版本历史

文档更新记录	文档名:	Lab15_Cache 模块设计（一）
	版本号	V0.1
	创建人:	计算机体系结构研讨课教学组
	创建日期 :	2019-12-18

更新历史

序号	更新日期	更新人	版本号	更新内容
1	2019/12/14	贾凡	-	草稿。
2	2019/12/18	邢金璋	V0.1	初版。

文档信息反馈: xingjinzhang@loongson.cn

1 实验十五 Cache 模块设计（一）

在学习并尝试本章节前，你需要具有以下环境和能力：

(1) 装有 Vivado 的电脑一台。

(2) 熟悉 Vivado。

(3) 掌握 Verilog。

(4) 熟悉龙芯体系结构实验箱（Artix-7）。

通过本章节的学习，你将获得：

(1) 掌握 Cache 的知识。

(2) 学会设计 Cache 模块。

本次实验需要参考的文档包括但不限于：

(1) Lab15 任务书（本文档）。

(2) 体系结构研讨课总讲义之第十章“高速缓存设计”。

1.1 实验目的

1. 掌握 Cache 的知识。
2. 学会设计 Cahe 模块。

1.2 实验设备

1. 装有 Xilinx Vivado 的计算机一台。
2. 龙芯体系结构教学实验箱（Artix-7）一套。

1.3 实验任务

本次实验只有一个任务：

1. 完成：
   a) 按照课程讲义规定的接口设计 Cache 模块。
   b) 通过简单的读写测试。
2. 本次实验要求以组为单位提交实验报告和 RTL 代码，以报告评分和现场检查评分作为最后的实验得分：
   (1) 报告评分：描述自己的设计方案，记录调试过程（错误记录应该配截图）。实验报告模板请使用 lab3 的模板。
   (2) 现场检查评分：检查包含仿真检查和上板检查。

1.4 实验检查

检查前需提交实验报告（每组一份）和调试好的 RTL 代码。本次实验在 2019 年 12 月 24 日进行检查。

现场检查，分仿真检查和上板检查：

1) 仿真检查：对照波形进行描述 Cache 读写的过程。

2) 上板检查：查看上板行为。

现场检查要求能正确应对检查者的提问，并根据要求进行正确的操作演示

1.5 实验提交

提交的作品包括纸质档和电子档。

(1) 纸质档提交

提交方式：课上现场提交，每组提交一份。

截止时间：2019 年 12 月 24 日 18:10。

提交内容：纸质档 lab15 实验报告。

(2) 电子档提交

提交方式：打包上传到 Sep 课程网站 lab15 作业下，每组提交一份。

截止时间：2019 年 12 月 24 日 18:10。

提交内容：电子档为一压缩包，文件名是“lab15_箱子号.zip”，目录层次如下（请将其中的“箱子号”替换为本组箱子号）。

|-lab15_箱子号/	目录，lab15 作品。
| |--lab15_箱子号.pdf/	Lab13 实验报告，实验报告模板参考“Lab03 实验报告模板_仅供参考.docx”
| |--cache /	**目录，自实现 Cache 模块**
| |--cache_test.bit	**功能测试 bit 文件**

1.6 实验环境

本次实验的环境与之前的环境不相同。本次实验的环境是针对 Cache 模块的验证，验证环境中，有 cache_test 模块向 cache 的每一个 Index 发出写和读的请求，并且响应 cache 模块发出的 rd 和 wr 请求。

本次实验环境的目录如下表所示：

|-cache_test/	实验环境目录。
| |--rtl/	包含 Cache 模块以及测试模块的设计代码目录。
| |--cache	目录，其中应当包含 Cache 模块，需要同学们自己完成。
| |--cache_test.v	cache_test 的顶层文件。
| |--testbench/	功能仿真验证目录。
| |--cache_testbench.v	仿真顶层。
| |--run_vivado/	Vivado 工程的运行目录。
| |--cache_test.xdc	Vivado 工程设计的约束文件。
| |--cache_test/	创建的 Vivado 工程:cache_test。
| | |--cache_test.xpr	Vivado 创建的工程文件。

本次实验的步骤是：

1) 准备好本次发布的 lab15 的实验环境，cache_test。
2) 阅读讲义和任务书，完成 cache 模块的设计和 RTL 编写，记为 cache.v。
3) 将 cache.v 放到 cache_test/rtl/cache/目录中。
4) 打开 cache_test 工程（cache_test/run_vivado/cache_test/cache_test.xpr）。
5) 通过“Add Sources”将 cache.v 添加到工程中。
6) 运行 cache_teset 工程的仿真（进入仿真界面后，点击 run all），开始 debug。
7) 仿真通过后，进行综合、布局布线和生成 bit 流文件，并进行上板验证。

1.7 实验说明

1.7.1 模块接口

在讲义第 10 章 1.2.3 节中规定了 Cache 模块与 AXI 总线模块之前的接口。讲义中没有写 CPU 与 Cache 模块的接口，因为这些接口可以在设计中随自己的习惯来定义。但是在本实验中需要使用以下规定的接口来接收请求和返回 cache 结果：

