

Lab 6 综合设计 report

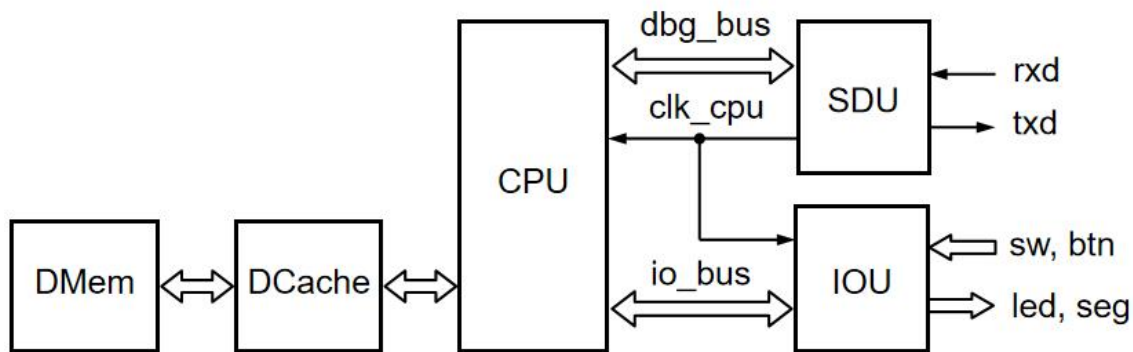
PB21111681 朱炜荣

实验目的与内容

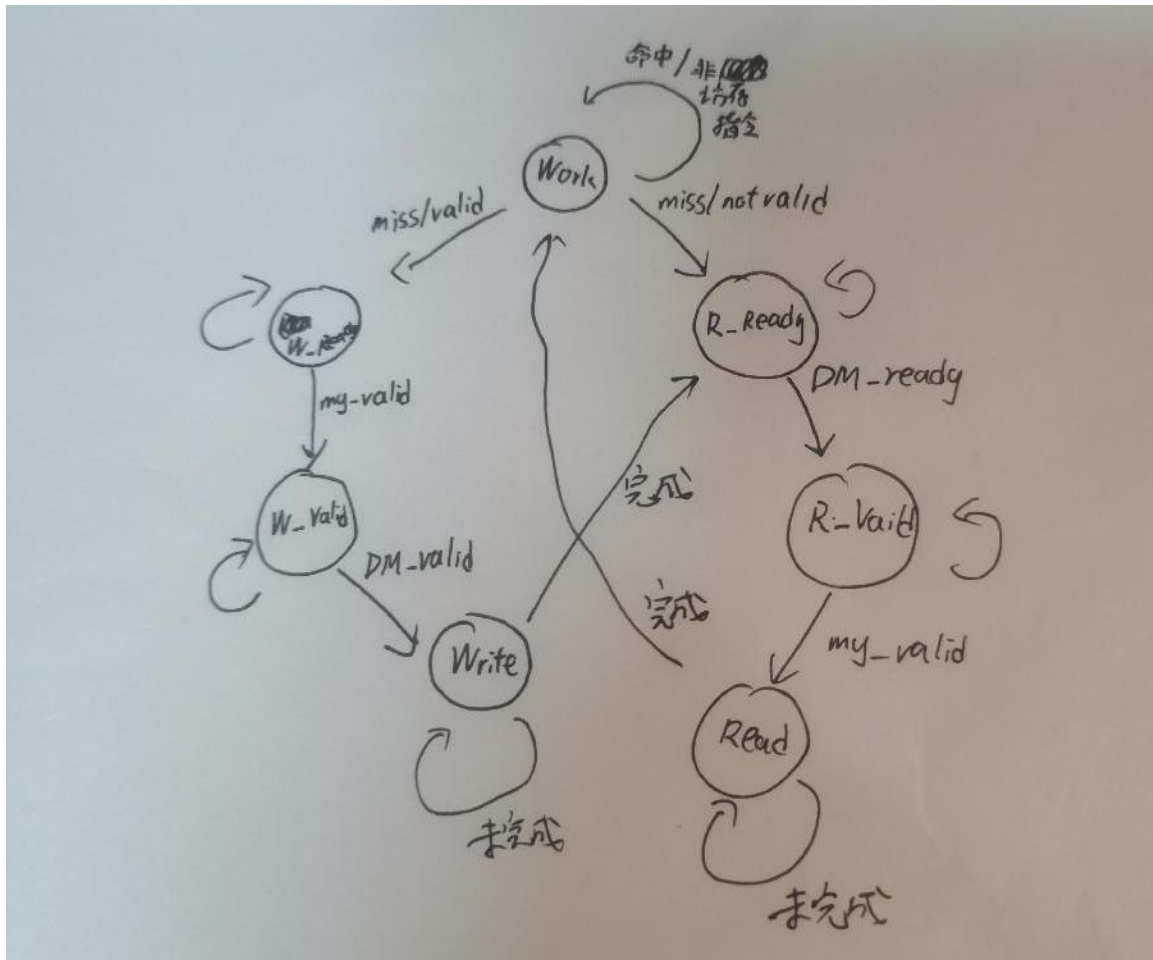
1. 学会 Cache 基本原理、结构、设计和调试方法
2. 掌握 CPU 输入/输出的编址和控制方式
3. 熟练掌握数据通路和控制器的设计和描述方法
4. 掌握有关 CPU 的 MMIO 外设输入输出的处理

