

**本科实验报告**

|  |  |
| --- | --- |
| 课程名称： | 计算机组成 |
| 姓 名： | 应周骏 |
| 学 院： | 计算机科学与技术学院 |
| 系： | 计算机科学与技术系 |
| 专 业： | 计算机科学与技术 |
| 学 号： | 3200103894 |
| 指导教师： | 马德 |

2022 年 6月 20 日

**浙江大学实验报告**

课程名称： 计算机组成 实验类型： 综合

实验项目名称： Cache设计与实现

学生姓名： 应周骏 专业：计算机科学与技术 学号：3200103894

同组学生姓名： 无 指导老师： 马德

实验地点： 东4-509实验日期：2022 年 6 月 5 日

**一、实验目的和要求**

1.理解高速缓存的基本概念和作用；

2.掌握缓存的组织结构和映射方式、替换策略；

3.理解缓存的工作原理、命中率和一致性问题；

4.设计缓存的控制器模块和存储模块；

**二、实验内容和原理**

**目标：**

熟悉数据缓存的工作原理，了解存储单元的组织结构，掌握缓存控制器的设计方法，设计并测试两路组关联Cache。

**内容：**

数据缓存模块的设计（two-way set associate cache）并进行数据缓存模块的仿真验证。

**原理：**

1. Cache的基本概念

Cache又叫高速缓冲存储器，位于CPU与内存之间，是一种特殊的存储器子系统。

目前比较常见的是两极cache结构，即cache系统由一级高速缓存L1 cache和二级高速缓存L2 cache组成，L1 cache通常又分为数据cache（D-Cache）和指令cache（I-Cache），它们分别用来存放数据和执行这些数据的指令。



图2.1 cache示意图

2. Cache的作用

Cache的作用就是为了提高CPU对存储器的访问速度。

电脑的内存是以系统总线的时钟频率工作的，这个频率通常也就是CPU的外频。但是，CPU的工作频率(主频)是外频与倍频因子的乘积。因此，内存的工作频率就远低于CPU的工作频率了。导致的直接结果是：CPU在执行完一条指令后，常常需要“等待”一些时间才能再次访问内存，极大降了CPU工作效率。

3. Cache的工作原理

CPU运行程序是一条指令一条指令地执行的，而且指令地址往往是连续的，即CPU在访问内存时，在较短的一段时间内往往集中于某个局部，这时候可能会碰到一些需要反复调用的子程序。系统在工作时，可把这些活跃的子程序存入比主存快得多的cache 中。

CPU在访问内存时，首先判断所要访问的内容是否在cache中，如果在，则称为命中（hit），此时CPU直接从cache中调用该内容；否则称为未命中（miss）， CPU会通过cache对主存中的相应内容进行操作。

4. Cache设计实现

·Cache控制采用FSM实现。

·实现标签比较，检测读写命中与否。

·实现Cache的数据更新，当miss后采用LRU完成数据替换。

·实现Mem的数据更新，当write miss后采用Write Back，被改写的数据块在被替换出cache时写回到Mem。同时实现Write Allocate。

**三、实验过程和数据记录**

**1. 工程文件建立**

新建工程文件，命名为“OxExp06\_cache”。

**2. Cache控制部分FSM设计**

新建Verilog文件 “cache.v”，输入以下代码。

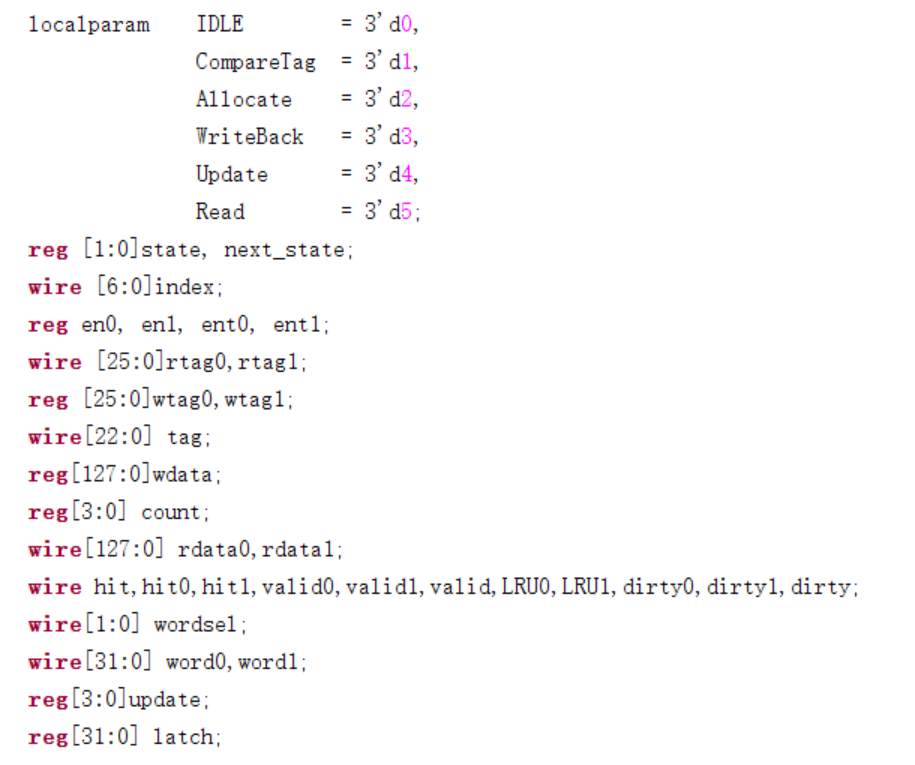
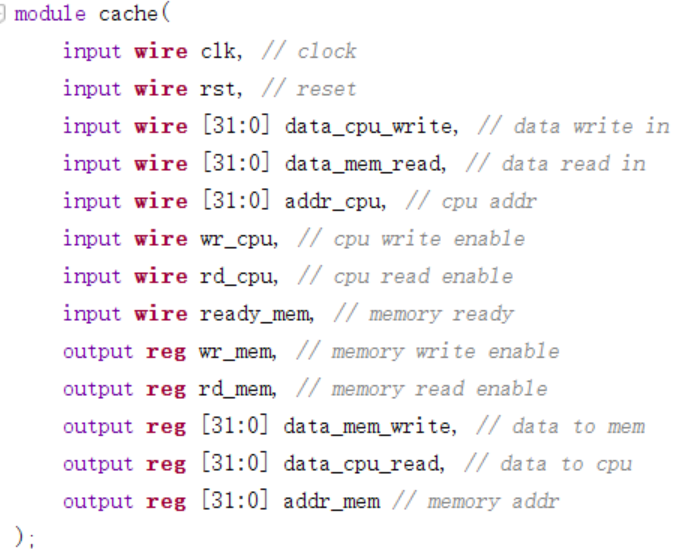


