

中国科学院大学

《计算机体系结构基础(研讨课)》实验报告

姓名 艾华春,李霄宇,王敬华

学号 2022K8009916011,2022K8009929029,2022K8009925009

实验项目编号 7 实验名称 高速缓存设计

一、 逻辑电路结构与仿真波形的截图及说明

• Cache 模块设计。

1. Cache 的逻辑组织结构

Cache 采用两路组相连的方式,每一路包含相同的 tag,v 表, d 表, 4 个 data_bank 表。所有的表同一时间只能接受一个读或写操作。

如下图所示:

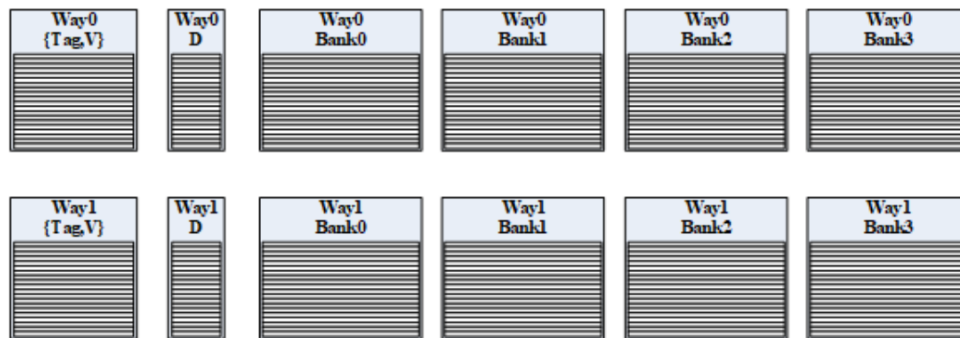


图 1: Cache 逻辑组织结构

(a) data_bank 表

```
generate
for (i = 0; i < 4; i = i + 1)begin: data_way0
    data_bank_ram data_way0(
        .clka (clk),
        .wea (data_way0_wen[i]),
        .addra (data_way0_index[i]),
        .dina (data_way0_wdata),
        .douta (way0_data[i])
    );
end
endgenerate
```

一个 Cache 行有 16 个字节,分为 4 个 data_bank 表。

每一路的 data_bank 表为一个 RAM 256 * 32(深度 * 宽度)。

(b) tag,v 表

```
tagv_ram tagv_ram_way0 (
```

```

.clka (clk),
.wea (tagv_way0_wen),
.addra (tagv_way0_index),
.dina ({tagv_way0_wdata}),
.douta ({way0_tag,way0_v})
);

```

每个 Cache 行有一个 tag 和 v 域,表示该行是否有效和该行的在主存中的地址。

由于所有 Cache 操作对 tag 和 v 的操作是完全一致的,所以,tag 和 v 合并在一起存储,tag 占高位,v 占低位。

每一路的 tagv 表用一个 RAM 256 * 21(深度 * 宽度) 存储。

(c) d 表

```

d_regfile d_way0(
.clk (clk),
.resetn (resetn),
.addr (d_way0_index),
.wen (d_way0_wen),
.wdata (d_way0_wdata),
.rdata (way0_d)
);

