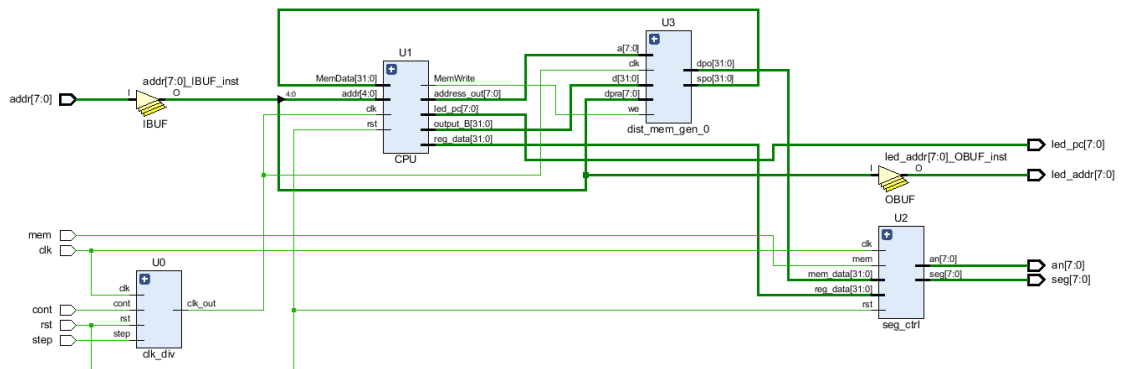


# Lab6 综合设计

## 用多周期 MIPS CPU 实现 Fibonacci 数列

### 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。同时此模块作为显示 Fibonacci 数列的模块，输入 Fibonacci 数列的项  $n$ ，输出  $f(n)$

```

) module ddu(
    input cont,
    input clk,
    input rst,
    input en,
    input [5:0]sw,
    output reg[15:0]led,
    output [6:0]seg,
    //output [31:0] data,
    output [7:0]an
    //output reg dp
);
    wire step;
    wire mem;
    wire inc;
    wire dec;
    wire run;
    reg clk_en;
    reg [7:0] addr=0;
    wire [7:0]addr0;
    assign addr0 = 1;
    wire [31:0] data;
    wire [7:0] pc;
    reg inc_sig=0;
    reg dec_sig=0;
    reg step_sig=0;

    always @(posedge clk,posedge rst)
    begin
        if (rst) addr<=0;
        else
            begin
                if (inc & ~inc_sig)
                begin
                    addr<=addr+1;
                    inc_sig=1;
                end
                if (~inc) inc_sig=0;
                if (dec & ~dec_sig)
                begin
                    addr<=addr-1;
                    dec_sig=1;
                end
                if (~dec) dec_sig=0;
            end
        end

        reg [3:0]seg_in;
        reg [7:0]seg_sel;
        seg_ctrl seg_ctrl(seg_in,seg_sel,seg,an);
        reg [23:0]cnt;
        always @(posedge clk,posedge rst)
        begin
            if (rst)
            begin
                seg_in<=0;
                seg_sel<=8'b11111111;
            end
        end
    end
endmodule

```

顶层模块只做输入输出接口作用，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。

```
30 reg clk_slow;
31 reg [31:0] cnt;
32 always @(posedge clk or posedge rst)
33 begin
34     if(rst)
35     begin
36         cnt<=0;
37         clk_slow<=0;
38     end
39     else if(cnt<=32'h 100_0000)
40         cnt<=cnt+1;
41     else
42     begin
43         cnt<=0;
44         clk_slow<=~clk_slow;
45     end
46 end
47 always@(*)
48 begin
49     if(const==1)
50         clk_out=clk_slow;
51     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\_ctrl, 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。



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==slli)    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

第二段通过时序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 ;
      PCSrc=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
  endcase
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; PCWriteCondi=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
    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
    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
|     end

```

```

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。

并且使用这个 IP 核来实现 I/O

```

// memory
dist_mem_gen_0 MEM(
    sw[5] = 0?Adr[7:0]:2'd2,
    sw[5] = 0?B:sw[4:0],
    sw[5] = 0?addr:2'd2,
    clk,
    sw[5] = 0?MemWrite:1'b1,
    MDR,
    data1);

```

(6) Fibonacci 数列汇编代码

```

1 | .text
2 |     j main
3 | .data
4 |     .word 0,7,0
5 |
6 | .text
7 | main:
8 |     addi $s0, $zero, 0
9 |     addi $s1, $zero, 1
10 |    add $s2, $zero, $zero
11 |    addi $s3, $zero, 1
12 |    lw $s4, 4($0)
13 |    #addi $s4, $zero, 12
14 | loop:
15 |    beq $s3, $s4, success
16 |    add $s2, $s1, $s0
17 |    add $s0, $zero, $s1
18 |    add $s1, $zero, $s2
19 |    addi $s3, $s3, 1
20 |    j loop
21 | success:
22 |    sw $s2, 12($zero)
23 |

```

