实验报告

实验题目：16 位 CPU 的设计　　日期：2018 年 12 月 14 日

姓名:廖洲洲　　学号:PB17081504　　成绩:_____

目录

1 功能

一 运行存储在 ROM 中的程序来实现某一功能，即可以

（1） 处理指令：严格按程序规定的顺序执行指令（16 位指令）

（2） 执行操作：根据指令的功能，产生相应的操作控制信号，发给相应的部件，从而控制这些部件按指令的要求进行动作

（3） 处理数据：可以对数据进行算术运算、逻辑运算、位运算等基本运算（16 位数据）

二 查看程序运行过程

（1） 通过开关控制在七段数码管显示寄存器文件中各个寄存器的值和当前的指令地址 PC 值

（2） 通过 16 个 LED 灯显示当前指令

三 控制程序进程-时钟控制和单步执行控制

（1） 为了能清楚地了解程序运行过程，给 CPU 一个单步控制开关，只有当开关打开，程序才会执行，关闭，程序则停止在当前指令状态。

（2） 设计了一个时钟控制器，用于选择 CPU 时钟为 5mhz 或 10hz。10hz 的时钟能清楚的观看程序运行的每一步，5mhz 的时钟能实现快速运算功能。

2 概要设计

一 指令系统的设计

（1） 单条指令的设计：一条指令根据操作码的不同，指令的化分也有所不同，大致设计如下

指令	1010	0001	0010	0011
1	操作码	操作数地址一	操作数地址二	操作数地址三
2	操作码	待保留的操作数地址	运算结果的保留地址	
3	操作码	操作数地址	指令跳转地址	
4	整体表示为立即数			

（2） 指令集的分类：根据操作的不同有下图

0000	add (加)	算术运算
0001	sub (减)	
0010	multi (乘)	
0011	div (除)	
0100	shiftL (左移)	位运算
0101	shitfR (右移)	
0110	and (与)	逻辑运算
0111	or (或)	
1000	cmp （比较）	比较
1001	equal (相等)	
1010	reg (寄存器值传递)	寄存器传输
1011	loadi (赋值)	装载
1100	jz0 (0跳转)	分支
1101	jz1 (1跳转)	
1110	store (存储)	存储
1111	halt (结束)	结束

例： memory[0]=16'b1011_0000_0000_0000;

memory[1]=16'b0000_0000_0000_0000;//R0=0

memory[19]=16'b0010_0111_0101_0110;//R11=R5*R6

memory[21]=16'b0101_0000_0000_0011;//R0=R0>>3

memory[45]=16'b1100_0011_0010_1001;//当 R3 为 0，跳转到地址为 0010_1000 的指令

二 模块化部件设计（由于本设计采用模块化设计，故便于修改）

（1） 显示系统 Display:能将任意 16 位和 8 位二进制数转换为 8421BCD 码，并在数码管显示

（2） CPU 部件设计

- ctrl：控制器，其是 CPU 主要部分，它的工作分为
  a) 获取指令
  b) 解码
  c) 发出控制信号控制运算器 ALU，存储器 ROM、RAM 进行工作
- AlU：算术逻辑单元，具体功能为
  a) 直通
  b) 判断是否为 0
  c) 加、减、乘、除
  d) 位左移、位右移
  e) 比较大小
  f) 与、或、非
- mux：二选一数据选择器
- register：寄存器
- registerfile：寄存器堆，本设计采用 4 根地址线，即 16*16 的寄存器堆