```

每个 Cache 行有一个 d 域,表示该行的脏位,由于每行只有一个脏位,所以用 regfile 存储。

每一路的 d 表用 256 个 1 位的寄存器堆存储。

2. Cache 的内部控制逻辑设计

3. Cache 的控制包括两个状态机,主状态机和 write buffer 状态机。

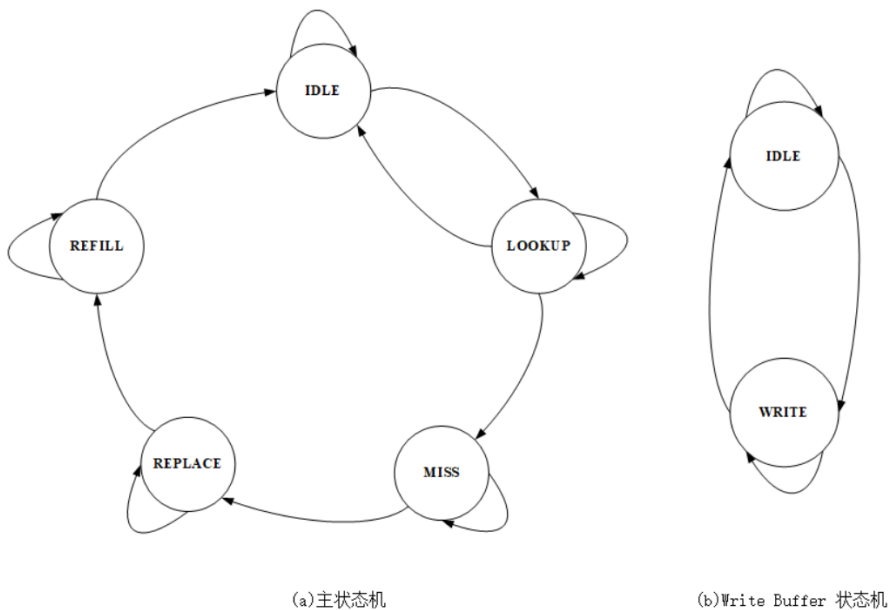


图 2: Cache 的状态机

主状态机包括 5 个状态,idle,lookup, miss, replace, refill。

(a) idle 状态

在 idle 状态下, Cache 等待外部的请求, 当有请求到来, 且与 Cache 中的 Hit write 不冲突时, 用组合逻辑拉高 addr_ok 信号, 并在下一拍 Cache 进入 lookup 状态。

```
assign addr_ok = main_current_state == LOOKUP & cache_hit & ~hit_write_conflict
               | main_current_state == IDLE & ~hit_write_conflict;
```

同时, 将请求信息中的 index 通过组合逻辑, 送到 Cache 进行查询, 将两路的对应的 Cache 行的数据 (tag, v, data) 在下一拍读出来。

并且, 在 request buffer 中存储请求的相关信息。

```
// request buffer, 存储接受到的请求信息
always @(posedge clk) begin
    if (~resetn) begin
        req_buffer_op <= 1'b0;
        req_buffer_index <= 8'b0;
        req_buffer_tag <= 20'b0;
        req_buffer_offset <= 4'b0;
        req_buffer_wstrb <= 4'b0;
        req_buffer_wdata <= 32'b0;
        req_buffer_type = 1'b0;
    end
    else if (addr_ok & valid) begin // next_state == LOOKUP, 存储请求信息
        req_buffer_op <= op;
        req_buffer_index <= index;
        req_buffer_tag <= tag;
        req_buffer_offset <= offset;
        req_buffer_wstrb <= wstrb;
        req_buffer_wdata <= wdata;
        req_buffer_type <= type;
    end
end
```

(b) lookup 状态

根据 Cache 的读出的两行的 tag 信息, 与锁存在 request buffer 中的请求信息进行比较, 如果有 Hit, Cache 进入 idle 状态, 写操作将数据传给 write buffer, 读操作直接返回 Cache 命中的数据。

否则进入 miss 状态。

```
// 请求的tag和Cache返回的tag比较, 判断是否命中
assign way0_hit = way0_v && (way0_tag == req_buffer_tag);
assign way1_hit = way1_v && (way1_tag == req_buffer_tag);
assign cache_hit = (way0_hit || way1_hit) && req_buffer_type;
```

为了在连续的 cache 命中时, 保证 ipc 为 1, 则如果在 lookup 命中, 且接受到流水线的有效的访存请求, 则直接当前 lookup 状态下, 向 cache 的发起对下一个请求的查询, 将对应的 Cache 行的数据 (tag, v, data) 在下一拍读出来, 并在下一拍直接进入 lookup 状态, 进行比较。


```

if(ret_valid & ret_last)
    main_next_state = IDLE;
else
    main_next_state = REFILL;
endcase
end

```

如果是读请求，等到返回 cpu 请求的数据时（即 `miss_buffer_cnt == req_buffer_offset[3:2]`），即可拉高 `data_ok` 信号，将数据传给 cpu。不必等到 Cache 进入 idle 状态。

```

// 读请求，等到返回cpu请求的数据时，即可向cpu返回data_ok信号，并将数据传给cpu
assign data_ok = (main_current_state == REFILL & ret_valid
    & miss_buffer_cnt == req_buffer_offset[3:2] & ~req_buffer_op)
    | ...;

