Lab 6 综合设计 report

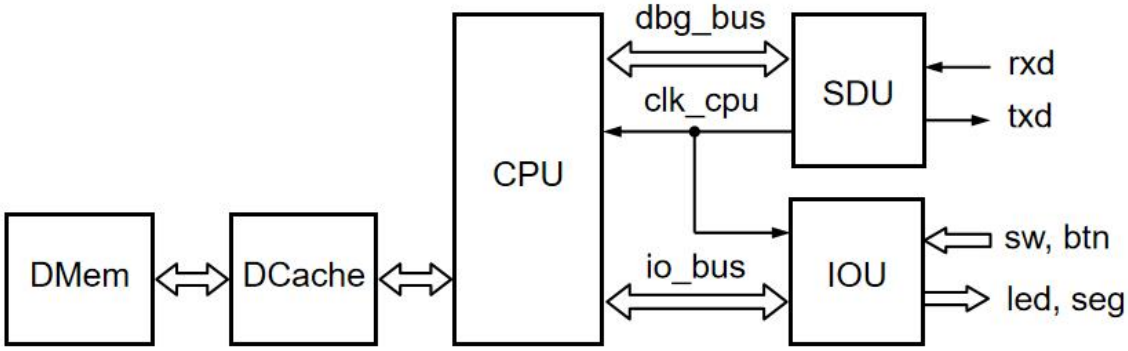
PB21111681 朱炜荣

实验目的与内容

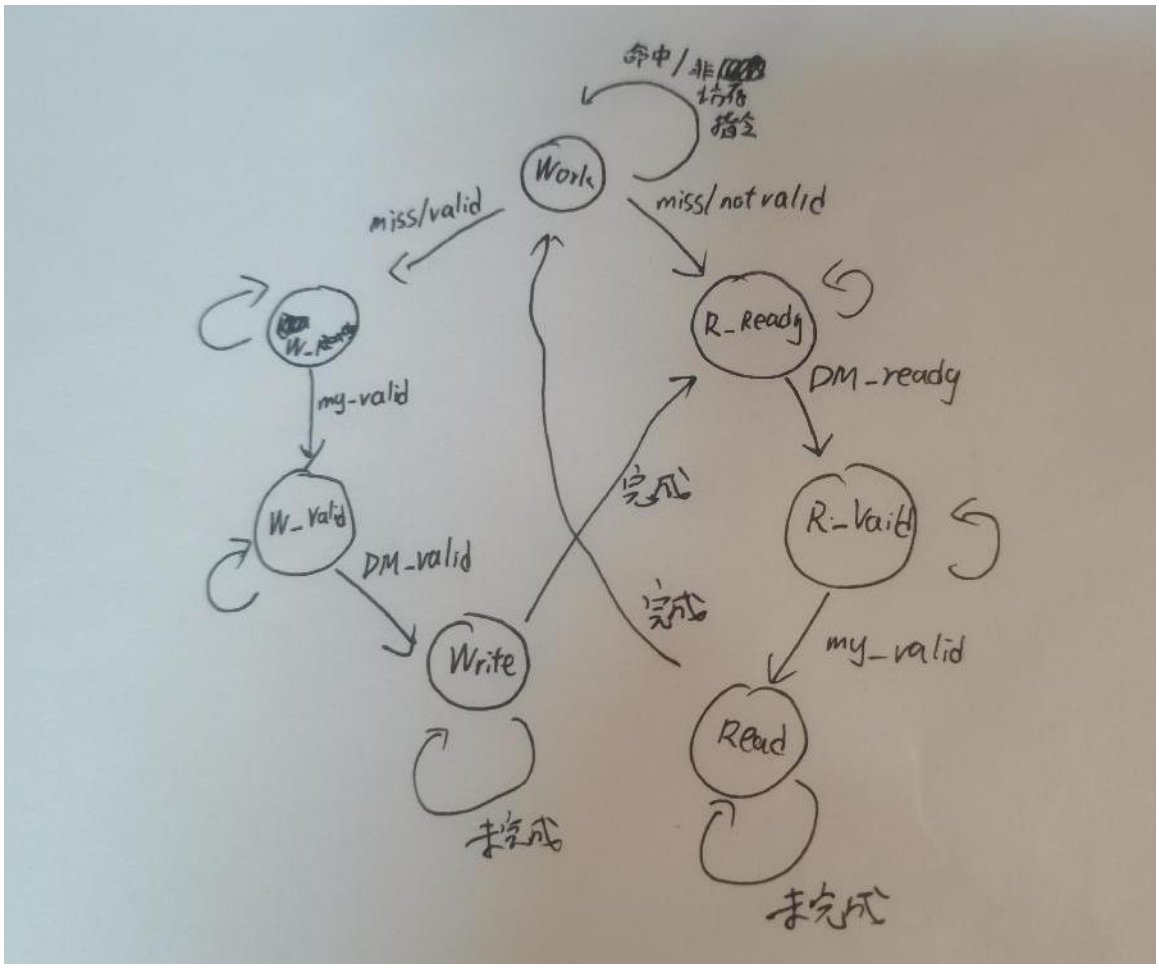
1. 学会 Cache 基本原理、结构、设计和调试方法
2. 掌握 CPU 输入/输出的编址和控制方式
3. 熟练掌握数据通路和控制器的设计和描述方法
4. 掌握有关 CPU 的 MMIO 外设输入输出的处理

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：

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 2023/06/04 17:10:49
// Design Name:
// Module Name: Dcache
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
```

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

//直接相联 Cache 8 行 一块 4 字 共 1KB
module Dcache(
    output [63:0] debug_data,
    input clk,
    input rstn,
    input we_r,          //外部读信号
    input we_w,          //外部写信号
    input [31:0] addr,       //外部尝试地址
    input [31:0] din,        //外部输入
    // input [31:0] addr_exchange_in,       //用于交换的输入值地址
    input [31:0] data_exchange_in,       //用于交换的输入值
    input ready_DM,          //Mem 的 ready 信号
    input valid_DM,          //Mem 的 valid 信号
    output reg[31:0] dout,      //外部输出
    output reg [31:0] addr_exchange_out,     //用于交换的输出值
    output reg [31:0] data_exchange_out,     //用于交换的输出值地址
    output valid_DC,         //Cache 的 valid 信号
    output ready_DC,         //Cache 的 ready 信号
    output signal,           //控制流水线的信号
    output reg [1:0] state
);
reg [7:0] valid = 8'b0;
reg [24:0] tag [0:7];       //32 - 3 - 2 - 2 = 25
reg [127:0] cache_data [0:7];
reg [31:0] miss_time = 32'b0;
reg [31:0] ls_time = 32'b0;

wire [1:0] offset = addr[3:2];
wire [2:0] mark = addr[6:4];
wire hit =(valid[mark] != 0) && (tag[mark] == addr[31:7]) && (we_r || we_w) && (addr < 32'
h7f00) && (addr >= 32'h2000);
// wire [7:0] part = (offset == 2'b00)?0:(offset == 2'b01)?32:(offset == 2'b10)?64:96;

parameter Work = 3'b000;
parameter Read_for_valid = 3'b001;
parameter Read_for_ready = 3'b010;
parameter Read = 3'b011;
parameter Write_for_valid = 3'b100;
parameter Write_for_ready = 3'b101;
parameter Write = 3'b110;
```

```
reg [2:0] current_state = Work;
reg [2:0] next_state = Work;
reg my_ready = 0;
reg my_valid = 0;
reg done = 1;
reg [2:0] count = 3'b0;

assign signal = done;
assign valid_DC = my_valid;
assign ready_DC = my_ready;

