Cache实验的预期优化方案

by 李梓煊 25/10/1

预期目标

1. 独立测试

由助教编写提供一个cache的单独测试环境（可以产生随机数种子），规定标准的顶层接口，支持仿真、上板测试，不再通过接入排序算法验证其正确性。

这样做有以下好处：

1. 与lab4解耦合，如果cache的实验需要运用lab4的排序来进行测试，当同学们完成前一个实验时，不会考虑为将来接入cache留出接口以及相应的状态机状态，因此当完成cache本身后，还需要来联合调试，而对于没有经验的同学们，这最后可能会导致排序和cache的接口与状态机都极为混乱（我当年就是）
2. 随机测试能够覆盖更多的边缘情况，能够更好的检测同学们设计cache的稳定性
3. 与后续竞赛班实验做好铺垫，如果同学们没有标准的cache接口与状态机，在cpu中集成cache时候，因为无法适配标准的接口与状态机，可能无法顺利接入。

2. 设置更为合理的难度梯度

建议在本次实验中只采用类SRAM接口的cache，在后续竞赛班实验中再引入AXI总线，写一个转接桥，这样既可以保证提高班实验不至于太难，而竞赛班实验又不至于太过于简单。

这样做有以下好处：

1. 对于没有学习过组原知识的同学们，设计cache本身的难度就较大了，此时加上AXI总线既显得难上加难，但是后续组原竞赛班实验只接入cache又显得过于简单，难度梯度不太合理，这样可以保证合理的难度梯度
2. cache直接接入AXI总线不符合真实的cpu设计中模块化的思想，在cpu中一般需要单独的一个AXI转接桥模块来实现，原来的AXI总线接口不利于后续组原实验的cpu接入，不如目前写一个合理的接口，等组原实验可以更为方便的接入。

优化方案

测试环境

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。

- 在仿真时，所有操作都完成后会打印PASS，如果在仿真中发现错误，控制台会打印出错误的原因。验证环境只会检查替换时的数据错误和Cache read的数据错误。
- 正确的上板运行，开发板上数码管的左边两位显示当前测试的index值，直到index为0xff的时候测试停止，并且基于开关可以控制随机数种子，通过改变随机数，可以更大程度的覆盖测试样例

测试完整性保证

全覆盖测试

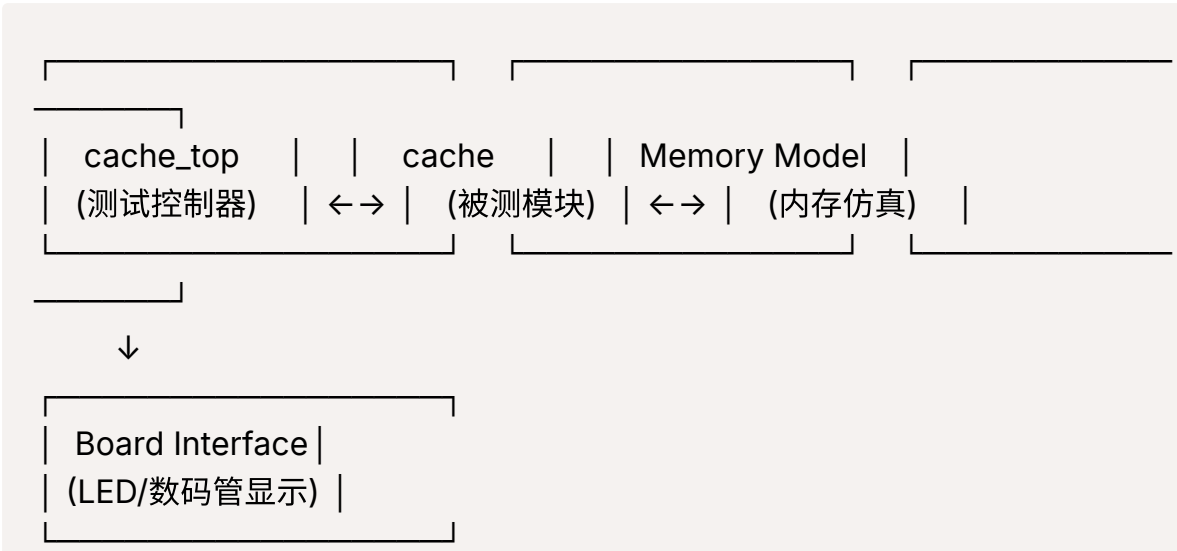
- 测试256个不同的Cache索引（0x00-0xFF）
- 每个索引测试4路关联的所有可能情况
- 验证Cache的命中、缺失、替换等所有功能

