

# lab2 单周期MIP处理器


---

## 实验目的

- 熟悉Vivado软件
- 熟悉在Vivado软件下进行硬件设计的流程
- 设计单周期MIPS处理器，包括：
  - 完成单周期MIPS处理器的设计
  - 在Vivado软件上进行仿真
  - 编写MIPS代码验证单周期MIPS处理器
  - 在NEXYS4 DDR板上进行验证

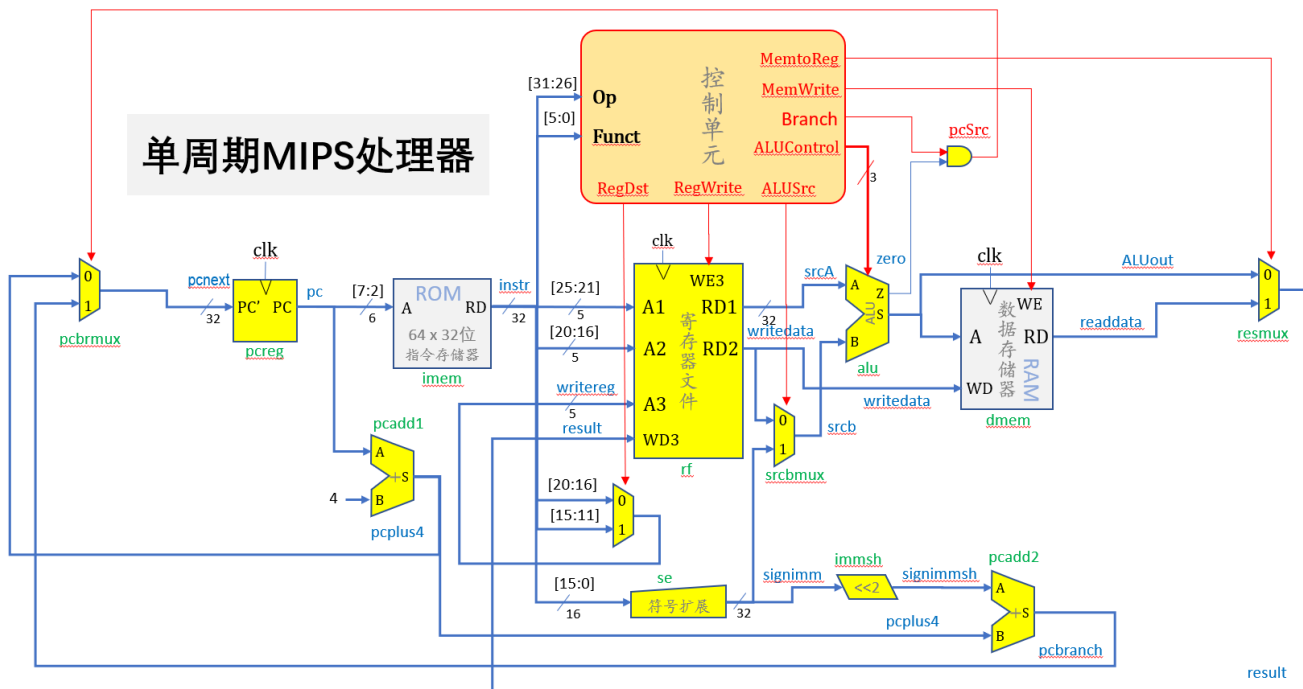
## 实验任务

### 1.设计单周期MIPS处理器

- ▼ ●  **top** (top.sv) (3)
  - ▼ ● **mips1 : mips** (mips.sv) (2)
    - ▼ ● **c : controller** (controller.sv) (2)
      - md : maindec (maindec.sv)
      - ad : aludec (aludec.sv)
    - ▼ ● **dp : datapath** (datapath.sv) (12)
      - pcreg : flopr (flopr.sv)
      - pcadd1 : Adder (Adder.sv)
      - immsh : sl2 (sl2.sv)
      - pcadd2 : Adder (Adder.sv)
      - pcbrmux : MUX2 (MUX2.sv)
      - pcmux : MUX2 (MUX2.sv)
      - rf : REGFILE (REGFILE.sv)
      - wrmux : MUX2 (MUX2.sv)
      - resmux : MUX2 (MUX2.sv)
      - se : signext (signext.sv)
      - srcbmux : MUX2 (MUX2.sv)
      - alu : ALU (ALU.sv)
  - imem : IMEM (IMEM.sv)
  - dmem : DMEM (DMEM.sv)

