Computer system III Lab 2 Report

3200105787 张云策

一. 实验目的

- 理解cache在CPU中的作用。
- 了解cache与流水线和内存的交互机制。
- 理解存储层次（Memory Hierarchy）。

二. 实验模块

- Cache：
    - D-cache：

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 2022/03/29 11:57:25
// Design Name:
// Module Name: Cache
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module DCache(
    input clk,
    input rst,
    input [6:0] debug_cache_index,
    output reg [31:0] debug_cache_out0,
    output reg [31:0] debug_cache_out1,
    output reg [31:0] debug_cache_out2,
    input [31:0] cache_req_addr,
    input [31:0] cache_req_data,
    input cache_req_wen,
    input cache_req_valid,
    output reg [31:0] cache_resp_data,
    output reg cache_resp_stall,
    output reg [31:0] mem_req_addr,
    output reg [31:0] mem_req_data,
```

```
    output reg mem_req_wen,
    output reg mem_req_valid,
    input [31:0] mem_resp_data,
    input mem_resp_valid
    );
    reg [31:0] cache_data[0:127];
    reg [22:0] cache_tag[0:127];
    reg [6:0] cache_index[0:127];
    reg [1:0] cache_offset[0:127];
    reg cache_V[0:127];
    reg cache_D[0:127];
    reg hit,miss,read,write;
    integer i;

    always@(*)begin
        debug_cache_out0<=cache_data[debug_cache_index];
        debug_cache_out1<=
{cache_tag[debug_cache_index],cache_index[debug_cache_index
],cache_V[debug_cache_index],cache_D[debug_cache_index]};
    end
    always@(negedge clk or posedge rst)begin
        if(rst==1)begin
            cache_resp_data<=0;
            cache_resp_stall<=0;
            mem_req_addr<=0;
            mem_req_data<=0;
            mem_req_wen<=0;
            mem_req_valid<=1'b0;
            miss<=0;
            hit<=0;
            read<=0;
            write<=0;
           for (i = 0; i < 128; i = i + 1) begin
            cache_data[i]<= 0;
            cache_tag[i]<= 0;
            cache_index[i]<=i;
            cache_V[i]<=0;
            cache_D[i]<=0;
            cache_offset[i]<=0;
           end
        end
        else if(cache_req_valid==1'b1)begin
            if(cache_req_wen==0)begin
                read<=1;
                write<=0;
                if(cache_req_addr==
{cache_tag[cache_req_addr[8:2]],cache_req_addr[8:2],cache_o
ffset[cache_req_addr[8:2]]}&&cache_V[cache_req_addr[8:2]]==
1'b1)begin//hit
                    hit<=1;
                    miss<=0;

 cache_resp_data<=cache_data[cache_req_addr[8:2]];
                    cache_resp_stall<=0;
                    mem_req_wen<=0;
                    mem_req_valid<=1'b0;
                end
                else begin
```

```
                    miss<=1;
                    hit<=0;

 if(cache_D[cache_req_addr[8:2]]==1'b1)begin
                        if(mem_resp_valid==1'b1) begin
                            cache_resp_stall<=1;
                            mem_req_wen<=0;
                            mem_req_valid<=1'b1;
                            mem_req_addr<=cache_req_addr;
                            cache_D[cache_req_addr[8:2]]
<=1'b0;
                        end
                        else begin
                            cache_resp_stall<=1;
                            mem_req_addr<=
{cache_tag[cache_req_addr[8:2]],cache_req_addr[8:2],cache_o
ffset[cache_req_addr[8:2]]};

 mem_req_data<=cache_data[cache_req_addr[8:2]];
                            mem_req_wen<=1'b1;
                            mem_req_valid<=1'b1;
                        end
                    end
                    else begin
                        if(mem_resp_valid==1'b1) begin
                            cache_resp_data<=mem_resp_data;
                            cache_resp_stall<=1'b0;
                            mem_req_wen<=1'b0;
                            mem_req_valid<=1'b0;
                            cache_data[cache_req_addr[8:2]]
<=mem_resp_data;
                            cache_tag[cache_req_addr[8:2]]
<=cache_req_addr[31:9];