（3） 只读存储器 ROM：存储程序机器码，本设计采用的是 8 根地址线，即 256*16 的 ROM

（4） 随机存储器 RAM：可以读取数据，本设计采用的 8 根地址线，即 256*16 的 RAM

三 系统整体设计

(1) 顶层模块 Cpu_Unity:连接 Cpu_mem 和 Display，因此可以查看 CPU 进程和控制 CPU 单步执行。

(2) Cpu_mem 的设计：负责连接 CPU 和 ROM、RAM 实现程序的读取和数据的存储

(3) Cpu 的设计：负责连接数据路径 datapath 和控制器 ctrl

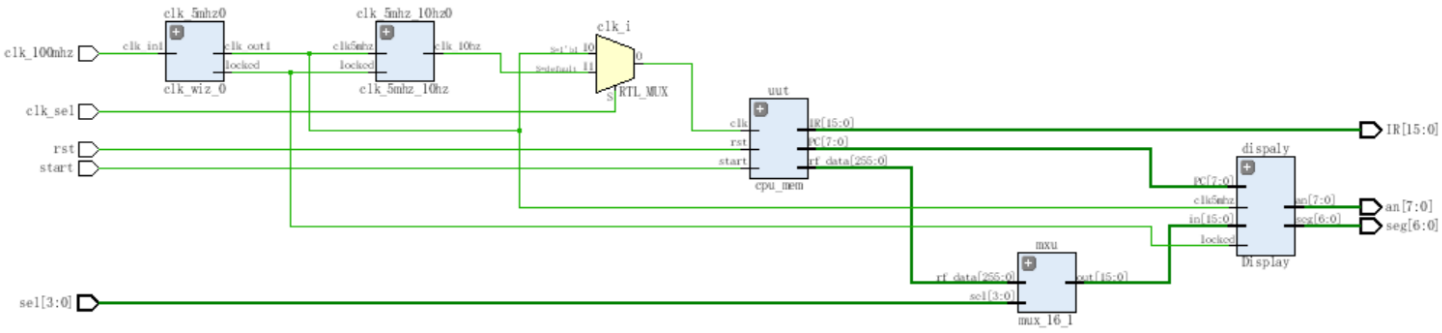
(4) datapath 的设计：负责连接 ALU、mux、register、registerfile 等单元，实现数据的各种操作