代码框架如下：

完整的单周期mips处理器：



## • 框架分析

- controller: 控制单元，分析处理指令中的op、func部分。
  - maindec: 内含名为ocntrols的变量（整合了若干输出，例如jump，aluop），根据op赋不同的值
  - aludec: 接受aluop与funct，输出alucontrol（控制alu行为）首先根据aluop判断是R型还是非R型指令。若为R型指令则根据funct为alucontrol赋值
- datapath: 数据通路，控制数据存储、处理与传送
  - ALU: 输入两个操作数和op，输出结果和zero（主要用于判断跳转指令）
  - MUX2: 二选一复用器，提供可选参数
  - signext: 符号扩展
  - sl2: 左移2位，寻址时偏移量的单位为指令条数，需要乘4转换为字节地址
  - regfile: 寄存器文件，两个读地址、两个写数据、一个写使能、一个写地址
  - adder: 32位加法器
  - flopr: 程序计数器PC

## • 执行一条指令的步骤：

- 根据PC值从指令存储器IMEM取指令
- 将指令的op与funct字段传入controller模块进行译码并分配控制信号的值
- 指令同时传入datapath模块，进行寄存器的读写，具体的操作由controller分配的控制信号决定（复用器值的选取、alu的操作）

## • 主要代码

主要列出对ppt上代码作出修改的部分

```
//maindec
//为实现beq、bne
assign {
```

```

regwrite,regdst,alusrc,branchbeq,branchbne,memwrite,memtoreg,jump,aluop } =
controls;

assign pcsrc = (branchbeq & zero)|(branchbne & (!zero));

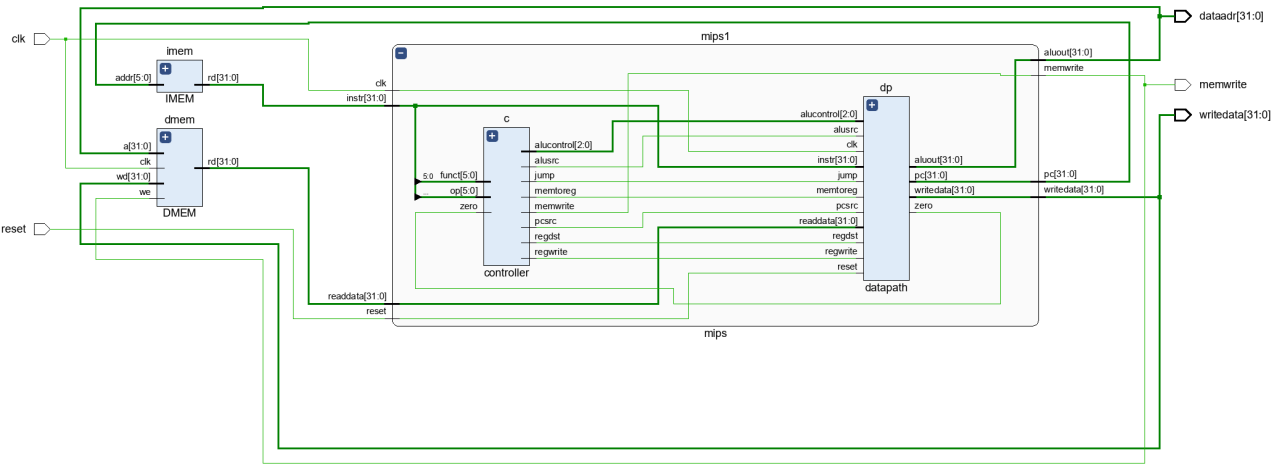
//aluop扩大到三位, 加入jump控制位, branch扩展为两位实现不同分支指令
always_comb
case(op)
    6'b000000:controls <= 11'b110_00_00_0_100; //r
    6'b100011:controls <= 11'b101_00_01_0_000; //lw
    6'b101011:controls <= 11'b001_00_10_0_000; //sw
    6'b000100:controls <= 11'b000_10_00_0_001; //beq
    6'b000101:controls <= 11'b000_01_00_0_001; //bne
    6'b001000:controls <= 11'b101_00_00_0_000; //addi 001000
    6'b001100:controls <= 11'b101_00_00_0_010; //andi
    6'b001101:controls <= 11'b101_00_00_0_011; //ori
    6'b001010:controls <= 11'b101_00_00_0_101; //slti
    6'b000010:controls <= 11'b000_00_00_1_000; //j
    default: controls <= 11'bxxxxxxxxxx;
endcase

//aludec
module aludec(
    input logic [5:0] funct,
    input logic [2:0] aluop,
    output logic [2:0] alucontrol
);

always_comb
case(aluop)
    3'b000: alucontrol <= 3'b010; //lw/sw/addi
    3'b001: alucontrol <= 3'b110; //branch
    3'b010: alucontrol <= 3'b000; //andi
    3'b011: alucontrol <= 3'b001; //ori
    3'b101: alucontrol <= 3'b111; //slti
    3'b100: case(funct)
        6'b100000:alucontrol <= 3'b010; //add
        6'b100010:alucontrol <= 3'b110; //sub
        6'b100100:alucontrol <= 3'b000; //and
        6'b100101:alucontrol <= 3'b001; //or
        6'b101010:alucontrol <= 3'b111; //slt
        6'b000000:alucontrol <= 3'b000; //nop
        default: alucontrol <= 3'bxxx;
    endcase
endcase
endmodule