逻辑设计

1. 实验的主体数据通路为：



2. DCache 中存在着 7 个状态的有限状态机



对应的 DMEM 中有着与 Dcache 对称分布（即读写交错）的有限状态机。

3. 请贴出较为核心的代码设计代码，并加以解释说明。

有关 Cache 的处理代码：

```

1. `timescale 1ns / 1ps
2. //////////////////////////////////////
3. // Company:
4. // Engineer:
5. //
6. // Create Date: 2023/06/04 17:10:49
7. // Design Name:
8. // Module Name: Dcache
9. // Project Name:
10. // Target Devices:
11. // Tool Versions:
12. // Description:
13. //
14. // Dependencies:
15. //

```

```

16. // Revision:
17. // Revision 0.01 - File Created
18. // Additional Comments:
19. //
20. ///////////////////////////////////////////////////
21.
22. //直接相联 Cache 8 行 一块4 字 共 1KB
23. module Dcache(
24.     output [63:0] debug_data,
25.     input clk,
26.     input rstn,
27.     input we_r,          //外部读信号
28.     input we_w,          //外部写信号
29.     input [31:0] addr,    //外部尝试地址
30.     input [31:0] din,     //外部输入
31.     // input [31:0] addr_exchange_in,    //用于交换的输入值地址
32.     input [31:0] data_exchange_in,      //用于交换的输入值
33.     input ready_DM,          //Mem 的 ready 信号
34.     input valid_DM,         //Mem 的 valid 信号
35.     output reg[31:0] dout,    //外部输出
36.     output reg [31:0] addr_exchange_out, //用于交换的输出值
37.     output reg [31:0] data_exchange_out, //用于交换的输出值地址
38.     output valid_DC,         //Cache 的 valid 信号
39.     output ready_DC,        //Cache 的 ready 信号
40.     output signal,          //控制流水线的信号
41.     output reg [1:0] state
42. );
43. reg [7:0] valid = 8'b0;
44. reg [24:0] tag [0:7];      //32 - 3 - 2 - 2 = 25
45. reg [127:0] cache_data [0:7];
46. reg [31:0] miss_time = 32'b0;
47. reg [31:0] ls_time = 32'b0;
48.
49. wire [1:0] offset = addr[3:2];
50. wire [2:0] mark = addr[6:4];
51. wire hit =(valid[mark] != 0) && (tag[mark] == addr[31:7]) && (we_r || we_w) && (addr < 32'h7f00) && (addr >= 32'h2000);
52. // wire [7:0] part = (offset == 2'b00)?0:(offset == 2'b01)?32:(offset == 2'b10)?64:96;
53.
54. parameter Work = 3'b000;
55. parameter Read_for_valid = 3'b001;
56. parameter Read_for_ready = 3'b010;
57. parameter Read = 3'b011;
58. parameter Write_for_valid = 3'b100;
59. parameter Write_for_ready = 3'b101;
60. parameter Write = 3'b110;
61.

