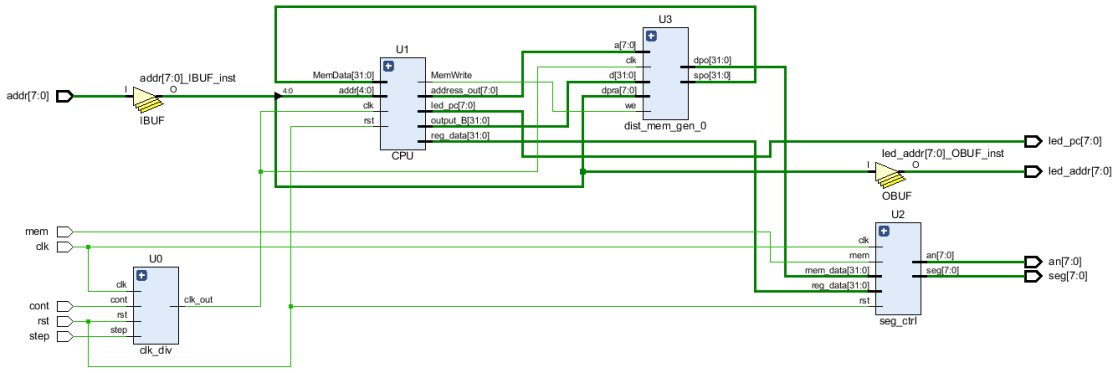
Lab5 多周期 MIPS-CPU

1. 设计逻辑&核心代码

使用模块化设计，大块模块一共有个：显示模块 DDU，时钟分频模块 clk_div，多周期 CPU，七段数码管控制模块 seg_ctrl，和 dist_mem_gen0。



(1) 显示模块 DDU，用以显示 CPU 的运行与输出情况，因此将此模块作为顶层模块。

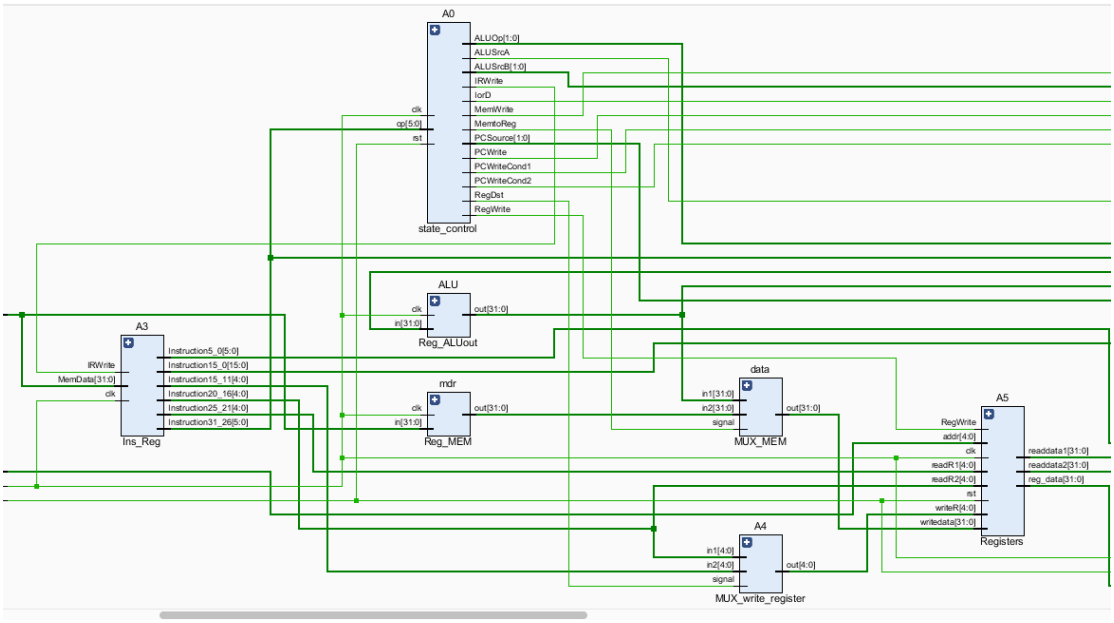
```
module DDU(
    input clk,
    input rst,
    input cont,
    input step,
    input mem,
    input [7:0]addr,
    output [7:0]led_pc,
    output [7:0]led_addr,
    output [7:0]an,
    output [7:0]seg
    );
    wire clk_out,MemWrite;
    wire [31:0]mem_data,reg_data,output_B,MemData;
    wire [7:0]address_out;
    reg inc_1,dec_1;
    assign led_addr=addr;

    clk_div U0(cont,step,clk,rst,clk_out);
    CPU U1(rst,clk_out,reg_data,addr[4:0],led_pc,address_out,MemWrite,MemData,output_B);
    seg_ctrl U2(clk,rst,mem,mem_data,reg_data,an,seg);
    dist_mem_gen_0 U3(.clk(clk_out),.a(address_out),.d(output_B),.we(MemWrite),.dpra(addr),.spo(MemData),.dpo(mem_data));
endmodule
```

