

# LAB3实验报告

PB1811757 陈金宝

## 实验目标

阅读并理解助教提供的简单cache的代码，将它修改为N路组相连的（要求组相连度使用宏定义可调）、写回并带写分配的cache。要求实现FIFO、LRU两种替换策略。并将实现的Cache添加到Lab1的CPU中（替换先前的data cache），并添加额外的数据通路，统计Cache缺失率，在Cache缺失时，bubble当期指令及之后的指令。要求能成功运行这个算法（所谓成功运行，是指运行后的结果符合预期）

## 实验环境

操作系统:windows10 20H2  
仿真工具:Vivado 2019.2

## cache实现

为实现组相联，需要将助教原先的cache.sv中部分数据结构增加一个维度：

```
reg [31:0] cache_mem [SET_SIZE][WAY_CNT][LINE_SIZE];  
reg [TAG_ADDR_LEN-1:0] cache_tags [SET_SIZE][WAY_CNT];  
reg valid [SET_SIZE][WAY_CNT];  
reg dirty [SET_SIZE][WAY_CNT];
```

判断命中时，使用for语句并行判断。若命中则break。

```
always @ (*) begin  
    for(integer i = 0; i < WAY_CNT; i++) begin  
        if(valid[set_addr][i] && cache_tags[set_addr][i] == tag_addr) begin  
            cache_hit = 1'b1;  
            hit_pos = i;  
            break;  
        end  
    end  
    else begin  
        cache_hit = 1'b0;  
    end  
end  
end
```

## FIFO

为实现FIFO，为每个SET维持两个数：队首指针和长度。队首指针指向的set内的块号就是当前队列中最早入队的，也即是要被换出的块（如果队列满）。队首指针和长度的初值均为0。当set内未滿时（队列长度 < WAY\_CNT），则不进行换出，队首指针不变，直接将新块换入到当前队列内队列长度对应的位置，并将队列长度+1。（如：当队列长度为0，也即set内是空时，新的line直接放入set内的第0个位置，队列长变为1）

```

if(mem_fifo_len < WAY_CNT) begin
    for(integer i=0; i<LINE_SIZE; i++)
        cache_mem[mem_rd_set_addr][mem_fifo_len][i] <= mem_rd_line[i];
    way_length[mem_rd_set_addr] <= mem_fifo_len + 1;
end

```

若set内已经满，则此时需要将队首指针对应的块换出并将新块换入。同时使队首指针+1模WAY\_CNT。队首指针的更新在状态SWAP\_IN\_OK中进行。

```

else begin
    for(integer i=0; i<LINE_SIZE; i++)
        cache_mem[mem_rd_set_addr][mem_fifo_pos][i] <= mem_rd_line[i];
    cache_fifo[mem_rd_set_addr] <= (mem_fifo_pos+1)%WAY_CNT;
end

```

## LRU

为实现LRU，为每个set维持一个队列以及队列长度。其中队头的块号是最近使用过的，队尾的块号是最近最少使用的。每当访问一个块(读、写)时，就将其块号放到队列的最前端。则队尾的块一定是要被换出的块。当队列非满时不进行换出。对队列的更新在IDLE状态且cache\_hit的情况下进行。

更新队列的部分代码如下,其中lru\_stack就是为每个set维持的队列。

```

if(cache_hit) begin
    if(rd_req) begin // 如果cache命中，并且是读请求，
        rd_data <= cache_mem[set_addr][hit_pos][line_addr];
        //则直接从cache中取出要读的数据
        for(integer i=1; i<WAY_CNT; i++) begin //update stack
            if(i>hit_pos) begin
                break; //只更新在hit_pos前的块在队列中的位置
            end
            else begin
                lru_stack[set_addr][i] <= lru_stack[set_addr][i-1];
                //位置向下移
            end
        end
        lru_stack[set_addr][0] <= hit_pos; //放在队首
    end else if(wr_req) begin // 如果cache命中，并且是写请求，
        cache_mem[set_addr][hit_pos][line_addr] <= wr_data;
        // 则直接向cache中写入数据
        dirty[set_addr][hit_pos] <= 1'b1;
        // 写数据的同时置脏位
        for(integer i=1; i<WAY_CNT; i++) begin
            if(i>hit_pos) begin
                break; //只更新在hit_pos前的块在队列中的位置
            end
            else begin
                lru_stack[set_addr][i] <= lru_stack[set_addr][i-1];
                //位置向下移
            end
        end
        lru_stack[set_addr][0] <= hit_pos; //放在队首
    end
end
end

