Lab 9 报告

学号: 2021K8009908024、2021K8009910001、2021K8009908004

姓名: 徐畅杰、张修梁、王致力

箱号: 34

一、实验任务

- 设计 Cache 模块，并且按照实验指导书中的设计要求实现；之后通过给定的验证框架验证
- 将上述 Cache 模块作为指令 Cache 接入 CPU，调整原有的取指通路与总线接口

二、实验设计

（一）总体设计思路

1、Cache 模块的设计

- Cache 结构：

  Cache 采用 2 路组相联的结构，每路包含 Tag、数据以及有效位等信息。每个 Cache 行由一个 Tag 数组、一个数据数组和一个有效位数组组成。Tag 数组存储每个 Cache 行的 Tag 信息，用于与地址进行比较以确定是否命中。数据数组存储每个 Cache 行对应的数据。有效位数组用于标识 Cache 行是否有效，如果有效位为 0，则表示 Cache 行为空或已被替换。

- 主状态机（Main State Machine）：

  主状态机控制 Cache 模块的整体操作流程，其中的状态包括 IDLE（空闲）、LOOKUP（查找）、MISS（缺失）、REPLACE（替换）和 REFILL（填充）等。较为重要的转换条件如下：

  1. 从 IDLE 状态到 LOOKUP 状态的转换：

     当有新的请求到来且 Cache 模块空闲时，主状态机从 IDLE 转换为 LOOKUP 状态。如果新请求是写操作，则将地址和数据存储到 Write Buffer 中，并将地址传递给 Cache 表进行查找。如果新请求是读操作，且没有冲突（Hit Write 冲突），则将地址传递给 Cache 表进行查找。

  2. 从 LOOKUP 状态到 LOOKUP 状态的转换：

     当前请求命中 Cache 且是写操作，或当前请求是读操作且没有冲突（Hit Write 冲突），主状态机保持在 LOOKUP 状态进行下一次查找。从 LOOKUP 状态到 MISS 状态的转换：当前请求未命中 Cache，主状态机从 LOOKUP 转换为 MISS 状态。

  3. 从 MISS 状态到 REPLACE 状态的转换：

MISS 状态发生时，根据 LFSR（线性反馈移位寄存器）生成替换的路号，用于确定要替换的 Cache 行。主状态机从 MISS 转换为 REPLACE 状态，同时将替换的路号传递给 Miss Buffer。

- Write Buffer：

  Write Buffer 用于暂存来自流水线方向的待写数据请求。可以使用 FIFO（First-In-First-Out）缓冲区来实现 Write Buffer。当有新的写请求到达时，将地址和数据存储到 Write Buffer 中，并设置相应的写使能信号。主状态机根据 Write Buffer 中的待写请求进行数据写入操作。

- Request Buffer：

  Request Buffer 用于记录来自流水线方向的请求信息。请求信息包括地址、读写类型等。当有新的请求到达时，将请求信息存储到 Request Buffer 中，并设置相应的写使能信号。主状态机根据 Request Buffer 中的请求信息进行 Cache 的查找操作。

- Miss Buffer：

  Miss Buffer 用于记录缺失 Cache 行的相关信息。包括替换的路号、已从总线返回的数据个数等。当缺失 Cache 行时，主状态机将替换的路号存储到 Miss Buffer 中，并将状态从 MISS 转换为 REPLACE。在 REFILL 状态时，根据 Miss Buffer 中的返回数据个数与缺失地址的 [3:2] 进行比较，决定是否将 ret_valid 信号置为 1。

2、随机数生成器的设计

根据在网上查找的资料，在设计 LFSR 时，最重要的是设计每次移位时，用来填充最低位的表达式逻辑（反馈多项式）。理论上，选择反馈多项式是 LFSR 设计中的一个重要决策，它直接影响着生成的伪随机序列的特性。在选择多项式时，需要考虑最大周期性、均匀分布性与较好的伪随机性；而选择一个不可约的反馈多项式可以增加生成的序列的伪随机性。

此处我们使用了 $f(x) = x^7 + x^3 + x^2 + x^1 + 1$，满足了其伪随机性要求。由于我们设计的 cache 只有两路，替换时非此即彼，最大周期性并不是特别重要，故没有在这个随机数生成器上深究。

3、ICache 接入 CPU 需要的调整

实际上所需要做出的调整较少，只需要将原来 MMU 与转接桥相连的线替换为 icache 的线，并且增加一个用于传递 burst 传输的 last 信息的位置即可。