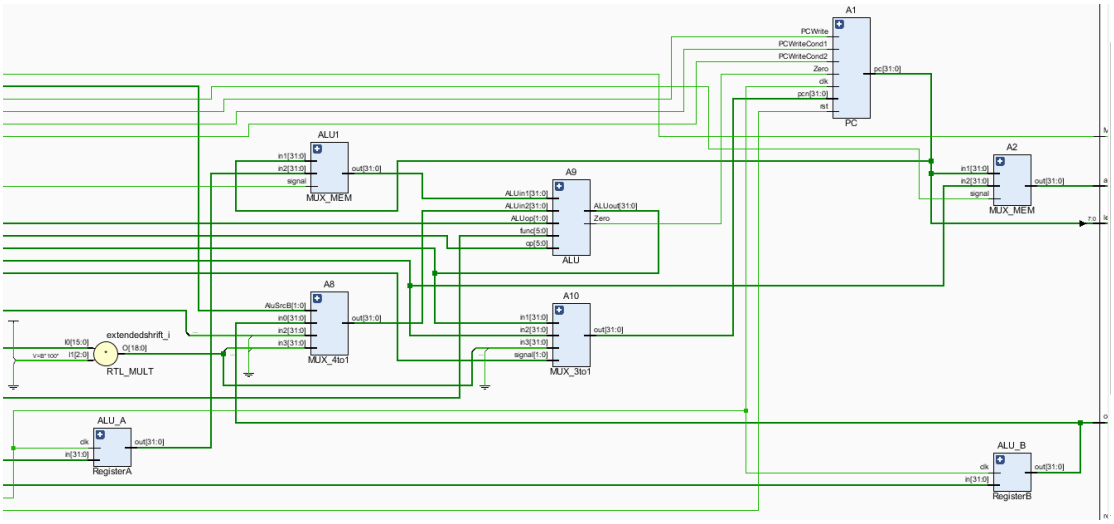
顶层模块只做输入输入接口作用，不对逻辑设计有任何影响。

(2) 时钟分频模块clk_div，把板载100MHz时钟分频成1MHz，当const=0，要求单步执行时，把时钟改为手剥时钟，即把 step 当作时钟，拨一下执行一个时钟周期。

```
reg clk_slow;
reg [31:0]cnt;
always @(posedge clk or posedge rst)
begin
    if(rst)
    begin
    cnt<=0;
    clk_slow<=0;
    end
    else if(cnt<=32'h 100_0000)
    cnt<=cnt+1;
    else
    begin
        cnt<=0;
        clk_slow<=~clk_slow;
    end
end
always@(*)
begin
    if(cont==1)
        clk_out=clk_slow;
    else clk_out=step;
```

(3) 多周期MIPS CPU模块，这是设计上最核心的模块。一共有state_contrl，PC，MUX_MEM，Ins_Reg，MUX_write_register，Registers，RegisterA，RegisterB，Reg_ALUout，Reg_MEM，MUX_MEM，MUX_4to1，ALU，MUX_3to1，MUX_MEM 这些小模块构成。





