信息工程系

教

案

课程名称： 可编程逻辑器件应用

教　　师： 陈彦彬

总 学 时 ： 54

理论学时： 0

实训学时： 54

上课班级： 电子 241、242

授课学期： 2025-2026-2

第 1 次教学活动设计

教学环节	内容设计与手段
学时	**3 学时**
重难点	HDL 的基本概念 VerilogHDL 基本程序结构 VerilogHDL 结构描述语句
导入新课	首先提问同学是否知晓 iPhone7 手机中处理器芯片的名称，然后再提问是否有同学知晓该处理器芯片的大致设计流程。当学生发表完自己的观点后，教师进行总结：电子产品在我们身边无处不在，集成电路芯片是它们的心脏，电子产品都会用到数字电路芯片。iPhone7 手机中处理器芯片的名称是 A11，有人说拆开 iPhone7 发现里面就是一块中高端 FPGA 板，那么它是如何设计出来的，以及如何设计一款电路芯片，从而导入本门课程第一讲内容，即 VerilogHDL 快速入门与上机实践。
讲授内容	**1、芯片设计完整流程描述** 工艺无关的前端设计：规格制定、详细设计、HDL 编码、仿真验证、逻辑综合、综合验证 工艺相关的后端设计：DFT 设计、布局布线、版图验证、寄生参数、仿真验证、芯片制造 **2、HDL 的基本概念** HDL 是一种用形式化方法来描述数字电路和系统的语言。用它可以表示逻辑电路图、逻辑表达式、数字逻辑系统所完成的逻辑功能等。通过 EDA 工具进行仿真验证，进而使用 ASIC 或 FPGA 布局布线工具进行电路和系统的实现。 **3、Verilog HDL 的基本概念** 它是 HDL 的一种，它们具有如下相同点： ■ 能形式化地抽象表示电路的行为和结构； ■ 支持逻辑设计中层次与范围的描述； ■ 可借用高级语言结构简化电路行为描述； ■ 具有电路仿真和验证机制以保证设计的正确性； ■ 支持电路描述由高层到低层的综合转换； ■ 硬件描述与实现工艺无关； ■ 便于文档管理和设计重用。 同时，也具有如下不同点： ■ VHDL 源自美国军方，1987 年成为 IEEE 标准； ■ Verilog 源自民间公司，1995 年成为 IEEE 标准； ■ VHDL 较难掌握，使用的客户群体匮乏； ■ VHDL 语法比 Verilog 严谨而冗长； ■ VHDL 语言几乎不支持电路级底层描述级别； ■ VHDL 语言系统级描述能力较好，而 Verilog 通过扩展为 System Verilog 才丰富了系统级描述能力。 **4、IP 核简介** ■ IP 就是知识产权核或知识产权模块之义。 ■ 美国 Dataquest 公司将半导体产业的 IP 定义为“用于 ASIC 或 FPGA 中的预先设计好

的电路功能模块”

- 软核：经过功能验证、可综合的、实现后电路结构的总门数在 5000 以上的 HDL 模型（编码）。
- 固核：在某一种 FPGA 上实现的、经验证是正确的、总门数在 5000 以上的电路结构编码文件（电路网表）。
- 硬核：在某一种 ASIC 上实现的、经验证是正确的、总门数在 5000 以上的电路结构版图掩膜。

5、Verilog 基本程序结构

```
module module_name(port_list);
    // 声明各种变量、信号
    // *变量如 reg、wire、parameter
    // *信号如 input、output、inout
    // 程序代码，描述
    // initial 块
    // always 块
    // …
endmodule
```

6、数据流描述语句简介

语法：

assign 线网型变量名 = 赋值表达式；

举例：

```
wire a;
wire [3:0] b,c;
assign a = 1;
assign b[1:0] = 2’b10;
assign c = {a,a};
```

注意：

assign 只能对 wire 型数据进行赋值；

数据流描述形式通常只采用 assign 语句进行实现。

7、如何编写一个简单的 Verilog 示例程序？

以移位器为例，使用 Xilinx ISE 开发套件实现电路功能模块和测试模块的编写。

```
module lshift(a,b);

input [3:0] a;
output [3:0] b;

assign b = a << 2;

endmodule
```

测试模块：

```
module testbench;

   reg [3:0] a;
   wire [3:0] b;

   // Instantiate the
   // Unit Under Test (UUT)
   lshift uut2 (
      .a(a),
      .b(b)
   );

   initial begin
      a = 4'b1000;

      // Wait 100 ns
      #100;

      a = 4'b1001;
   end

endmodule
```

8、结构描述语句简介

语法：

门类型 <实例名> (输出,输入 1,输入 2,…,输入 N)