名称	位宽	方向	含义
clk	1	IN	时钟信号
resetn	1	IN	复位信号
Cache 与 CPU core 交互接口			
valid	1	IN	表明请求有效
op	1	IN	1：WRITE；0：READ
index	8	IN	地址的 index 域(addr[11:4])
tlb_tag	20	IN	从 tlb 查到的 pfn 与地址的部分位组合而成的 tag。
offset	4	IN	地址的 offset 域(addr[3:0])
wstrb	4	IN	写字节使能信号
wdata	32	IN	写数据
addr_ok	1	OUT	该次请求的地址传输 OK，读：地址被接收；写：地址和数据被接收
data_ok	1	OUT	该次请求的数据传输 OK，读：数据返回；写：数据写入完成
rdata	32	OUT	读 Cache 的结果
Cache 与转换桥的交互接口			
rd_req	1	OUT	读请求有效信号。高电平有效。
rd_type	3	OUT	读请求类型。3’b000——字节，3’b001——半字，3’b010——字，3’b100——Cache 行。
rd_addr	32	OUT	读请求起始地址.
rd_rdy	1	IN	读请求能否被接收的握手信号。高电平有效。
ret_valid	1	IN	返回数据有效吸纳后。高电平有效。
ret_last	2	IN	返回数据是一次读请求对应的最后一个返回数据。
ret_data	32	IN	读返回数据。
wr_req	1	OUT	写请求有效信号。高电平有效。
wr_type	3	OUT	写请求类型。3’b000——字节，3’b001——半字，3’b010——字，3’b100——Cache 行。
wr_addr	32	OUT	写请求起始地址。
wr_wstrb	4	OUT	写操作的字节掩码。仅在写请求类型为 3’b000、3’b001、3’b010 情况下才有意义。
wr_data	128	OUT	写数据。