(3.1) state_control 模块

状态机状态为一个周期内执行指令的步骤。

```
parameter IF=4'b0000;
parameter ID=4'b0001;
parameter EXC_LS=4'b0010;
parameter EXC_R=4'b0011;
parameter EXC_BEQ=4'b0100;
parameter EXC_BNE=4'b0101;
parameter EXC_J=4'b0110;
parameter EXC_Itype=4'b1011;
parameter MEM_LW=4'b0111;
parameter MEM_SW=4'b1000;
parameter REG_R=4'b1001;
parameter REG_LW=4'b1010;
parameter REG_Itype=4'b1100;
```

两段式状态机，第一段组合逻辑指明下一个状态

```
case(currentstate)
   IF: nextstate=ID;
   ID:
   begin
       if(op==Rtype)    nextstate=EXC_R;
       else if(op==lw||op==sw)    nextstate=EXC_LS;
       else if(op==beq)    nextstate=EXC_BEQ;
       else if(op==bne)    nextstate=EXC_BNE;
       else if(op==jump)    nextstate=EXC_J;
       else if(op==addi||op==andi||op==ori||op==xori||op== slti)    nextstate=EXC_Itype;
   end
   EXC_LS:
   begin
   if(op==sw)
       nextstate=MEM_SW;
   else
       nextstate=MEM_LW;
   end
   EXC_R:nextstate=REG_R;
   MEM_LW:nextstate=REG_LW;
   REG_R:nextstate=IF;
   REG_LW:nextstate=IF;
   MEM_SW:nextstate=IF;
   EXC_J:nextstate=IF;
   EXC_BEQ:nextstate=IF;
   EXC_BNE:nextstate=IF;
   EXC_Itype:nextstate=REG_Itype;
   REG_Itype:nextstate=IF;
```

4'b1011

第二段通过时许逻辑电路来指明每一个状态执行怎样的操作。

```
always@(*)
begin
    case(currentstate)
    IF:begin
           MemRead=1;
           ALUSrcA= 0;
           IorD = 0;
           IRWrite=1;
           ALUSrcB=2'b 01;
           ALUOp=2'b00;
           PCWrite=1 ;
           PCSource =2'b 00;
           PCWriteCond1=0;
           PCWriteCond2=0;
           MemWrite=0;
           RegWrite=0;
       end
    ID:begin
        IRWrite=0;
        ALUSrcA= 0;
        ALUSrcB=2'b11;
        ALUOp=2'b00;
        PCWrite=0;
       end
    EXC_LS:
    begin
        ALUSrcA= 1; ALUSrcB= 2'b10; ALUOp =2'b00;
    end
```

```
            EXC_R:
            begin
                ALUSrcA=1; ALUSrcB=2'b00; ALUOp=2'b 10;
            end
            EXC_BEQ:
            begin
                ALUSrcA=1; ALUSrcB=2'b00; ALUOp=2'b 01; PCWriteCond1=1; PCSource=2'b 01;
            end
            EXC_BNE:
            begin
                ALUSrcA=1; ALUSrcB=2'b00; ALUOp=2'b 01; PCWriteCond2=1; PCSource=2'b 01;
            end
            EXC_J:
            begin
                PCWrite=1; PCSource=2'b 10;
            end
            MEM_SW:
            begin
                IorD=1; MemWrite=1;
            end
            MEM_LW:
            begin
                IorD=1; MemRead=1;
            end
            REG_R:
            begin
                RegDst=1; RegWrite=1; MemtoReg=0;
            end
                    REG_LW:
                    begin
                        RegDst=0; RegWrite=1;   MemtoReg=1; MemRead=0;
                    end
                    EXC_Itype:
                    begin
                        ALUSrcA=1; ALUSrcB=2'b10; ALUOp=2'b11;
                    end
                    REG_Itype:
                    begin
                        RegDst=0; RegWrite=1; MemtoReg=0;
                    end
                endcase
            end
```