常用的 8 个门类型关键字：

and、nand、or、nor、xor、xnor、buf、not

举例：

```
nand na01(na_out, a, b, c);
xor xo01(xo_out, a, b);
```

注意：

门类型关键字区分大小写；多输入单输出；

Verilog HDL 中的结构描述本质上为门级描述。

9、实例：一个简单的全加器例子

```
module ADD(A, B, Cin, Sum, Cout);
    input A, B, Cin;
    output Sum, Cout;

    wire S1, T1, T2, T3;

    xor X1(S1, A, B), X2(Sum, S1, Cin);

    and A1(T1, A, B), A2(T2, B, Cin), A3(T3, A, Cin);
    or O1(Cout, T1, T2, T3);

endmodule
```

10、行为描述语句简介

行为级描述包含四个方面：

- ◆ 流控制　　典型代表为 if，case, while…
- ◆ 语句块　　典型代表为 begin+end 模块
- ◆ 时序控制　　典型代表为边沿/电平触发事件控制
- ◆ 过程结构　　典型代表为 always 模块

	行为级描述主要用于时序功能的实现。 **11、实例：一个八位计数器的例子** `module led_flash(reset, clk, led);` `  input reset, clk;` `  output led;` `  reg [7:0] cnt = 0;` `  wire led;` `  always @(posedge clk)  begin` `  if (reset)` `  cnt <= 0;` `  else` `  cnt <= cnt + 1;` `  end` `  assign led = cnt[7];` `endmodule`
归纳总结	① 扼要地介绍了基本概念； ② 重点介绍了 Xilinx ISE 套件对 Verilog HDL 程序的仿真与测试； ③ 布置第一次实验报告： 给出 c17 电路的结构描述形式，如何进行仿真测试？写出实验报告。 附 c17 电路网表如下： INPUT(G1) INPUT(G2) INPUT(G3) INPUT(G4) INPUT(G5) OUTPUT(G16) OUTPUT(G17) G8 = NAND(G1,G3) G9 = NAND(G3,G4) G12 = NAND(G2,G9) G15 = NAND(G9,G5) G16 = NAND(G8,G12) G17 = NAND(G12,G15)

第 2 次教学活动设计

教学环节	内容设计与手段
学时	**3 学时**
重难点	Verilog 中的标志符 Verilog 数据类型 Verilog 行为描述语句 阻塞与非阻塞赋值
导入新课	1、复习上节课内容，包括几个基本概念：即 HDL、软核、固核、硬核； 2、回顾如何进行 Verilog 程序的仿真和测试。 3、如何界定数据流描述语句和结构描述语句。
讲授内容	**1、Verilog 中的标志符** ◆ 字母、下划线（开头），数字、$符； ◆ 区分大小写？ 例： IF; _ce; CLK_100MHz **2、数据类型** 共 19 种，本讲只介绍常用的四种： wire 型；reg 型；memory 型；parameter 型。 ■wire 型 ・常用来表示组合逻辑信号，常与 assign 一起使用； ・默认值为高阻抗状态，高阻态如何表示？ ■reg 型 ・常用来表示时序逻辑信号，常在 always 块中使用； ・默认值为未知(不定)状态，不定态如何表示？ ※ wire 型需持续驱动，而 wire 型保持最后一次赋值。 变量不指定类型？always 块中被赋值变量的类型？ ■memory 型 常用于将 reg 型扩展为寄存器数组(二维),对存储器建模. 定义格式：reg[n-1:0] 存储器名[m-1:0]; 定义例：reg[7:0] X_ROM[3:0]; 赋值例：X_ROM[0] = 4’hf; X_ROM[1] = 2’b10; X_ROM[2] = 8’o7; X_ROM[3] = 4’d9; 思考：存储位宽？存储深度？存储结果？ ※ 被赋值变量应当在时序逻辑块中； 不能只对某一位赋值，未赋值部分被补 0？ 分析区别：reg[n-1:0] rega; reg memb[n-1:0]; ■parameter 型

用来表示常量（整形、实数型、字符串型）

定义格式：parameter 参数名 ＝ 数据;

例：parameter pai = 3.14; //科学计数法？

parameter[1:0] a = 2'd3,b = 4'd6;//基数表示法

parameter hi = "hi"; //二进制存储 ASCII,1 符 8 位

reg[1:8*7] str = "counter";

3、Verilog 语言的运算符和表达式

算术运算符（简单了解）

赋值运算符

连续赋值 vs. 过程赋值（阻塞、非阻塞）

关系运算符 成立返 1，不成立返 0，不定返？

逻辑运算符

条件运算符 三目运算符

位运算符 ^; ^~; ~&; ~|

移位运算符

拼接运算符 {a,b}

一元约简运算符

4、形为描述语句的过程结构

① initial 模块

定义格式：

initial begin/fork

…

end/join

※ 只执行一次,只面向仿真(不可综合),多个则并行执行.