```

```

62. reg [2:0] current_state = Work;
63. reg [2:0] next_state = Work;
64. reg my_ready = 0;
65. reg my_valid = 0;
66. reg done = 1;
67. reg [2:0] count = 3'b0;
68.
69. assign signal = done;
70. assign valid_DC = my_valid;
71. assign ready_DC = my_ready;
72.
73. always @(posedge clk,negedge rstn)
74. begin
75.     if(!rstn)begin
76.         current_state <= Work;
77.     end
78.     else begin
79.         current_state <= next_state;
80.     end
81. end
82.
83. always @(*)
84. begin
85.     if(!rstn)begin
86.         next_state = Work;
87.         addr_exchange_out = 32'b0;
88.         data_exchange_out = 32'b0;
89.         dout = 32'b0;
90.         state = 2'b00;
91.         done = 1;
92.     end
93.     else begin
94.         addr_exchange_out = 32'b0;
95.         data_exchange_out = 32'b0;
96.         dout = 32'b0;
97.         case (current_state)
98.             Work: begin
99.                 state = 2'b00;
100.                if(hit == 1)begin
101.                    next_state = Work;
102.                    done = 1;
103.                    if(we_r == 1)begin
104.                        case(offset)
105.                            2'b00: begin
106.                                dout = cache_data[mark][31:0];
107.                            end
108.                            2'b01:
109.                                begin

```

```

110.                dout = cache_data[mark][63:32];
111.            end
112.            2'b10:
113.            begin
114.                dout = cache_data[mark][95:64];
115.            end
116.            2'b11:
117.            begin
118.                dout = cache_data[mark][127:96];
119.            end
120.        endcase
121.    end
122.    end
123.    else begin
124.        if((we_r == 1 || we_w == 1) && (addr < 32'h7f00) && (addr >= 32'h2000)
    )begin
125.            done = 0;
126.            if(valid[mark] == 1)
127.                next_state = Write_for_ready;
128.            else
129.                next_state = Read_for_ready;
130.            end
131.        else begin
132.            next_state = Work;
133.            done = 1;
134.        end
135.    end
136.    end
137.    Read_for_ready:begin
138.        state = 2'b10;
139.        done = 0;
140.        if(valid_DM == 1)
141.            next_state = Read_for_valid;
142.        else
143.            next_state = Read_for_ready;
144.        end
145.    Read_for_valid:begin
146.        state = 2'b10;
147.        done = 0;
148.        if(my_ready == 1)
149.            next_state = Read;
150.        else
151.            next_state = Read_for_valid;
152.        end
153.    Read:begin
154.        addr_exchange_out = addr;
155.        state = 2'b10;
156.        done = 0;

```

```

157.         if(count == 3'b100)
158.             next_state = Work;
159.         else
160.             next_state = Read;
161.         end
162.         Write_for_ready:begin
163.             state = 2'b01;
164.             done = 0;
165.             if(my_valid == 1)
166.                 next_state = Write_for_valid;
167.             else
168.                 next_state = Write_for_ready;
169.             end
170.         Write_for_valid:begin
171.             state = 2'b01;
172.             done = 0;
173.             if(ready_DM == 1)
174.                 next_state = Write;
175.             else
176.                 next_state = Write_for_valid;
177.             end
178.         Write:begin
179.             state = 2'b01;
180.             done = 0;
181.             case(count)
182.                 3'b000:begin
183.                     data_exchange_out = cache_data[mark][31:0];
184.                     addr_exchange_out = {tag[mark],mark[2:0],2'b00,2'b00};
185.                 end
186.                 3'b001:begin
187.                     data_exchange_out = cache_data[mark][63:32];
188.                     addr_exchange_out = {tag[mark],mark[2:0],2'b01,2'b00};
189.                 end
190.                 3'b010:begin
191.                     data_exchange_out = cache_data[mark][95:64];
192.                     addr_exchange_out = {tag[mark],mark[2:0],2'b10,2'b00};
193.                 end
194.                 3'b011:begin
195.                     data_exchange_out = cache_data[mark][127:96] ;
196.                     addr_exchange_out = {tag[mark],mark[2:0],2'b11,2'b00};
197.                     state = 2'b11;
198.                 end
199.             endcase
200.             if(count == 3'b100)
201.                 next_state = Read_for_ready;
202.             else
203.                 next_state = Write;
204.             end

