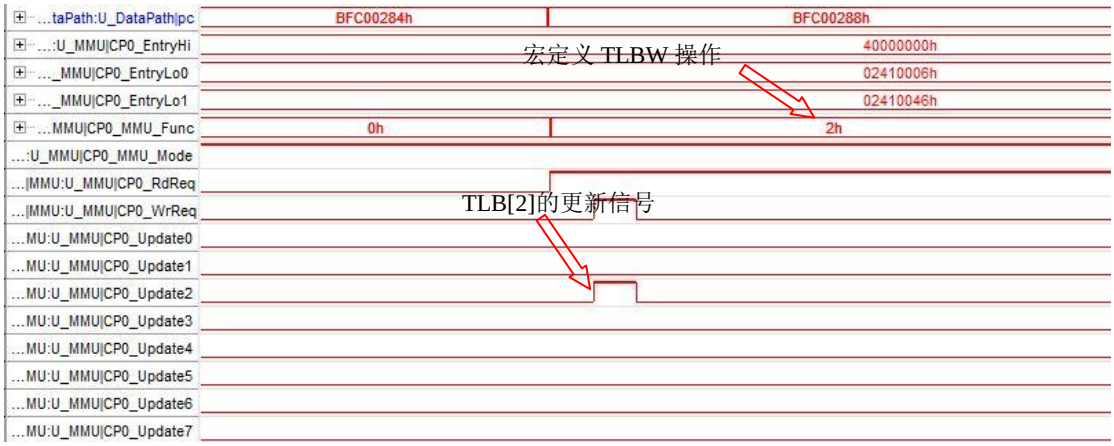


图 4.8 tlbwr 的执行

处在图 4.8 下部的 CP0_Update2 为高，可知此次写入的是 MMU 中的第三个 TLB 页面。具体是否写入成功可以通过执行后续的用户程序验证，如果写入失败，则再执行用户程序，还会跳转到 TLB 缺页异常，如果写入成功，则用户程序可以正常执行。

在处理完 TLB 缺页异常之后，PC 跳回用户程序继续执行，此时检验之后的地址能否被正常的翻译。如图 4.9 所示， 0x4000_0000 翻译完之后的地址为 0x9040_0000，地址 0x4000_0084 被翻译为 0x9040_0084。地址低 12 位是页内偏移，页号能够被正常翻译，则在 4M 空间内的程序不会再发生 TLB 缺页情况，符合最初的设计，此时的用户程序能够流畅运行，整个 MMU 工作稳定正常。

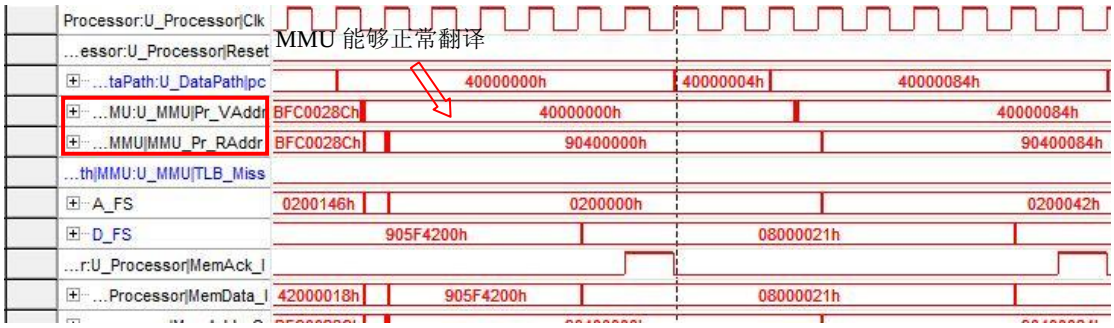


图 4.9 用户程序的执行

### 4.3.4 非法访问异常的抛出

在模块测试时，非法访问异常的抛出功能能够起作用，但是需要在异常发生时，改

变 CPU 特权级别。正常运行时的 CPU 处在用户级别，非法访问之后 CPU 处在内核级别，这时 MMU 可以提供访问高 2G 地址空间的服务，否则不能访问。在整体测试时要求 CPU 能够正确管理的运行级别，比如开机时 CPU 应该处于内核级别，异常时也应该处于内核级别，平常运行用户软件时，处于用户级别。