相比 PPT 上的状态图，补充了 EXC_Itype，REG_Itype，EXE_BEQ，EXE_BNE 这几个状态

(3.2) PC 模块对 PC 的刷新操作。执行完一条指令之后，更新 PC 的值。

```
module PC(
input clk,
input rst,
input PCWrite,
input PCWriteCond1,
input PCWriteCond2,
input Zero,
input [31:0]pcn,
output reg [31:0]pc
    );
    always@(posedge clk or posedge rst)
    begin
        if(rst)
            pc<=0;
        else
        begin
            if(PCWrite||(PCWriteCond1&&Zero)||(PCWriteCond2&&~Zero))
            begin
                pc<=pcn;
            end
         end
     end
endmodule
```

(3.3) MUX_MEM 模块，内存中指出的多选器模块，其他的多选器模块都是同一个逻辑，不再展开赘述。

```
module MUX_MEM(
    input signal,
    input [31:0]in1,
    input [31:0]in2,
    output reg [31:0]out
);
always@(*)
begin
    if(signal==0)
        out=in1;
    else     out=in2;
end
endmodule
```

(3.4) Ins_Reg 指令寄存器模块，内部包含 Decoder 译码器模块。译码器模块把 32 位指令翻译成操作码、操作数、地址等。

```
module IR_Decoder(
input [31:0]instruction,
output  [5:0]Instruction31_26,
output  [4:0]Instruction25_21,
output [4:0]Instruction20_16,
output [4:0]Instruction15_11,
output  [15:0]Instruction15_0,
output [5:0]Instruction5_0,
output [25:0]Jaddr
    );
    assign Instruction31_26=instruction[31:26];
    assign Instruction25_21=instruction[25:21];
    assign Instruction20_16=instruction[20:16];
    assign Instruction15_11=instruction[15:11];
    assign Instruction15_0=instruction[15:0];
    assign Instruction5_0=instruction[5:0];
    assign Jaddr=instruction[25:0];
endmodule
```

寄存器作为寄存单元用来保存数据一个时钟周期，接下来的寄存器不再赘述。

```
module Ins_Reg(
input clk,
input IRWrite,
input [31:0]MemData,
output  [5:0]Instruction31_26,
output  [4:0]Instruction25_21,
output [4:0]Instruction20_16,
output [4:0]Instruction15_11,
output  [15:0]Instruction15_0,
output  [5:0]Instruction5_0,
output [25:0]Jaddr
);
reg [31:0]instruction;
always@(posedge clk)
begin
    if(IRWrite)
        instruction<=MemData;
end
IR_Decoder C(instruction,Instruction31_26,Instruction25_21,Instruction20_16,Instruction15_11,Instruction15_0,Instruction5_0,Jaddr);
endmodule
```

(3.5) Registers 模块，寄存器堆。

```
reg [31:0] regFile [0:31];
integer i;
initial
begin
    for (i = 0; i < 32; i = i + 1)
        regFile[i] = 32'b0;
end
assign reg_data=regFile[addr];
always@(posedge clk)
if(rst)
begin
    for (i = 0; i < 32; i = i + 1) regFile[i] = 32'b0;
end
else
begin
    if (RegWrite) regFile[writeR] <= writedata;
end

assign readdata1=regFile[readR1];
assign readdata2=regFile[readR2];
```

(3.6) ALU 模块，CPU 运算的核心模块。执行 add and or xor slt 运算。用状态机来写。

```
    case(ALUop)
    2'b 00:
    begin
        ALUout=ALUin1+ALUin2;
    end
    2'b 01:
    begin
        ALUout=ALUin1-ALUin2;
    end
    2'b 10:
    begin
       case(func)
       6'b 100000:ALUout=ALUin1+ALUin2;
       6'b 100001:ALUout=ALUin1+ALUin2;
       6'b 100010:ALUout=ALUin1-ALUin2;
       6'b 100011:ALUout=ALUin1-ALUin2;
       6'b 100100:ALUout=ALUin1&ALUin2;
       6'b 100101:ALUout=ALUin1|ALUin2;
       6'b 100110:ALUout=ALUin1^ALUin2;
       6'b 100111:ALUout=~(ALUin1|ALUin2);
       6'b 101010:
       begin
           if(ALUin1<ALUin2 )
               ALUout=1;
           else
               ALUout=0;
       end
       default ALUout=0;
       endcase
```

```
            2'b 11:
            begin
               case(op)
               addi :ALUout= ALUin1+ALUin2;
               andi :ALUout=ALUin1&ALUin2;
               ori : ALUout=ALUin1|ALUin2;
               xori : ALUout=ALUin1^ALUin2;
               slti :
               begin
                   if(ALUin1<ALUin2)
                       ALUout=1;
                   else
                       ALUout=0;
                end
               endcase
             end
             endcase
        end
        assign Zero=ALUout==0?1:0;
        endmodule
```

