

# 实验报告

## cache 设计实验

姓名： 陈锦赐

学号： 141220008

14 级计算机科学与技术系 1 班

邮箱： njucjc@163.com

时间： 2017 年 1 月 日 星期日

## 一、实验目的

1. 了解 cache 的内部构造
2. 掌握 cache 控制器的原理及其设计方法
3. 了解 cache 替换算法
4. 学会设计有限状态机

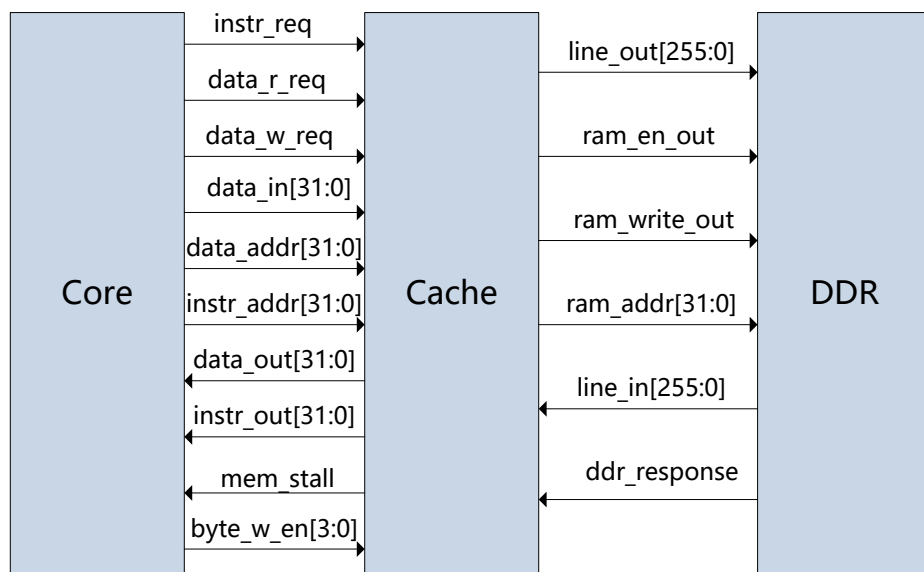
## 二、实验环境

1. 装有 Vivado 的计算机一台
2. Nexy4 开发板

## 三、实验原理

### 1. Cache 的对外接口

#### (1) 处理器-Cache-DDR 之间的接口图

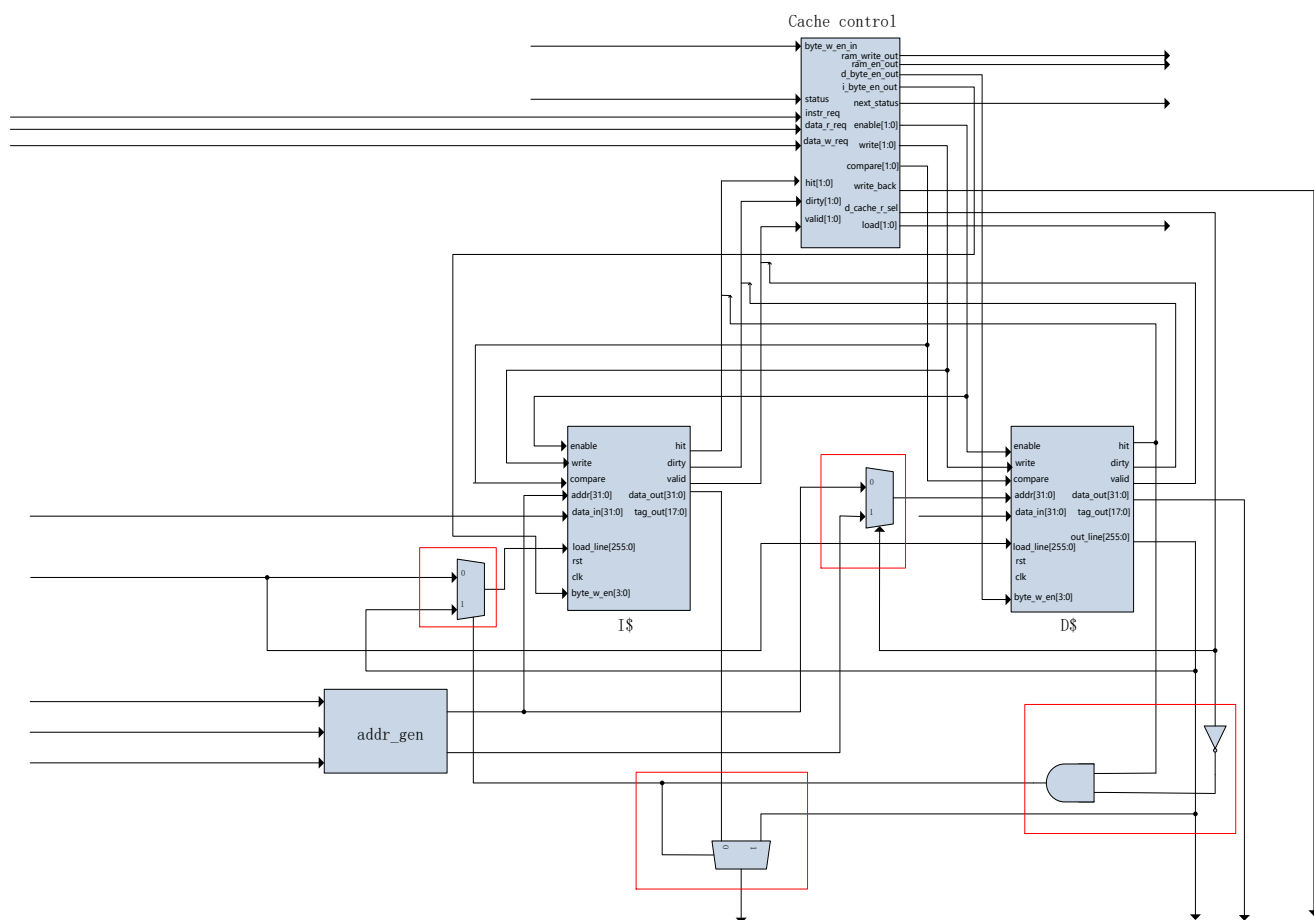


## (2) 信号含义说明表

信号	方向	宽度	信号说明
instr_req	core→cache	1	来自 core 的读指令请求
data_r_req	core→cache	1	来自 core 的读数据请求
data_w_req	core→cache	1	来自 core 的写数据请求
byte_w_en	core→cache	4	来自 core 的数据字节写使能
instr_addr	core→cache	32	来自 core 的指令地址
data_addr	core→cache	32	来自 core 的读/写数据地址
data_in	core→cache	32	来自 core 的写入数据
ddr_response	ddr→cache	1	来自 DDR 的访存反馈
line_in	ddr→cache	255	ddr 载入 cache 的数据块
mem_stall	cache→core	1	标识当前还有来自 core 的请求未完成
data_out	cache→core	32	从 cache 读取的数据
instr_out	cache→core	32	从 cache 读取的指令
ram_en_out	cache→ddr	1	Cache 发送给 DDR 的信号，表示要读写 DDR
ram_write_out	cache→ddr	1	1 表示写 DDR，0 表示读 DDR