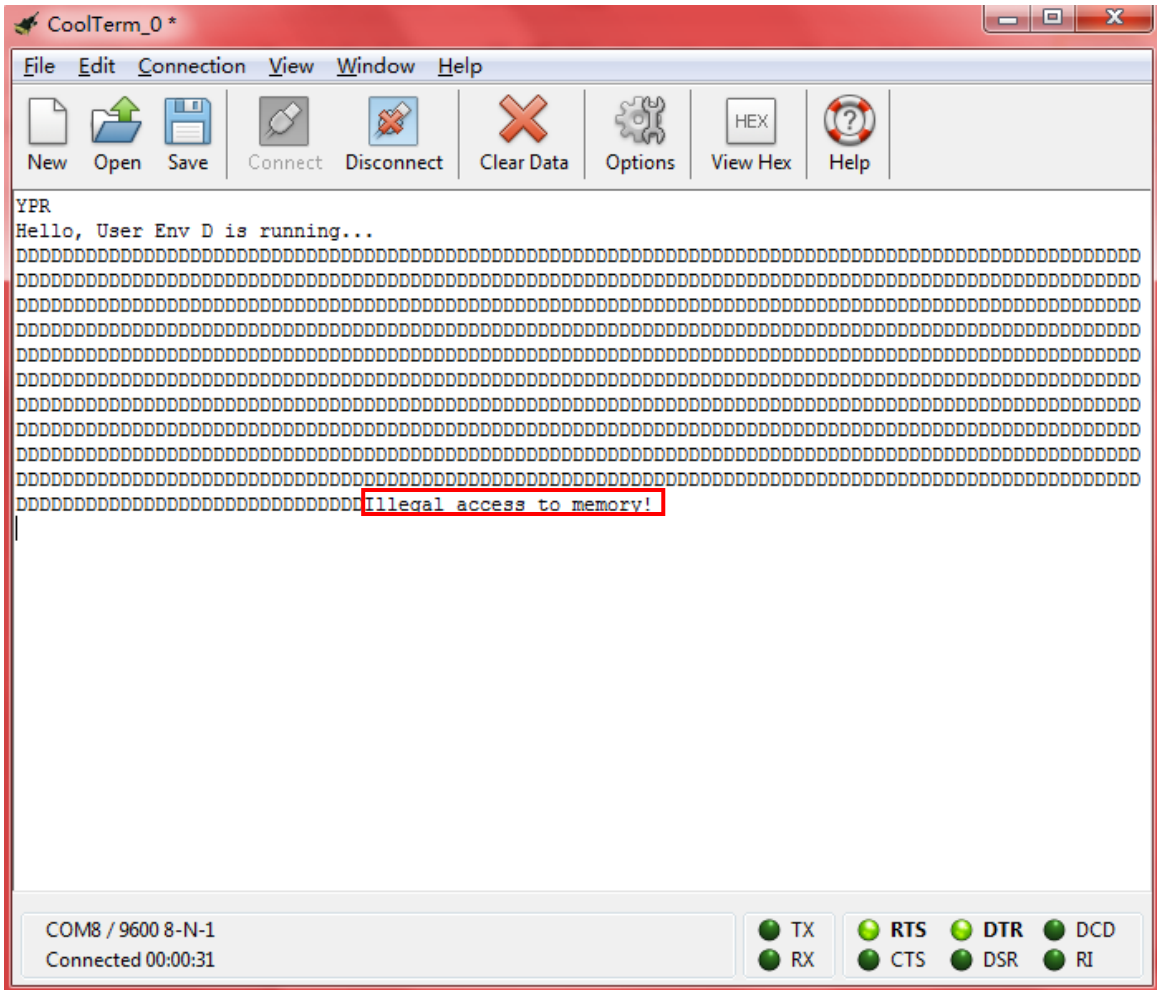


图 4.10 串口显示结果

由于最终剩余时间不充裕，设计中将 CPU 封在了内核级别且不能调节。遇到非法地址时 CPU 须抛出异常，设计在操作系统中添加了非法地址检测代码。此处操作系统做的错误处理工作很简单，打印出错误并进入死循环。

```
void    MMU_Error(){
    Print("Illegal access to memory!\n");//打印字符串
    while (1) ;//进入死循环
}
```

用户程序中被强制了一个非法地址的指针，强制访问这个非法地址时，从串口输出的结果可以知道此错误是否被正确的捕获。附录二中的第三段代码列出了用户程序，这里对用户程序中非法地址访问的原理做一解释。

用户程序中定义了一个指针*p 并强制其地址为一个高 2G 的非用户访问地址 0xF000_0000，然后再强制给这个指针赋值*p=12345678 模拟一次非法访问操作。

操作系统在检测到异常之后，会调用 MMU_Error()处理这个异常，这里通过串口观察预期的结果。图 4.10 中可以看出，操作系统捕获了这个异常，并正确调用了异常处理程序，打印出了字符串“Illegal access to memory!”，然后进入了死循环。整个设计符合预期的要求。