实时验证

- 每次读操作都立即验证结果
- 发现错误立即停止测试并指示错误位置
- 成功完成所有测试后显示PASS状态

测试环境原理

1. 整体架构



2. 测试状态机

项目实现了一个三阶段的测试流程：

- **PREPARE阶段**：生成随机测试数据
- **WRITE阶段**：向Cache写入测试数据
- **READ阶段**：从Cache读取数据并验证

3. 核心测试原理

数据生成机制

```
// 伪随机数生成器，基于开关状态和线性反馈移位寄存器
pseudo_random_23 <=
   {pseudo_random_23[21:0], pseudo_random_23[22] ^ pseudo_random_2
3[17]};
```

测试数据结构

- **Tag数组**：`reg [19:0] tag [3:0]` - 存储4组20位标签
- **Data数组**：`reg [127:0] data [3:0]` - 存储4组128位数据块
- **测试索引**：`test_index` 从0到255，覆盖所有Cache行

写测试流程

```
// 每个测试轮次写入4个32位字（构成128位缓存行）
for(counter_i = 0; counter_i < 4; counter_i++) {
    for(counter_j = 0; counter_j < 4; counter_j++) {
        写入 data[counter_i][counter_j*32 +: 32] 到 Cache
    }
}
```

读验证流程

```
// 读取并验证每个32位字
cacheres_right = out_valid && (cacheres == data[res_counter_i][31:0]);
cacheres_wrong = out_valid && (cacheres != data[res_counter_i][31:0]);
```

4. 内存模型仿真

项目内置了完整的内存响应模拟：

读响应模拟

```
// 模拟AXI总线读响应，返回正确的测试数据
assign ret_data = round_state==WRITE ? 32'hffffffff : rd_true_value;
```

写响应模拟

```
// 验证写入数据的正确性
assign data_right = {8'hff,wr_hit_data[119:0]} == wr_data_r;
```

5. 硬件接口与显示

数码管显示

- 实时显示当前测试的`test_index`值（0x00-0xFF）
- 通过扫描显示技术实现8位数码管动态显示

LED指示

- 所有LED常亮表示系统正常运行

- 可通过开关控制LED状态用于调试

错误检测

```
// 两种错误检测机制
replace_wrong = do_wr && {8'hff,wr_hit_data[119:0]} != wr_data_r;  // 替换错误
cacheres_wrong = out_valid && cacheres != data[res_counter_i][31:0]; // 读取错误
```