(4) seg_ctrl 模块，用来控制寄存器内容或者内存地址输出到七段数码管上。

```
always @(posedge clk or posedge rst)
begin
if(rst)
    cnt<=0;
 else if(cnt<32'h 1000_0000)
 cnt<=cnt+1;
 else cnt<=0;
end
always@(*)
case(cnt[16:14])
3'b000:begin  an=8'b1111_1110; data=out[3:0]; end
3'b001:begin an=8'b1111_1101; data=out[7:4]; end
3'b010:begin an=8'b1111_1011; data=out[11:8]; end
3'b011:begin an=8'b1111_0111; data=out[15:12]; end
3'b100:begin an=8'b1110_1111; data=out[19:16]; end
3'b101:begin an=8'b1101_1111; data=out[23:20]; end
3'b110:begin an=8'b1011_1111; data=out[27:24]; end
3'b111:begin an=8'b0111_1111; data=out[31:28]; end
endcase
assign out=mem==1? mem_data:reg_data;
```

```
always@(*)
case(data)
0: seg= 8'b 1100_0000;
1: seg= 8'b 1111_1001;
2: seg= 8'b 1010_0100;
3: seg= 8'b 1011_0000;
4: seg= 8'b 1001_1001;
5: seg= 8'b 1001_0010;
6: seg= 8'b 1000_0010;
7: seg= 8'b 1111_1000;
8: seg= 8'b 1000_0000;
9: seg= 8'b 1001_0000;
10:seg= 8'b 1000_1000;
11:seg= 8'b 1000_0011;
12:seg= 8'b 1100_0110;
13:seg= 8'b 1010_0001;
14:seg= 8'b 1000_0110;
15:seg= 8'b 1000_1110;
```

(5) dist_mem_gen0 模块, 存储器, 引入 coe 文件完成测试, 相当于内存。

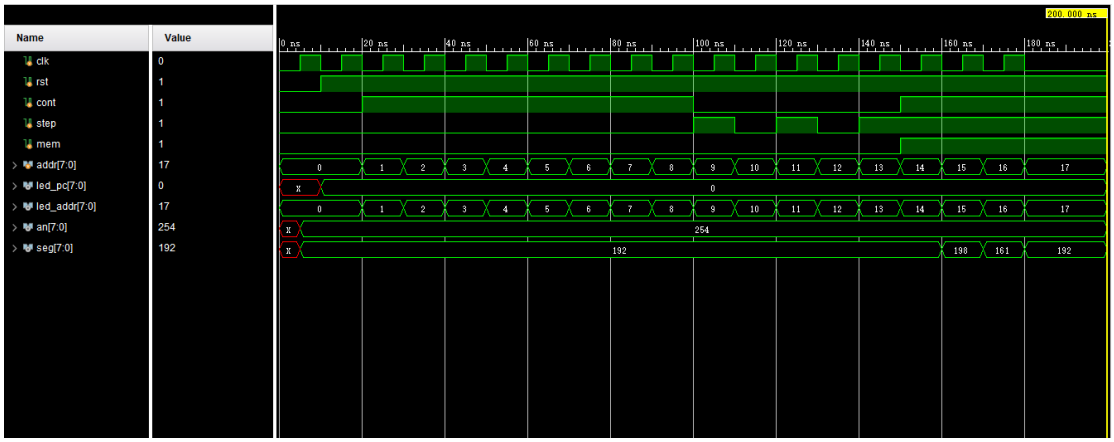
2. 仿真结果与下载结果:

(1) 仿真

```
DDU DUT(.clk(clk),.rst(rst),.cont(cont),.step(step),.mem(mem),.addr(addr),.led_pc(led_pc),.led_addr(led_addr),.an(an),.seg(seg));
initial
begin
    #8000 $finish;
end
initial
begin
    clk=0; rst=0; cont=0; step=0; addr=0; mem=0;  #5
    clk=1; #5
    clk=0; rst=1; #5
    clk=1; #5
    clk=0; cont=1; addr=1; #5
    clk=1; #5
    clk=0; addr=2; #5
    clk=1; #5
    clk=0; addr=3; #5
    clk=1; #5
    clk=0; addr=4; #5
    clk=1; #5
    clk=0; addr=5; #5
    clk=1; #5
```

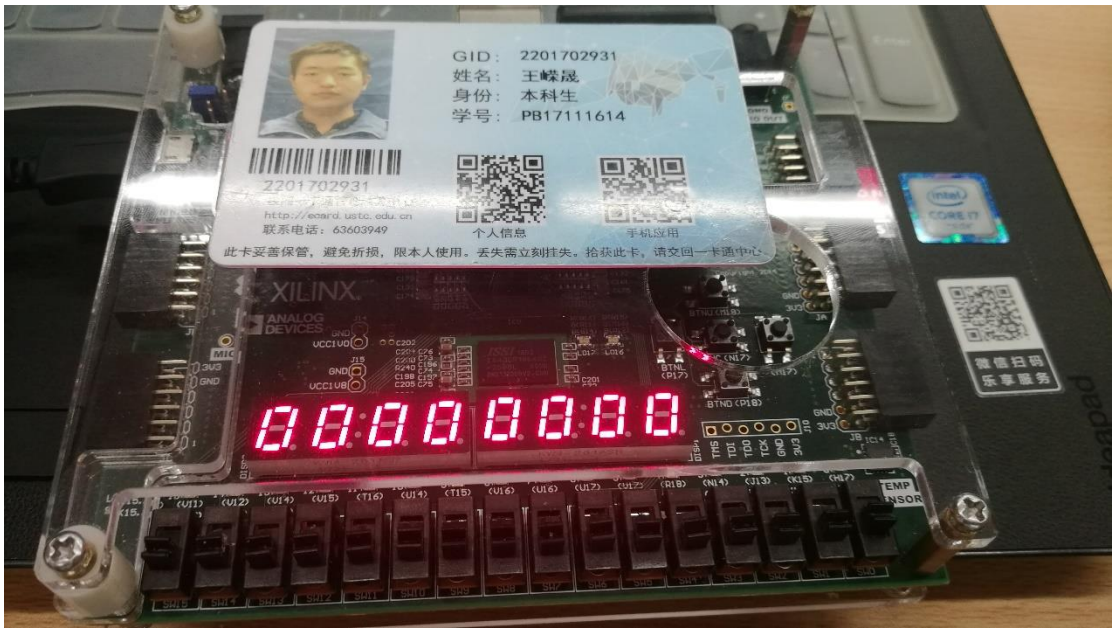
```
clk=0; addr=8; #5
clk=1; #5
clk=0; addr=9; cont=0; step=1; #5
clk=1; #5
clk=0; addr=10; step=0; #5
clk=1; #5
clk=0; addr=11; step=1; #5
clk=1; #5
clk=0; addr=12; step=0; #5
clk=1; #5
clk=0; addr=13; step=1; #5
clk=1; #5
clk=0; addr=14; cont=1; mem=1; #5
clk=1; #5
clk=0; addr=15; #5
clk=1; #5
clk=0; addr=16; #5
clk=1; #5
clk=0; addr=17; #5;
```

