哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY

实验报告

开课学期：2025 春季

课程名称：计算机组成原理（实验）

实验名称：高速缓存器设计

实验性质：设计型

实验学时：4 地点：T2506

学生班级：7 班

学生学号：2023311709

学生姓名：宁中昊

作业成绩：

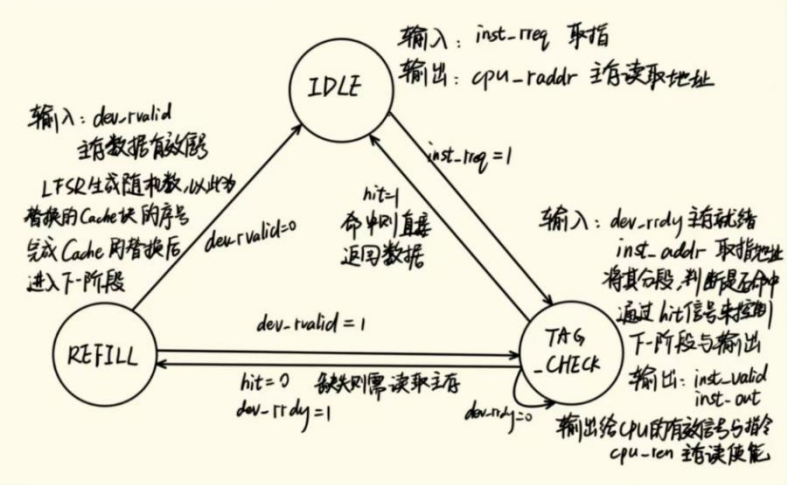
实验与创新实践教育中心制

2025 年 4 月

1、Cache 详细设计

要求：绘制 ICache 的状态转换图，并结合关键代码，**详细描述**状态转移关系、转移条件、各状态的输入输出信号以及需要完成的操作。**若完成了附加题，则分别绘制 DCache 的读、写状态转换图，并配以文字详细描述相应的内容。*

状态转换图：



转移关系与转移条件：

```
always @(*) begin
    case(cur_state)
    IDLE: begin
        next_state = inst_rreq ? TAG_CHECK : IDLE;
    end

    TAG_CHECK: begin
        if(hit) begin
            next_state = IDLE;
        end else begin
            next_state = dev_rrdy ? REFILL : TAG_CHECK;
        end
    end

    REFILL: begin
        next_state = dev_rvalid ? TAG_CHECK : REFILL;
    end

    default: begin
        next_state = IDLE;
    end
    endcase
end
```

体现在三段式的第二段中，根据输入的 inst_rreq 信号来判断是否开始取值（IDLE=>TAG_CHECK），由 hit 信号来决定取值是否完成（TAG_CHECK=>IDLE），由 dev_rrdy 来决定是否需要替换 Cache 块（TAG_CHECK=>REFILL），由 dev_rvalid 信号来决定替换是否已经顺利完成（REFILL=>TAG_CHECK）。

各阶段输入输出与操作：

在 IDLE 状态中，等待 CPU 的取值请求，根据 inst_rreq 来选择是否进入 TAG_CHECK 状态。将 cpu_addr 准备好即对齐主存地址。

```
IDLE: begin
    cpu_raddr <= inst_rreq ? {inst_addr[31:5], 5'h0} : 0;
end
```

（对齐主存地址）

在 TAG_CHECK 状态中，检查请求的指令是否存在 Cache 中，如果是则命中，hit=1，否则未命中 hit=0，并以此作为 inst_valid 的输出。命中时将地址对应的 32 位指令输出到 inst_addr，未命中时待主存就绪（dev_rrdy=1）时将主存读使能信号赋值为 4b'1111，发起主存读操作，进入到 REFILL 状态。

```
wire hit = (cur_state == TAG_CHECK) && (
    (valid[0] && (tag[0] == tag_from_cpu)) ||
    (valid[1] && (tag[1] == tag_from_cpu)) ||
    (valid[2] && (tag[2] == tag_from_cpu)) ||
    (valid[3] && (tag[3] == tag_from_cpu))
);
```

（判断是否命中）

```
inst_valid = hit;
inst_out   =  data[hit_index][32*offset[4:2] +: 32];
```

（命中输出逻辑）

在 REFILL 状态中，从主存加载块到 Cache 块中，用 LSFR 生成随机数作为被替换的 Cache 块的序号，替换 Cache 块。

```
// LFSR 生成随机数
always @(posedge cpu_clk or posedge cpu_rst) begin
    if(cpu_rst) begin
        lfsr <= 2'b01;
    end else begin
        lfsr <= {lfsr[0]^lfsr[1], lfsr[1]};
    end
end
```

（LSFR 生成随机数）

```
always @(posedge cpu_clk or posedge cpu_rst) begin
    if(cpu_rst) begin
        replace_index <= 2'b00;
    end else if(free_index != 3'b100) begin
        replace_index <= free_index[1:0];
    end else begin
        replace_index <= lfsr;
    end
end
```