3 源代码

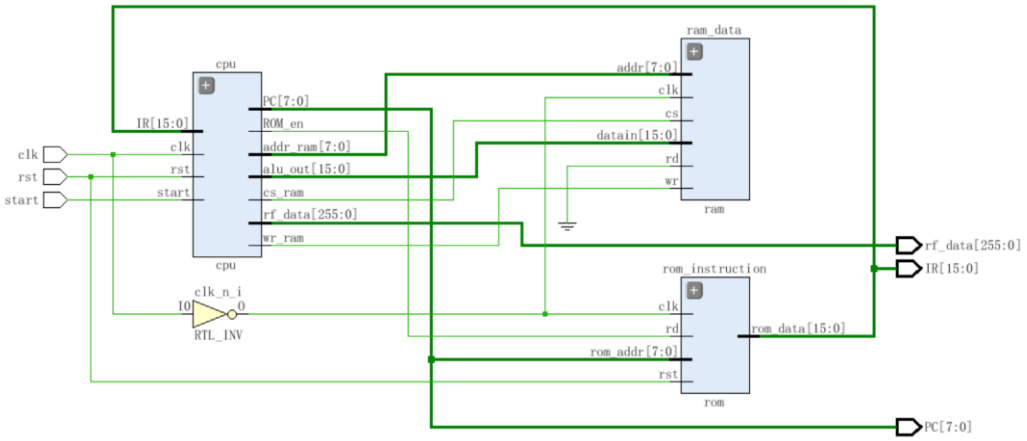
（见文件 CPU_demo.docx_约 950 行）

4 电路图

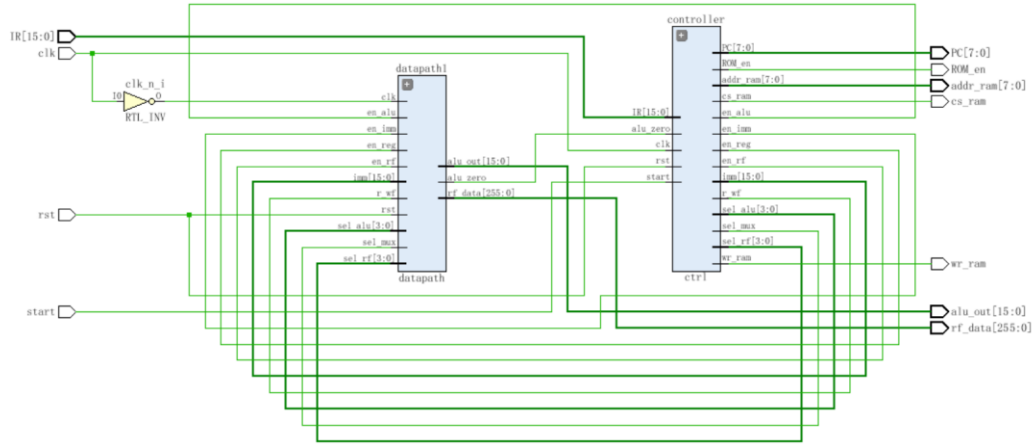
一 整体电路



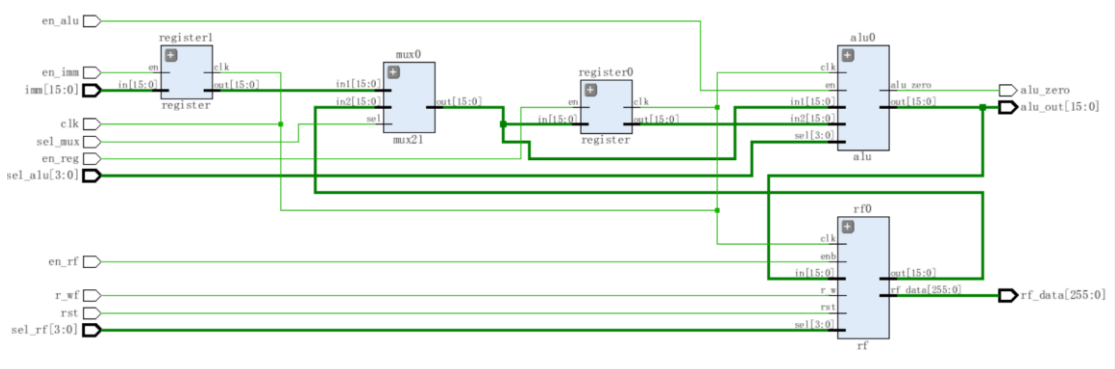
二 cpu_mem 电路



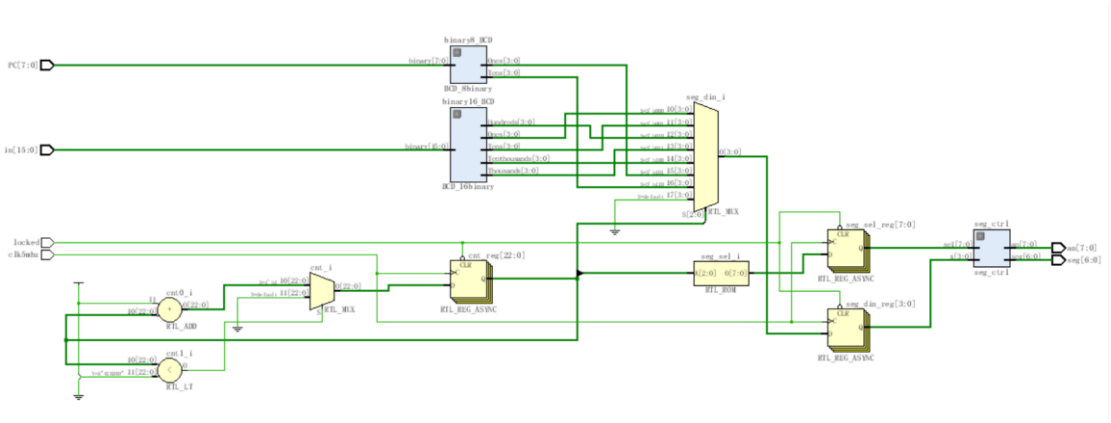
三 cpu 电路



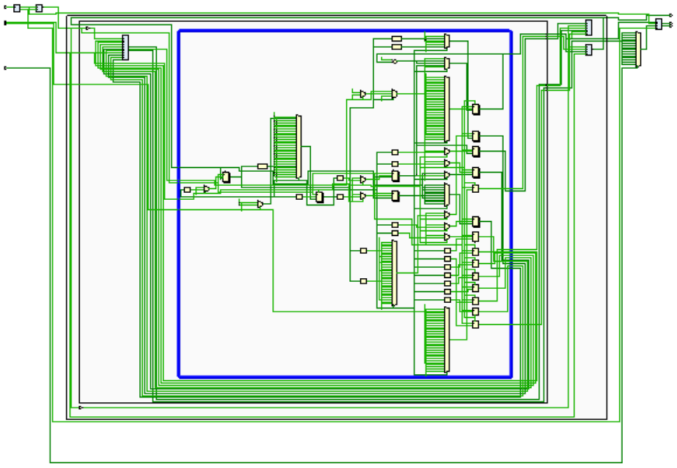
四 datapath 电路



五 Display 电路



六 ctrl 电路



5 测试分析

第一次测试：

（1） 编写 c 语言代码：

```
main()
    {
    int total=0;
    int i;
    for(i=10;i>0;i--)
        total+=2*I;
    int multi1=13,multi2=3;
    int result1=multi1*multi2;
    int result2=result1/multi2;
    multi1=multi1>>3;
    int flag=0;
```

```
        flag=multi1&&multi2;
        flag=result1&&i;
        flag=result1||i;
        if(flag==0){
            int tb=100;
            for(;i<10;i++)
                tb++;
        }
        else{
            int tb=1;
            for(;i<10;i++)
                tb++;
        }
}
```