实验要求

1. 设计Cache模块。
2. 利用Cache模块级验证环境对所设计的Cache进行验证，通过仿真和上板验证。

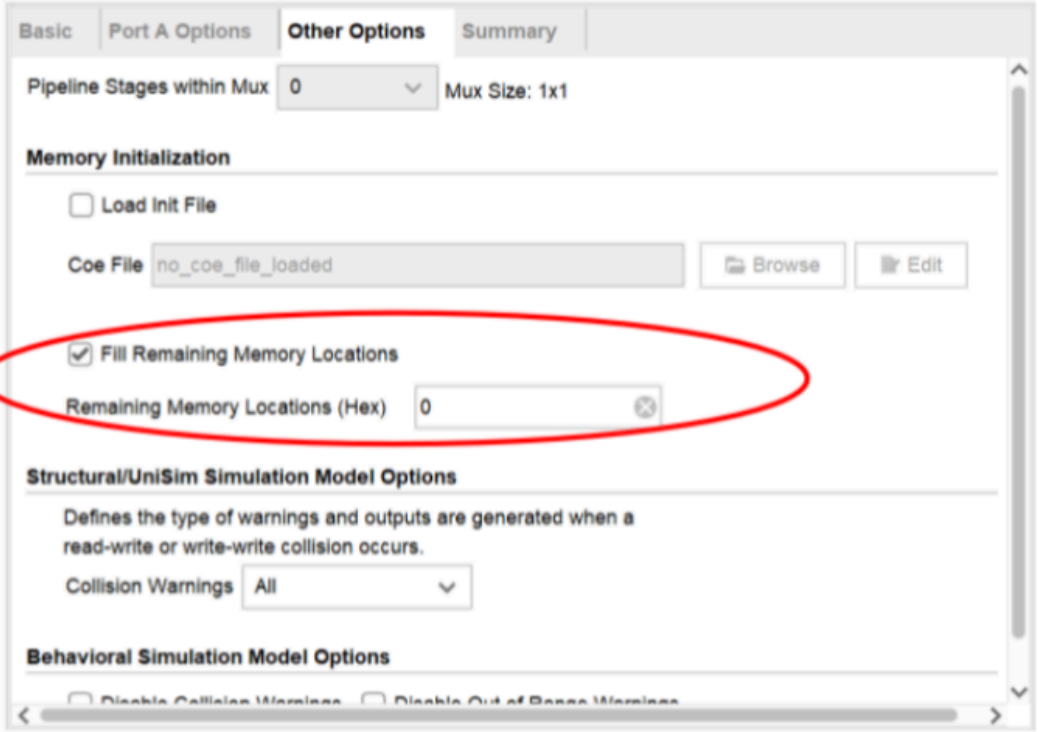
实验配置

Cache模块的设计规格要求：2路组相连，每路大小4KB，LRU或伪随机替换算法，推荐硬件初始化。

在LoongArch精简版指令集中，Cache可以通过软件进行初始化。处理器复位结束之后，CSR.CRMD的DATF和DMTM域都是0值，此时取指和访存都是强序非缓存，软件可以使用CACOP指令将Cache的Tag部分置为0值。

但在我们的实践任务中，这就引发了一个问题：在没有实现CACHE指令的时候，如何在上板验证的时候确保Cache被初始化过。于是在我们的实验场景下，要考虑Cache的硬件初始化问题。

Cache初始化至少要把Cache中每一项的Tag、V、D的状态置为确定的无效值。由于Tag、V信息都存放在RAM中，因此该问题的解决方案是设计一个小的硬件电路，将存放Cache Tag和V信息的RAM的每一行写成全0值。还有一个偷懒的方法：利用实验中采用FPGA 硬件平台这一特点来简化Cache硬件初始化的实现。读者可以在生成Cache所用的RAM的时候，选择将RAM初始化成全0。具体来说，是在RAM IP生成对话框的“Other Options”标签下，勾选“Fill Remaining Memory Locations”，同时将初始值设为0值，如图所示



Cache接口

表 10.3: Cache模块与AXI总线的交互接口

名称	位宽	方向	含义
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	返回数据有效信号后。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。此处要求wr_rdy要先于wr_req置起，wr_req看到wr_rdy后才可能置上。所以wr_rdy的生成不要组合逻辑依赖wr_req，它应该是当AXI总线接口内部的16字节写缓存为空时就置上。

```
cache cache(
```

```
    .clk    (clk_g),
    .resetn (resetn),
    .valid  (memref_valid),
    .op     (memref_op ),
    .index  (in_index  ),
    .tag    (in_tag    ),
    .offset (in_offset ),
    .wstrb  (memref_wstrb),
    .wdata  (memref_data),

    .addr_ok(cache_addr_ok),
    .data_ok(out_valid),
    .rdata  (cacheres ),

    .rd_req   (rd_req   ),
    .rd_type  (rd_type  ),
    .rd_addr  (rd_addr  ),
    .rd_rdy   (rd_rdy   ),
    .ret_valid(ret_valid),
    .ret_last (ret_last ),
    .ret_data (ret_data ),

    .wr_req  (wr_req  ),
    .wr_type (wr_type ),
    .wr_addr (wr_addr ),
    .wr_wstrb(wr_wstrb),
    .wr_data (wr_data ),
    .wr_rdy  (wr_rdy  )
);
```