```

```

205.         default:begin
206.             next_state = Work;
207.             state = 2'b00;
208.             done = 1;
209.         end
210.     endcase
211.
212. end
213. end
214.
215.
216. always@(posedge clk,negedge rstn)
217. begin
218.     if(!rstn)begin
219.         my_valid <= 0;
220.         my_ready <= 0;
221.         count <= 3'b000;
222.         miss_time <= 32'b0;
223.         ls_time <= 32'b0;
224.     end
225.     else begin
226.         case (current_state)
227.             Work:begin
228.                 if(hit == 1) begin
229.                     ls_time <= ls_time + 1'b1;
230.                     if(we_w == 1)begin
231.                         case(offset)
232.                             2'b00: begin
233.                                 cache_data[mark][31:0] <= din;
234.                             end
235.                             2'b01:
236.                                 begin
237.                                     cache_data[mark][63:32] <= din;
238.                                 end
239.                             2'b10:
240.                                 begin
241.                                     cache_data[mark][95:64] <= din;
242.                                 end
243.                             2'b11:
244.                                 begin
245.                                     cache_data[mark][127:96] <= din;
246.                                 end
247.                         endcase
248.                     end
249.                 end
250.             else begin
251.                 if((we_r == 1 || we_w == 1) && (addr < 32'h7f00) && (addr >= 32'h2000))begin
252.                     miss_time <= miss_time + 1;

```

```

253.         if(valid[mark] == 1)begin
254.             my_valid <= 1;
255.         end
256.         else begin
257.             ;
258.         end
259.     end
260.     else begin
261.
262.     end
263.
264.     end
265. end
266. Read_for_ready:begin
267.     if(valid_DM == 1)
268.         my_ready <= 1;
269.     end
270. Read_for_valid:begin
271.     if(my_ready == 1)begin
272.         count <= 3'b0;
273.         // addr_exchange_out <= addr;
274.     end
275. end
276. Read:begin
277.     case(count)
278.         3'b001:begin
279.             cache_data[mark][31:0] <= data_exchange_in;
280.         end
281.         3'b010:begin
282.             cache_data[mark][63:32] <= data_exchange_in;
283.         end
284.         3'b011:begin
285.             cache_data[mark][95:64] <= data_exchange_in;
286.         end
287.         3'b100:begin
288.             cache_data[mark][127:96] <= data_exchange_in;
289.             tag[mark][24:0] <= addr[31:7];
290.             valid[mark] <= 1;
291.         end
292.         default:begin
293.
294.         end
295.     endcase
296.     if(count == 3'b100) begin
297.         my_ready <= 0;
298.         my_valid <= 0;
299.         count <= 0;
300.     end