## 1. cache 顶层模块

### (1) 顶层模块图

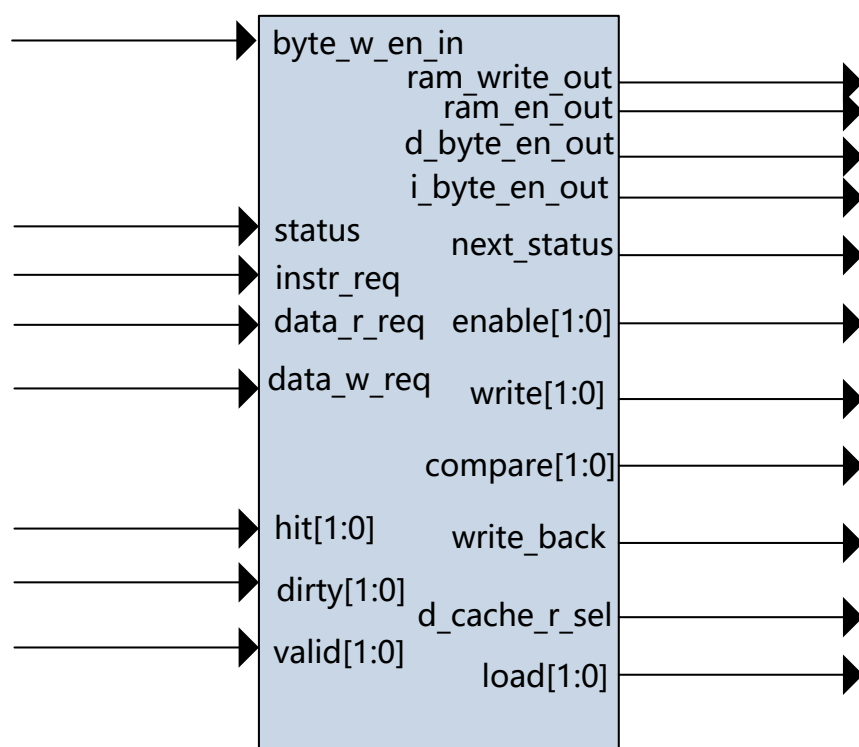


上图红色方框为处理I\$ read miss时先读D\$的逻辑图

注：上图主要分为三个模块：cache\_control、i\_cache、d\_cache 分别表示 cache 控制器、指令 cache、数据 cache

## (2) cache 控制器

### i) 控制器接口图

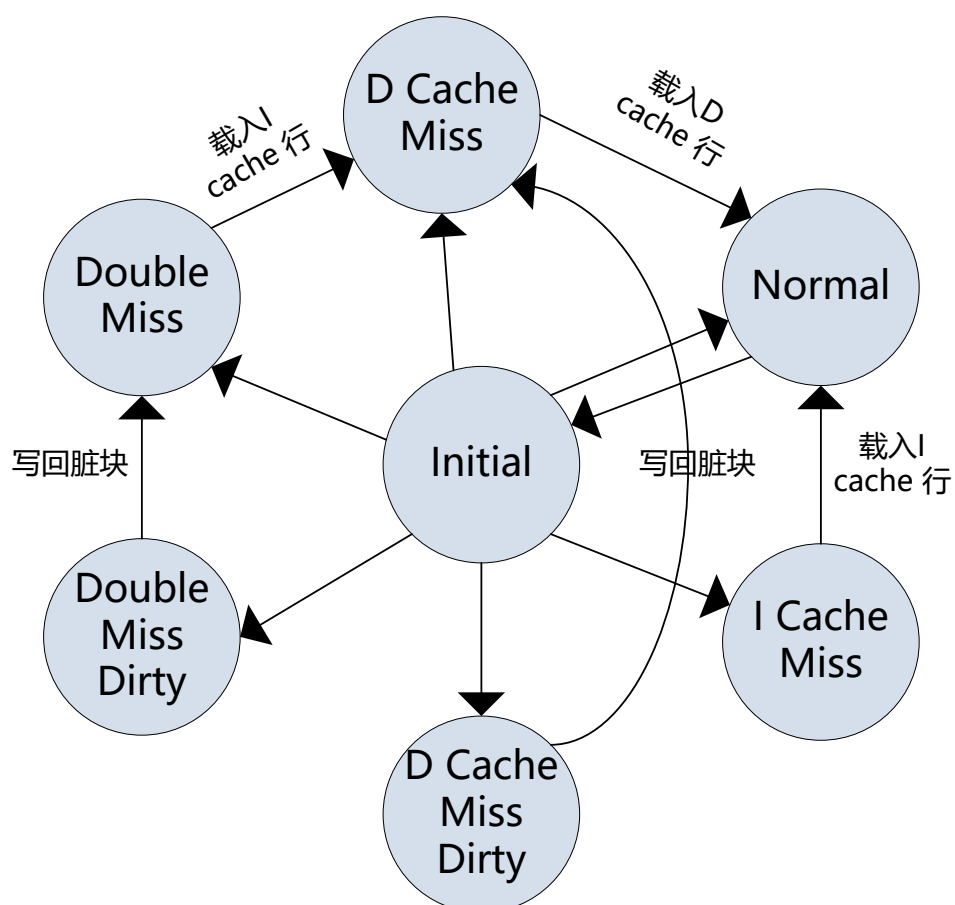


### ii) 控制器接口信号说明

信号名	类型	宽度	信号说明
instr_req	input	1	读指令请求
data_r_req	input	1	读数据请求
data_w_req	input	1	写数据请求
hit	input	2	cache 读写是否命中 (i/d_cache)
dirty	input	2	cache 行是否为脏
valid	input	2	cache 行是否有效
byte_w_en_in	input	4	字节写使能
status	input	3	当前 cache 状态
enable	output	2	i/d_cache 使能信号
write	output	2	i/d_cache 读写信号

compare	output	2	结合 write 产生读写模式
i_byte_w_en_out	output	4	i_cache 字节写使能
d_byte_w_en_out	output	4	d_cache 字节写使能
load	output	2	请求载入 cache 行(cache 内部信号)
ram_en_out	output	1	DDR 使能
ram_write_out	output	1	DDR 读写(1:写 0:读)
d_cache_r_sel	output	1	d_cache 行载入端选择(i_cache 或 ddr)
next_status	output	3	转移的下一个状态