② always 模块

定义格式：

always @(敏感事件列表) begin/fork

…

end/join

5、形为描述语句的时序控制

① 延迟控制

② 事件控制（对应于敏感事件列表）

·电平触发事件控制

·边沿触发事件控制

上机实践：1、编写 nor3，并进行测试。

2、编写一个八位计数器，并进行测试。

6、**形为描述语句的语句块**

parameter d = 50;

reg[7:0] r;

	begin: block1 // 有名块、串行块 #d r = 'h10; #d r = 'o10; #d r = 'd10; end 如何修改为并行块？ 串行块与并行块的执行特点？混合使用？ **7、形为描述语句的流控制** ① 跳转（if 语句） 注意 else；串行。 ② 分支(case 语句) 注意 default；并行。 ③ 循环 for 语句 与 C 语言语法相同 while 语句while(表达式) begin … end forever 语句 forever begin … end 须在 initial 中 repeat 语句 repeat(次数) begin … end **8、理解阻塞与非阻塞赋值** module eg1 (out, a, b, c, d); input a, b, c, d; // 假设该组值均从 0→1 output out; reg t1, t2, out; always @(a or b or c or d) begin t1 = a & b; // t1 <= a & b; t2 = c & d; // t2 <= c & d; out = t1 | t2; // out <= t1 | t2; end endmodule 分析替换为注释部分的语句，程序将如何响应。
归纳总结	① 扼要地介绍了 Verilog 数据类型和运算符； ② 重点介绍了 Verilog 行为描述语句，完成了上机实践（仿真与测试） ③ 布置第二次实验报告： 试给出 D 触发器的行为描述形式，如何进行仿真测试？写出实验报告。 附一种并未实现 D 触发器功能的代码： module dff (clk, d, q1, q2); input clk, d; output q1, q2; reg q1, q2; always @(negedge clk) begin q1 = d; q2 = q1; end endmodule

第 3 次教学活动设计

<table>
<tr><td>教学
环节</td><td>内容设计与手段</td></tr>
<tr><td>学时</td><td>2 学时</td></tr>
<tr><td>重难点</td><td>过程结构的 task 语句
过程结构的 function 语句</td></tr>
<tr><td>导
入
新
课</td><td>1、复习上节课内容，包括几个基本数据类型的使用、默认值等。
2、回顾 Verilog 程序的仿真和测试基本流程。
3、回顾 Verilog 的描述语句，如数据流描述语句、结构描述语句、形为描述语句。其中，行为描述语句的重要组成是过程结构。在上一节课程中介绍了过程结构中的 initial 语句和语句块。提问还有其他的过程结构吗？</td></tr>
<tr><td>讲
授
内
容</td><td>1、形为描述语句中过程结构的 task 语句
定义格式：
task 任务名;
// 端口及数据类型声明语句
// …
endtask
※ 分解模块,定公共代码,增强程序可读性,可备后续调用。示例：
task my_task;
input a, b;
inout c; // 非必使用
output d, e;
// …
c = xxx1;
d = xxx2;
e = xxx3;
endtask
任务调用：my_task(p, q, u, v, w);

上机实现一个简易的 ALU 任务，代码如下：
module aluTask(a, b, sum, diff);
input [1:0] a, b;
output reg [2:0] sum;
output reg [1:0] diff;
always@(a, b) begin tsk(a, b, sum, diff); end
task tsk;
input [1:0] x, y;
output [2:0] s;
output [1:0] d;
begin s = x + y; d = x - y; end
endtask</td></tr>
</table>

endmodule

使用 task 语句，上机实现一个交通灯轮询的功能模块，代码如下：

```
module traffic_lights;
    reg  clock, red, amber, green;
    parameter  on =1, off=0, red_tics=350,
           amber_tics=30, green_tics=200;
        //交通灯初始化
        initial     red=off;
        initial     amber=off;
        initial     green=off;
    //交通灯控制时序
        always
          begin
                red=on;                    //开红灯
                light(red,red_tics);       //调用等待任务
                green=on;                  //开绿灯
                light(green,green_tics);   //等待
                amber=on;                  //开黄灯
                light(amber,amber_tics);   //等待
          end

    //定义交通灯开启时间的任务
    task  light;
        output  color;
        input[31:0] tics;
    begin
        repeat(tics)
            @(posedge clock);
        color=off;
        end
    endtask
    //产生时钟脉冲的 always 块
    always
        begin
          #100 clock=0;
          #100 clock=1;
        end
endmodule
```