```

memory_initialization_radix=16;
memory_initialization_vector=
08000003
00000000
00000111
00000000
20080001
20090001
200b0000
8c100008
216b0001
11700004
01285020
21280000
21490000
08000007
ac0a0004;

```

## 2. 仿真结果与下载结果：

### (1) 仿真

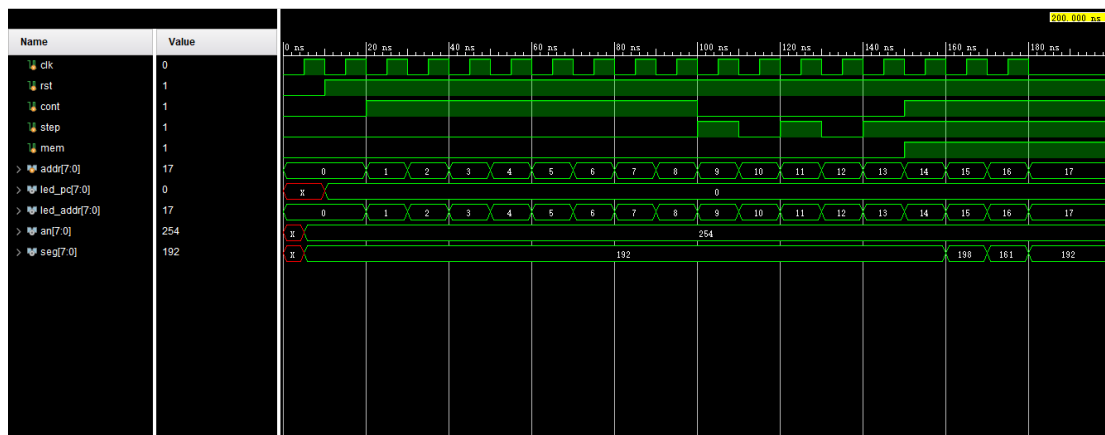
```