```



```

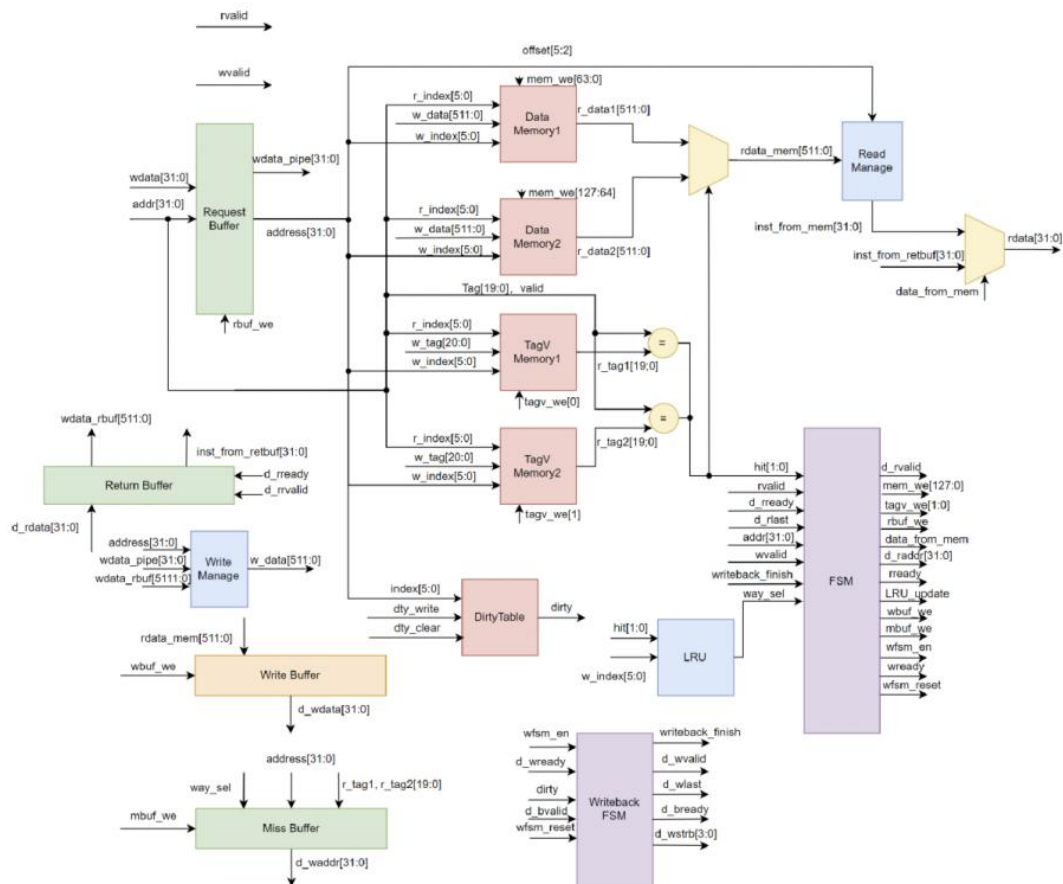
301.         else begin
302.             count <= count + 3'b1;
303.         end
304.     end
305.     Write_for_ready:begin
306.         if(my_valid == 1)begin
307.
308.             end
309.         end
310.     Write_for_valid:begin
311.         if(ready_DM == 1)begin
312.             count <= 3'b0;
313.         end
314.     end
315.     Write:begin
316.         if(count == 3'b100) begin
317.             // valid[mark] <= 1;
318.             my_ready <= 0;
319.             my_valid <= 0;
320.             count <= 0;
321.         end
322.         else begin
323.             count <= count + 2'b1;
324.         end
325.     end
326.     default: begin
327.
328.     end
329. endcase
330. end
331.
332. end
333.
334.
335. assign debug_data = {miss_time,ls_time};
336. endmodule

```

其中用的是三段式米粒型有限状态机，主要是处理了各种状态下与 MEM 如何交换数据、与 CPU 如何交换数据以及 Dcache 如何维护自身的 tag、valid 和 cache_data 等各种寄存器。具体的设计思路来自于上面给出的有限状态机和课本中关于直接相联 Cache 的介绍。例如，判断 Cache 是否命中时不仅仅要判断 tag 标签，还需要查看当前行是否有效。同时由于 Dmem 中使用的是块式存储器，读写都需要延迟一个时钟周期，所以在握手协议完成之后，Cache 中数据的修改相较于 DMem 中对于读取地址的处理要晚上一个时钟周期。但是在写入时由于时钟延迟时间充足，在进行数据的传输交换时是同频地处理地址和数据。