2、形为描述语句中过程结构的 function 语句

定义格式：

function 返回值的类型或范围 函数名;

// 端口及数据类型声明语句

// …

endfunction

※ 用于分解大模块,定义公共代码,增强程序可读性和可维护性,可备其他模块调用。

※ 返回值类型或范围为可选项，默认返回一位 reg 类型。

示意性代码：

```
function [7:0] getLowByte;
    input [15:0] dataA;
    begin
        // …
        getLowByte = result_expression;
    end
endfunction
```

函数调用：result = getLowByte(dataB);

※ 函数返回值的类型和范围，对返回值的赋值。

上机实现一个阶乘的例子，要求使用 function 实现，代码如下：

```
module  tryfact;
  //函数的定义------------------------------
    function[31:0]factorial;
      input[3:0]operand;
      reg[3:0]index;
      begin
        factorial = 1;  //0 的阶乘为 1，1 的阶乘也为 1
        for(index=2; index<=operand; index=index+1)
        factorial = index * factorial;
      end
    endfunction
    //函数的测试------------------------------
    reg[31:0]result;
    reg[3:0]n;
    initial
    begin
      result=1;
      for(n=2;n<=9;n=n+1)
      begin
        $display("Partial result n= %d result= %d", n, result);
        result = factorial(n) ;
      end
      $display("Finalresult=%d",result);
    end
  endmodule //模块结束
```

3、task 语句和 function 语句的注意点

※ 函数的使用规则：

1、函数的定义中不能包含任何时间控制语句（#,@…）；
2、函数不能启动任务（任务能启动任务和函数）；
3、函数的定义至少包含一个输入参数；
4、函数的定义中须有一条赋值语句，用于给函数中一个内部变量赋以函数结果值，且该内部变量与函数名同名

※ 函数与任务的主要不同点：

1、函数只能与主模块共用同一个仿真时间单位，而任务可以定义自己的仿真时间单位,任务中可有时间控制语句
2、同上面 2；
3、函数输入变量至少有一，任务可有任何个数/类型变量；
4、函数返回一个值，而任务则不返回值。

4、简介 Verilog 语言常用的系统任务

※ $display; $write; $fopen; $fdisplay; $fclose;

※ $monitor; $time; $realtime; $stop; $finish;

程序示例：

```
`timescale 10 ns/1 ns
  module test;
    reg set;
    parameter  p=1.55;
    initial
      begin
        $monitor( $realtime,,"set=",set);
        #p set=0;
        #p set=1;
      end
endmodule
```

输出结果为：

```
0 set=x
1.6 set=0
3.2 set=1
```

5、简介 Verilog 语言的编译预处理

※ `timescale; `define; `ifdef; `else; `endif;

※ `include

程序示例：

	`` //条件编译 `ifdef  TEST              //若设置 TEST 标志,则编译 test 模块 module test;     initial         $display("Module %m compiled"); endmodule `else //在默认情况下,则编译 stimulus 模块 module stimulus; initial $display("Module %m compiled"); endmodule `endif                     // `ifdef 语句的结束 ```
归纳总结	① 重点地介绍了 Verilog 任务与函数定义与应用； ② 扼要介绍了 Verilog 常见系统任务和编译预处理，完成了上机实践（仿真与测试）。

第 4 次教学活动设计

教学环节	内容设计与手段
学时	**2 学时**
重难点	分频时序逻辑电路设计 用 always 块实现较复杂的组合电路 Verilog 中函数与任务的使用
导入新课	1、复习上节课内容，包括几个基本概念：Verilog 任务与函数和编译预处理 2、回顾函数的使用规则，函数与任务的区别。 3、回顾 Verilog 的几种描述语句，以及各种描述语句的组成与使用方法。
讲授内容	本节主要进行如下内容的集中上机实践： 一、简单的组合逻辑设计（设计比较器） 二、简单的分频时序逻辑电路设计（1/2 分频） 三、利用条件语句实现计数分频时序电路（1/20 分频） 四、深入理解阻塞与非阻塞的区别 五、用 always 块实现较复杂的组合电路 六、在 Verilog 中使用函数与任务 其中，部分代码参考如下： 一、比较器的设计 `module compare(equal,a,b);` `input a,b;` `output equal;` `reg equal;` `always @(a or b)` `if(a==b)` `equal =1;` `else` `equal = 0;` `endmodule`

```
`timescale 1ns/1ns
`include   "./compare.v"
module t;
    reg a,b;
    wire equal;
    initial
      begin
        a=0;
        b=0;
      #100    a=0; b=1;
      #100    a=1; b=1;
      #100    a=1; b=0;
      #100    a=0; b=0;
      #100    $stop;
      end
      compare  m(.equal(equal),.a(a),.b(b));
endmodule
```