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

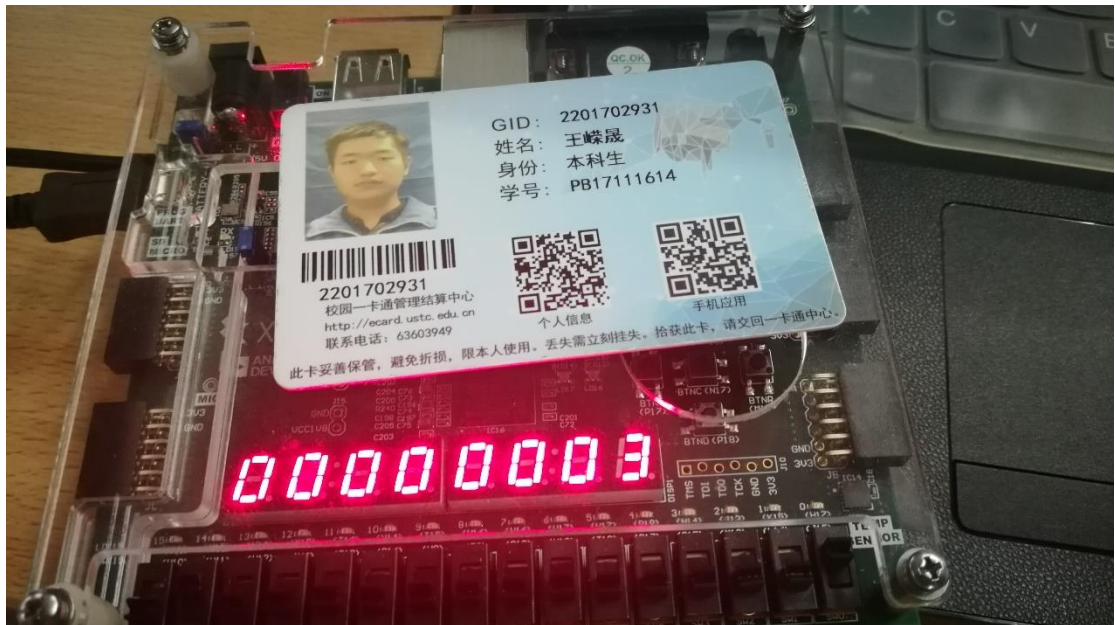


### 初始化 rst 后

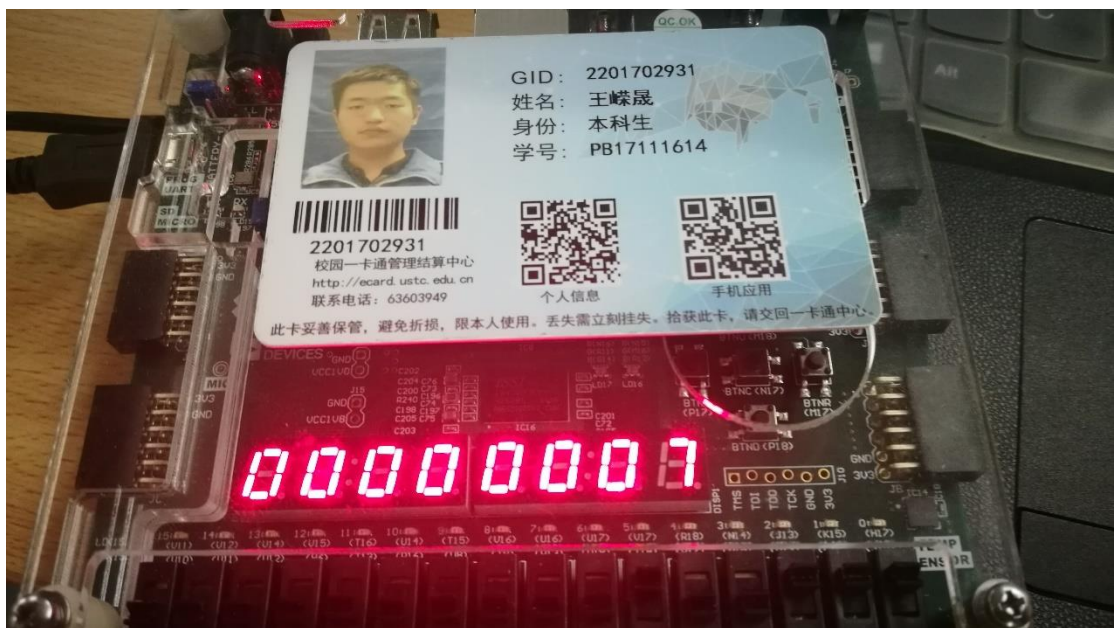


### 输入 n=1

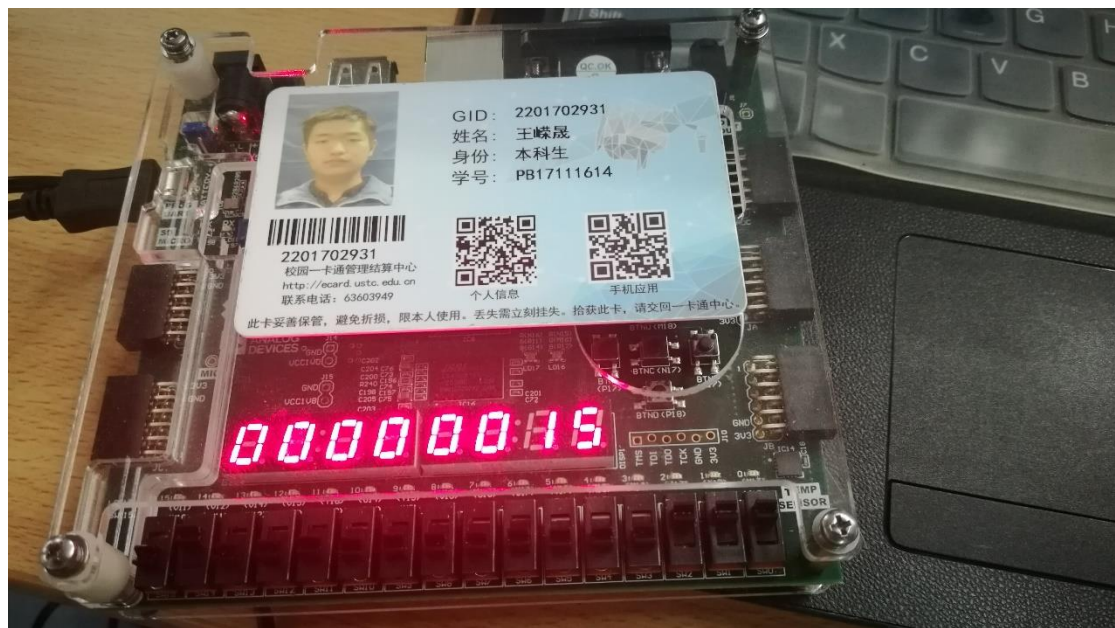




输出  $F(1)=3$



输入  $n=7$



输出  $F(7)=21,16$  进制 15



$F(10)=89, 16$  进制 59

### 3. 结果分析

下载结果正确，可以直接由输入的值  $n$  来决定输出的  $F(n)$

### 4. 实验总结

本次试验是一次综合实验，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，并且要自己完成 MIPS 汇编代码，才能完成这次实验，难度的确不小。使用之前的 CPU 通过调用 IP 核加入了 I/O 来完成实验。使用 Mars 编写汇编代码的时候，由于一开始使用不熟练导致转换成十六进制代码时出了 bug 很久没有解决。在处理 verilog 硬件代码时，由于大部分代码都是上一次实验的内容，并没有太大的问题。