 cache_offset[cache_req_addr[8:2]]<=cache_req_addr[1:0];
                            cache_V[cache_req_addr[8:2]]
<=1'b1;
                        end
                        else begin
                            cache_resp_stall<=1;
                            mem_req_addr<=cache_req_addr;
                            mem_req_wen<=1'b0;
                            mem_req_valid<=1'b1;
                        end
                    end
                end
            end
            else begin
                read<=0;
                write<=1;
                if((cache_req_addr==
{cache_tag[cache_req_addr[8:2]],cache_req_addr[8:2],cache_o
ffset[cache_req_addr[8:2]]}&&cache_V[cache_req_addr[8:2]]==
1'b1)||cache_D[cache_req_addr[8:2]]==1'b0)begin//hit
                    hit<=1;
                    miss<=0;
                    cache_resp_stall<=1'b0;
                    mem_req_wen<=1'b0;
```

```
                    mem_req_valid<=1'b0;
                    cache_data[cache_req_addr[8:2]]
<=cache_req_data;
                    cache_tag[cache_req_addr[8:2]]
<=cache_req_addr[31:9];
                    cache_offset[cache_req_addr[8:2]]
<=cache_req_addr[1:0];
                    cache_V[cache_req_addr[8:2]]<=1'b1;
                    cache_D[cache_req_addr[8:2]]<=1'b1;
                end
                else begin
                    miss<=1;
                    hit<=0;
                    if(mem_resp_valid==1'b1) begin
                        cache_resp_stall<=1'b0;
                        mem_req_wen<=1'b0;
                        mem_req_valid<=1'b0;
                        cache_data[cache_req_addr[8:2]]
<=cache_req_data;
                        cache_tag[cache_req_addr[8:2]]
<=cache_req_addr[31:9];
                        cache_offset[cache_req_addr[8:2]]
<=cache_req_addr[1:0];
                        cache_V[cache_req_addr[8:2]]<=1'b1;
                        cache_D[cache_req_addr[8:2]]<=1'b1;
                    end
                    else begin
                        cache_resp_stall<=1;
                        mem_req_addr<=
{cache_tag[cache_req_addr[8:2]],cache_req_addr[8:2],cache_o
ffset[cache_req_addr[8:2]]};