（二）重要模块 1 设计：主状态机模块

该模块为整个 Cache 状态控制的核心部件

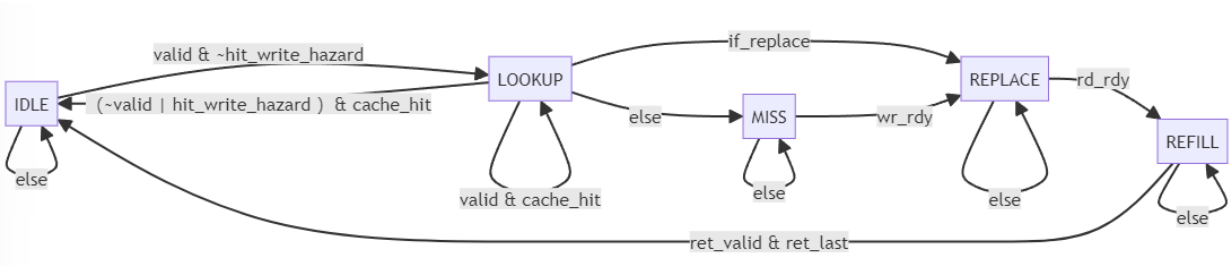
1、工作原理

以下是根据阅读讲义以及自己的理解所总结的各个状态转移逻辑：

- IDLE（空闲状态）：
  - 在空闲状态下，如果有一个有效的请求且没有写冲突（~hit_write_hazard），则进入 LOOKUP 状态进行缓存查找，否则保持在空闲状态。
- LOOKUP（查找状态）：
  - 在查找状态下，根据不同的条件判断下一个状态：
  - 如果缓存命中且没有写冲突，则保持在 LOOKUP 状态，继续进行缓存查找。
  - 如果缓存未命中且有替换请求，则进入 REPLACE 状态，准备替换缓存行。
  - 如果缓存未命中且没有替换请求，则进入 MISS 状态，表示需要从主存中读取数据。
- MISS（缺失状态）：
  - 在缺失状态下，根据是否就绪（wr_rdy）判断下一个状态：
  - 如果就绪，表示可以进行替换操作，进入 REPLACE 状态，准备替换缓存行。
  - 如果未就绪，则保持在 MISS 状态，等待就绪。
- REPLACE（替换状态）：
  - 在替换状态下，根据是否就绪（rd_rdy）判断下一个状态：
  - 如果就绪，表示可以进行填充操作，进入 REFILL 状态，将数据从主存填充到缓存行。
  - 如果未就绪，则保持在 REPLACE 状态，等待就绪。
- REFILL（填充状态）：
  - 在填充状态下，根据返回是否有效（ret_valid）和是否是最后一个数据（ret_last）判断下一个状态：
  - 如果返回有效且为最后一个数据，则回到 IDLE 状态，表示填充完成。
  - 如果返回有效但不是最后一个数据，则保持在 REFILL 状态，继续等待填充。

2、状态转移图

由于状态转移图上无法表示优先级，故此图仅表示逻辑，具体的优先级顺序以代码为准。



3、功能描述

需要指明的是，此处有一个 `hit_write_hazard` 信号，用来表示存在写冲突

- 写冲突判断：写冲突的判断通过 `hit_write_hazard` 条件来实现。该条件表示在当前周期内是否存在与当前写请求冲突的先前写请求。此处主要判断的是在需要写入 cache 时，如果新接收到需要读取的请求，是否与还未写入的数据存在冲突。

- 写冲突处理：

  写冲突的处理是通过在 LOOKUP 状态中根据写冲突情况决定下一个状态来实现。如果存在写冲突，且缓存命中（`cache_hit` 为真），则将下一个状态设置为 IDLE，表示阻塞当前的读请求。这样做是为了确保在存在写冲突的情况下，不会给出错误的读结果；总体设计思路与 CPU 流水线的阻塞设计相似。

（三）重要模块 2 设计：Request/Write Buffer 模块

1、工作原理

为什么要这么设计，其基本工作机制是否合理。

2、接口定义

大部分接口与对应的写入/请求信号相同，主要承担缓存的功能；较为特别的一个写请求信号 (wr_req_r) 在下方的功能描述中有更加详细的介绍。

3、功能描述

Request Buffer:

在请求缓冲区中，根据下一个状态（`next_state`）的值将请求相关的信息存储到相应的寄存器中。主状态机根据当前状态（`curr_state`）和其他条件，决定下一个状态（`next_state`）的转换。当主状态机进入 LOOKUP 状态时，会从请求缓冲区中读取存储的请求信息，并将其传递给缓存控制逻辑进行处理。请求缓冲区的数据存储和读取与主状态机的状态转换配合，确保请求信息在合适的时间传递给缓存控制逻辑。写缓冲区与主状态机的联合工作：