二、1/2 分频率时序电路设计

```
module half_clk(reset,clk_in,clk_out);
  input clk_in,reset;
  output clk_out;
  reg clk_out;

  always @(posedge clk_in)
    begin
      if(! reset)   clk_out=0;
      else          clk_out=~clk_out;
    end
  endmodule


  `timescale 1ns/100 ps
  `define clk_cycle 50
  module  top;
  reg clk,reset;
  wire clk_out;

  always    #`clk_cycle   clk = ~clk;

  initial
    begin
            clk = 0;
            reset = 1;
      #10 reset = 0;
      #110 reset = 1;
      #100000   $stop;
    end
```

```
    half_clk m0(. reset(reset),. clk_in(clk),. clk_out(clk_out));
endmodule
```

三、1/20 分频率时序电路设计

```
module fdivision(RESET,F10 MB,F500 KB);
    input F10 MB,RESET;
    output F500 KB;
    reg F500 KB;
    reg [7:0]j;
      always @(posedge F10 MB)
        if(! RESET)              //低电平复位
          begin
            F500 KB <= 0;
            j<= 0;
          end
        else
          begin
            if(j==19)            //对计数器进
              begin
                j <= 0;
                F500 KB <= ~F500 KB;
              end
            else
              j <= j+1;
          end
    endmodule


`timescale 1ns/100 ps
`define clk_cycle 50
 module division_Top;
reg F10 MB,RESET;
 wire F500 KB_clk;
always # `clk_cycle    F10 MB_clk =~ F10 MB_clk;
 initial
   begin
     RESET=1;
     F10 MB=0;
     #100 RESET=0;
     #100 RESET=1;
     #10000 $stop;
   end

  fdivision fdivision (. RESET(RESET),. F10 MB(F10 MB),. F500 KB(F500 KB_clk));


 endmodule
```

四、深入理解阻塞/非阻塞的区别

```
module blocking(clk,a,b,c);
  output [3:0] b,c;
  input  [3:0] a;
  input        clk;
  reg    [3:0] b,c;
  always @(posedge clk)
  begin
    b = a;
    c = b;
    $display("Blocking: a = %d, b = %d, c = %d.",a,b,c);
  end
endmodule
module non_blocking(clk,a,b,c);
  output [3:0] b,c;
  input  [3:0] a;
  input        clk;
  reg    [3:0] b,c;

  always @(posedge clk)

  begin
    b <= a;
    c <= b;
    $display("Non_Blocking: a = %d, b = %d, c = %d.",a,b,c);
  end

endmodule
```

归纳总结

① 通过若干练习题进一步巩固了前三讲基础知识；
② 重点介绍了 Verilog 分频电路设计，函数/任务的使用
完成了上机实践（仿真与测试）。

第 5 次教学活动设计

教学环节	内容设计与手段
学时	**1 学时**
重难点	Verilog 基础知识 实训项目
导入新课	本讲只有 1 学时，为课程的总复习。通过本讲的学习，系统性梳理本课程的基本学习内容，给出部分练习题，加深对课程整体内容的理解，最后对实验报告出现的问题进行答疑和总结。
讲授内容	1、浏览 PPT，回顾课程内容，总结。 2、讲解实验报告中出现的问题，并上机演示容易出错的问题。 3、通过 PPT 浏览部分练习题，加深对 verilog 知识的理解和掌握。 4、简述考核及成绩评定方式，结课。
归纳总结	总结课程知识点 总结实验报告 总结成绩评定 结课