图3.2.1 接口和临时变量

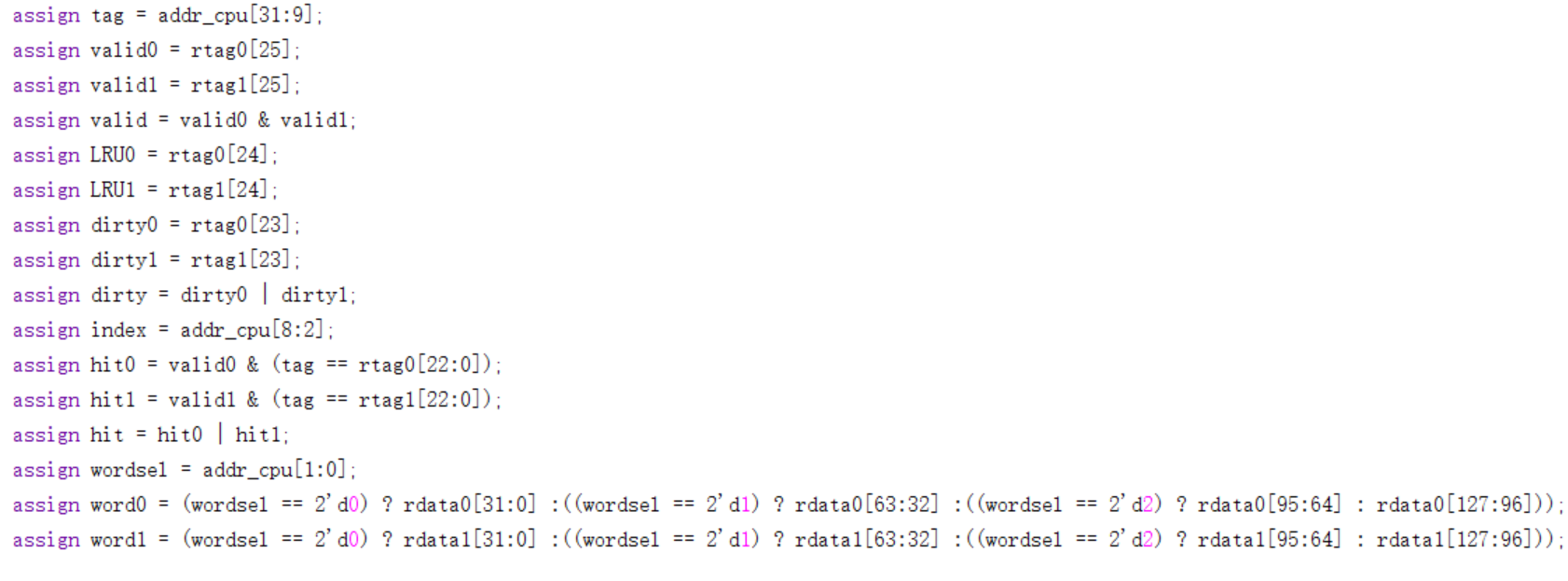


图3.2.2 寄存器赋值

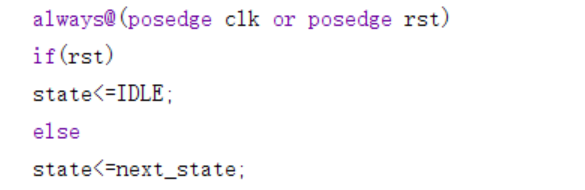


图3.2.3 时序逻辑状态转换

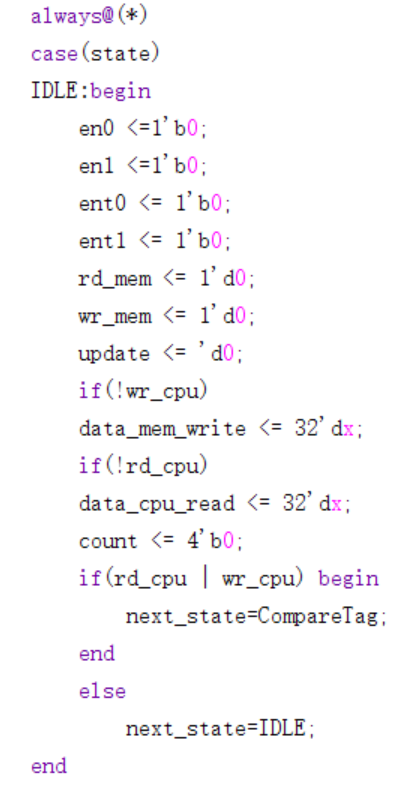