Write Buffer:

写缓冲区用于暂存写请求的相关信息，以便在合适的时间进行处理。当存在写命中时，写缓冲区会将请求缓冲区中的相关信息存储到相应的寄存器中，如标签（wr_tag_r）、命中的路（wr_way_r）、Bank（wr_bank_r）、索引（wr_index_r）、写数据（wr_wdata_r）和写使能信号（wr_wstrb_r）。写缓冲区的数据存储与主状态机的状态转换配合，确保在写命中时将相关信息传递给缓存控制逻辑进行处理。

写请求信号（wr_req_r）：

写请求信号用于告知缓存控制逻辑是否需要进行替换操作。写请求信号的生成依赖于当前状态（curr_state）、下一个状态（next_state）和写就绪信号（wr_rdy）。当主状态机处于 MISS 状态且下一个状态为 REPLACE 时，写请求信号被设置为逻辑 1，表示需要进行替换操作。当写就绪信号为真时，写请求信号被设置为逻辑 0，表示写操作已经完成。写请求信号的生成与主状态机的状态转换和写就绪信号的产生相结合，确保在合适的时机通知缓存控制逻辑进行替换操作。

（四）重要模块 3 设计：转接桥的新连线

1、工作原理

增加部分连线，使得转接桥可以支持 burst 传输。

2、接口定义

仅仅描述相比之前增加的接口，不考虑因为截断而改名的接口

表 1: 接口定义表

接口名称	方向	位宽	功能描述
r_last	axi->cache	1	burst 传输的最后一段消息到达
ar_len	cache->axi	8	翻译需要进行 burst 传输的长度

3、功能描述

通过储存 burst 长度以及将上层连线直接连接到相应的接口，使得转接桥可以支持 burst 传输。

三、实验过程

（一）实验流水账

12.25 08：00-16：00 设计 icache 模块并实现

12.27 15：00-17：00 将 icache 接入 CPU。

（二）错误记录

1、错误 1：burst 传输长度错误

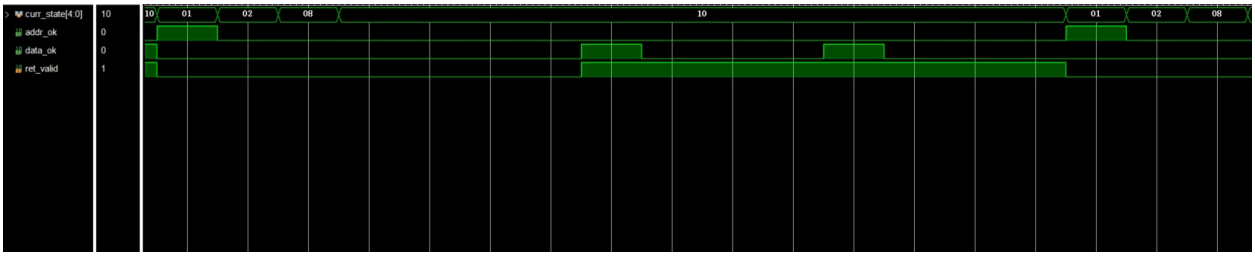
（1）错误现象

从 icache 中取出了错误的指令。

```
------------------------------------------------------------
[11374167 ns] Error!!!
    reference: PC = 0x1c04d95c, wb_rf_wnum = 0x0c, wb_rf_wdata = 0xfda5e000
    mycpu    : PC = 0x1c04d958, wb_rf_wnum = 0x0c, wb_rf_wdata = 0xfda5e000
------------------------------------------------------------
```

（2）分析定位过程

直接查看 icache，发现在进行 burst 传输时因为错误指定了长度导致了多命中，即同一个地址被填了两次，其中第二次是错误的。



（3）错误原因

如前所述，长度指定错误，应当为 4，这里为 8.

```
assign arlen = {8{selecter[0]}} & 8'b11;
```

（4）归纳总结

只关注具体模块外设计，缺乏对于模块内实现的了解，对于 icache 的具体设计并不了解。

四、实验总结（可选）

不得不说这部分的验证相当奇怪，程序几乎没有局部性，还是使用之前的测试框架，一套测试流程走到底。一看 icache 的测试情况就是 miss 然后命中几下然后继续 miss，导致 icache 的行为过于确定。