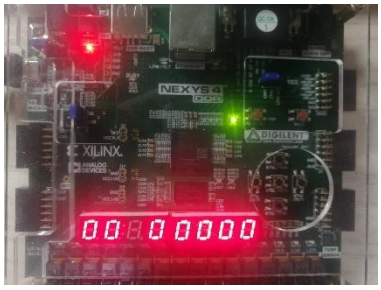
该程序运行结束后各个变量的值应total=110,result1=39,result2=13,multi1=1,flag=1,tb=11;

（2） 程序的机器码表示存储入 ROM

```
memory[0]=16'b1011_0000_0000_0000;
memory[1]=16'b0000_0000_0000_0000;
memory[2]=16'b1011_0001_0000_0000;
memory[3]=16'b0000_0000_0000_1010;
memory[4]=16'b1011_0010_0000_0000;
memory[5]=16'b0000_0000_0000_0001;
memory[6]=16'b1011_0011_0000_0000;
memory[7]=16'b0000_0000_0000_0000;
memory[8]=16'b1100_0001_0000_1100;
memory[9]=16'b0000_0000_0000_0001;
memory[10]=16'b0001_0001_0001_0010;
memory[11]=16'b1100_0011_0000_1000;
memory[12]=16'b1010_0100_0000_0000;
memory[13]=16'b0100_0100_0000_0001;
memory[14]=16'b1110_0100_0000_1010;
memory[15]=16'b1011_0101_0000_0000;
memory[16]=16'b0000_0000_0000_1101;
memory[17]=16'b1011_0110_0000_0000;
memory[18]=16'b0000_0000_0000_0011;
memory[19]=16'b0010_0111_0101_0110;
memory[20]=16'b0011_1000_0111_0110;
memory[21]=16'b0101_0101_0000_0011;
memory[22]=16'b1011_1001_0000_0000;
memory[23]=16'b0000_0000_0000_0000;
memory[24]=16'b0110_1001_0101_0101;
memory[25]=16'b0110_1001_0111_0001;
memory[26]=16'b0111_1001_0111_0001;
memory[27]=16'b1101_1001_0010_0101;
memory[28]=16'b1011_1010_0000_0000;
memory[29]=16'b0000_0000_0110_0100;
memory[30]=16'b1011_1011_0000_0000;
memory[31]=16'b0000_0000_0000_1010;
memory[32]=16'b1000_1100_1011_0001;
memory[33]=16'b1100_1100_0010_1110;
memory[34]=16'b0000_0001_0001_0010;
memory[35]=16'b0000_1010_1010_0010;
memory[36]=16'b1100_0011_0010_0000;
memory[37]=16'b1011_1010_0000_0000;
memory[38]=16'b0000_0000_0000_0001;
memory[39]=16'b1011_1011_0000_0000;
memory[40]=16'b0000_0000_0000_1010;
memory[41]=16'b1000_1100_1011_0001;
memory[42]=16'b1100_1100_0010_1110;
memory[43]=16'b0000_1010_1010_0010;
memory[44]=16'b0000_0001_0001_0010;
memory[45]=16'b1100_0011_0010_1001;
memory[46]=16'b1111_0000_0000_0000;
```