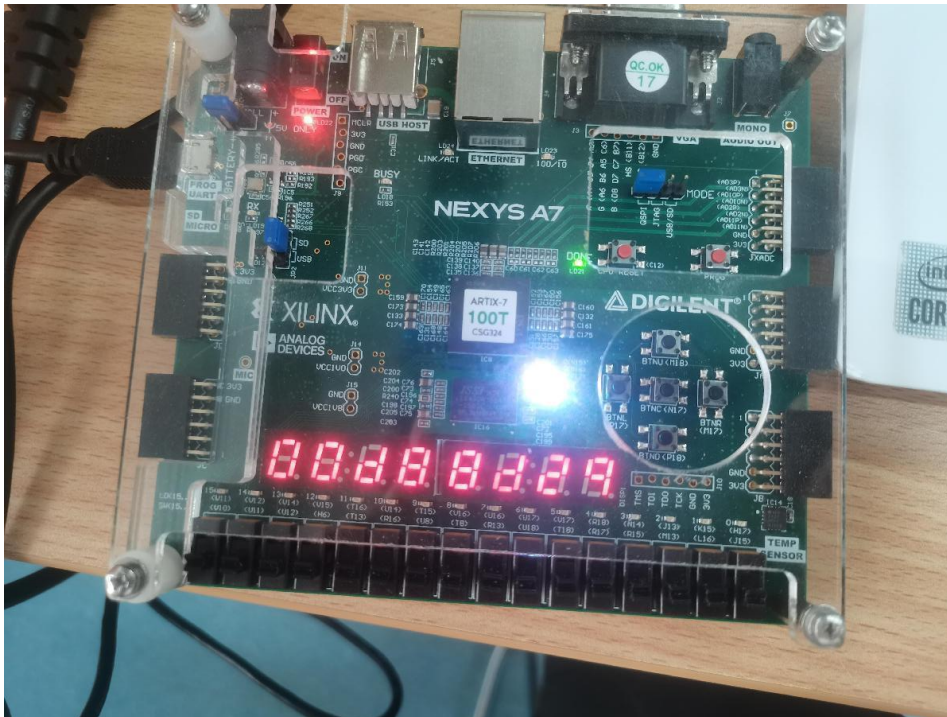
电路设计与分析

下面是在编写二路组相联和直接相联程序时仿照的 Cache 数据通路图



测试结果与分析

首先是直接相联的 Cache，这里输入的生成数组长度为 1000，即十六进制下的 3e8；第一个数设置为 35，即十六进制下的 23。直接相联的命中率较低，所以程序运行的会比较慢，大概 4-5 分钟可以跑完一次，以下为上板情况展示：



AT&T XCOM V2.0

```
IF/ID-: pc_id= 0000_00e0  ir_id= 01ff_4663
ID/EX-: pc_ex= 0000_0000  ir_ex= 0000_0000  rrd1= 0000_0000
      : rrd2= 0000_0000  imm= 0000_000c
EX/MEM: ir_mem= 000e_af83  res= 0000_0000  dwd= 0000_0000
MEM/WB: ir_wb= 01d6_0eb3  res_wb= 0000_2160  drd= 0000_0000
WB/-:  rwd= 0000_2160

R - Registers:
x0 (r0): 0000_0000  x1 (ra): 0000_0000  x2 (sp): 0000_0000  x3 (gp):
0000_0000
x4 (tp): 0000_0000  x5 (t0): 0000_0000  x6 (t1): 0000_009c  x7 (t2):
0000_0057x8 (s0): 0000_0000  x9 (s1): 0000_0000  x10(a0): 0000_7f00  x11
(a1): 0000_0001x12(a2): 0000_2004  x13(a3): 0000_2000  x14(a4): 000d_3a0c
x15(a5): 0000_0000x16(a6): 0000_0000  x17(a7): 0000_0000  x18(s2): 0000_0040
x19(s3): 0000_0000x20(s4): 0000_0000  x21(s5): 0000_0000  x22(s6): 0000_0000
x23(s7): 0000_0000x24(s8): 0000_0000  x25(s9): 0000_0000  x26(sa): 0000_0000
x27(sb): 0000_0000x28(t3): 0000_2274  x29(t4): 0000_2160  x30(t5): 0000_0000
x31(t6): 0000_0000

P - State info -
CTR-: debug1= 0005_5a28  debug2= 001e_5ae6  debug3= 0000_0005
/IF-: npc= 0000_0164  pc= 0000_0160  ir= 0000_0000
IF/ID-: pc_id= 0000_015c  ir_id= 0000_006f  rrd1= 0000_0000
ID/EX-: pc_ex= 0000_0000  ir_ex= 0000_0000  rrd2= 0000_0000
      : rrd2= 0000_0000  imm= 0000_0000
EX/MEM: ir_mem= 0000_0000  res= 0000_0000  dwd= 0000_0000
MEM/WB: ir_wb= 0000_006f  res_wb= 0000_0000  drd= 1234_5678
WB/-:  rwd= 0000_0160
```