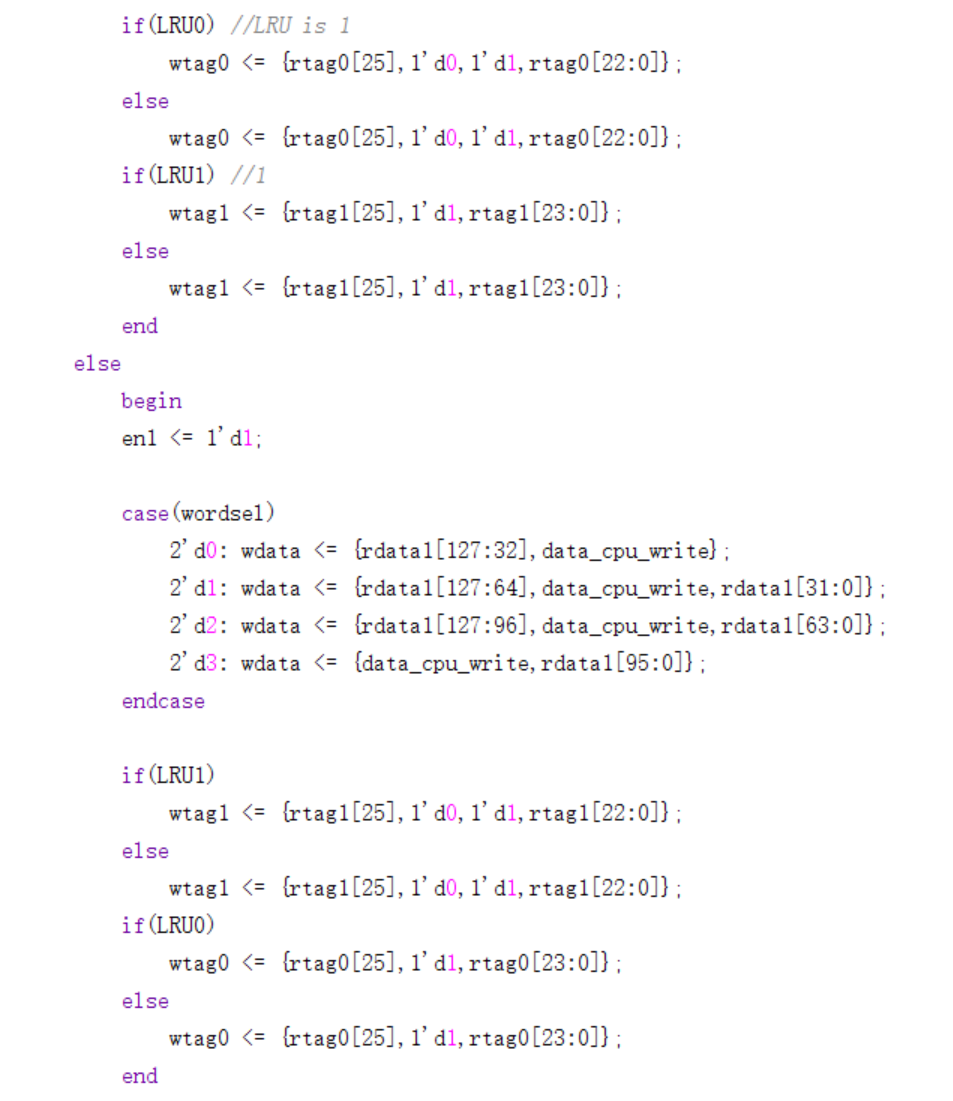
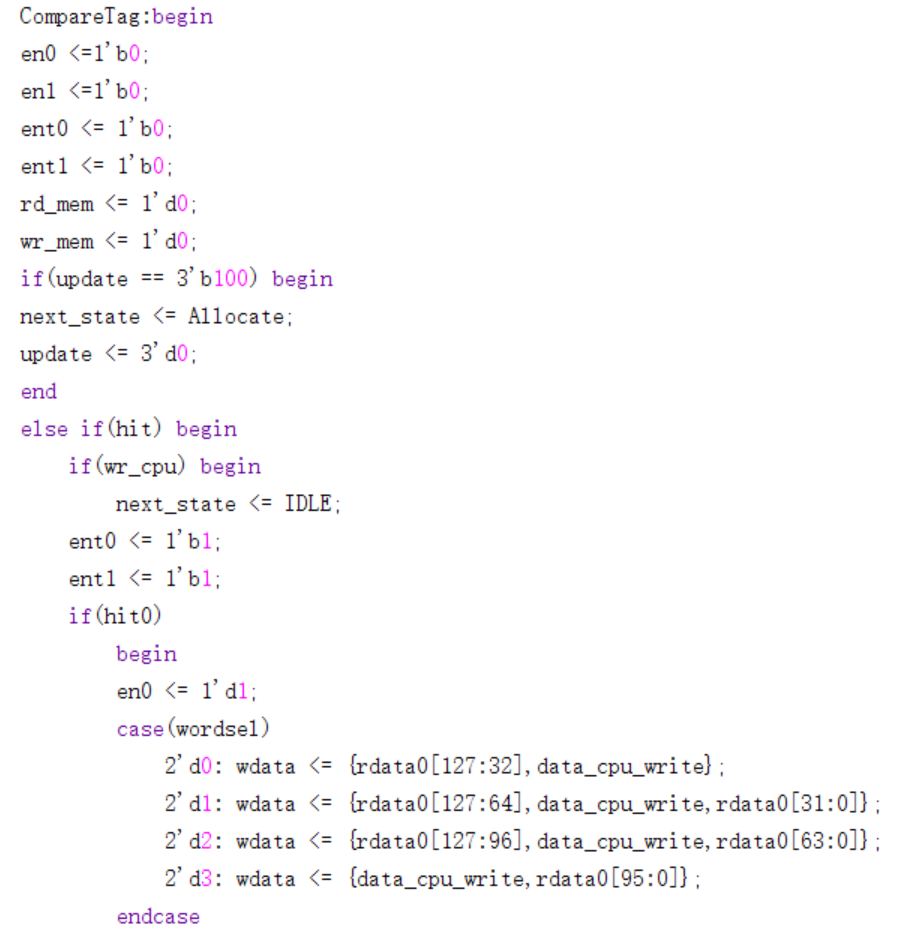


图3.2.4 初始状态IDLE



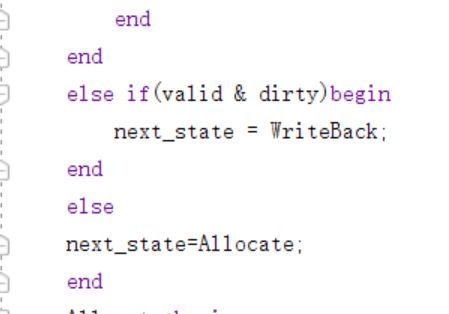
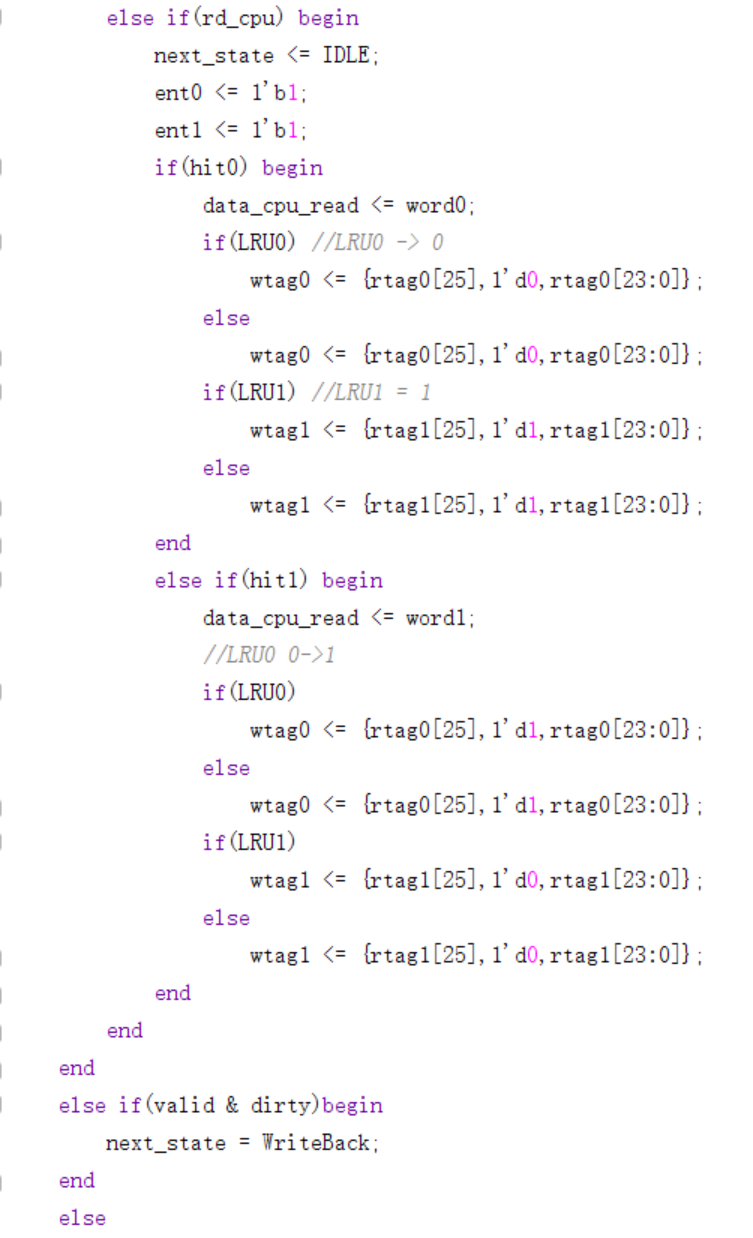