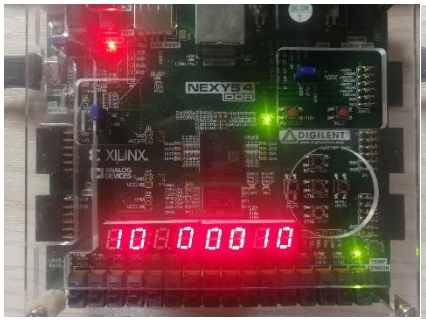
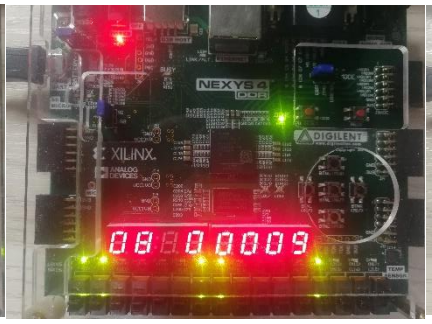
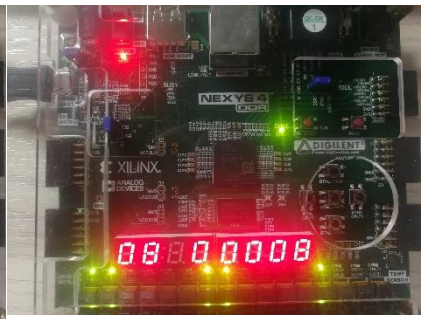
（3） FPGA 演示

- 开始未执行，控制开关向上拨后，程序执行

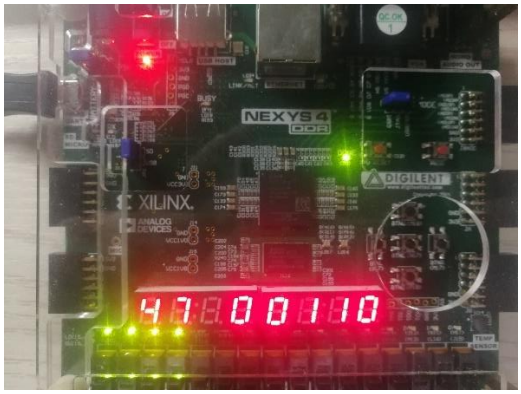
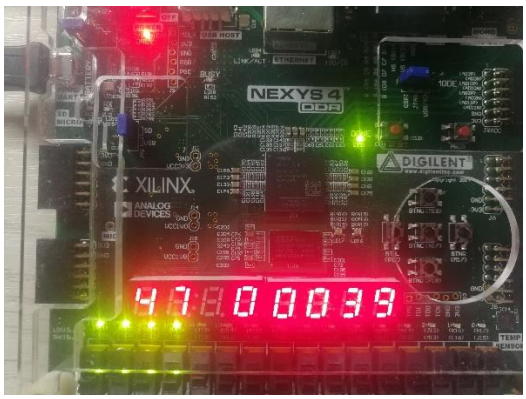


开始未执行时，PC=0，各个寄存器的值都为 0，指令值为 0

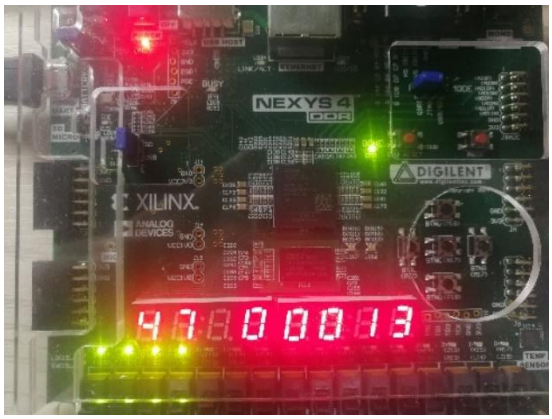
- start 可以控制 CPU 单步运行，我们可以看到 i 从 10 到 0 的过程，当然，由于时钟的频率较小，不用拨下控制开关，我们也可以看清 CPU 运算过程