串口选择: COM4: USB-SERIAL

波特率: 9600

停止位: 1

数据位: 8

奇偶校验: 无

串口操作: ☒ 关闭串口

保存窗口 清除接收

☐ 16进制显示 ☐ 白底黑字

☐ RTS ☐ DTR

☐ 时间戳(以换行回车断帧)

单条发送 多条发送 协议传输 帮助

☐ D 00000000 0 ☐ P 5 ☒ 发送新行

☐ G 1 ☐ D 00000008 6 ☐ 16进制发送

☐ T 2 ☐ D 00000010 7 ☐ 关联数字键盘

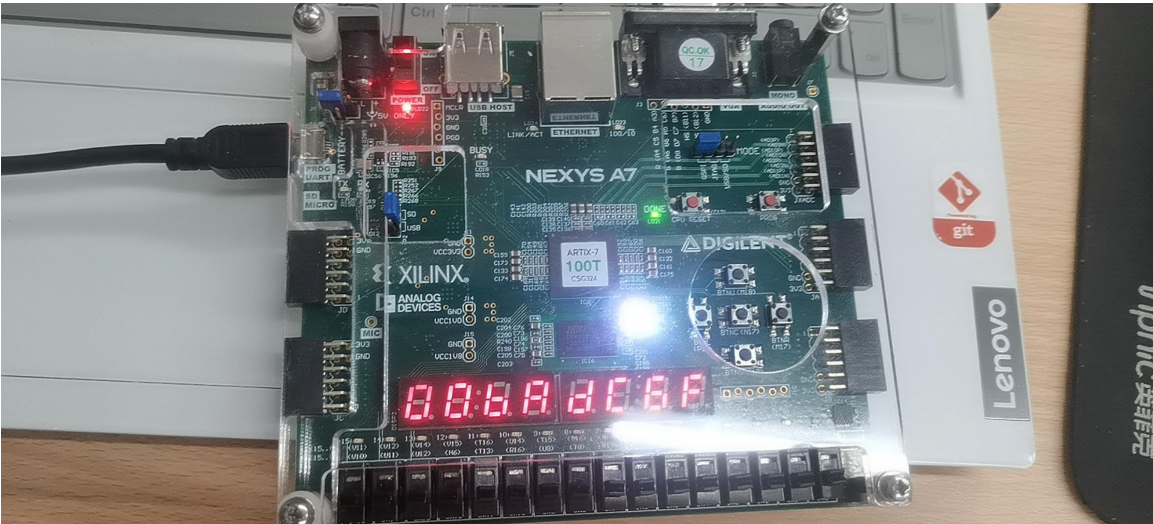
☐ R 3 ☐ D 00000018 8 ☐ 自动循环发送

☐ H 4 ☐ 9 周期: 1000 ms

首页 上一页 下一页 尾页

www.openedv.com S:30 R:3950 CTS=1 DSR=0 DCD=0 当前时间 19:31:08

然后是二路组相联的 Cache（在 Verilog 程序中需要更改选用的模块），以下为上板子的情况：



```
Verilog code snippet:
Write_for_valid = 3'b100;
Write_for_ready = 3'b101;
Write = 3'b110;

current_state = Work;
next_state = Work;
ready = 0;
lid = 0;
count = 3'b0;

signal = done;
lid_DC = my_valid;
ready_DC = my_ready;

posedge clk, negedge rstn)
begin
current_state <= Work;

begin
current_state <= next_state;
```

```
Serial monitor output (XCOM V2.0):
IF/ID- pc_id= 0000_00e8  ir_id= 01e6_0e33  rrd1= 0000_021f
ID/EX- pc_ex= 0000_00e8  ir_ex= 0023_1e13  rrd2= 0000_021f
EX/MEM ir_mem= 02e3_8863  res= 0000_087e  dwd= 0000_021f
MEM/WB ir_wb= 0000_0000  res_wb= 0000_087e  drd= 1234_5678
WB/-_ rwd= 0000_0000

R - Registers:
z0 (z0): 0000_0000  z1 (r4): 0000_0000  z2 (sp): 0000_0000  z3 (sp):
0000_0000
z4 (tp): 0000_0000  z5 (t0): 0000_0000  z6 (t1): 0000_021f  z7 (t2):
0000_01f2e8 (z0): 0000_0000  z8 (z1): 0000_0000  z9 (z0): 0000_7600
(a1) 0000_0001z12(z2): 0000_2004  z13(z3): 0000_2000  z14(z4): 0008_e820