图3.2.5 CompareTag状态

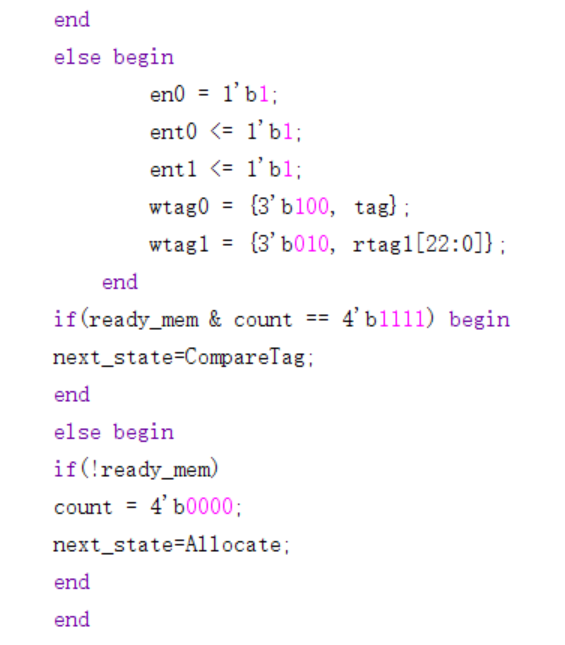
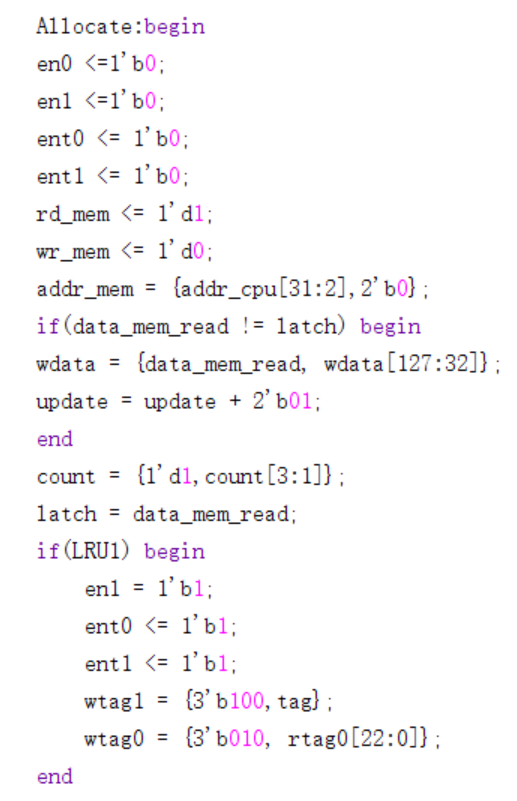