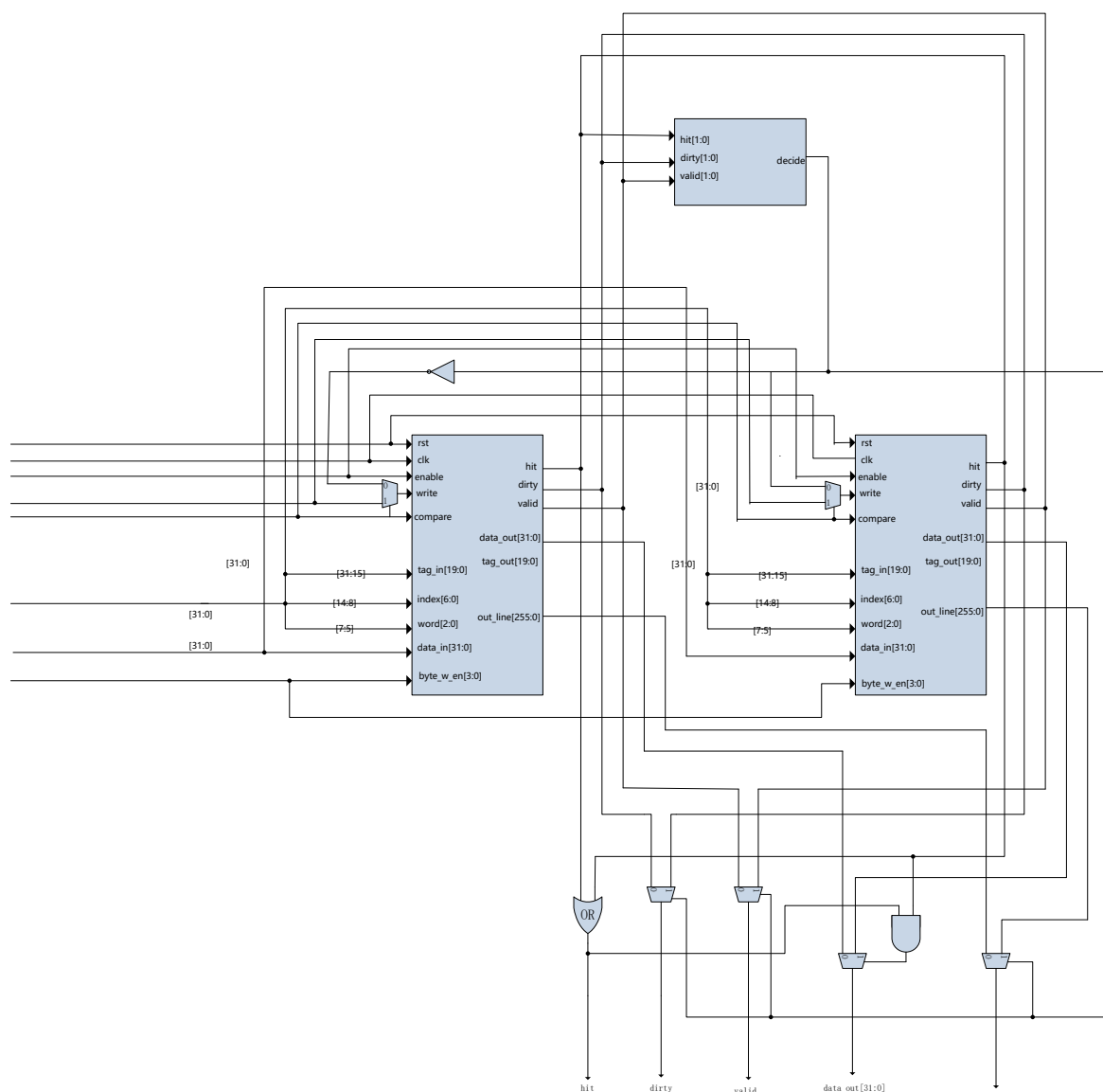
iii) 控制器状态转换图



注:状态信号取值见随附表格

### (3) i\_cache 和 d\_cache 设计

#### i) 二路组相联 cache 图



tag\_out、data\_out、valid、dirty、的选择  
依赖于victim\_line的选择逻辑decide

注：上图为一个二路组相联 cache，其内部构造为两个 128 行每  
行 256 个字节的直接相联 cache 并联拼接而成。

## ii) 信号说明

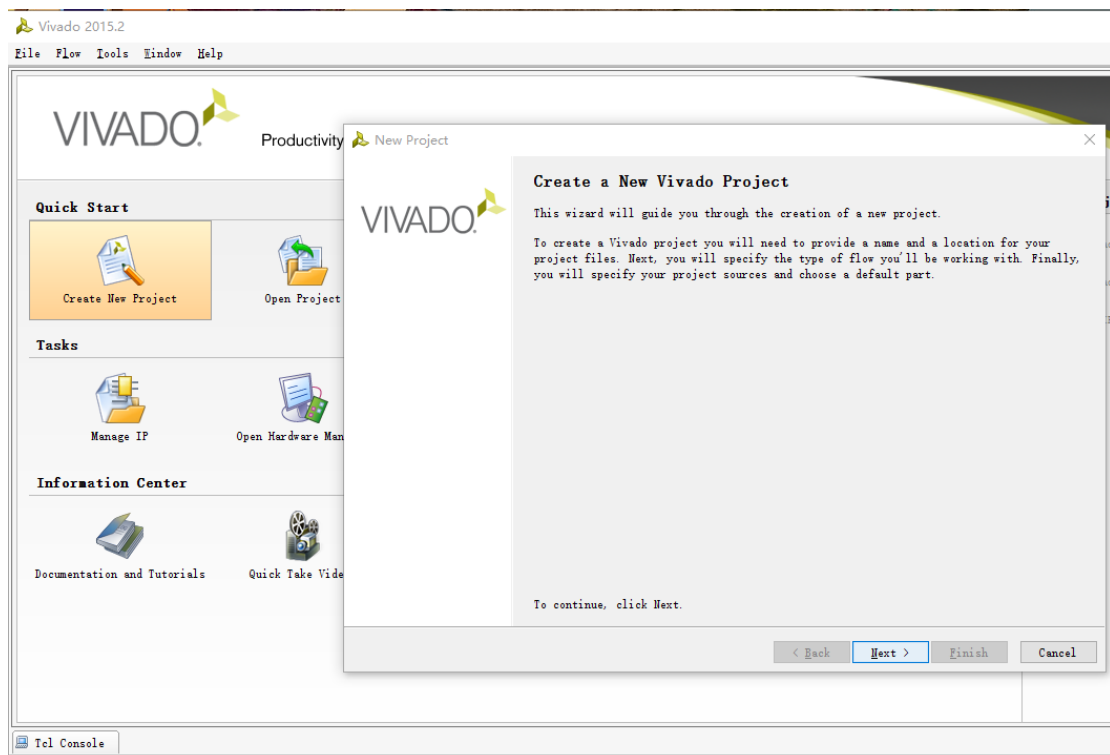
基本信号与顶层模块大致相同，故仅说明几个内部信号

信号	类型	宽度	信号说明
tag_in	input	20	tag 字段
index	input	6	cache 组索引
word	input	3	cache 行内字索引
victim		1	cache 替换行组内行索引


## 四、实验步骤

### 1. cache 仿真

#### (1) 新建测试工程





 New Project ✕

**Project Name**


Enter a name for your project and specify a directory where the project data files will be stored.

Project name:

Project location:

☒ Create project subdirectory

Project will be created at: E:/digital/cache\_test

 New Project ✕

**Default Part**

Choose a default Xilinx part or board for your project. This can be changed later.