项目名称：3 位二进制数的数码管显示电路设计

一、任务要求：实现 3 位二进制数的数码管显示电路设计

输入			输出数码管显示
a	b	c	
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

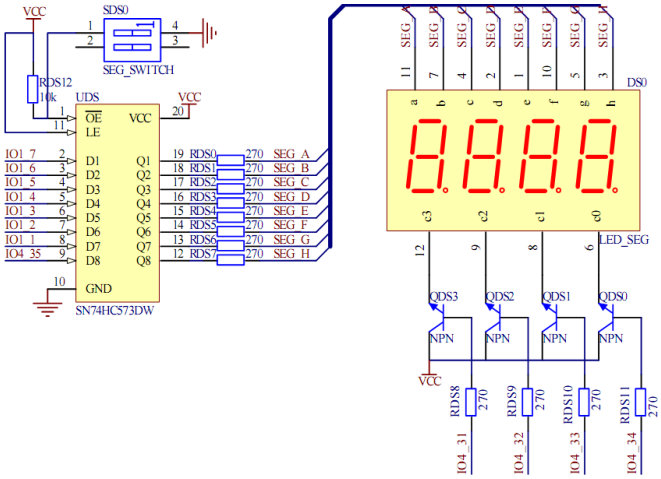
二、实验环境

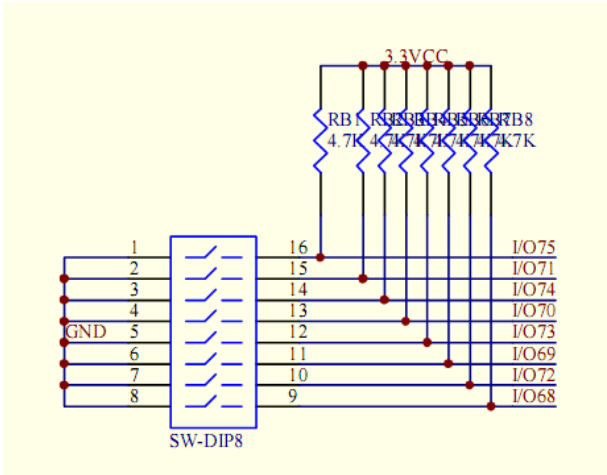
Cyclone II FPGA 开发板

Quartus II

PC 机

三、电路原理图：





3-8 译码器控制 LED 原理图略

四、任务实施步骤：

（一）子任务 1：数码管显示

1、新建工程，输入以下实验参考代码；

```
    module SEG_2 (clk, en, seg       );
  input clk;
  output [3:0] en;
  output [7:0] seg;
reg[7:0] seg_n;
 reg [24:0] count1;
```

```
 reg [3:0] count;
 wire [7:0]seg;
  assign en = 4'b1111;
  always @( posedge clk )
  begin
       count1 <= count1 + 25'd1;
   end
  always @( posedge count1[24] )
  begin
       count <= count + 4'd1;
   end
  always @(count)
  begin
    case (count)
      4'b0000 :
        seg_n <= 8'b11111100;
      4'b0001 :
        seg_n <= 8'b01100000;
      4'b0010 :
        seg_n <= 8'b11011010;
      4'b0011 :
        seg_n  <= 8'b11110010;
      4'b0100 :
        seg_n  <= 8'b01100110;
      4'b0101 :
        seg_n  <= 8'b10110110;
      4'b0110 :
        seg_n  <= 8'b10111110;
      4'b0111 :
        seg_n  <= 8'b11100000;
      4'b1000 :
        seg_n  <= 8'b11111110;
      4'b1001 :
        seg_n  <= 8'b11110110;
      4'b1010 :
        seg_n  <= 8'b11101110;
      4'b1011 :
        seg_n  <= 8'b00111110;
      4'b1100 :
        seg_n  <= 8'b10011100;
      4'b1101 :
        seg_n  <= 8'b01111010;
      4'b1110 :
        seg_n  <= 8'b10011110;
      default  :
        seg_n <= 8'b10001110;
    endcase
  end
assign seg=~seg_n;
endmodule
```

2、绑定引脚并编译工程，下载完成后，SW1 编码开关 1 置于 ON 状态，程序自动运行。

（二）子任务 2：3-8 译码器设计

1、调试 3-8 译码器的代码（见课本 P135）。

2、绑定引脚并编译工程，下载完成后，程序自动运行。

（三）系统总设计

1、修改子任务 1、2，完成本项目任务。

2、引脚绑定记录：
请截图说明引脚绑定详细情况。
3、综合编译报告：
完成以上步骤后，请编译工程，并把编译报告截图。
4、系统调试
程序下载，下载完成后，运行程序。
五、任务拓展
1、更改为 8-3 编码器的控制电路；
2、数码管动态显示控制电路。

参考代码：

```
module decoder_38(out,key_in);
output[7:0] out;        //3 8 译码器输出有 8 钟状态，所以要 8 个 LED 灯。
                            //如果没有 8 个 LED 灯也没有关系，只是有的状态就看不到了
input[2:0] key_in;     //(1 2 3)key1 key2 key3  作为数据输入
reg[7:0] out;
always @(key_in)
begin
case(key_in)
3'd0: out=8'b11111110;   //LED 作为状态显示,低电平有效
3'd1: out=8'b11111101;
3'd2: out=8'b11111011;
3'd3: out=8'b11110111;
3'd4: out=8'b11101111;
3'd5: out=8'b11011111;
3'd6: out=8'b10111111;
3'd7: out=8'b01111111;

endcase
end
endmodule
```