 mem_req_data<=cache_data[cache_req_addr[8:2]];
                        mem_req_wen<=1'b1;
                        mem_req_valid<=1'b1;
                    end
                end
            end
        end
    end
endmodule
```

- I-cache:

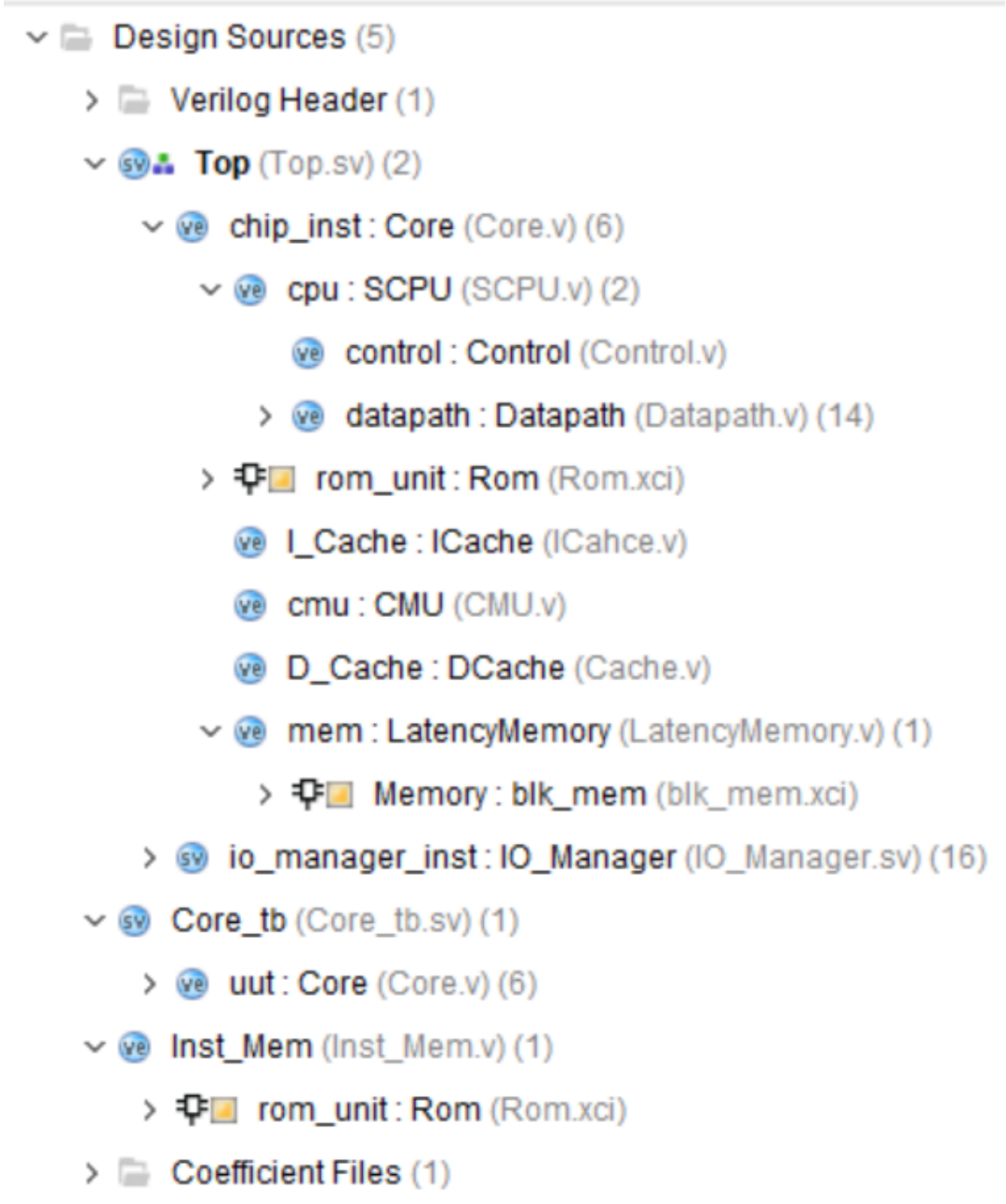
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 2022/03/29 12:27:42
// Design Name:
// Module Name: Cache
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
```