```

- 电路图



2.仿真

- 模块regfile测试

```
timescale 1ns / 100ps
module test_regFile( )
    reg      clk;
    reg      regWriteEn; // 写入使能
    reg [4:0] regWriteAddr;
    reg [31:0] regWriteData; // 1个写入寄存器
    reg [4:0] RsAddr;
    reg [4:0] RtAddr;
    wire [31:0] RsData; // Rs寄存器
    wire [31:0] RtData; // Rt寄存器

    //实例化
    regfile MUT(clk, regWriteEn, regWriteAddr, regWriteData,
                RsAddr, RtAddr, RsData, RtData);

    //初始化
    initial begin
        clk = 0;
        regWriteEn = 0;
        regWriteAddr = 0;
        regWriteData = 0;
        RsAddr = 0;
        RtAddr = 0;
        //Wait 100ns
        #100;
        //添加激励信号
        regWriteEn = 1;
        regWriteData = 32'h1234abcd;
    end
```

测试代码

```
28 //设置时钟
29 parameter PERIOD = 20;
30 always begin
31     clk = 1'b0;
32     #(PERIOD/2) clk = 1'b1;
33     #(PERIOD/2);
34 end
35 //激励信号
36 always begin
37     regWriteAddr = 8;
38     RsAddr = 8;
39     #PERIOD;
40 end
41 endmodule
```