Select: ☒ Parts ☐ Boards

Filter






Product category:  Package:

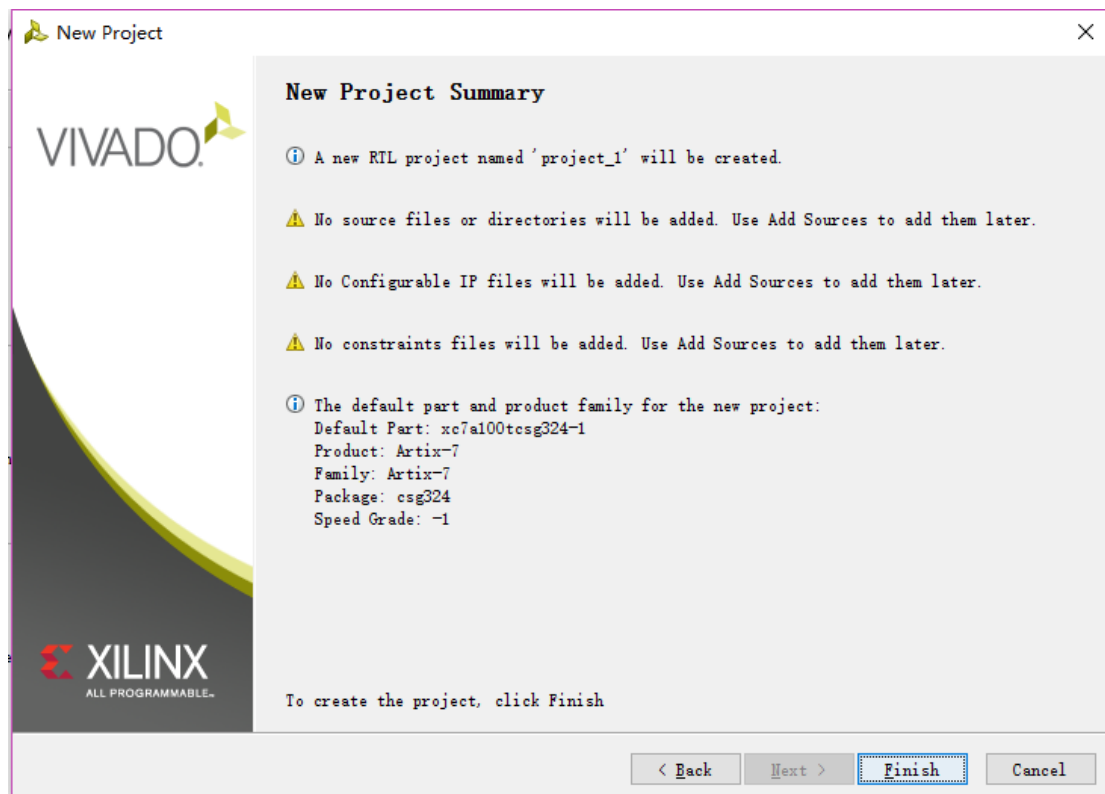
Family:  Speed grade:

Sub-Family:  Temp grade:

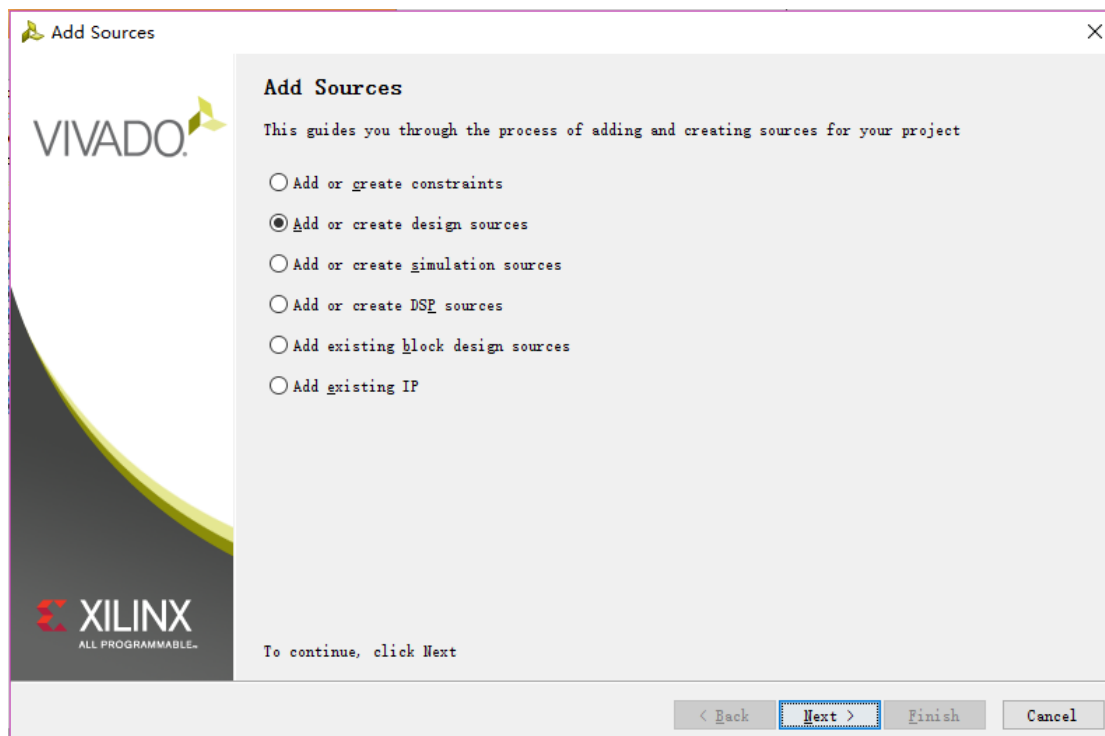
Si Revision:

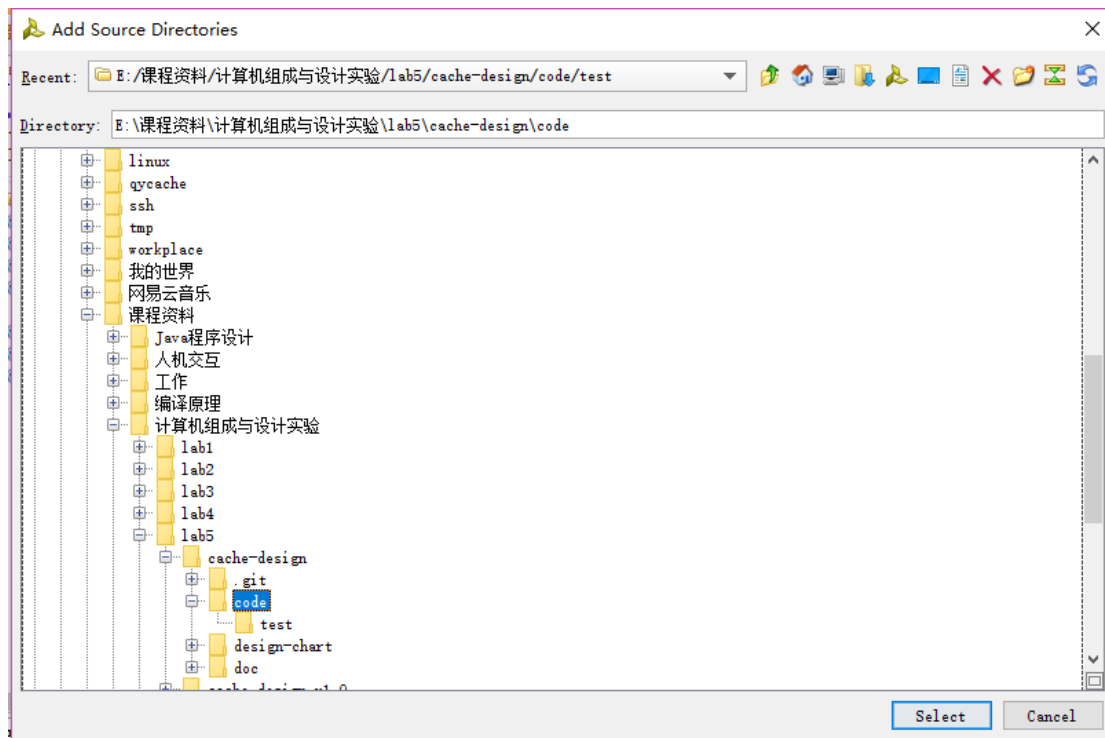
Search:

Part	I/O Pin Count	Available IOBs	LUT Elements	FlipFlops	Block RAMs	DSPs	Gb Transceivers	GIPE2 Trans
 xc7a15tcs324-1	324	210	10400	20800	25	45	0	0
 xc7a35tcs324-1	324	210	20800	41600	50	90	0	0
 xc7a50tcs324-1	324	210	32600	65200	75	120	0	0
 xc7a75tcs324-1	324	210	47200	94400	105	180	0	0
 xc7a100tcs324-1	324	210	63400	126800	135	240	0	0



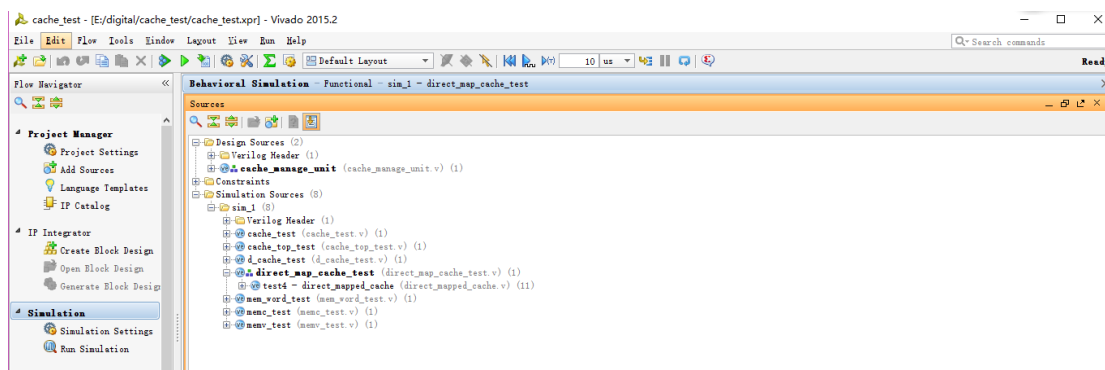
## (2) 添加源文件及测试文件





点击 Select 选择添加

最终模块结构如下：



### (3) 进行仿真

注：测试仿真过程自底向上地进行

#### i) direct\_mapped\_cache(直接相联 cache)

测试代码：

```
1  `timescale 1ns / 1ps
2
3  module direct_map_cache_test();
4      reg clk;
5      reg rst;
6      reg enable;
7      reg write;
8      reg compare;
9      // input load,
10     reg [19:0] tag_in;
11     reg [6:0] index;
12     reg [2:0] word;
13     reg [31:0] data_in;
14     reg [255:0] line_in;
15     reg [3:0] byte_w_en;
16
17     wire hit;
18     wire dirty;
19     wire valid;
20     wire [19:0] tag_out;
21     wire [31:0] data_out;
22     wire [255:0] line_out;
23
```

```
24     direct_mapped_cache test4(
25         .clk(clk),
26         .rst(rst),
27         .enable(enable),
28         .write(write),
29         .compare(compare),
30         .tag_in(tag_in),
31         .index(index),
32         .word(word),
33         .data_in(data_in),
34         .line_in(line_in),
35         .byte_w_en(byte_w_en),
36
37         .hit(hit),
38         .dirty(dirty),
39         .valid(valid),
40         .tag_out(tag_out),
41         .data_out(data_out),
42         .line_out(line_out));
```

```
44     initial begin
45         clk = 1;
46         rst = 0;
47         enable = 1;
48         write = 1;
49         compare = 0;
50         tag_in = 20'h01000;
51         index = 7'd128;
52         word = 0;
53         data_in = 32'h0c0c_0c0c;
54         line_in = 255'h11111111_22222222_33333333_44444444_55555555_66666666_77777777_88888888;
55         byte_w_en = 4'b0000;
56         #20;
57
58         write = 0;
59         compare = 1;
60         #20;
61
62         word = 1;
63         #20;
64
65         word = 2;
66         #20;
67
```