图3.2.6 Allocate状态

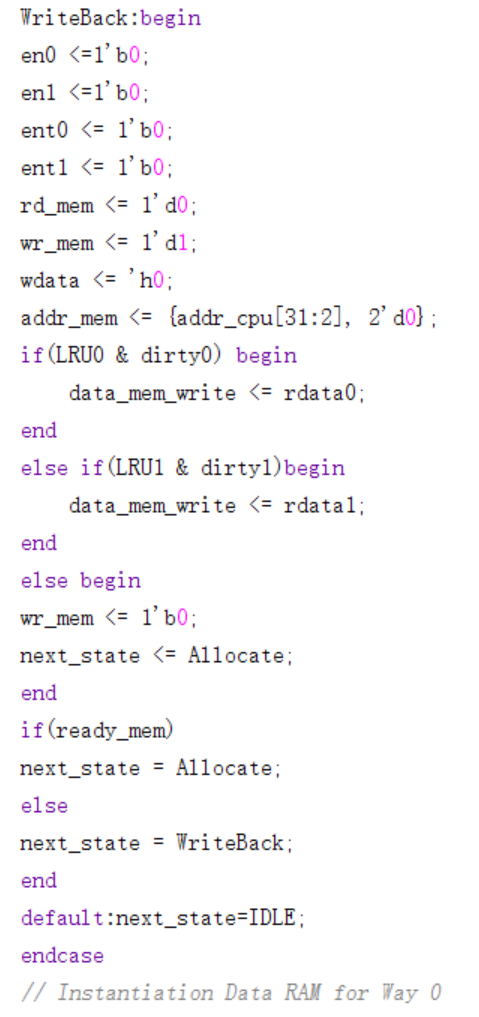


图3.2.7 WriteBack状态

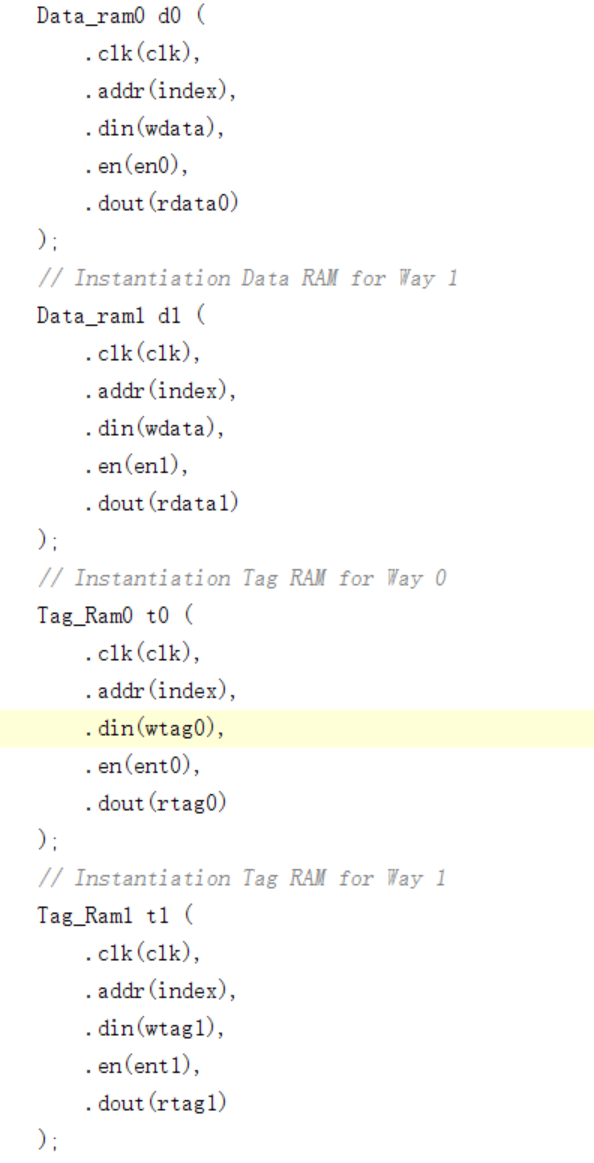


图3.2.8 存储器接口

**3. 存储器模块设计**

新建 “Data\_ram0.v” “Data\_ram1.v” “Tag\_ram0.v” “Tag\_ram1.v” ，实现对应的存储模块。

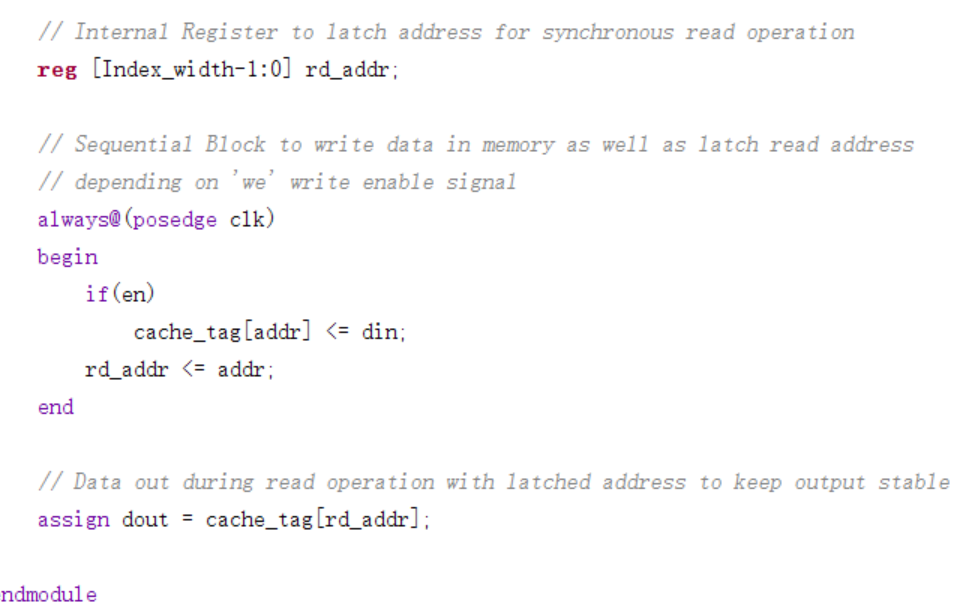
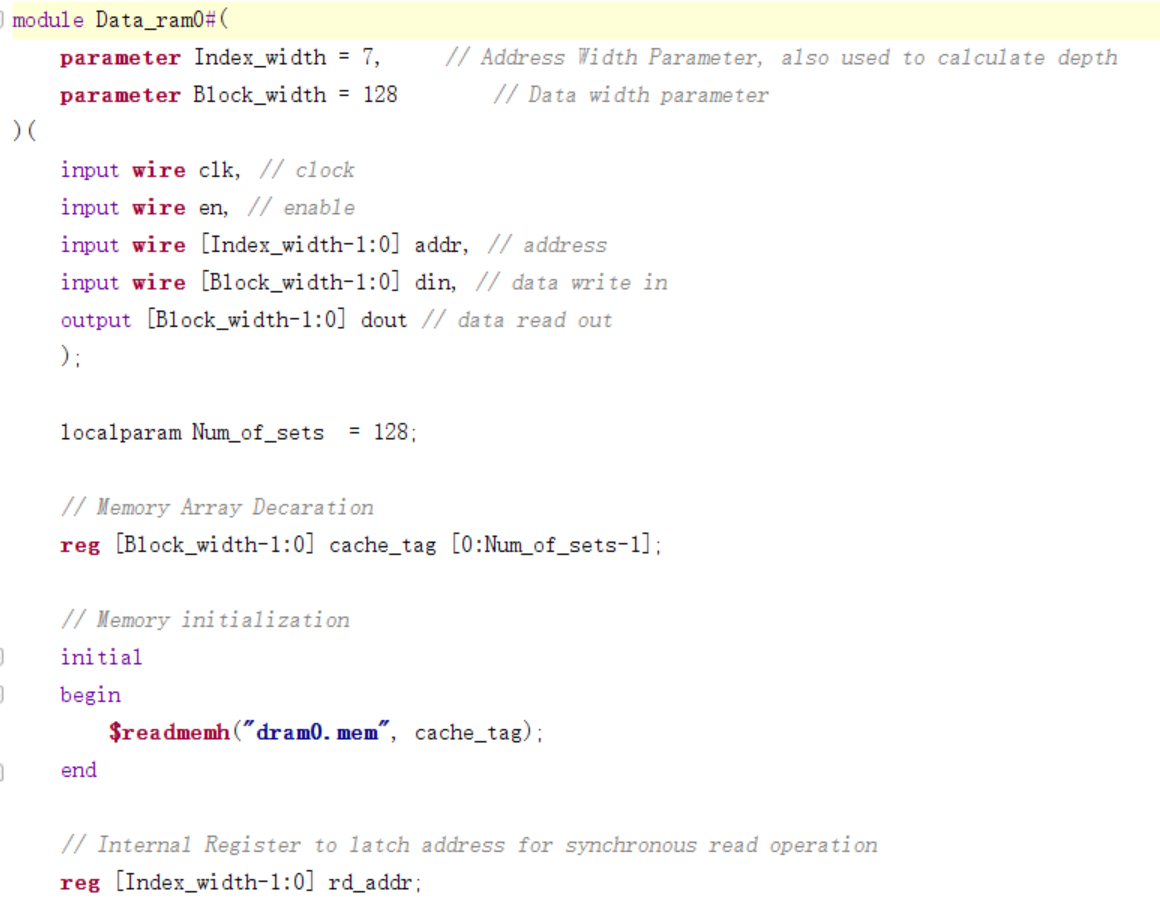