时regWriteData获得数据"1234abcd"  
时，时钟上升沿，RsData获得数据

运行结果：



100ns时regWriteData获得数据"1234abcd"; 110ns时，时钟上升沿，RsData获得数据

- 测试汇编代码+机器代码

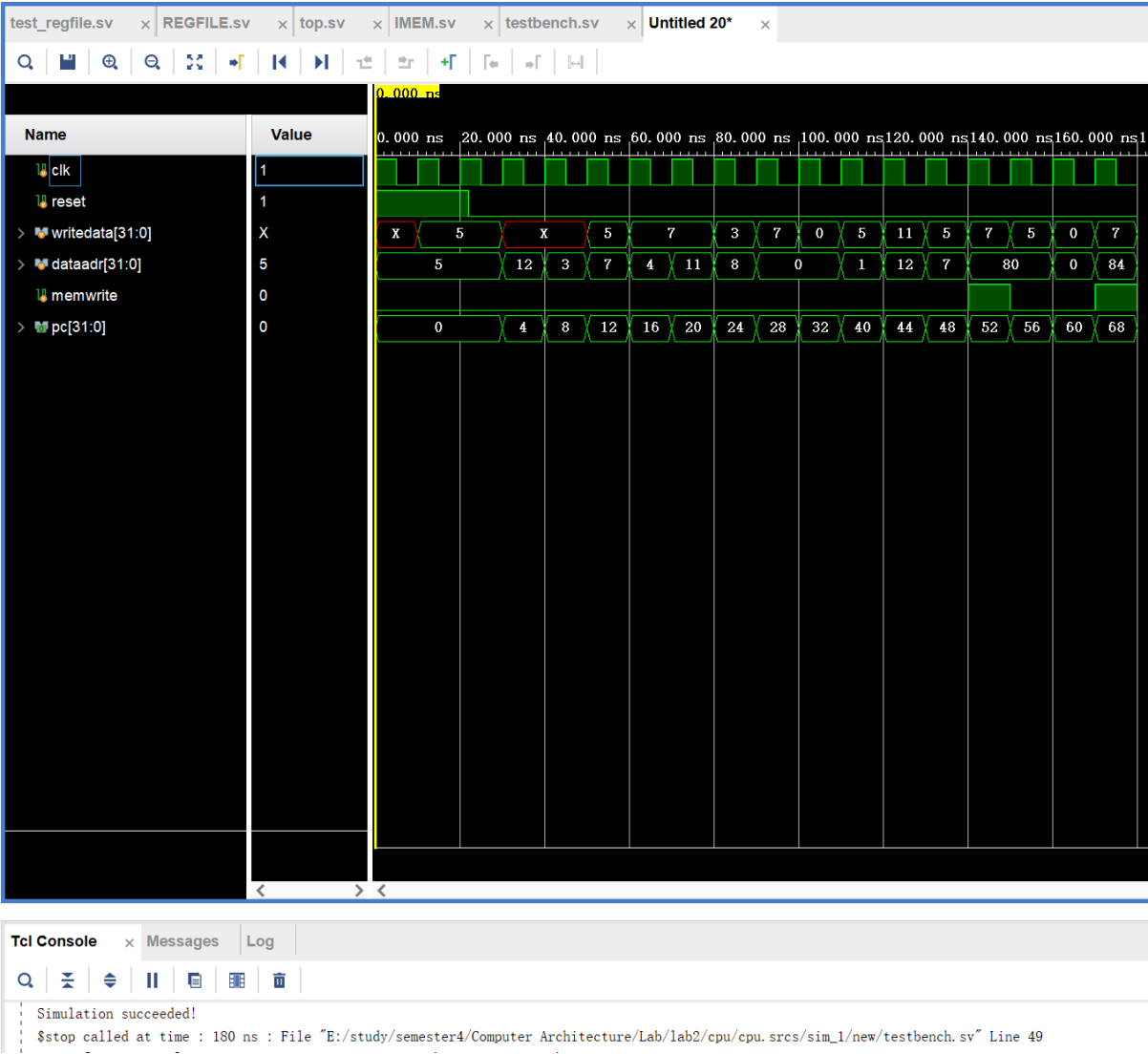
编写memfile.dat文件并导入，编写模拟文件testbench.sv

- 仿真代码:

```
1  `timescale 1ns / 100ps
2  module testbench();
3      logic      clk;
4      logic      reset;
5      logic [31:0] writedata, dataadr;
6      logic      memwrite;
7
8      // instantiate device to be tested
9      top dut(clk, reset, writedata, dataadr, memwrite);
10
11     // initialize test
12     initial begin
13         reset <= 1; # 22; reset <= 0;    end
14
15     // generate clock to sequence tests
16     always begin
17         clk <= 1; # 5; clk <= 0; # 5;    end
18
19     // check that 7 gets written to address 84
20     always@(negedge clk) begin
21         if(memwrite) begin
22             if(dataadr === 84 & writedata === 7) begin
23                 $display("Simulation succeeded");
24                 $stop;
25             end
26             else if (dataadr !== 80) begin
27                 $display("Simulation failed");
28                 $stop;
29             end
30         end
31     end
32 endmodule
```



运行结果



3.验证

- 编写mips代码并转为机器码验证。程序作用为对一个长度为5的数组求和

```
or $8,$0,$0 # $8  
ori $9,$0,9 # $9  
sw $9,0($8)  
addi $8,$8,4  
ori $9,$0,7  
sw $9,0($8)  
addi $8,$8,4  
ori $9,$0,15  
sw $9,0($8)  
addi $8,$8,4  
ori $9,$0,19  
sw $9,0($8)  
addi $8,$8,4  
ori $9,$0,20  
sw $9,0($8)  
ori $10,$0,5 # $t2=5  
or $8,$0,$0 # $t0=0
```



```

# index
or $9,$0,$0 # $t1=0
bne $10,$0,loop
loop:
slt $11,$9,$10 # if($t1<6) $t3=1
beq $11,$0,exit
add $12,$9,$9 #
add $12,$12,$12 # $t4=4*$t1 018c6020
lw $13,0($12)
add $8,$8,$12
# index++
addi $9,$9,1
j loop
exit:
sw $8,84($0)