## ii)two\_ways\_cache(2 路组相联 cache)仿真

测试代码:

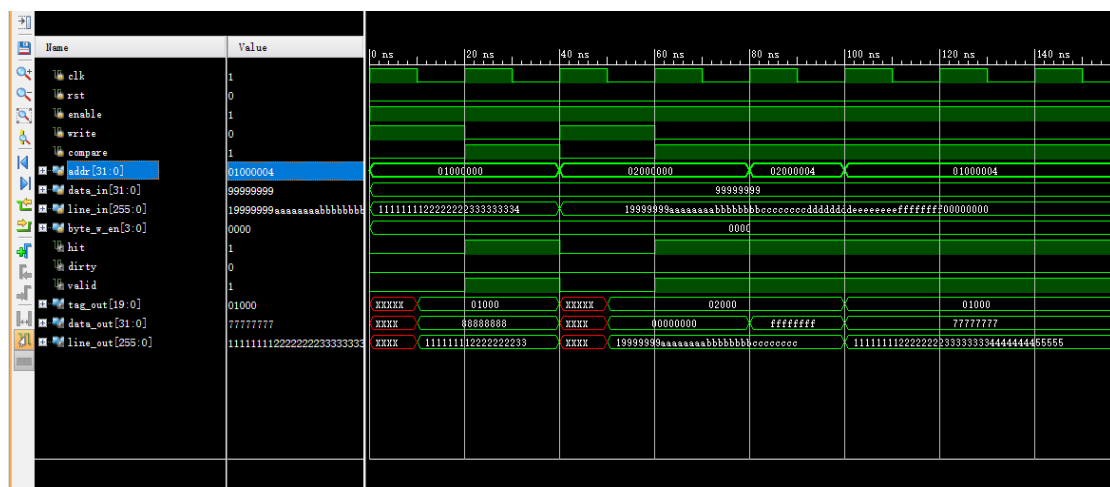
```
1 |timescale 1ns / 1ps
2
3 module d_cache_test();
4     reg clk;
5     reg rst;
6     reg enable;
7     reg write;
8     reg compare;
9     //input load,
10    reg [31:0] addr;
11    reg [31:0] data_in;
12    reg [255:0] line_in;
13    reg [3:0] byte_w_en;
14
15    wire hit;
16    wire dirty;
17    wire valid;
18    wire [19:0] tag_out;
19    wire [31:0] data_out;
20    wire [255:0] line_out;
21
22    two_ways_cache test5(
23        .clk(clk),
24        .rst(rst),
25        .enable(enable),
26        .write(write),
27        .compare(compare),
28        .addr(addr),
29        .data_in(data_in),
30        .line_in(line_in),
31        .byte_w_en(byte_w_en),
32
33        .hit(hit),
34        .dirty(dirty),
35        .valid(valid),
36        .tag_out(tag_out),
37        .data_out(data_out),
38        .line_out(line_out));
39
40    initial begin
41        clk = 1;
42        rst = 0;
43        enable = 1;
44        write = 1;
45        compare = 0;
46        addr = 32'h0100_0000;
47        data_in = 32'h9999_9999;
48        line_in = 255'h11111111_22222222_33333333_44444444_55555555_66666666_77777777_88888888;
49        byte_w_en = 4'b0000;
50        #20;
51
52
53        write = 0;
54        compare = 1;
55        addr = 32'h0100_0000;
56        #20;
57
58        write = 1;
59        compare = 0;
60        addr = 32'h0200_0000;
61        line_in = 255'h99999999_aaaaaaa_bbbbbbb_ccccccc_ddddddd_eeeeeee_ffffff_00000000;
62        #20;
```

```

65     write = 0;
66     compare = 1;
67     addr = 32'h0200_0000;
68     #20;
69
70     addr = 32'h0200_0004;
71     #20;
72
73     addr = 32'h0100_0004;
74     #20;
75 end
76
77 always begin
78     #10 clk = ~clk;
79 end
80 endmodule

```

仿真图：



### 仿真分析：

此段代码测试了一个二路组相联 cache 对同一组的两行进行  
操作的过程，构造了两次读 cache miss 从而引发 cache 行  
载入同一组的两行，之后依次读取它们的数据验证 cache 基  
本数据读写正确性。结果如下表所示：

addr	data_out
32'h0200_0000	00000000
32'h0200_0004	ffffffff
32'h0100_0004	77777777