always @(posedge clk,negedge rstn)
begin
    if(!rstn)begin
        current_state <= Work;
    end
    else begin
        current_state <= next_state;
    end
end

always @(*)
begin
    if(!rstn)begin
        next_state = Work;
        addr_exchange_out = 32'b0;
        data_exchange_out = 32'b0;
        dout = 32'b0;
        state = 2'b00;
        done = 1;
    end
    else begin
        addr_exchange_out = 32'b0;
        data_exchange_out = 32'b0;
        dout = 32'b0;
        case (current_state)
            Work: begin
                state = 2'b00;
                if(hit == 1)begin
                    next_state = Work;
                    done = 1;
                    if(we_r == 1)begin
                    case(offset)
                        2'b00: begin
                            dout = cache_data[mark][31:0];
                        end
                        2'b01:
                         begin
```

```
                            dout = cache_data[mark][63:32];
                        end
                        2'b10:
                         begin
                            dout = cache_data[mark][95:64];
                        end
                        2'b11:
                         begin
                            dout = cache_data[mark][127:96];
                        end
                    endcase
                end
                end
                else begin
                    if((we_r == 1 || we_w == 1) &&  (addr < 32'h7f00) && (addr >= 32'h2000)
    )begin
                        done = 0;
                        if(valid[mark] == 1)
                            next_state = Write_for_ready;
                        else
                            next_state = Read_for_ready;
                    end
                    else begin
                        next_state = Work;
                        done = 1;
                    end
                end
            end
            Read_for_ready:begin
                state = 2'b10;
                done = 0;
                if(valid_DM == 1)
                    next_state = Read_for_valid;
                else