图3.3.1 Data\_ram模块

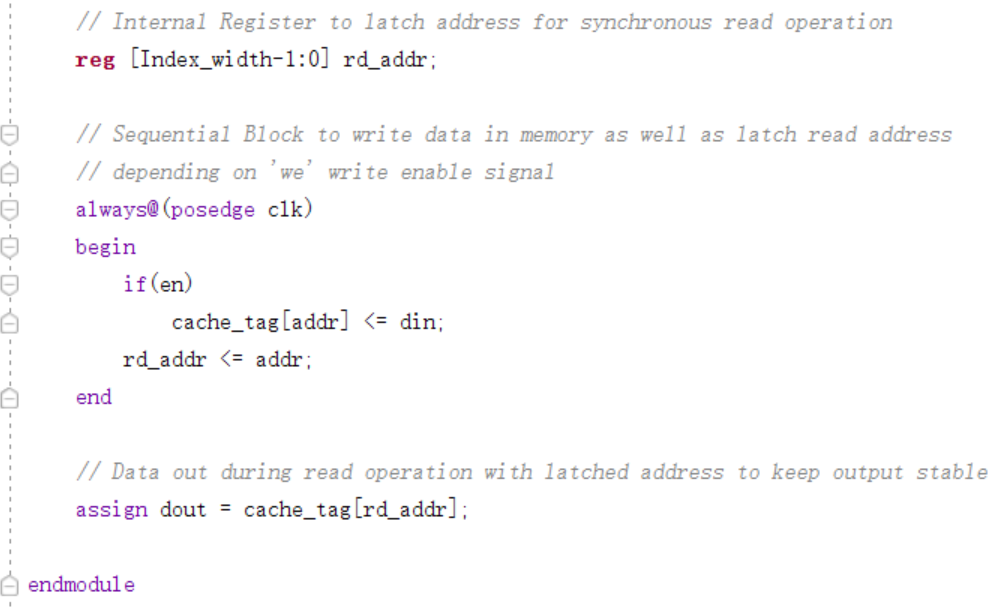
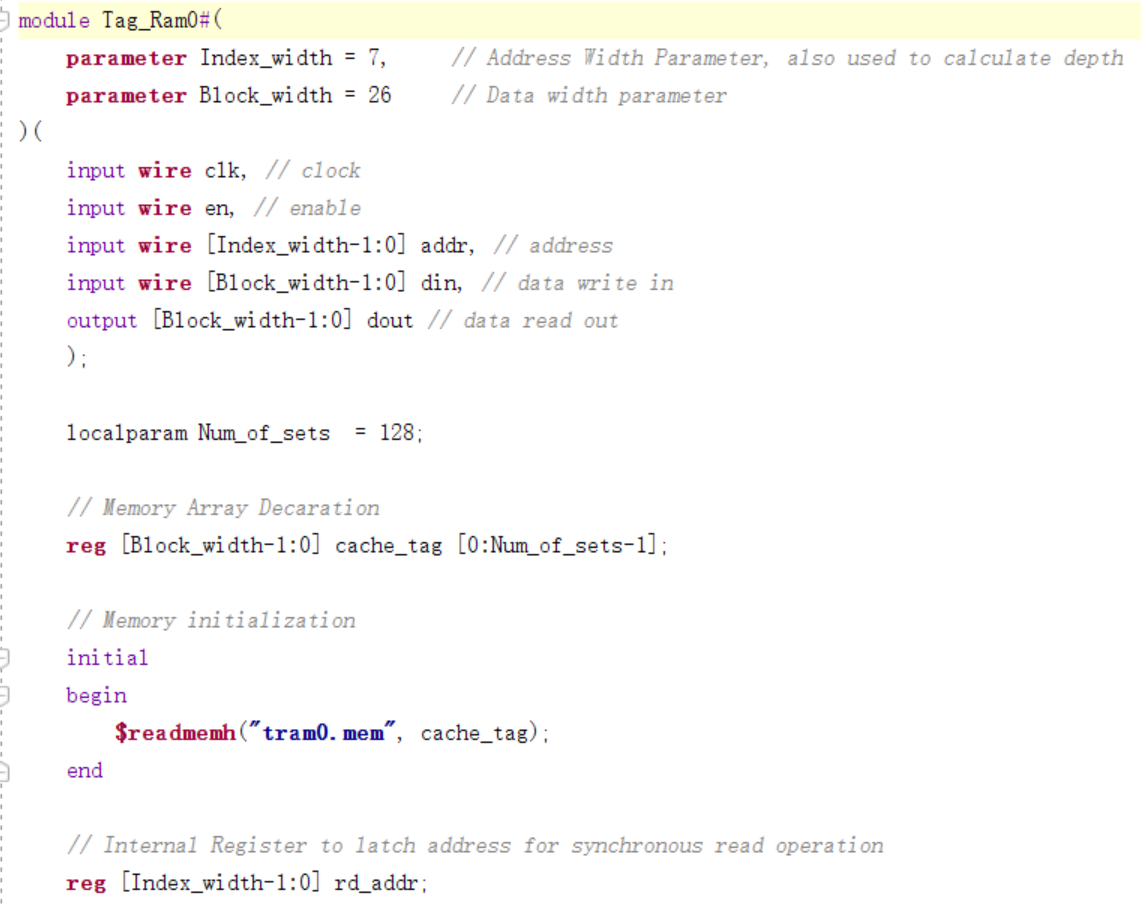