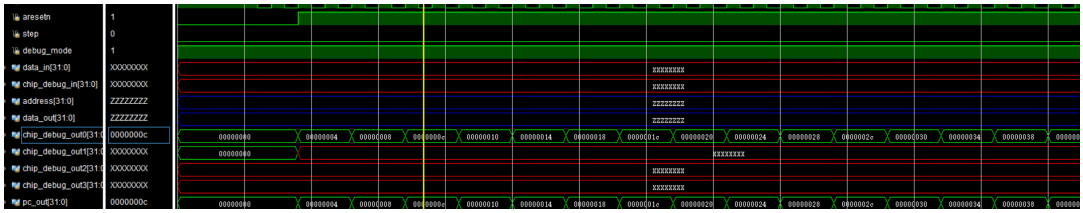
```
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module ICache(
    input [31:0] cache_req_addr,
    output reg [31:0] cache_resp_data,
    output reg [31:0] mem_req_addr,
    input [31:0] mem_resp_data
    );
    reg [31:0] cache_data[0:127];
    reg [22:0] cache_tag[0:127];
    reg [6:0] cache_index[0:127];
    reg [1:0] cache_offset[0:127];
    reg cache_V[0:127];
    reg cache_D[0:127];
    integer i;
    initial begin
        cache_resp_data<=0;
        for (i = 0; i < 128; i = i + 1) begin
             cache_data[i] <= 0;
             cache_tag[i] <= 0;
             cache_index[i]<=i;
             cache_V[i]<=0;
             cache_D[i]<=0;
             cache_offset[i]<=0;
        end
    end
    always@(*)begin
        if(cache_req_addr==
{cache_tag[cache_req_addr[8:2]],cache_req_addr[8:2],cache_of
fset[cache_req_addr[8:2]]}&&cache_V[cache_req_addr[8:2]]==1'
b1)begin//hit

 cache_resp_data<=cache_data[cache_req_addr[8:2]];
        end
        else begin//miss
            cache_resp_data<=mem_resp_data;
            cache_data[cache_req_addr[8:2]]<=mem_resp_data;
            cache_tag[cache_req_addr[8:2]]
<=cache_req_addr[31:9];
            cache_offset[cache_req_addr[8:2]]
<=cache_req_addr[1:0];
            cache_V[cache_req_addr[8:2]]<=1'b1;
            cache_D[cache_req_addr[8:2]]<=1'b0;
        end
    end
endmodule
```

- Cache内部交互设计：

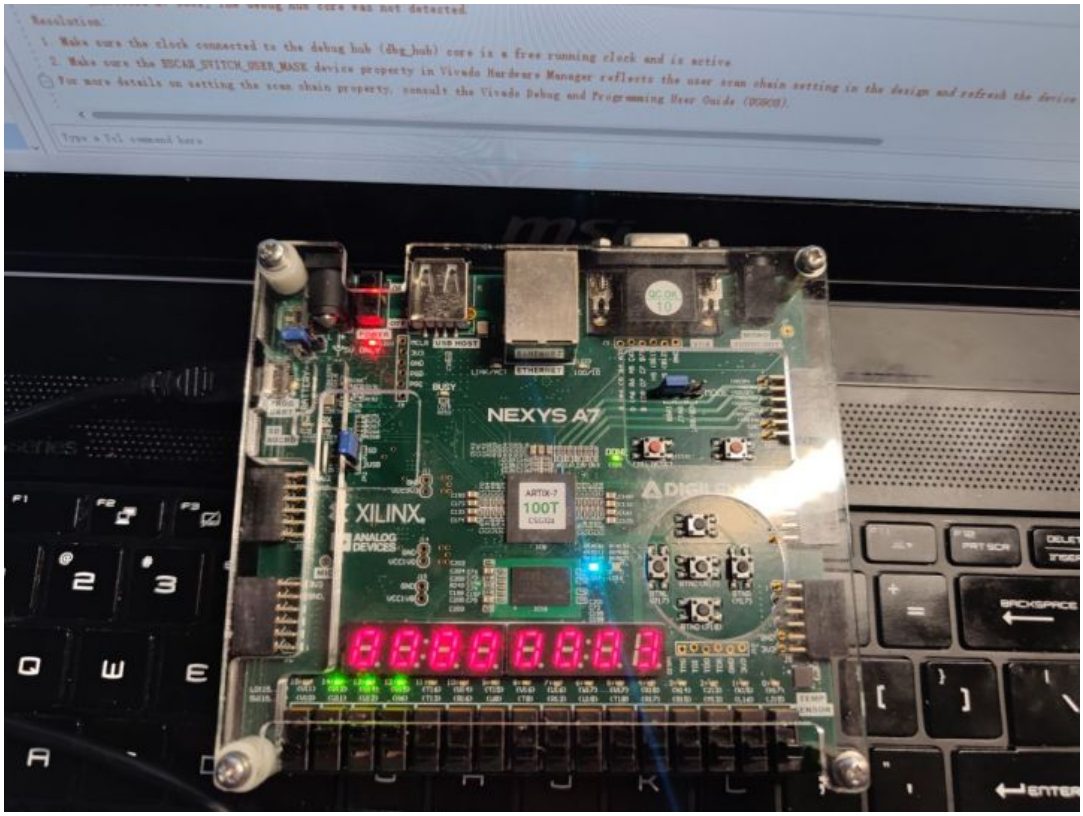
```
module CMU(
    input [6:0] op_code,
    output reg cache_req_wen,
    output reg cache_req_valid
    );
    always @(*) begin
          case(op_code)
              7'b0100011:begin cache_req_wen <= 1'b1;cache_req_valid <=1'b1; end //SW
              7'b0000011:begin cache_req_wen <= 1'b0;cache_req_valid <=1'b1; end //LW
              default:begin cache_req_valid<=1'b0; cache_req_wen <= 1'b0;end
          endcase
      end
endmodule
```