```

- 转为机器码

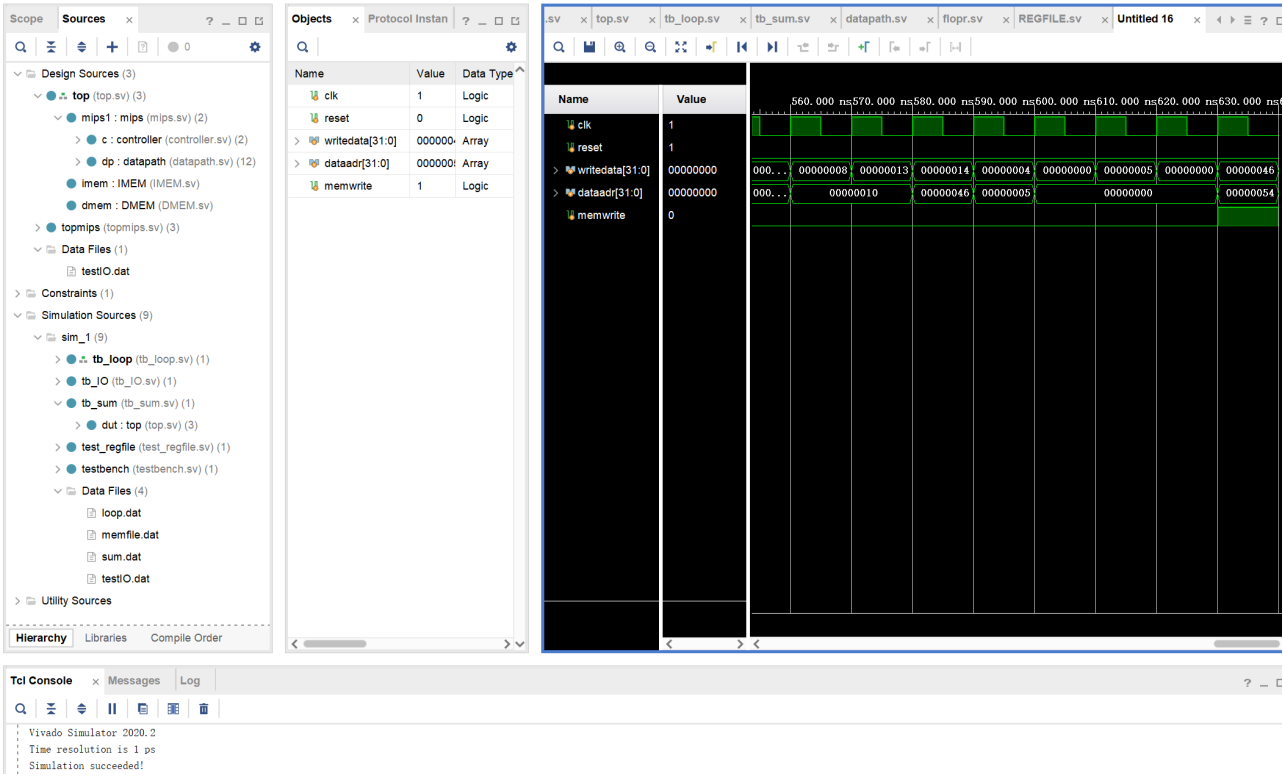
```

00004025
34090009
ad090000
21080004
34090007
ad090000
21080004
3409000f
ad090000
21080004
34090013
ad090000
21080004
34090014
ad090000
340a0005
00004025
00004825
140a0000
012a582a
100b0006
01296020
018c6020
8d8d0000
010d4020
21290001
08000013
ac080054

```

- 运行结果

- 修改testbench，预期结果为70。最终测试成功



4.板上验证

- topmips (topmips.sv) (3)
  - iMemory : IMEM (IMEM.sv)
- mips2 : mips (mips.sv) (2)
  - c : controller (controller.sv) (2)
  - dp : datapath (datapath.sv) (12)
- mdecoder : dMemoryDecoder (dMemoryDecoder.sv) (3)
  - dmemory : DMEM (DMEM.sv)
  - io : IO (IO.sv)
  - mux7seg : mux7seg (mux7seg.sv) (1)
- Data Files (1)
  - testIO.dat