图3.3.2 Tag\_ram模块

**4. 仿真设计**

依据给出的仿真代码，根据本实验状态机进行修改，代码如下。

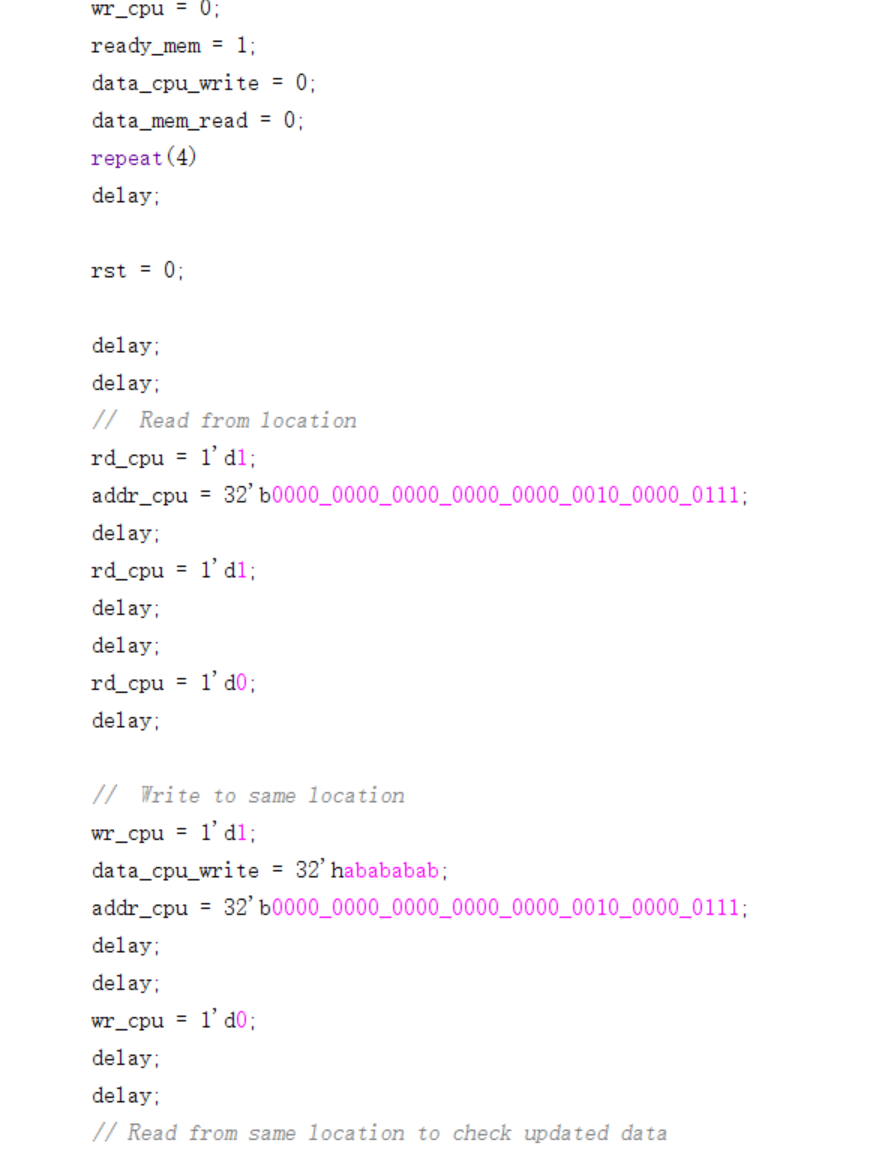


图3.4.1 cache仿真（详见附件）

**四、实验结果分析**

**1. cache控制模块设计思路**

本实验状态机依据实验PPT中给出的状态机设计。在IDLE状态时，对一些基本变量进行赋值。有请求时，进入CompareTag状态，此时判断是否命中，若命中，则直接读写数据，在写数据时设置dirty位，并更新LRU状态；若没有命中且当前块不是dirty，则直接进入Allocate状态，若为dirty，则进入WriteBack状态。在Allocate状态下，从mem中连续读入4个block值，采用比较读入值是否改变（类似上升沿触发）来判断是否读入，从而防止因为时序而导致反复读入。在WriteBack状态下，写回cache中当前block到mem中，并进入Allocate状态。具体细节依据下面的流程图（来自wiki百科）。

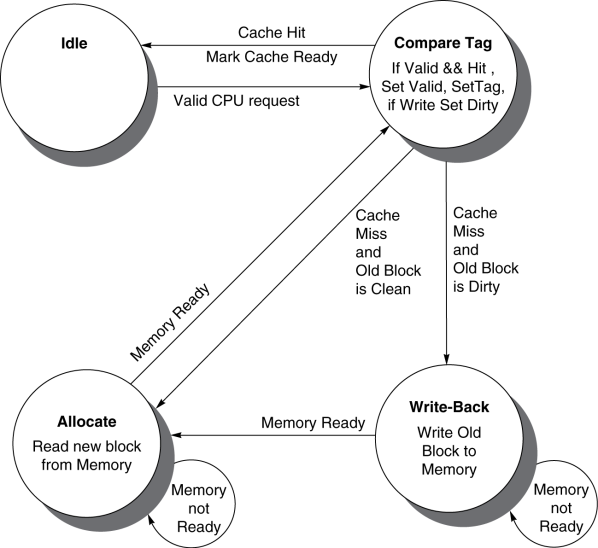


图4.1.1 cache控制状态机

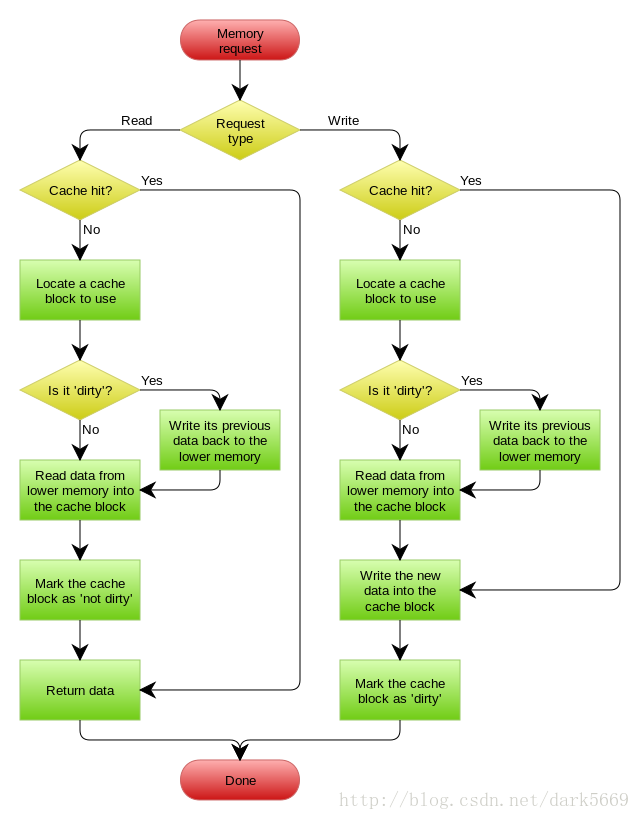


图4.1.2 cache with WB&WA

**2. 仿真结果分析**

依据实验给出的仿真，测试了read hit, read miss with dirty bit 0, write miss with dirty bit 0三种情况，结果如图，符合实验预期。