项目名称：LED 发光

一、任务要求：实现四个 LED 发光二极管循环自动点亮

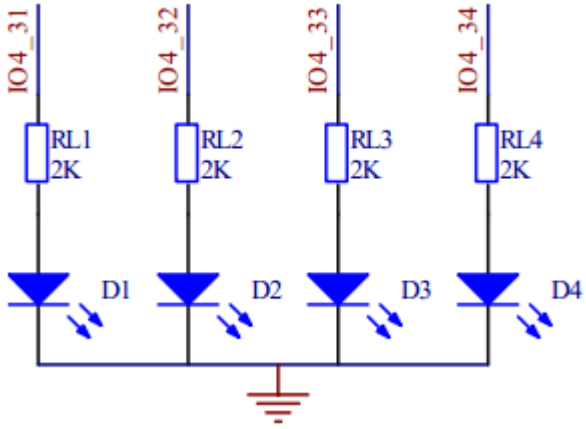
二、实验环境

Cyclone II FPGA 开发板

Quartus II

PC 机

三、电路原理图：



四、任务实施步骤：

1、新建工程，输入以下实验参考代码；

```
module led_demo(rst,clk,led);
input            clk,rst;
output [3:0] led;
reg       [3:0] led;

wire          clk_slow;
reg[23:0] clk_div;
reg           dir;

//div clk
always @(posedge clk or negedge rst)
begin
if(!rst)
   clk_div<=0;
else
   clk_div<=clk_div+1;
end

assign clk_slow= clk_div[23];

//show led
always @(posedge clk_slow or negedge rst)
```

```
begin
  if(!rst)
      begin
      led <= 4'b0001;
      dir <= 1'b0;
      end
  else
      begin
      if((led == 4'b1000) & (!dir))
            dir <= 1'b1;
      else
          begin
        if((led == 4'b0010) & (dir))
              dir <= 1'b0;
         led <= dir? (led >>1) : (led <<1);
        end
     end

end

endmodule
```

2、引脚绑定记录：

请截图说明引脚绑定详细情况。

3、功能仿真、时序仿真记录

请截图说明功能仿真和时序仿真。

4、综合编译报告：

完成以上步骤后，请编译工程，并把编译报告截图。

5、系统调试

程序下载，下载完成后，S1 为复位键，复位后程序自动运行。

五、任务拓展

1、更改 LED 循环闪烁的时间间隔；

2、更改 LED 循环闪烁的方向。

项目名称：分计数系统

一、任务要求：分计数系统。

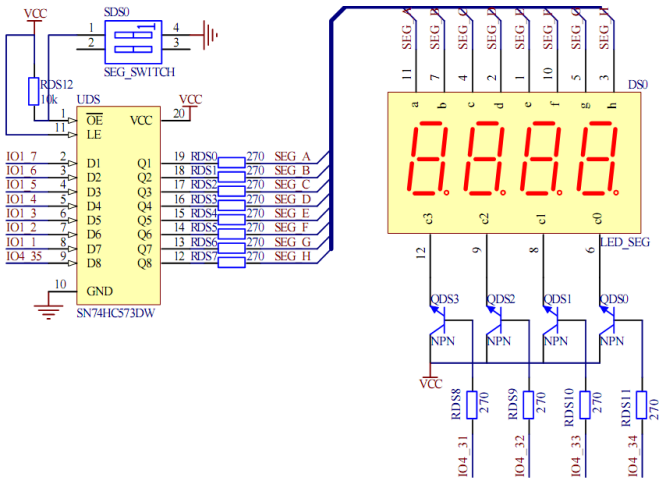
二、实验环境

Cyclone II FPGA 开发板

Quartus II

PC 机

三、电路原理图：



四、任务实施步骤：

（一）子任务 1：分频 1s

1、新建工程，输入以下实验参考代码；

参考代码：

```
module ledwater (clk_50M,led_out);
input     clk_50M;          //系统时钟输入 50M
output    led_out;

reg [24:0] count;   //分频计数器，25000000 分频
reg    div_clk;       //利用分频计数器得到显示一秒的闪烁效果
reg    led_out;

//分频计数器。得到一秒的频率
always @ ( posedge clk_50M )
begin
if ( count==25000000 )
  begin
    div_clk<=~div_clk;
      count<=0;
    end
else
    count<=count+1;
```

```
    led_out<=~div_clk;     //利用分频计数器得到显示一秒的闪烁效果
end
endmodule
```