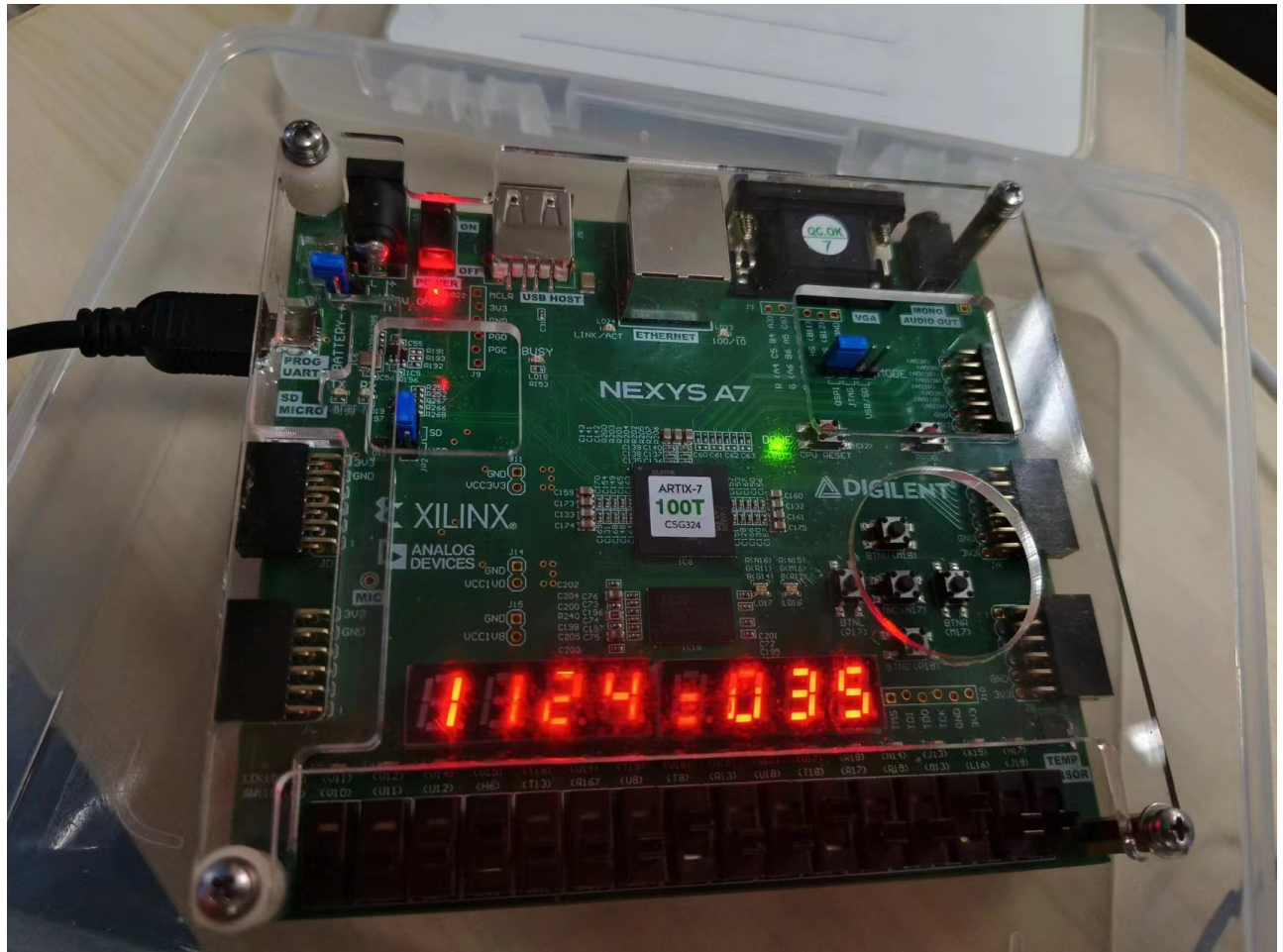
- 代码框架
  - dMemoryDecoder: 地址译码器, 控制IO, 调用原DMEM, 将各地址分配到对应的DMEM、IO中, 并新增七段数码管译码器mux7seg
  - 顶层文件topmips加入开发板上的硬件作为输入/输出
- 主要代码
  - dMemoryDecoder:

```
assign pread=(addr[7])?1:0;
assign pwrite=(addr[7])?writeEN:0;
assign mwrite=(addr[7]==0)?writeEN:0;
assign readData=(addr[7]==1)?readData2:readData1;
```

根据数据地址（高位）决定数据来自内存还是IO部分

其余代码在ppt上提及，这里不做赘述

- 运行结果



## 实验心得

- 掌握了模拟单周期mips处理器的方法，控制信号与数据路径的处理，并通过增加控制信号的方式扩展指令
- 在vivado寻找bug来源很困难，有时候接口不一致会被优化，从而寻找不到问题出处。因此要养成良好的习惯，在实例化模块的时候需要注意输入输出的顺序，最好按照名称对应，不要偷懒。
- 未解决的问题
  - 我还编写了一个求和sum.dat文件，由于时间限制模拟未成功
  - 2.4部分的仿真led始终为0，寻找问题多次无果，但因上板实验成功，猜测不是底层逻辑问题，应是上层仿真文件或者IO逻辑问题