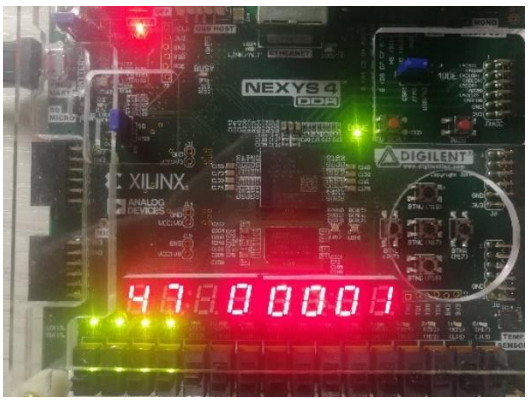
- 最后查看各个变量的值-发现结果均正确

total=110

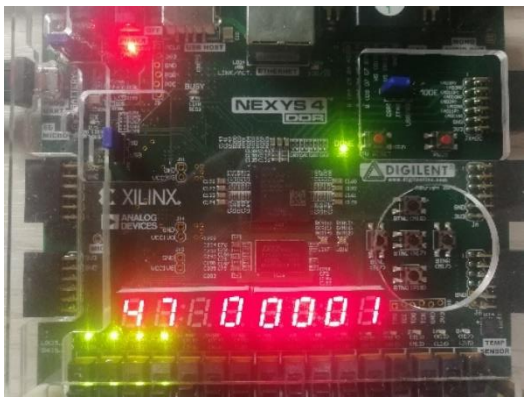


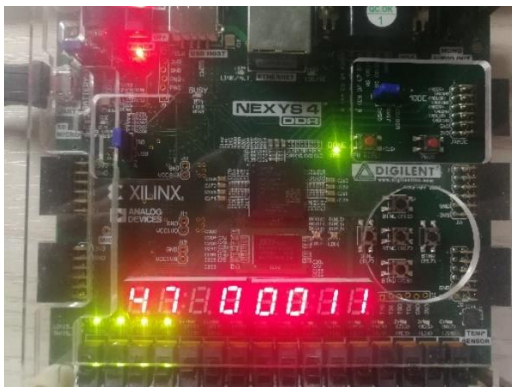
result1=39



result2=13

multi1=1





flag=1

tb=11

- 若更改频率为 5mhz，可以发现程序结果可以瞬间出来

第二次测试：

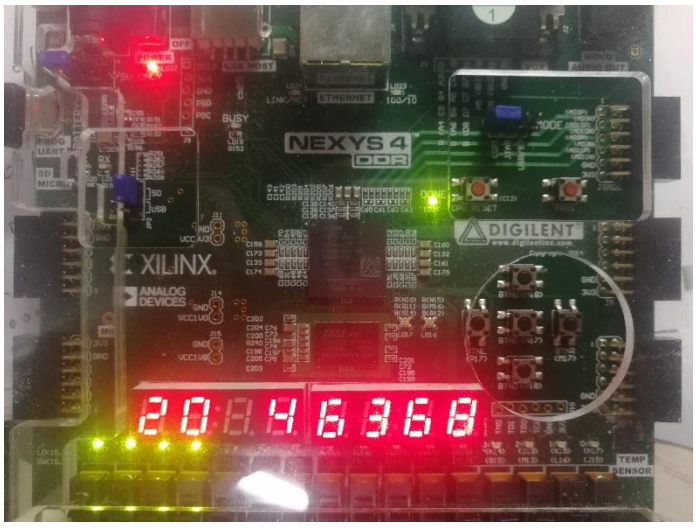
（1） 编写 c 程序代码—斐波那契数列

```
main(){
    int i=0;
    int f,Fn=0,FN=1;
    for(i=0;  i<=22;i++){
        f=Fn+FN;
        Fn=FN;
        FN=f;
        }
```

（2） 程序的机器码表示存储入 ROM

```
memory[0]=16'b1011_0000_0000_0000;
memory[1]=16'b0000_0000_0000_0000;
memory[2]=16'b1011_0001_0000_0000;
memory[3]=16'b0000_0000_0000_0001;
memory[4]=16'b1011_0010_0000_0000;
memory[5]=16'b0000_0000_0000_0000;
memory[6]=16'b1011_0011_0000_0000;
memory[7]=16'b0000_0000_0000_0001;
memory[8]=16'b1011_0100_0000_0000;
memory[9]=16'b0000_0000_0001_0110;
memory[10]=16'b1000_0101_0000_0100;
memory[11]=16'b1101_0101_0001_0011;
memory[12]=16'b0000_0001_0010_0011;
memory[13]=16'b1010_0010_0011_0000;
memory[14]=16'b1010_0011_0001_0000;
memory[15]=16'b1011_0110_0000_0000;
memory[16]=16'b0000_0000_0000_0001;
memory[17]=16'b0000_0000_0000_0110;
memory[18]=16'b1101_0110_0000_1010;
memory[19]=16'b1111_0000_0000_0000;
```

(3) FPGA 演示：得到第 24 个斐波那契数 46368



6 实验小结

- 通过本次实验，我了解了 Cpu 的工作原理，并设计了 Cpu 的基本组件 ALU、Mux、Register、RegisterFile、Cpu 控制模

块 Ctrl 等和与 Cpu 相连的 ROM、RAM。本次实验增强了我对 Verilog 的熟练程度，也让我学会了复杂电路的设计，同时体会到了 FPGA 的强大。