```

• 在 CPU 中集成 ICache。

以 tlb 地址映射为例，如下图所示：

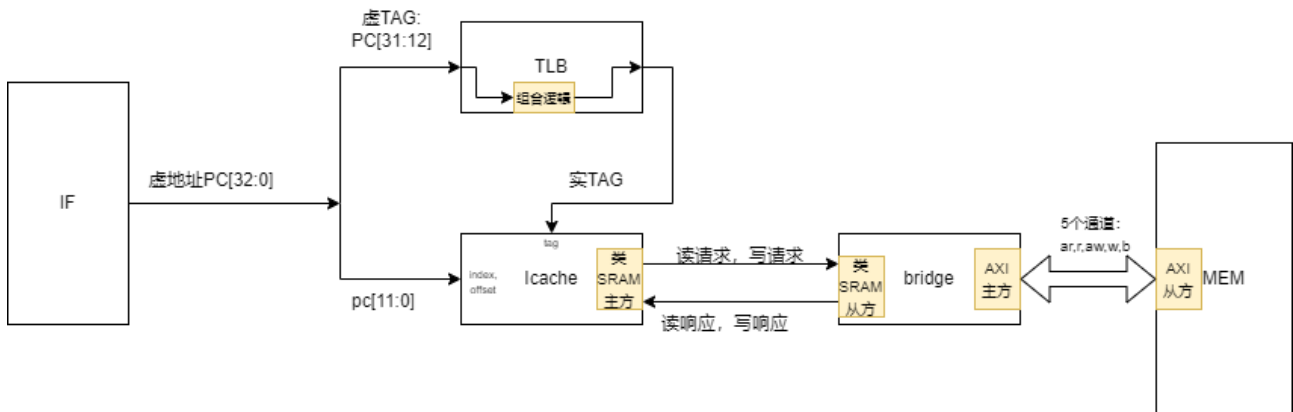


图 4: ICache 集成在 CPU 中示意图

1. ICache 与 CPU 的接口设计

由于 Cache 采用的是“虚 Index 实 Tag”的方式，所以当 cpu 取指的时候，把虚拟地址的 Index 直接传递给 Cache，Cache 便开始根据虚拟地址的 Index 进行查询，

同时，将虚拟地址传递给虚实地址转换单元，它能够通过组合逻辑在同一拍，将虚拟地址转换为物理地址的 TAG，并传递给 Cache。

上述方式使得 Cache 和 TLB 查询能并行执行，提高了效率。

2. ICache 与转接桥的接口设计

ICache 将访存请求发送到转接桥，转接桥将类 sram 的访存请求转换为 AXI 总线的访存请求，并且如果是多个字的访存请求，采用突发 burst 的方式。

以写通道的 burst 控制为例，

```

assign awlen = awtype_reg == 3'b100 ? 8'd3 : 8'd0; // 如果是写整个cache行，burst长度为3
assign awsize = 3'b010; // 每拍发送一个字
assign awburst = 2'b01; // 地址增加突发
assign wlast = awtype_reg == 3'b100 ? wdata_cnt == 2'b11 // 最后一个字，拉高wlast

```

```

        :1'b1;

always @(posedge clk)begin
    if(~resetn)begin
        wdata_cnt <= 2'b0;
    end
    else if(aw_current_state == AW_SEND_DATA && wready)begin
        wdata_cnt <= wdata_cnt + 1;
    end
end
end

```

二、 实验过程中遇到的问题、对问题的思考过程及解决方法(比如 RTL 代码中出现的逻辑 bug, 逻辑仿真和 FPGA 调试过程中的难点等)

● Cache 模块向 cpu 的 data_ok 信号设置。最初, 由于下意识认为 cpu 是串行工作, 所以不论是读还是写, 都等到 Cache 的状态机走完一圈后, 返回 idle 状态, 再向 cpu 返回 data_ok 信号。

但是, 实际上, 流水线和 Cache 是并行工作的, 所以只要 Cache 可以返回读数据或者接受了写数据, 就可以向 cpu 返回 data_ok 信号, 让流水线可以继续工作。

至于要保证 Cache 采用阻塞式设计(即一旦发生缺失, Cache 就不再接受新的请求, 直到缺失的数据返回), 需要用 addr_ok 信号来控制, 等到 Cache 的状态机走完一圈后, 接受完缺失的数据, 再接受新的请求。

```

assign addr_ok = main_current_state == LOOKUP & cache_hit & ~hit_write_conflict //
    如果发生数据缺失, Cache拉低addr_ok, 不接受新的请求
    |main_current_state == IDLE & ~hit_write_conflict;
assign data_ok =
(main_current_state == REFILL & ret_valid & miss_buffer_cnt == req_buffer_offset[3:2] &
    ~req_buffer_op)
// 读缺失时, 在refill状态下, 等到返回cpu请求的数据时, 即可向cpu返回data_ok信号
|(main_current_state == LOOKUP & (cache_hit | req_buffer_op));
// 写操作, 不管是否命中, 都将向cpu返回data_ok信号, 表示请求已经接受

```

三、 小组成员分工合作情况

王敬华负责 TLB 模块实现, 部分 TLB 指令的添加和冲突的处理。

李霄宇负责 TLB 相关指令的添加, 部分冲突的处理, 接口连接。

艾华春负责 Cache 模块设计, 集成 iCache 到 cpu 中。

实验报告为根据每人负责代码的部分, 写相应部分的报告。