（Cache 块替换序号确定）

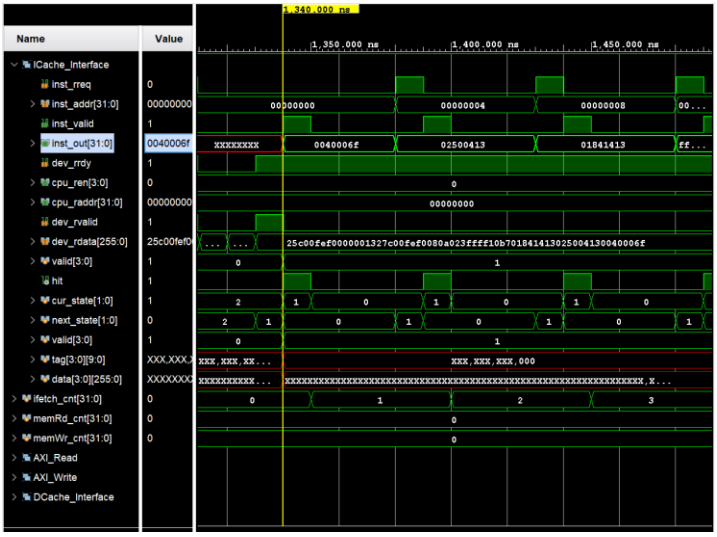
```
always @(posedge cpu_clk or posedge cpu_rst) begin
    if(cpu_rst) begin
        valid      <= 0;
        free_index <= 0;
    end else begin
        if((cur_state == REFILL) && dev_rvalid) begin
            valid[replace_index] <= 1'b1;
            tag  [replace_index] <= tag_from_cpu;
            data [replace_index] <= dev_rdata;

            if(free_index != 3'b100) begin
                free_index <= free_index + 1;
            end else begin
                free_index <= free_index;
            end
        end else begin
            valid <= valid;
        end
    end
end
```

（Cache 块替换逻辑）

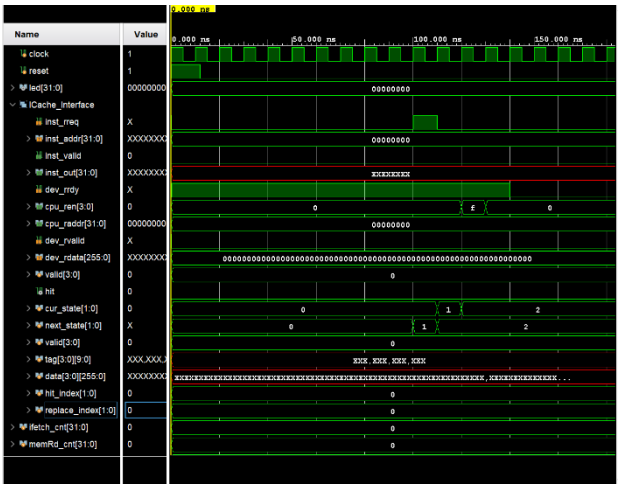
2、调试报告

要求：结合仿真波形截图对 ICache 作详细的时序分析，要求包含读命中、读缺失 2 种情形，每种情形列举 1 个测试用例**详细分析**。分析过程参考实验原理中的时序解读，但需**把模块内部关键信号添加到波形，并结合信号的实际取值**进行分析。*若完成了附加题，则需额外给出 DCache 的仿真波形截图及其详细文字分析，要求包含写命中、写缺失和 Uncached 访问 3 种情形，每种情况列举 1 个测试用例详细分析。*



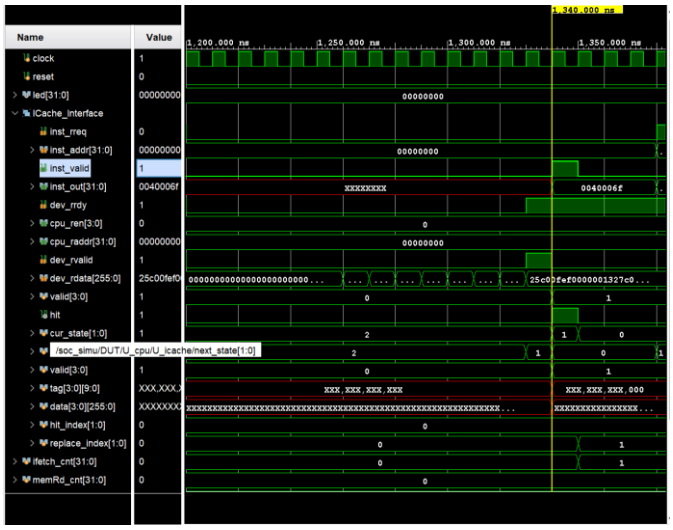
（图 2-1）

如图 2-1，在标出的位置后第一个时钟内，inst_rreq 被拉高，由 IDLE 状态进入 TAG_CHECK 状态。随后命中 Cache 块，hit 信号拉高，没有进入 REFILL 状态直接更新 inst_out 输出。