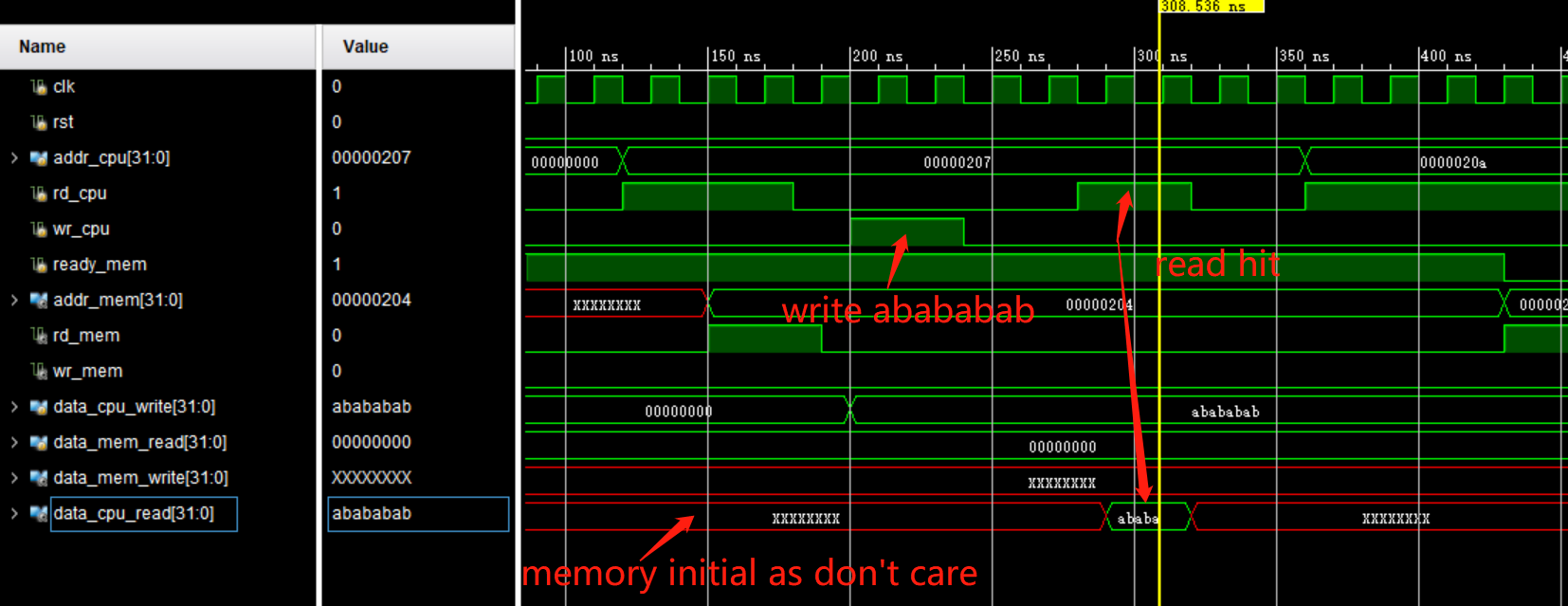


图4.2.1 read hit

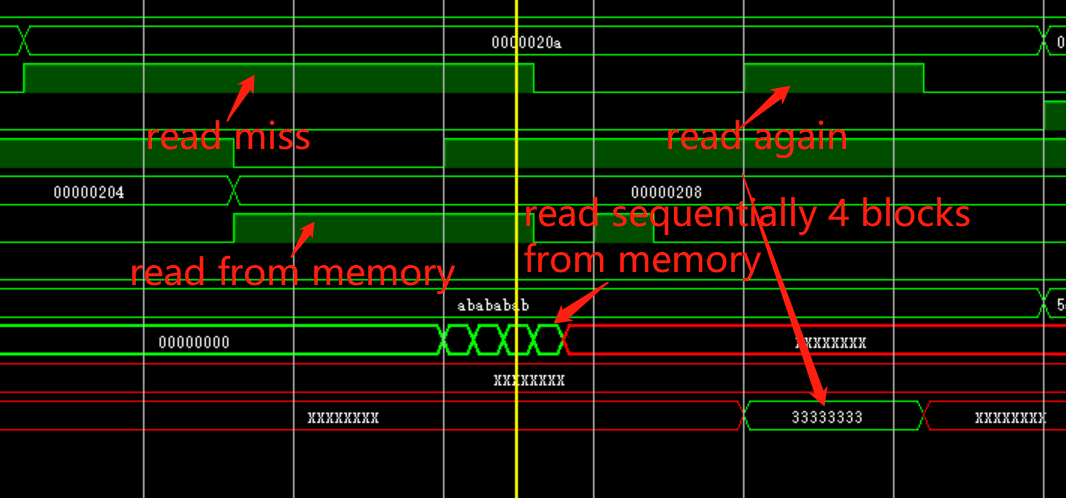


图4.2.2 read miss with dirty bit 0

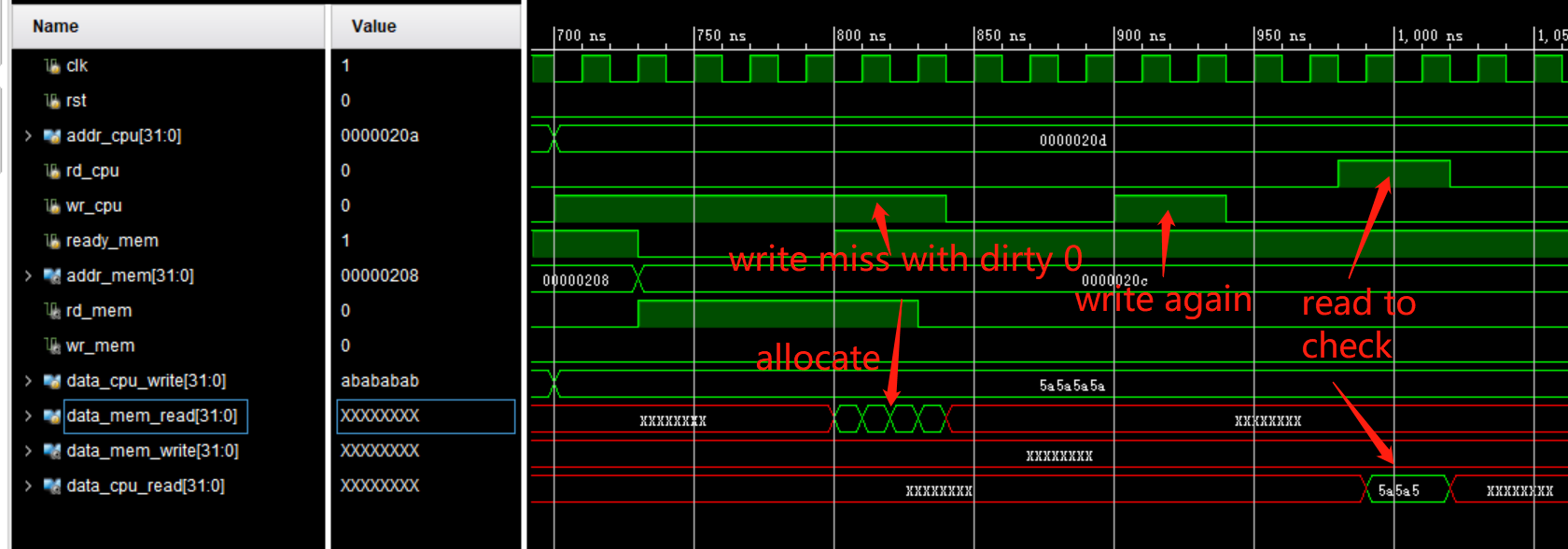


图4.2.3 write miss with dirty bit 0

**五、讨论与心得**

1. 通过本次实验，我掌握了Cache的基本原理，基本了解了Cache存储的优势以及LRU、Write Back、Write Allocate等策略的具体实现，厘清了cache内部存储以及数据的组织（包括几位tag/index、LRU位、dirty位、valid位等等）。

2. 本次实验中，虽然给出了相应的教师版参考答案，但是其中的状态机与PPT中给出的状态机有较大区别，其中细分了很多二级的状态，因此我重新按照PPT给出状态机书写了cache控制器，基本能够实现功能。在过程中遇到了不少状态转移问题，例如涉及较多assign操作时，其顺序与我预想的不一致，导致了状态内部操作赋值存在一些问题；同时，对阻塞和非阻塞赋值，也花费了我不少时间，整体上比较艰难，不过通过这个实验，加深对cache的印象，在日后学习过程中也不会忘记。

3. 该实验进一步锻炼了我书写复杂状态机的能力，熟练度得到更多提升。