浙 江 大 學

本科生实验报告

课程名称 计算机体系结构

姓　　名 吴同

学　　院 计算机科学与技术学院

专　　业 计算机科学与技术

学　　号 **3170104848**

指导教师 陈文智

浙江大学实验报告

专业：计算机科学与技术
姓名：吴同
学号：3170104848
日期：2020 年 1 月 5 日

课程名称：计算机体系结构　指导老师：陈文智　电子邮件：wutongcs@zju.edu.cn
实验名称：Cache 设计　实验类型：综合型　同组同学：徐欣苑

一、 实验目的和要求

1. 实验目的

- 理解 Cache line
- 理解 CMU 的原理和 CMU 的状态机
- 掌握 CMU 的设计方法
- 掌握 Cache line 的设计方法
- 掌握 Cache line 的验证方法

2. 实验要求

- 设计 Cache line 和 CMU
- 验证 Cache line 和 CMU
- 观察仿真波形

二、 实验内容和原理

1. Cache

本实验设计的 Cache 采用直连、写回和写分配的策略，即主存中的一个块只能映射到 Cache 的某一特定块中去；一个 Cache line 只有在被选中替换回主存时，才会将脏（dirty）数据写回主存；所要写的位置不在 Cache 中时，将要写的内容直接写回主存。

在本实验设计的 Cache 中，每个字包括 4 个字节（32 位），每一行有 4 个字，各需要 2 位来表示偏移量；每个标签用 22 位来表示；地址线为 32 位。则行索引为 6 位，一共有 64 个行。

Cache 的结构如下：一共有 64 个行，每行有一个 vaild 位，一个 dirty 位，一个 22 位宽的 tag，四个 32 位的数据。

V	D	tag	Data

图 1: Cache 结构图

在每一个 32 位的地址中，第 10 到第 31 位为 tag，第 2 到第 9 位为 index，第 0 到第 1 位为偏移量。

图 2: 地址结构图

2. Cache 管理模块（CMU）

Cache 管理模块为 CPU 提供存储器管理的接口。当 CPU 发出访问内存的请求时，CMU 首先根据是读或写。如果是读，当 Cache 命中时，直接返回数据；当 Cache 未命中时，选择一个 Cache line，如果该行是脏的，则先将其写回主存。接下来将数据从主存加载到 Cache 中，将其 dirty 位清零，返回数据。如果 CPU 发出的是写存储器的请求，若 Cache 命中，则将其写入 Cache，将其标记为 dirty；若 Cache 失配，则先选一个 Cache line，并将脏数据写回，再写入 Cache。

三、 实验过程记录

1. 设计 Cache

根据实验框架，Cache 的输入输出接口如下：

clk	时钟
rst	复位
addr	所存/取数据的地址
store	存储信号，设置 valid 位为 1，dirty 位为 0
edit	修改信号，设置 dirty 位为 1
invalid	失效信号，设置 valid 位为 0
din	送入的数据
hit	输出信号，表示 Cache 是否命中
dout	输出的数据
valid	输出信号，表示该位置是否有效
dirty	输出信号，表示该数据是否为脏数据
tag	输出当前地址的 tag

Cache 的内部实现逻辑为：

- 根据 index 值选取当前的行，如果该位为有效，且 tag 与输入地址的 tag 相同，则为命中。根据 index 值，输出对应位置的数据。
- 如果输入 store 信号为高电平，或输入 edit 信号为高电平且命中，则将输入数据写入对应位置。
- 如果输入 invalid 信号为真，则将输入地址的 index 所在行的 valid 和 dirty 位清零；否则，如果输入 store 信号为真，则将输入地址的 index 所在行的 valid 位置 1，dirty 位信号清零，tag 设为当前地址的 tag；否则，如果输入 edit 信号为真，则将输入地址的 index 所在行的 valid 位置 1，tag 设为当前地址的 tag。

2. 设计 CMU

根据实验框架，CMU 的输入输出接口如下：

clk	时钟
rst	复位
addr_rw	所存/取数据的地址
en_r	读取数据的控制信号
data_r	从 Cache 中读到的数据
en_w	写入数据的控制信号
data_w	写入 Cache 的数据
stall	CMU 输出的停顿信号
mem_cs_o	访问主存的请求信号
mem_we_o	访问主存的写入信号
mem_addr_o	访问主存的地址
mem_data_i	从主存获取的数据
mem_data_i	向主存输出的数据
mem_ack_i	主存 ack 信号

CMU 有五个状态：S_IDLE 空闲状态，S_BACK 写回状态，S_BACK_WAIT 写回后等待状态，S_FILL 读取状态，S_FILL_WAIT 读取后等待状态。状态转移逻辑如下：

- S_IDLE 状态下，若收到 CPU 的读写请求，若 Cache 命中，保持为 S_IDLE 状态；若需要进行脏数据的替换，则进入 S_BACK 状态，否则进入 S_FILL 状态。
- S_BACK 状态下，在主存发回 ACK 信号后，每个周期将计数器加 1。如果计数器的值显示这一行已经全部写回，则进入 S_BACK_WAIT 状态，否则保持 S_BACK 状态。
- S_BACK_WAIT 状态下，将计数器清零，回到 S_IDLE 状态。
- S_FILL 状态下，在主存发回 ACK 信号后，每个周期将计数器加 1。如果计数器的值显示这一行已经全部读入 Cache，则进入 S_FILL_WAIT 状态，否则保持 S_FILL 状态。
- S_FILL_WAIT 状态下，将计数器清零，回到 S_IDLE 状态。