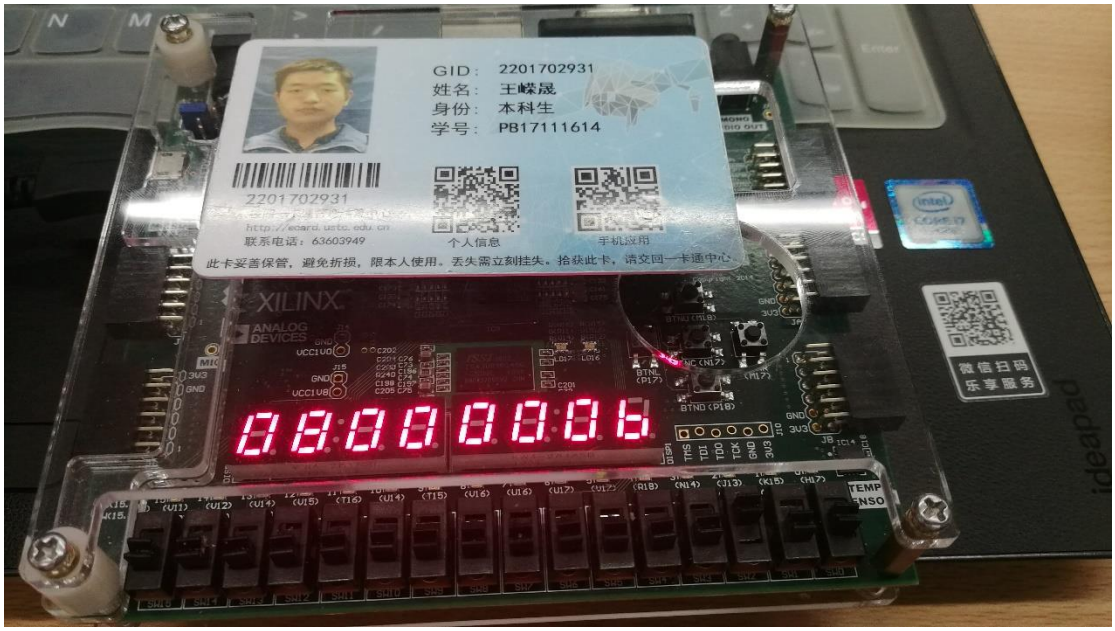
仿真代码



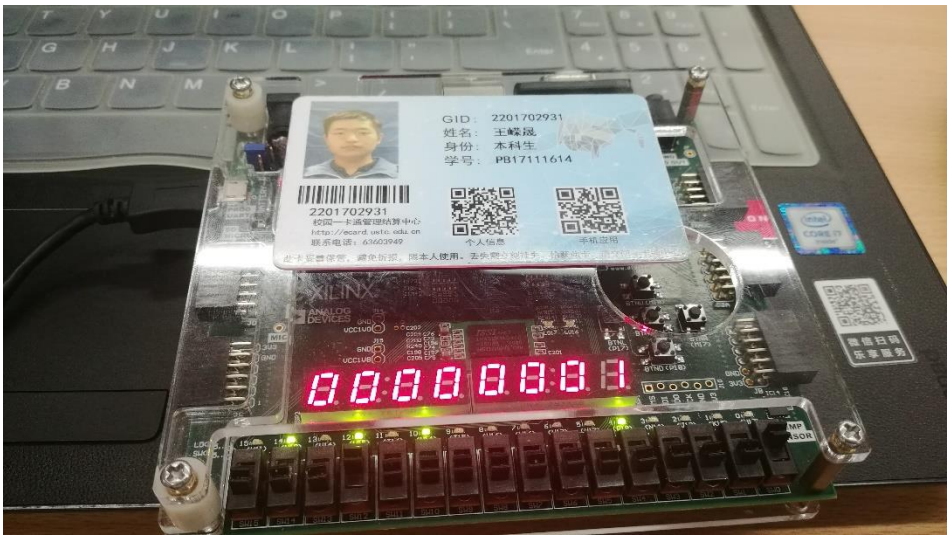
仿真结果

(2) 下载

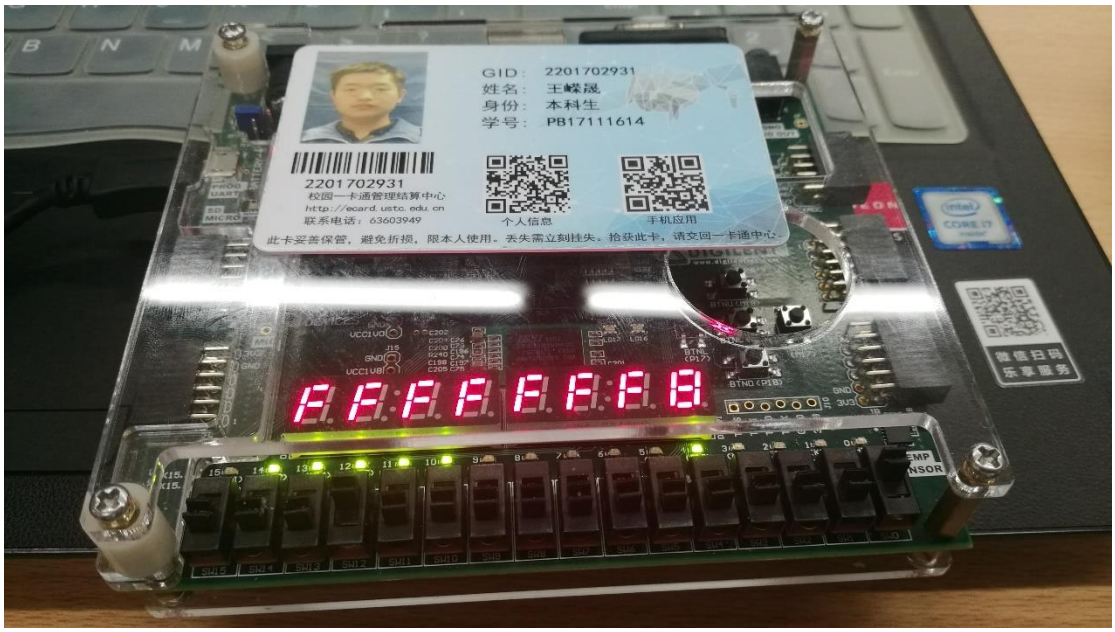

GID： 2201702931
姓名： 王嵘晟
身份： 本科生
学号： PB17111614
2201702931
http://eeard.ustc.edu.cn
联系电话：63603949
个人信息
手机应用
此卡妥善保管，避免折损，限本人使用。丢失需立刻挂失。拾获此卡，请交回一卡通中心
XILINX
ANALOG DEVICES
微信扫码
乐享服务


GID： 2201702931
姓名： 王嵘晟
身份： 本科生
学号： PB17111614
2201702931
http://eeard.ustc.edu.cn
联系电话：63603949
个人信息
手机应用
此卡妥善保管，避免折损，限本人使用。丢失需立刻挂失。拾获此卡，请交回一卡通中心
XILINX
ANALOG DEVICES
微信扫码
乐享服务


GID： 2201702931
姓名： 王嵘晟
身份： 本科生
学号： PB17111614
2201702931
个人信息
手机应用