- 仿真:





- 上板：



三. 实验思考题

- 由题意得： Block Memory为2^{13} = 8192KB，所以地址位占13；而Cache中存在2^{9}=512个block，所以index寻址位占9位；标定tag占4位；而valid bit和dirty bit各占一位，故得出：

  | 4bit | 9bit | 0bit |

  |——tag——|———index———|—offset—|

- D-cache：

```
Begin:
    # Reg[1-5]: Used to record data
0x00    addi    x1,     x0,     123         # x1 = 123  Data
0x04    addi    x2,     x1,     111         # x2 = 234  Data
0x08    addi    x3,     x2,     111         # x3 = 345  Data
0x0c    addi    x4,     x3,     111         # x4 = 456  Data
0x10    addi    x5,     x4,     111         # x5 = 567  Data
    # Reg[6-10]: Used to store test address
0x14    addi    x6,     x0,     0x0
0x18    addi    x7,     x0,     0x4
0x1c    addi    x8,     x0,     0x94
0x20    addi    x9,     x0,     0x98
0x24    addi    x10,    x0,     0x404
    # Reg[20]: Used to record score
0x28    addi    x20,    x0,     0           # x20 = 0   以上均与D-cache功能无关
Test_Hit:
```

```
0x2c    sw      x1,     0(x6)               # hit 写
0x30    sw      x2,     0(x7)               # hit 写
0x34    lw      x11,    0(x6)               # hit
0x38    sw      x3,     0(x8)               # hit 写
0x3c    sw      x4,     0(x9)               # hit 写
0x40    lw      x13,    0(x8)               # hit
0x44    lw      x12,    0(x7)               # hit
0x48    lw      x14,    0(x9)               # hit
0x4c    sub     x11,    x11,    x13         # 无关
0x50    sub     x12,    x12,    x14         # 无关
0x54    bne     x11,    x12,    Test_Replacement    # 无关
0x58    addi    x20,    x20,    1           # 无关
Test_Replacement:
0x5c    sw      x5,     0(x10)              # miss，写缺失，由dirty-bit处理
0x60    lw      x11,    0(x7)               # miss，读缺失
0x64    lw      x12,    0(x10)              # miss，读缺失
0x68    addi    x11,    x11,    333         # 无关
0x6c    bne     x11,    x12,    Test_Dirty_Block    # 无关
0x70    addi    x20,    x20,    1           # 无关
Test_Dirty_Block:
0x74    lw      x11,    0(x6)               # miss，读缺失
0x78    lw      x12,    0(x10)              # miss，读缺失
0x7c    addi    x11,    x11,    444         # 无关
0x80    bne     x11,    x12,    End          #无关
0x84    addi    x20,    x20,    1           # 无关
End:
    #------------Useless Loop-----------------
0x88    addi    x21,    x0,     3
0x8c    bne     x21,    x20,    End
    #------------Useless Loop-----------------
```