                    next_state = Read_for_ready;
            end
            Read_for_valid:begin
                state = 2'b10;
                done = 0;
                if(my_ready == 1)
                    next_state = Read;
                else
                    next_state = Read_for_valid;
            end
            Read:begin
                addr_exchange_out = addr;
                state = 2'b10;
                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
```

```
            default:begin
                next_state = Work;
                state = 2'b00;
                done = 1;
            end
        endcase

    end
end


always@(posedge clk,negedge rstn)
begin
    if(!rstn)begin
        my_valid <= 0;
        my_ready <= 0;
        count <= 3'b000;
        miss_time <= 32'b0;
        ls_time <= 32'b0;
    end
    else begin
    case (current_state)
        Work:begin
            if(hit == 1) begin
                ls_time <= ls_time + 1'b1;
                if(we_w == 1)begin
                    case(offset)
                        2'b00: begin
                            cache_data[mark][31:0] <= din;
                        end
                        2'b01:
                         begin
                            cache_data[mark][63:32] <= din;
                        end
                        2'b10:
                         begin
                            cache_data[mark][95:64] <= din;
                        end
                        2'b11:
                         begin
                            cache_data[mark][127:96] <= din;
                        end
                    endcase
                end
            end
            else begin
            if((we_r == 1 || we_w == 1) &&  (addr < 32'h7f00) && (addr >= 32'h2000))begin
                miss_time <= miss_time + 1;
```

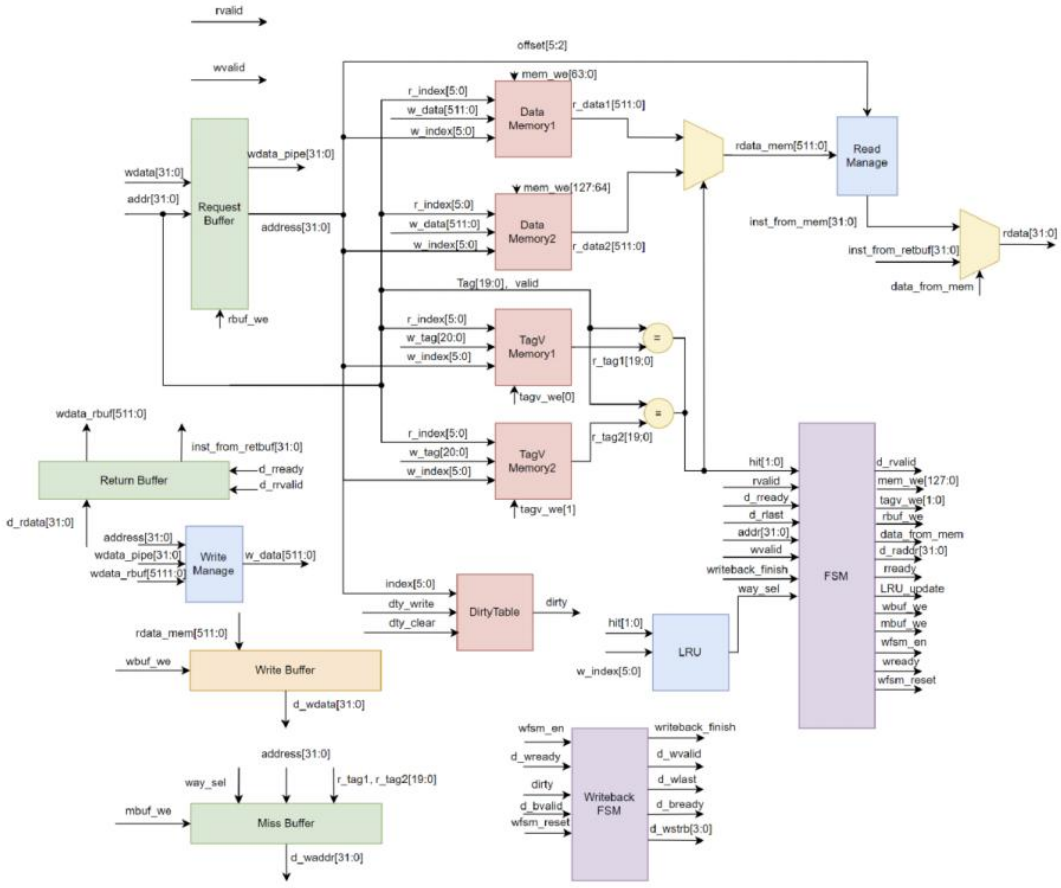
```
253.                 if(valid[mark] == 1)begin
254.                     my_valid <= 1;
255.                 end
256.                 else begin
257.                     ;
258.                 end
259.             end
260.             else begin
261.