### iii) cache\_top(l cache 和 d cache 的顶层)模块

测试代码:

```
1 | timescale 1ns / 1ps
2 |
3 | module cache_top_test();
4 |     //from CPU
5 |     reg clk;
6 |     reg rst;
7 |     reg instr_req;
8 |     reg data_r_req;
9 |     reg data_w_req;
10 |    reg [3:0] byte_w_en;
11 |    reg [31:0] instr_addr;
12 |    reg [31:0] data_addr;
13 |    reg [31:0] data_in;
14 |
15 |    //from RAM
16 |    reg ddr_response; //write back ready or load data ready
17 |    reg [255:0] line_in; //load data
18 |
19 |    //TO CPU
20 |    //output instr_response,
21 |    //output data_response,
22 |    wire mem_stall;
23 |    wire [31:0] data_out;
24 |    wire [31:0] instr_out;
25 |
26 |    //TO RAM
27 |    wire ram_en_out;
28 |    wire ram_write_out;
29 |    //output [19:0] tag_out,
30 |    wire [31:0] ram_addr;
31 |    wire [255:0] line_out;
32 |
33 | cache_top test6(
34 |     .clk(clk),
35 |     .rst(rst),
36 |     .instr_req(instr_req),
37 |     .data_r_req(data_r_req),
38 |     .data_w_req(data_w_req),
39 |     .byte_w_en(byte_w_en),
40 |     .instr_addr(instr_addr),
41 |     .data_addr(data_addr),
42 |     .data_in(data_in),
43 |     .ddr_response(ddr_response),
44 |     .line_in(line_in),
45 |
46 |     .mem_stall(mem_stall),
47 |     .data_out(data_out),
48 |     .instr_out(instr_out),
49 |     .ram_en_out(ram_en_out),
50 |     .ram_write_out(ram_write_out),
51 |     .ram_addr(ram_addr),
52 |     .line_out(line_out));
53 |
54 | initial begin
55 |     clk = 1;
56 |     rst = 0;
57 |     instr_req = 0;
58 |     data_r_req = 0;
59 |     data_w_req = 1;
60 |     byte_w_en = 4'b1111;
61 |     instr_addr = 32'h0100_0000;
62 |     data_addr = 32'h0100_0000;
63 |     data_in = 32'hffff_ffff;
64 |     ddr_response = 1'b0;
65 |     line_in = 255'h11111111_22222222_33333333_44444444_55555555_66666666_77777777_88888888;
66 |     #20;
67 |
68 |     ddr_response = 1'b1;
69 |     #40;
70 |
71 |     data_r_req = 1;
72 |     data_w_req = 0;
73 |     ddr_response = 1'b0;
74 |     #20;
```

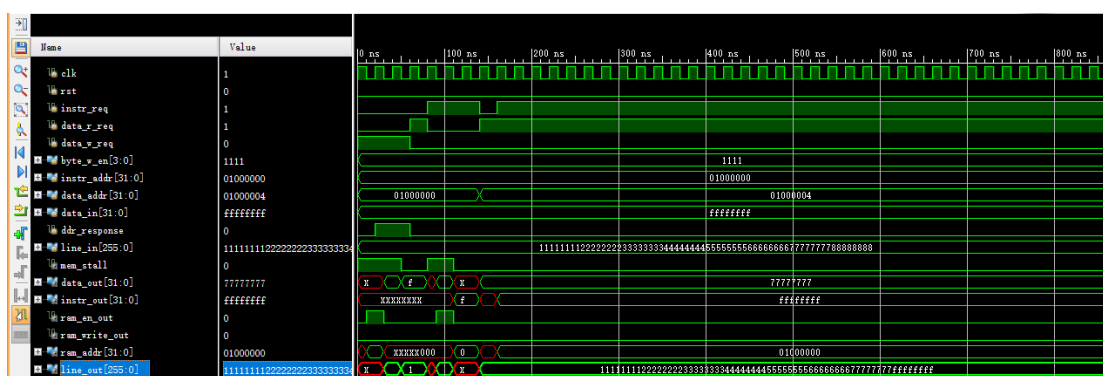


```

75
76     instr_req = 1;
77     data_r_req = 0;
78     #60;
79
80     instr_req = 0;
81     data_r_req = 1;
82     data_addr = 32'h0100_0004;
83     #20;
84
85     data_r_req = 1;
86     instr_req = 1;
87     #24;
88
89 end
90
91 always begin
92     #10 clk = ~clk;
93 end
94
95 endmodule

```

仿真图：



仿真分析：

此段测试代码测试是 cache 的最顶层模块，构造了一些 i\_cache 及 d\_cache 的缺失载入用例，以此测试 cache 行的替换算法正确性，先在 d\_cache 中载入一行以 0x01000000 为首地址的数据，然后再让 i\_cache 读这一块数据，发现 l cache miss 后重新从 d cache 中载入了该行，从而验证了 cache 一致性问题。

## 2. 生成二进制文件并用 FPGA 测试

本次上板测试所使用的方式是将已有的流水线 CPU 的 cache 模块替换为自己实现的 cache 并进行测试，主要测试内容分为两大块，一是基本的读/写数据功能，而是 cache 行的替换和写回功能，，这两大块各有一个测试文件。

### (1)基本读写功能测试

使用一段快速排序的代码进行测试，最终将排序结果打印在屏幕上。

### (2)cache 行的替换和写回功能测试

由于用的是现成的代码,故生成二进制文件和调试时没有出现太大的问题。

## 五、实验心得

本次实验认识和掌握 Cache 控制器的原理及其设计方法以及掌握 Cache 控制器的实现方法，代码实现方法。

对于 cache 的理解更加深入，工作的机制和原理比较清晰。

其中遇到了比较多的问题,逐一上网查询解决办法,并逐个进行尝试,并最终得到了解决,对于个人的信心有极大的提高。

## 六、cache 代码

见随附 code 目录文件