```

## 缺失率统计

对缺失率的统计在WBSEgReg中进行。

miss时cache\_miss会持续50个周期。统计时只统计一次，用状态机来实现。cache\_miss时最终会转化为不miss的情况(目标块会调入)，所以统计总次数时只统计不cache\_miss的情况

相关实现如下:

```
wire we;
assign we = |WE;
reg [31:0] miss_cnt;
reg [31:0] total_cnt;
reg state;

always@(posedge clk or posedge rst) begin
    if(rst)begin
        state <= 0;
        miss_cnt <= 0;
    end else begin
        case(state)
            1'b0:begin
                if(DCacheMiss) begin
                    miss_cnt <= miss_cnt + 1;
                    state <= 1'b1;
                end
            end
            1'b1:begin
                if(!DCacheMiss) begin
                    state <= 1'b0;
                end
            end
        endcase
    end
end

always@(posedge clk or posedge rst) begin
    if(rst)begin
        total_cnt <= 0;
    end else begin
        if((MemReadM || we)&&!DCacheMiss) begin
            total_cnt <= total_cnt + 1;
        end
    end
end
```

仿真后通过miss\_cnt和total\_cnt的值就可以计算缺失率，通过最后一次访存的时间估算运行时间

同时对cache分别综合，得到不同策略，不同参数所使用的硬件相应信息。

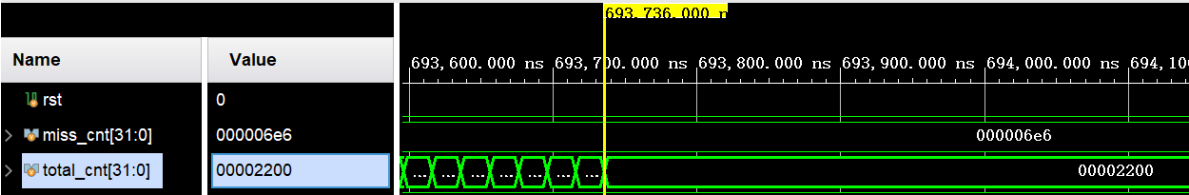
## 统计分析

对FIFO,LRU分别使用16\*16矩阵乘法，256个数的快排进行仿真。仿真时取cache参数为3, 3, 6, 3和3, 3, 6, 4。

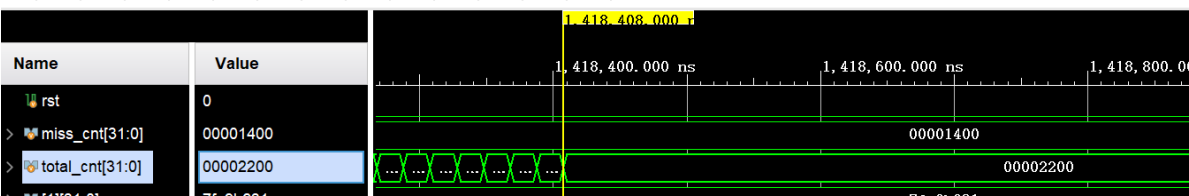
分析结果如下。

当cache的参数为3, 3, 6, 4时的结果

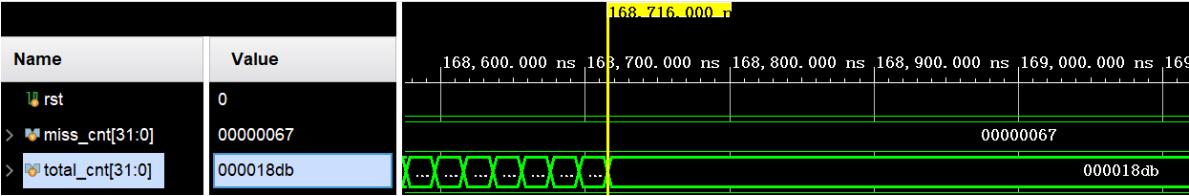
fifo,16\*16矩阵乘:



lru,16\*16矩阵乘



fifo,256个数快排



lru,256个数快排



fifo所需硬件资源

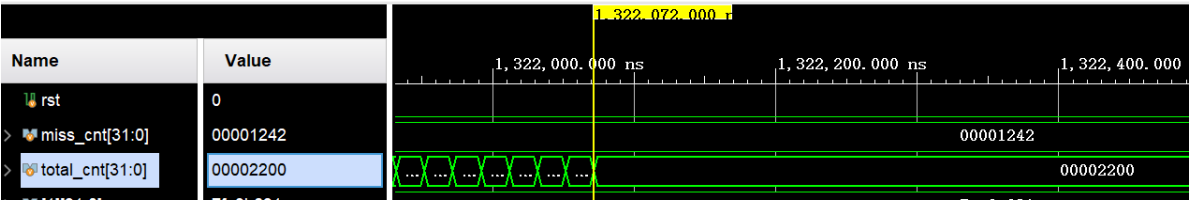
LUT	FF	BRAM	URAM	DSP	Start	Elapsed
3843	9461	4.0	0	0	5/24/21, 11:55 PM	00:01:31

lru所需硬件资源

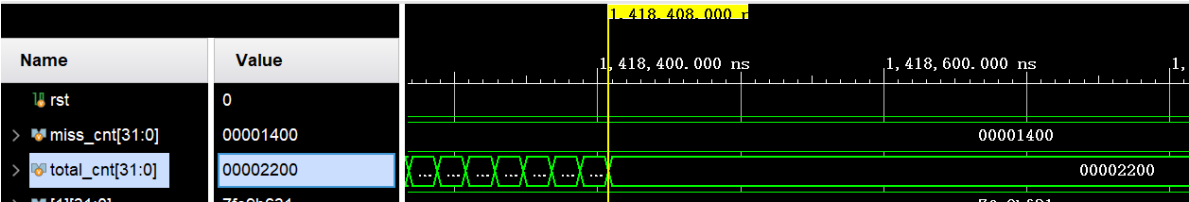
LUT	FF	BRAM	URAM	DSP	Start	Elapsed
4662	9510	4.0	0	0	5/25/21, 12:26 AM	00:01:45

当cache参数为3363时的结果:

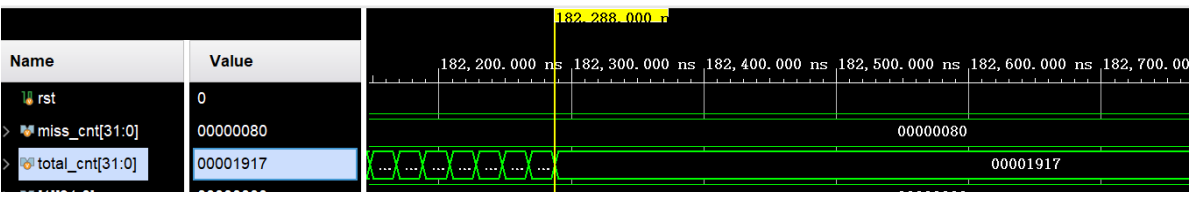
fifo,16\*16矩阵乘:



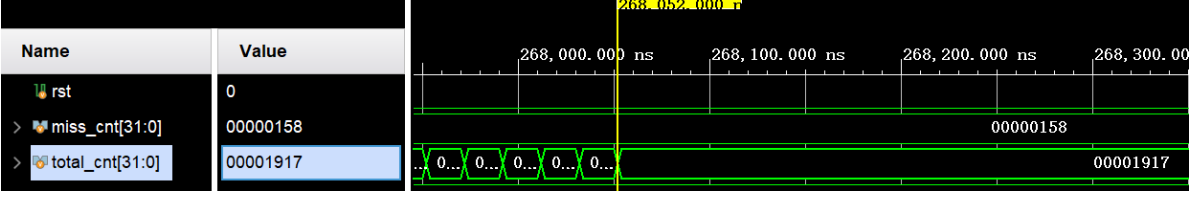
lru,16\*16矩阵乘



fifo,256个数快排



lru,256个数快排



fifo所需硬件资源

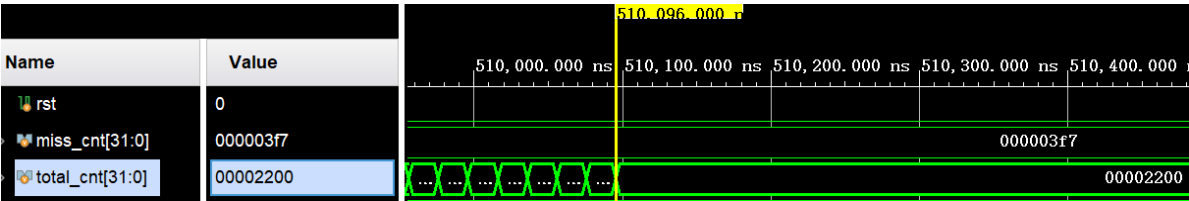
LUT	FF	BRAM	URAM	DSP	Start	Elapsed
2946	7336	4.0	0	0	5/24/2	00:01:25

lru所需硬件资源

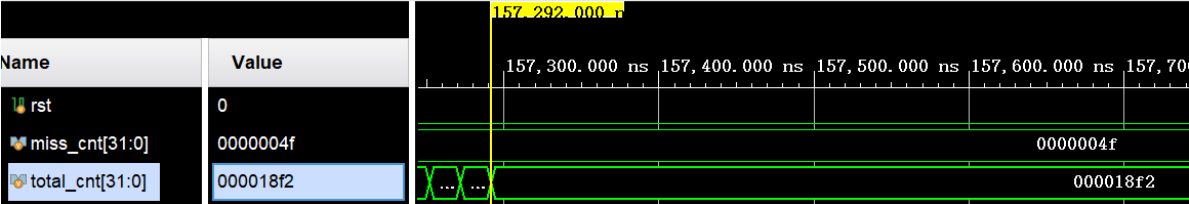
LUT	FF	BRAM	URAM	DSP	Start	Elapsed
3036	7372	4.0	0	0	5/25/21, 12:23 AM	00:01:15

当参数为3365时:

fifo,16\*16矩阵乘



fifo 256个数快排



fifo所需硬件资源

LUT	FF	BRAM	URAM	DSP
5048	11593	4.0	0	0

将上述结果制成表格

策略	参数	硬件资源(LUT,FF)	算法	运行时间(ns)	缺失率
FIFO	3364	3843,9461	MatMul 16*16	693,736	20.29%
LRU	3364	4662,9510	MatMul 16*16	1,418,408	58.82%
FIFO	3364	3843,9461	QuickSort 256	168,716	1.62%
LRU	3364	4662,9510	QuickSort 256	259,948	5.14%
FIFO	3363	2946,7336	MatMul 16*16	1,322,072	53.70%
LRU	3363	3036,7372	MatMul 16*16	1,418,408	58.82%
FIFO	3363	2946,7336	QuickSort 256	182,288	1.99%
LRU	3363	3036,7372	QuickSort 256	268,052	5.36%
FIFO	3365	5048,11593	MatMul 16*16	510,096	11.66%
FIFO	3365	5048,11593	QuickSort 256	157,292	1.24%

通过以上分析可发现，当ram\_cell地址长度不变，cache内组相联度增加时，所需硬件资源会较大幅增加。当组相联度从3变到4时，所需硬件资源增加，但缺失率和运行时间有明显的下降。从4到5时，所需硬件资源进一步增加，miss率和运行时间也有所下降，但没有3-4下降得明显。综合考虑到硬件成本和运行时间，若保持cache前三个参数为336，则cache最佳参数应为3364。此时可在控制一定硬件成本的情况下使得缺失率也较低。