## 4.4 本章小结

本章测试工作分为两部分进行，第一部分的单个模块的功能测试分别进行了 TLB、MMU、CP0 3 个单个的模块的测试工作。经测试这 3 个模块都符合预想的设计，能够正常的工作并满足设计需求。第二部分是 TLB、MMU、CP0 等几个模块接入 CPU 形成一个整体之后的测试，这部分主要使用了逻辑分析仪，将断点设置到关键代码位置，验证了在 TLB 缺失之后，CPU 能够进入正确的异常处理入口，并能够正确执行异常处理指令，完成 TLB 的重填工作，重填之后的 MMU 能够正常并稳定的工作。

由于时间关系，没有在硬件上实现 MMU 对于非法访问的捕获，但是 MMU 模块设计之初预留了接口，稍微修改设计便能够实现此功能。操作系统中添加了相应的非法访问的捕获和处理，实现了硬件上的功能缺失。

# 结论

## 工作总结

本文分析了 32 位 MIPS 处理器的体系结构和内存组织管理机构，结合设计时实际所用的硬件环境以及实现复杂度，最终采用了层次化的设计思想来设计 MMU。

本文首先编写了 TLB 的硬件逻辑代码，设计了 TLB 的工作逻辑，然后在 MMU 中声明了 8 个 TLB 页面并详细讨论了 MMU 的工作逻辑；为了减少 MMU 和 CP0 的通讯时间，在 MMU 和 CP0 之间设置多个数据交换口，为了配合 TLBR、TLBWI、TLBWR 和 TLBP 四条指令的实现，在 CP0 中新加入了 9 个寄存器，并编写了它们的读写逻辑。

完成了模块设计之后，为了让 MMU 匹配整个 CPU 的工作时序修改了 CU 的设计，在 CU 中加入 MMU 控制逻辑，让地址翻译时的时序与整个系统的时序相匹配，并且在 MMU 遇到错误时能够正确的报出错误跳入正确的异常入口。为了配合 TLB 缺失的异常处理，操作系统中设计了相应的处理代码，让 TLB 缺失异常能够自动化处理。

最后的工作是测试与验证。测试工作主要分两部分来进行，第一部分是单个模块的功能以及几个简单模块的集成测试，将局部错误减到了最少甚至消除。第二部分是几个模块拼接成为大的模块的局部测试以及 MMU 接入 CPU 之后的整体测试。此部分综合运用 Quartus II 7.2、ModelSim 的和逻辑分析仪三个强大的工具，将错误减到了最少。

通过这一系列的仿真以及测试工作证明了整个系统的工作符合最初的设计要求，能够达到预期的目标。

## 下一步工作

本文通过硬件编码和更改操作系统，基本实现了将 MMU 添加到 MIPS CPU 的设计目标，现版本的 CPU 能够支持虚拟地址到实地址的翻译，并能够对越界访问进行报错，但是本设计还有一定的缺点，若时间充足，我觉得需要做以下改进和完善：

1) 加入 CPU 内核运行级别的代码。在 CP0 中的 SR 寄存器应该加入 CPU 运行级别的管理，并给操作系统留出接口，这样操作系统能够更改 CPU 运行级别。CPU 运行级别自动化管理，更能够体现 MIPS 体系结构的特点。
2) 探索一个 MMU 中包含多少个 TLB 页面才能让 CPU 的地址翻译工作达到最高效率。现版本 MMU 中包含有 8 个 TLB 页面，当初设计时候考虑到了设计复杂性以及成本问题，并没有考虑效率的问题。

   探索 TLB 页面个数与翻译效率的关系是一个非常有趣的课题，我们可以从 4 个页面开始探索，逐渐将页面个数增加到 8 个，16 个，32 个，64 个，让其运行同一段程序并统计其运行时间并绘制 TLB 数量与运行时间折线图，运用统计学方法找出理论最佳点，并通过理论最佳点规划出实际最佳点。
3) 现版本 CPU 的 MMU 翻译要两个时钟周期。由于使用同步时钟 CPU 在上升沿检测信号，因此 CU 发出请求信号 MMU 得到响应需要一个时钟周期，MMU 发出应答信号到 CU，CU 检测到应答信号还需要一个时钟周期，因此翻译需要两个时钟周期。下一步工作是争取将其变为一个时钟周期，将现在 MMU 的翻译周期由两个时钟周期变为一个时钟周期，则翻译效率会提升 100%，这对于 CPU 来说是一个质的飞跃。这个工作的难度不是非常大，但需要重新设计 TLB，由于时间的原因我没有完成此工作。若以后时间充足，这是一项令人兴奋的工作。
4) 与彭德生同学所做的流水线和 cache 合并，形成一个完成的设计。MIPS CPU 的存储系统由片内的寄存器和 cache，再到虚存构成一个有机的整体，彭德生同学所做的 cache 将现版本 CPU 的速度大为提升，配合流水线，使得 CPU 从取址到执行的速度，都有了质的飞跃。MMU 为 CPU 运行用户程序提供了安全保障，这如果能够两者结合在一起， COCOA 的系统完备性将有飞跃性的提升。但是这项工作的难度并不小，由于流水线处理器和非流水线处理器的控制逻辑