z15(z5): 0000_0000z16(z6): 0000_0000  z17(z7): 0000_0000  z18(z2): 0000_0040
z19(z3): 0000_0000z20(z4): 0000_0000  z21(z5): 0000_0000  z22(z6): 0000_0000
z23(z7): 0000_0000z24(z8): 0000_0000  z25(z9): 0000_0000  z26(z1): 0000_0000
z27(z1): 0000_0000z28(z3): 0000_2880  z29(z4): 0000_27f4  z30(z5): 0000_0000
z31(z6): 0000_0000

f - State info:
CTR= debug1= 0003_d137  debug2= 001e_640e  debug3= 0000_0000
/IF_ npc= 0000_0164  pc= 0000_0160  ir= 0000_0000
IF/ID- pc_id= 0000_015e  ir_id= 0000_006f  rrd1= 0000_0000
ID/EX- pc_ex= 0000_0000  ir_ex= 0000_0000  rrd2= 0000_0000
EX/MEM ir_mem= 0000_0000  res= 0000_0000  dwd= 0000_0000
MEM/WB ir_wb= 0000_006f  res_wb= 0000_0000  drd= 1234_5678
WB/-_ rwd= 0000_0160
```

```
Serial port settings:
串口号: COM4
波特率: 9600
停止位: 1
数据位: 8
奇偶校验: 无
串口操作: 关闭串口
保存窗口: 清除接收
16进制显示: 16进制发送
16进制接收: 16进制接收
时间戳: 时间戳
```

不难看出失误率大概为之前的 1/2。这也符合我们对于 1000 个数排序的预期。由于需要访存 1001 个地址的数据，而两种 Cache 的容量大小均为 1KB，而我采用的冒泡排序的算法需要固定数组中的一个数，然后依次与后面的数比较大小，这样的情况下，二路组相联借助两个 tag 的灵活性，大幅度降低了自身的失误率。

总结

在本次实验中，我从零到有地学习、理解、构建了两种 1KB 地 Cache，同时又仿照着给出的 PDU 仿制了一个自己的 IOU 外设处理单元。总的来说收获颇丰，学习到了很多新东西，也掌握了有关 Cache 工作的有限状态机，以及失效时与存储器之间如何进行握手协议处理。在 MMIO 的处理中，我进一步理解了外设输入输出、CPU 和汇编程序之间的关系，也复习了之前在 ICS 课上学到的轮询处理方法。作为一个长达三周的 Verilog 收尾实验，总体的过程并不是很顺利，出现了许许多多我从未设想过的 bug，如 IOU 处理单元的时钟应与 CPU 调试时钟分开、Cache 与 Mem 之间握手的对应关系甚至是 Cache 的出现对于流水线停顿的影响等等。自主地发现并解决问题后，感觉又学到了很多，甚至萌生了明年去当数电实验助教的想法。虽然收官并不完美，但是我依旧觉得值得了，大二的课程让我真真正正可以实现徒手搭 CPU 的愿景，感谢老师和助教的付出，至于 Vivado 还是别再见了吧。