wr_rdy	1	IN	写请求能否被接收的握手信号。高电平有效。

1.7.2 RAM 组织结构

本次实验要求 ICache 和 DCache 实现为相同的结构：2 路组相连，每路 4KB，Cache 行大小为 16 字节。

讲义第 10 章 1.2.2 已经分析了 Cache 的组织。针对 4KB*2ways 的 Cache 组织结构，列举如下：

(1) {Tag,Valid}RAM：调用 256 项*21 比特的 RAM，共需要 2 块 RAM。

(2) {Dirty} RAM：使用 regfiles 写法实现，实现为 2 个 256 项*1 比特的结构，或 1 个 256 项*2 比特的结构。

(3) {Data} RAM：调用 256 项*32 比特的 RAM，共需要 8 块 RAM。

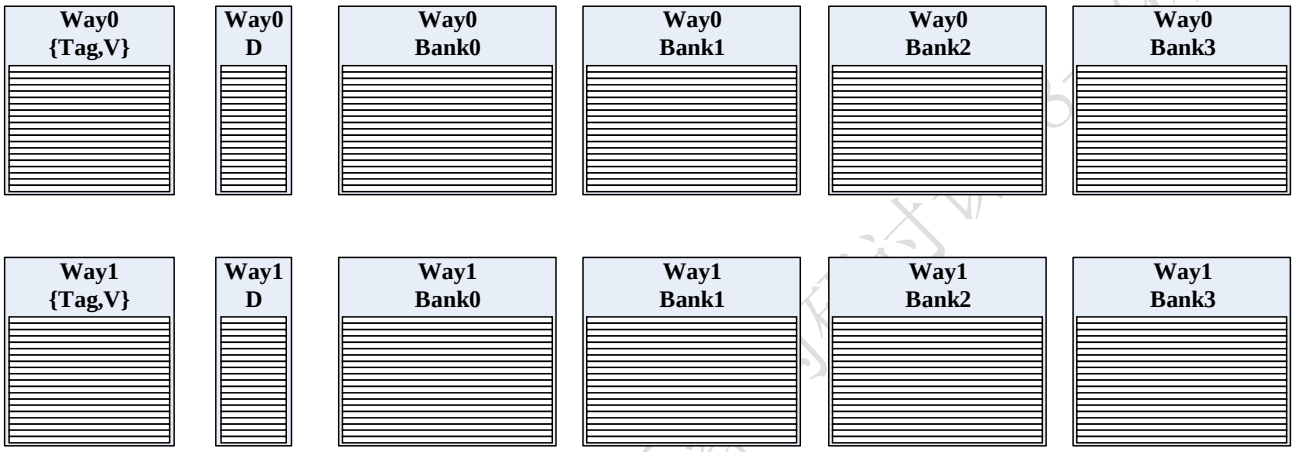

图 1、Cache 的逻辑组织结构

在本实验中，可以参考实验 2 的任务书生成 block ram，但是要注意 Bank 需要开启字节使能。

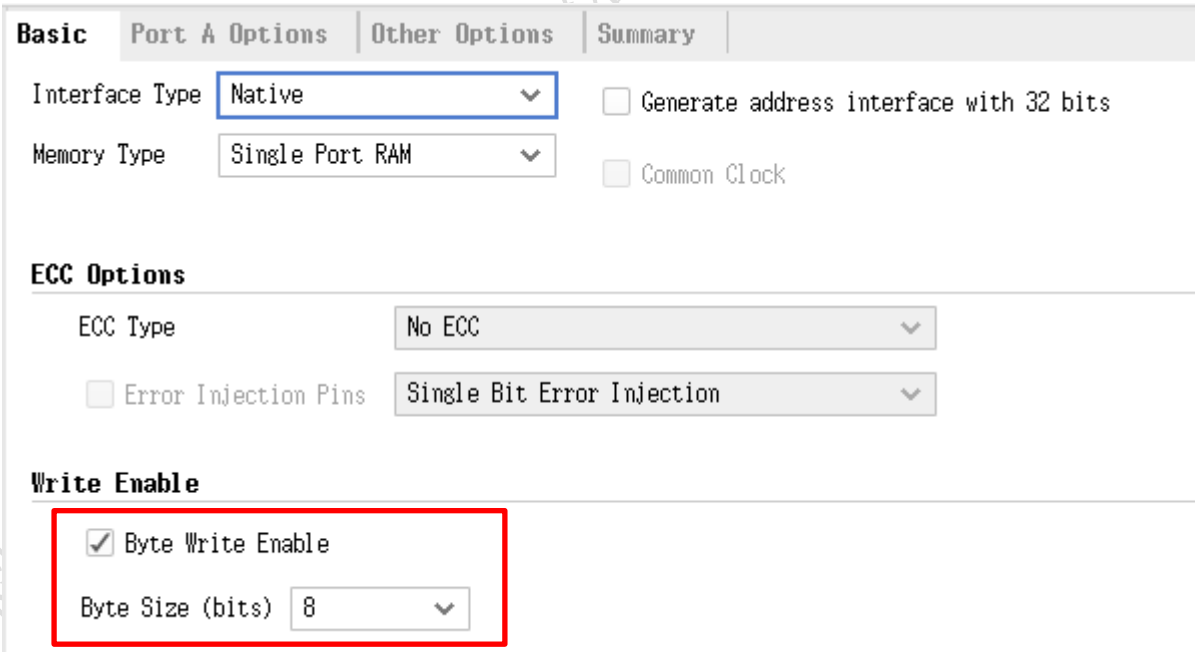

图 2、RAM 生成的注意事项

1.7.3 Cache 内部模块

根据讲义第 10 章 1.2 节的介绍，Cache 内部模块大致分为：

(1) Ramwrap：就是讲义中 3 类 Cache 表（TAG/V，Dirty，Data）的调用逻辑。

(2) TagCompare&DataSelect：根据 Cache 表（TAG/V 和 Data）读出数据进行 Tag 比较，比较结果也就是命中与否的信息，使用比较结果选择 Data 返回给 CPU。

(3) Request Buffer：维护 Store 命中的写 Cache，和 Cache miss 时的处理：对外发出 AXI 请求，对 Cache 进行 Replace 和 Refill。

(4) LSFR：也就是线性反馈移位寄存器，用来产生伪随机数，在 Cache 进行替换时（Replace&Refill）选择替换的路号。关于 LSFR 的写法，可以参考 cache_test/rtl/cache_test.v 里第 40 行的 pseudo_random_23 的写法。

1.7.4 Cache 状态机

讲义第 10 章 1.3.1 节的给出了一个 5 个状态的状态机，以供参考。

需要提醒的是：当状态机处于 LOOKUP 状态时，如果是一条命中的 Store 请求（Hit Store），那状态的转换需要仔细考虑下。讲义中对于这一点没有描述清楚。

1.7.5 Cache 模块级验证说明

模块级验证会从 index=0 的时候开始验证，针对每个 index，生成四组随机的 tag 和 data 对。首先生成写请求将这四组数写进 cache，然后再生成读请求读它们。如果中间没有发生错误，index 递增，重新生成 tag 和 data 对进行相同的测试，直到 index==ff 的测试完成为止。

对于写 cache 请求。验证环境期望看到的结果是，写请求发出后会出现 Cache miss，Cache 模块会发出 rd 请求，验证环境返回全 1 值（0xFFFFFFFF）。写请求可能会引发替换操作，这时验证环境会拿 wr_addr 和 wr_data 和前述的 tag/data 组合做对比，如果 replace 的值有错，测试会中止。

写操作全部进行完之后会有读操作，验证环境会做同样的检测。当 cache 返回读操作的结果之后，验证环境会检测读到的结果与之前写入的结果是否相同。

(1) 仿真效果

在仿真时，会对每一个 index 生成四个 cache 行的先写再读的操作，所有操作都完成后会打印 PASS。

```
[   2705 ns] index 00 finishd
……
============================================================
Test end!
----PASS!!!
```

如果仿真中发现错误，请进行 debug，控制台会打出错误的原因。验证环境只会检查 replace 时的数据错误和 cacheres 的数据错误。

(2) 上板效果

图 3、上板效果示意图

在本次实验中，实验板上数码管的左边两位会显示当前测试的 index 值，直到 index 为 0xff 的时候测试停止。