差距很大，加之 cache 有写回操作，TLB 其实也是页面的缓存，也存在脏数据这一问题，这就要求 MMU 和 cache 两者必须配合严密，否则脏数据问题会困扰合并后的设计，使得整个系统无法保证正确运行。

# 致谢

在本论文即将完成之际，我想把心里最诚挚的感谢送给所有关心我帮助我的人们。

首先，我要感谢我的导师高小鹏副教授。高老师是一位知识渊博的学者，他作风严谨，一丝不苟，在教学方面更是提倡我们自己独立思考，引导我们如何去学习，令我们受益匪浅。加入嵌入式系统实验室后，我深深地感受到一种科研的氛围。在最初选题的时候，高老师与我做了几次深入的交流，让我不仅对毕业设计课题有了进一步理解，还对高老师渊博的知识以及严谨的治学风格敬佩不已。在毕业设计期间，高老师也给予我很多指导和帮助，使我能够约束自己努力完成毕业设计项目。再次感谢高老师。

同样也感谢彭德生同学。彭德生同学在大三暑假时候参加了与 COCOA 项目关系密切的一个实习，提前接触了 MIPS 方面的许多工具和 COCOA 之前版本的设计，彭同学在整个毕业设计过程中给了我很大的帮助。毕业设计开始阶段，彭同学耐心给我讲解 CPU 各个逻辑部件以及他们之间的互联关系，还画出了 CPU 片内互联图，让我清楚了各个器件之间的连接关系。在收尾阶段，我的设计无法完成 MMU 越界访问报错功能时，彭同学提出了从操作系统入手的方法，并给我阐述了具体的思想。彭同学总是耐心地与我做讨论，使我能够在很多关键技术和难点的地方得到突破口。没有彭同学的帮助，就很有可能没有这篇论文的顺利完成，在此向他表示最真挚的谢意 。

在实验室工作和学习的日子是让人难以忘怀的，老师们兢兢业业的工作作风是我们的学习榜样，同学们的珍贵友情是本人毕生最大的财富。在毕业设计项目的开发过程中，他们也多次提出宝贵意见，令我在很多地方茅舍顿开。在此，一并送上我对他们的祝福。

感谢我院的各位老师以及徐毅、张萌两位辅导员。感谢他们在大学四年里对所有学生的无私奉献，感谢他们对我的帮助与指导，使我顺利完成了大学四年的学业，并顺利的拿到了自己最喜欢的工作的 offer。特别感谢章张锴同学在百忙中为我的毕业设计论文模板做出的调整以及为我的毕业设计论文格式做出的指导。

在这里，我还要把最真诚的感谢献给我的父母、亲人，他们在生活上给予了莫大的关怀，在精神上给予了我支持与鼓励。感谢 228 宿舍的舍友们，几年来 228 始终是 17 号宿舍楼的文明宿舍，舍友们都很团结，劲能够往一处使，大家齐心协力做了很多事情，这让我的大学生活也少了很多遗憾。感谢 380605 这个非常有凝聚力的集体，每次集体

活动大家的出勤率都非常高，有这么多能够说到一起玩到一起的同学。正是带着他们对我的期望，我才能勤勉自励，才能够坚持不懈地完成我的工作。今后，我也依然需要他们对我的关怀与支持，我会用我的表现来回报他们。

最后，衷心感谢为评阅本文而付出辛勤劳动的各位老师和评委！

# 参考文献

[1] Dominic Sweetman 著. 李鹏 等译.MIPS 体系结构透视[M]. 机械工业出版社, 2008.

[2] 孙庆平. 现场可编程逻辑门阵列（FPGA）技术及其应用[J]. 数字通信,1996.01.

[3] 高小鹏.FPGA 设计规范-Cyclone3 [Z]. 2011.