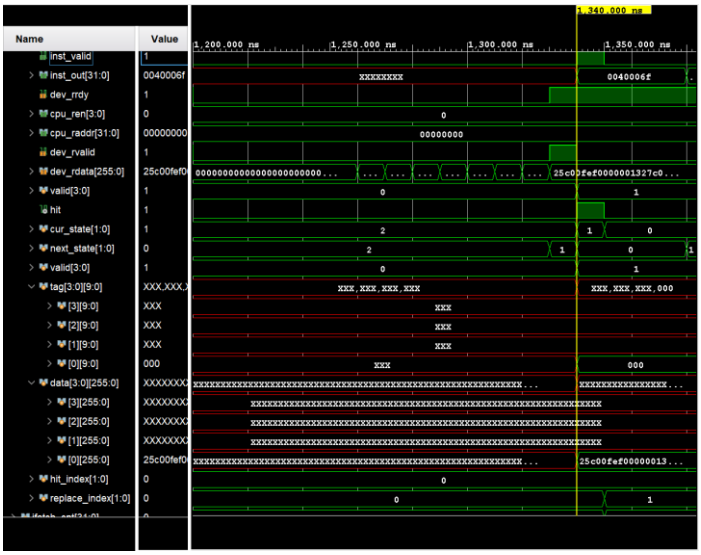


（图 2-2）

如图 2-2 为第一个时钟周期，Cache 块此时为空，因此导致读缺失，因此需要进入 REFILL 状态，经过若干个时钟周期访问主存更新 Cache 块，完成后更新 Cache 块，如图 2-3。查看 valid、tag、data 信号发现在 REFILL 状态结束后更新了 Cache 块，如图 2-4，标示位置即为 REFILL 状态结束位置，更新 Cache 块


Name
Value
clock
reset
led[31:0]
iCache_interface
inst_rreq
inst_addr[31:0]
inst_valid
inst_out[31:0]
dev_rrdy
cpu_ren[3:0]
cpu_raddr[31:0]
dev_rvalid
dev_rdata[255:0]
valid[3:0]
hit
cur_state[1:0]
/soc_simu/DUT/U_cpu/U_icache/next_state[1:0]
valid[3:0]
tag[3:0][9:0]
data[3:0][255:0]
hit_index[1:0]
replace_index[1:0]
ifetch_cnt[31:0]
memRd_cnt[31:0]
1,200.000 ns
1,250.000 ns
1,300.000 ns
1,340.000 ns
1,350.000 ns

（图 2-3）


Name
Value
inst_valid
inst_out[31:0]
dev_rrdy
cpu_ren[3:0]
cpu_raddr[31:0]
dev_rvalid
dev_rdata[255:0]
valid[3:0]
hit
cur_state[1:0]
next_state[1:0]
valid[3:0]
tag[3:0][9:0]
[3][9:0]
[2][9:0]
[1][9:0]
[0][9:0]
data[3:0][255:0]
[3][255:0]
[2][255:0]
[1][255:0]
[0][255:0]
hit_index[1:0]
replace_index[1:0]
1,200.000 ns
1,250.000 ns
1,300.000 ns
1,340.000 ns
1,350.000 ns

（图 2-4）

3、思考与讨论

分别给出无 ICache 时和有 ICache 时，SoC 运行测试程序的总时间的截图，并谈谈你对该测试结果的理解。

无 ICache 时 SoC 运行测试程序总时间：

```
All tests passed!
Total: 1972 instruction fetching, 26 memory access (18 RDs and 8 WRs)
$finish called at time : 362540 ns : File "N:/Study/Coding/RISC-V/lab3_stu/miniRV_axi/miniRV_axi.srcs/sim_1/new/soc_simu.v" Line 185
run: Time (s): cpu = 00:00:07 ; elapsed = 00:00:08 . Memory (MB): peak = 2974.281 ; gain = 0.000
```

有 ICache 时 SoC 运行测试程序总时间：

```
All tests passed!
Total: 1968 instruction fetching, 26 memory access (18 RDs and 8 WRs)
ICache hit rate: 84.451%
$finish called at time : 501350 ns : File "N:/Study/Coding/RISC-V/lab3_stu/miniRV_axi/miniRV_axi.srcs/sim_1/new/soc_simu.v" Line 185
run: Time (s): cpu = 00:00:05 ; elapsed = 00:00:10 . Memory (MB): peak = 2960.227 ; gain = 0.000
```

可以看到，无 ICache 时运行时间为 362540ns，比有 ICache 时的 501350ns 更少，加上 ICache 后似乎没有提升运行效率，反而减低了运行速度。我的理解是，在未命中的时候主存访问时间太慢，CPU 需要等待较长的主存访问时间，就导致在未命中时额外的时间开销相比命中时带来的效率收益更大，最后使得运行时间变得更长。