此卡妥善保管，避免折损，限本人使用。丢失需立刻挂失。拾获此卡，请交回一卡通中心。
XILINX
ANALOG DEVICES


GID： 2201702931
姓名： 王嵘晟
身份： 本科生
学号： PB17111614
2201702931
校园一卡通管理结算中心
http://ecard.ustc.edu.cn
联系电话：63603949
个人信息
手机应用
XILINX
ANALOG DEVICES


GID： 2201702931
姓名： 王嵘晟
身份： 本科生
学号： PB17111614
2201702931
校园一卡通管理结算中心
http://ecard.ustc.edu.cn
联系电话：63603949
个人信息
手机应用
此卡妥善保管，避免折损，限本人使用。丢失需立刻挂失。拾获此卡，请交回一卡通中心。
XILINX
ANALOG DEVICES



3. 结果分析

下载结果正确， 连续运行时稍微有点慢。

4. 实验总结

本次试验算是一次综合实验，把之前做的状态机、ALU、寄存器堆等试验都连接到了一起，同时结合计算机组成原理这门课上所学的多周期 MIPS CPU 原理，才能完成这次实验，难度的确不小。写控制信号的状态机是一个难点，另外由于模块众多，对连接总线也是充满了难度。不过学习过这个试验之后，对 CPU 的组成有了一定的深入理解。