Cache 的控制逻辑如下：

- S_IDLE 状态下，送入 Cache 的地址为 CPU 请求的地址，传入 Cache 的 edit 信号为 CPU 的写信号，送入 Cache 的数据为 CPU 传入的数据。
- S_BACK, S_BACK_WAIT 状态下，送入 Cache 的地址高 28 位为 CPU 请求的地址的高 28 位，第 2 和 3 位是计数器的值，最低 2 位是 0。
- S_FILL, S_FILL_WAIT 状态下，送入 Cache 的地址高 28 位为 CPU 请求的地址的高 28 位，第 2 和 3 位是计数器的值，最低 2 位是 0。送入 Cache 的数据是从内存获取的数据，控制 Cache 的 store 信号是主存发回的 ACK 信号。

主存的控制逻辑如下：

- S_IDLE, S_BACK_WAIT, S_FILL_WAIT 状态下，主存请求信号，写入信号，地址值全部为 0。
- S_BACK 状态下，主存请求信号和写入信号均为 1，主存地址由 Cache line 的 tag，CPU 请求的地址的 index，计数器组成高 30 位，低 2 位为 0。

- S_FILL 状态下，主存请求信号为 1，写入信号均为 0，主存地址由 CPU 请求的地址的高 28 位，计数器，两位 0 组成。

3. 综合顶层模块

根据实验框架的顶层代码，综合整个工程，生成的 RTL 图如下：

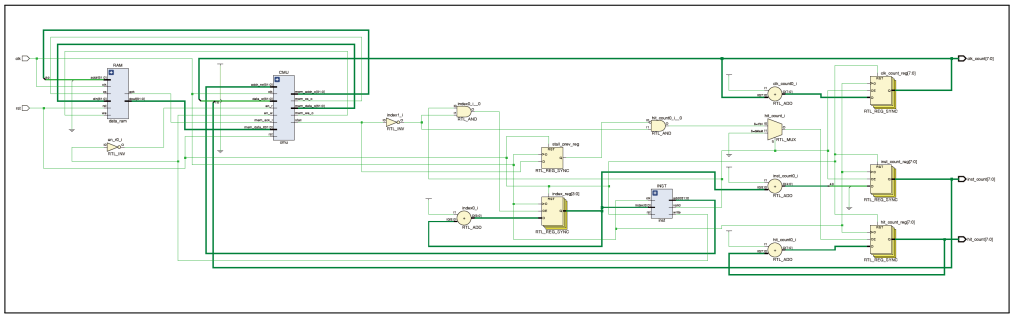

图 3: 本实验的 RTL 电路图

四、 实验结果分析

1. 仿真测试 Cache

使用实验框架所给的测试程序，对 Cache 模块进行仿真测试，分别对 0x0，0x4，0x8，0x1c 地址进行存储，然后读取 0xb4 地址，再修改 0x8 地址，再读取 0x0 地址。测试结果如下：

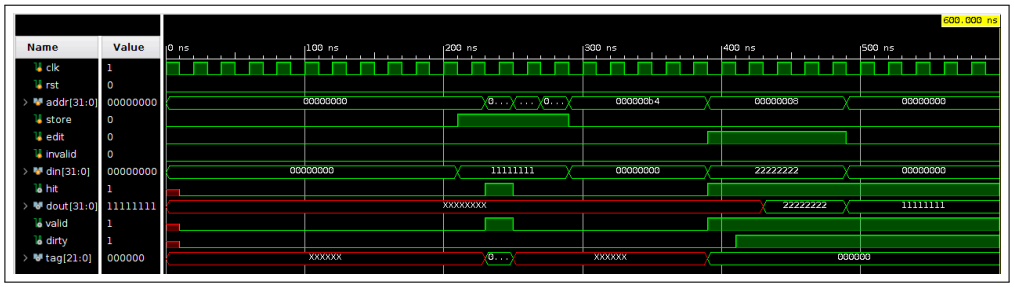

图 4: Cache 模块仿真测试图

2. 仿真测试 CMU

对 Top 模块进行仿真，测试 CMU。完成 inst.v 文件中对存储的访问请求一共花费 92 个时钟周期，由 3 次 Cache 命中。

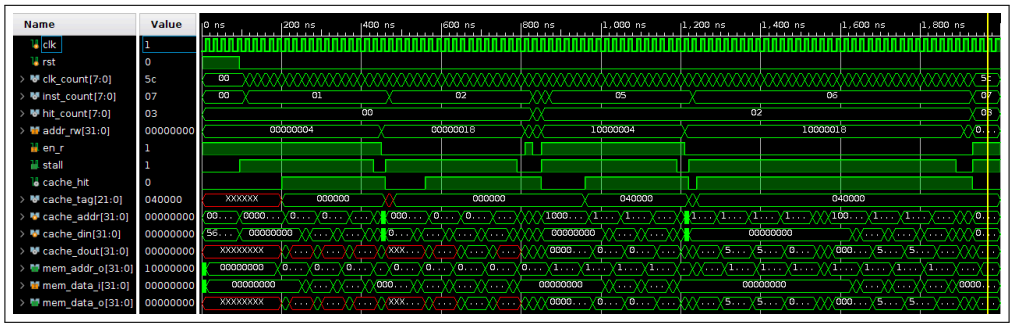

图 5: Cache 模块仿真测试图

五、 讨论与心得

本次实验的内容是设计并验证 Cache。通过这次实验，我对 Cache 的原理有了更加深入的理解，掌握了 Cache 管理模块（CMU）的状态转换过程。Cache 的设计较为复杂，但由于实验指导的 PPT 上已经给出了代码的框架，只需要添加 CMU 中状态转换的逻辑，所以实际做的时候并不是十分困难。

在本次实验中，我使用运行在 docker 中的 Vivado 2019.1 完成。相比于在虚拟机中搭建的开发环境，docker 使用起来更加便捷，对系统资源的占用更少。但是 docker 中的 GUI 界面由 X11 转发，显示的效果不是很好。