[4] David A. Patterson, John L. Hennessy 著 郑纬民等译. 计算机组成与设计[M]. 机械工业出版社, 2008.

[5] 胡仁霖，胡传国，卢克盛等. RISC 技术参考大全[M]. 电子工业出版社，1992 年.

[6] 王爱英. 计算机组成和结构[M]. 清华大学出版社, 1999 年.

[7] 卢仕听，尤凯迪，韩军，曾晓洋等. MIPS 内存管理单元的设计与实现[J]. 计算机工程. 2010.10.

[8] 邓子君. 32 位 MIPS 微处理器内存管理单元的设计和验证[D], 西安：西安电子科技大学,2008.

[9] Willian Stallings. Computer Organization and Architecture-Design for Performance [M]. 清华大学出版社. Feb, 1997.

[10] Silicore Technologies, Inc. AHB-Lite Overview [Z]. 2003.

[11] Silieore Technologies, Inc. AMBA Specification Rev 2.0 [Z]. 2003.

[12] S.Furber. ARM SOC 体系结构 [M]. 北京航空航天大学出版社. 2002 年 12 月.

[13] 李亚民. 计算机原理与设计——Verilog HDL 版[M]. 清华大学出版社. 2011.06.

[14] Ulrich Golze. VLSI Chip Design with the Hardware Description Language—Verilog [M]. 北京航空航天大学出版社. 2005.01.

[15] 夏宇闻. Verilog 数字系统设计教程（第二版）[M], 北京航空航天大学出版社. 2008.06.

[16] 吴厚航. 深入浅出玩转 FPGA [M], 北京航空航天大学出版社.2010.05.

[17] 王诚，吴继华，范丽珍，薛宁. Altera FPGA/CPLD 设计[M], 人民邮电出版社.2005.07.

[18] 乔庐峰，王志功. Verilog HDL 数字系统设计与验证[M], 电子工业出版社. 2009.04.

[19] MIPS Technologies, Inc. MIPS32 M4Kc Processor Core Software User’s Manual [Z], Revision 1.0, August 2002.

[20] MIPS Technologies, Inc. MIPS32 Architecture for Programmers [Z]. Revision 1.0, August

2002.

[21]MIPS Technologies, Inc. MIPS32 4Kc Processor Core Datasheet [Z]. Revision1.0, August 2002.

[22]MIPS Technologies, Inc. MIPS32 4K [Z]. Revision1.0, August 2002.

[23]Processor Core Family Integrator’s Manual [Z]. Revision 1.0, August 2002.

[24]MIPS Technologies, Inc. MIPS 指令集 [Z]. August 2002.

[25]于斌，米秀杰. ModelSim 电子系统分析及仿真[M]，电子工业出版社. 2011.12.

# 附录

## 附录 A 操作系统中 MMU 缺页处理的代码

```
.section .text.tlb_entry
.set noreorder
.global tlb_entry
tlb_entry:
mfc0    k1, CP0_CONTEXT     ；载入错误地址的页表所在地址
lw      k0, 0(k1)           ；载入页表项
lw      k1, 8(k1)           ；载入另一个相邻页表项
mtc0    k0,CP0_ENTRYLO0     ；将载入的页表项送入相应寄存器
mtc0    k1,CP0_ENTRYLO1     ；将载入的页表项送入相应寄存器
nop
tlbwr                       ；随机写入 MMU 中的 TLB 页面
nop
eret                        ；返回到异常发生处
nop
```

# 附录 B 测试用例代码

在启动时，操作系统中初始化了四张页表，具体值如下所示：

```
unsigned int page_table[4] =
{0x90400006,0x90401006,0x90402006,0x90403006};
```

启动用户程序之前，需要对于 CP0 进行必要的初始化工作，这是操作系统在切换进程时候的必要操作，具体代码如下：

```
li      k0,0
mtc0 k0, CP0_INDEX
mtc0 k0, CP0_PAGEMASK
li      k0, 0x40000000
mtc0 k0, CP0_ENTRYHI

jal    get_pagetable_addr
move      k0,v0
mtc0 k0, CP0_CONTEXT

jal    user_D_prog_init
```

下面列出的代码是用于测试的用户程序代码，包含非法访问的指针。

```
int main()
{
        int i=0,n=10000;
        unsigned int *p;
        p=(unsigned int *)(0xF0000000);

        printstr("Hello, User Env D is running...\n");
        while(1)
        {
                n=++i;
```

```
            printcharc('D');
          if (i==1000)
            {
                *p=12345678;
            }
    }
    printstr("Bye, User Env D end running...\n");
    return 0;
}
```