262.             end
263.
264.             end
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
```

```
            else begin
                count <= count + 3'b1;
            end
        end
        Write_for_ready:begin
            if(my_valid == 1)begin

            end
        end
        Write_for_valid:begin
            if(ready_DM == 1)begin
                count <= 3'b0;
            end
        end
        Write:begin
            if(count == 3'b100) begin
                // valid[mark] <= 1;
                my_ready <= 0;
                my_valid <= 0;
                count <= 0;
            end
            else begin
                count <= count + 2'b1;
            end
        end
        default: begin

        end
    endcase
    end

end


assign debug_data = {miss_time,ls_time};
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。

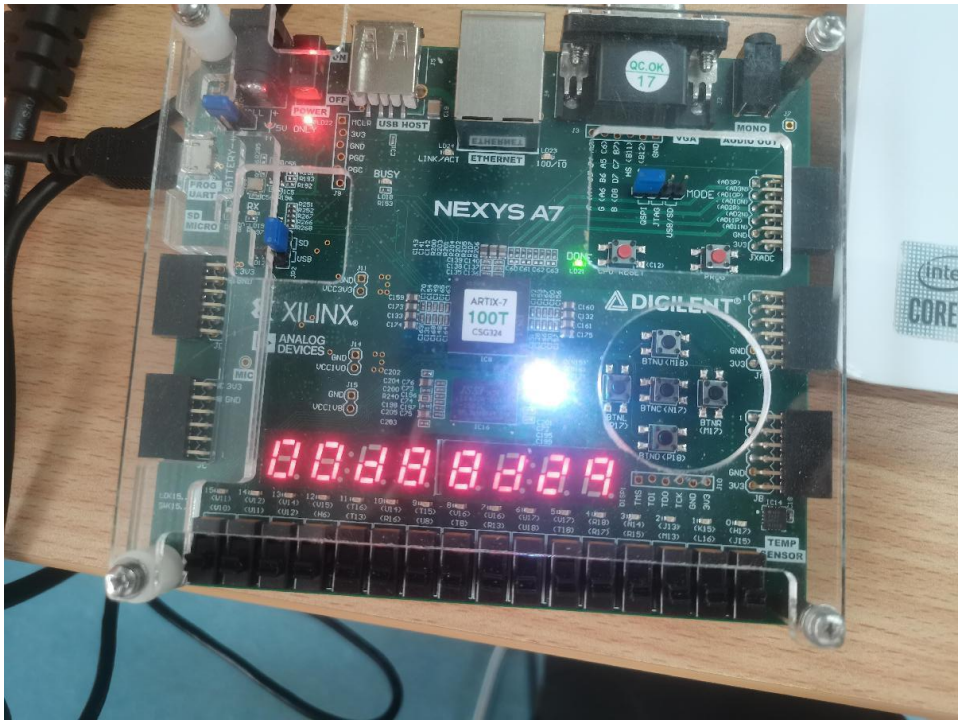
电路设计与分析

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：



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 (zo): 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_3e0c
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
ID/EX-: pc_ex——= 0000_0000      ir_ex——= 0000_0000      rrd1——= 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)

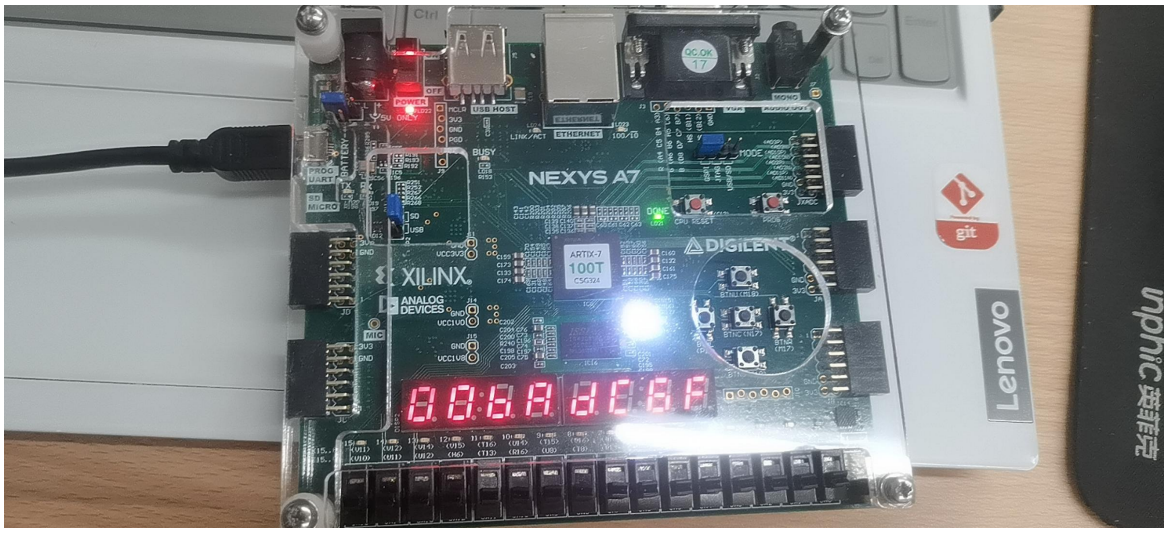
单条发送 多条发送 协议传输 帮助

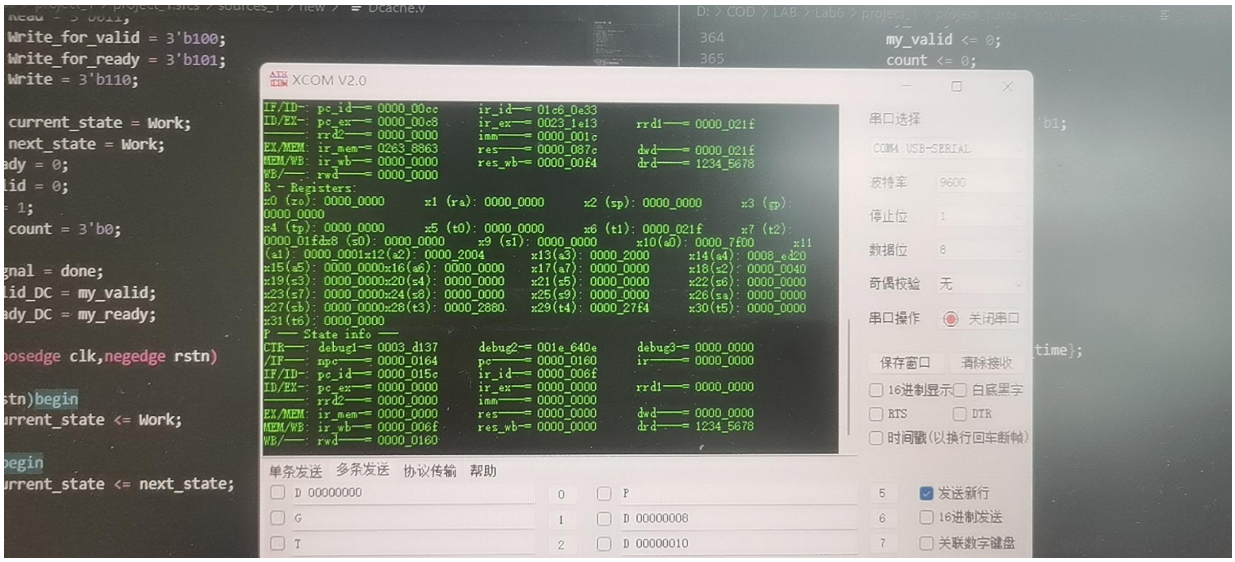
D 00000000	0	P	5	发送新行
G	1	D 00000008	6	16进制发送
T	2	D 00000010	7	关联数字键盘
R	3	D 00000018	8	自动循环发送
H	4		9	周期: 1000 ms

首页 上一页 下一页 尾页 导入导出条目

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：





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。