2、led_out 输出控制 LED，绑定引脚并编译工程，下载完成后，程序自动运行，观看效果。

（二）子任务 2：计数系统，实现对按键按下次数的计数。

1、调试 4 位同步计数器代码。（P195）

2、自行编写数码管显示电路。

3、绑定按键，锁定引脚并编译工程，下载完成后，程序自动运行。

（三）系统总设计

1、调试上面两个子任务，实现对 1s 脉冲的计数。

2、引脚绑定记录：

请截图说明引脚绑定详细情况。

3、综合编译报告：

完成以上步骤后，请编译工程，并把编译报告截图。

4、系统调试

程序下载，下载完成后，S1为复位键，复位后程序自动运行。

五、任务拓展

基于矩阵键盘的加法器并用数码管显示

一．任务解析

用 Verilog 语言编写矩阵键盘驱动程序，实现“0~9，+，=”输入，数码管显示“加数”，按“+”键后再显示“被加数”，按“=”键后显示结果“和”。

二．实验方案

键盘的识别有两种编程方法：

方法一：

```
module keyboard(clk,clr,keyv,C,L,k1);//输入行线[3:0]L
input clk,clr;                  //输出扫描列线[3:0]C;
input [3:0]L;                    //输出键值[3:0]keyv。
output[3:0]C;
output k1;
output [3:0]keyv;
reg [1:0]scan;
reg [3:0]C;
reg k1;
always @(posedge clk or negedge clr)begin//clk 大概 1000HZ
  if(!clr) scan=0;
  else begin
        scan=scan+2'h1;
      case(scan)
                2'H0:C=4'b1110;//列
                2'H1:C=4'b1101;
                2'H2:C=4'b1011;
                2'H3:C=4'b0111;
        endcase
  end
end
reg [3:0]keyv;
always @(negedge clk or negedge clr)begin
  if(!clr) begin keyv=0;k1=0;end
  else begin
    case(L)
                4'HE:begin keyv={2'H0,scan};k1=1;end
                4'HD:begin keyv={2'H1,scan};k1=1;end
                4'HB:begin keyv={2'H2,scan};k1=1;end
```

```
                4'H7:begin keyv={2'H3,scan};k1=1;end
                default:k1=0;
     endcase
     end
end
endmodule
```

	8	4	0
	9	5	1
+		6	2
=		7	3

方法二：

```
module key_recognize(clk,reset,row,col,key_value,key_flag);
//50MHz clk,复位，行，列，键值
input clk,reset;
input [3:0]row;
output [3:0]col;
output [3:0]key_value;
output key_flag;
reg [3:0]col;
reg [3:0]key_value;
reg [5:0]count;//delay_20ms
reg [2:0]state;//状态标志
reg key_flag;//按键标志位
reg clk_500khz;//500KHZ 时钟信号
reg [3:0]col_reg;//寄存扫描列值
reg [3:0]row_reg;//寄存扫描行值
always @(posedge clk or negedge reset)
 if(!reset) begin clk_500khz<=0; count<=0; end
 else begin
    if(count>=50) begin clk_500khz<=~clk_500khz;count<=0;end
    else count<=count+1;
      end
always @(posedge clk_500khz or negedge reset)
    if(!reset) begin col<=4'b0000;state<=0;end
    else
       begin
       case (state)
       0:begin col[3:0]<=4'b0000;key_flag<=1'b0;
           if(row[3:0]!=4'b1111) begin state<=1;col[3:0]<=4'b1110;end
//有键按下，扫描第一行
           else state<=0;
```

```
          end
        1:begin
            if(row[3:0]!=4'b1111) begin state<=5;end//判断是否是第一行
            else begin state<=2;col[3:0]<=4'b1101;end//扫描第二行
          end
        2:begin
            if(row[3:0]!=4'b1111) begin state<=5;end//判断是否是第二行
            else begin state<=3;col[3:0]<=4'b1011;end//扫描第三行
          end
        3:begin
            if(row[3:0]!=4'b1111) begin state<=5;end//判断是否是第三一
    行
            else begin state<=4;col[3:0]<=4'b0111;end//扫描第四行
          end
        4:begin
            if(row[3:0]!=4'b1111) begin state<=5;end//判断是否扫描第一
    行
            else state<=0;
          end
        5:begin
            if(row[3:0]!=4'b1111) begin
                          col_reg<=col;row_reg<=row;//保存扫描列、行值
                          state<=5;key_flag<=1'b1;//有键按下
                               end
            else begin state<=0;end
           end
         endcase
        end
always @(clk_500khz or col_reg or row_reg)
 begin
 if(key_flag==1'b1)
    begin
    case({col_reg,row_reg})
    8'b1110_1110:key_value<=0;
    8'b1110_1101:key_value<=1;
    8'b1110_1011:key_value<=2;
    8'b1110_0111:key_value<=3;
    8'b1101_1110:key_value<=4;
    8'b1101_1101:key_value<=5;
```

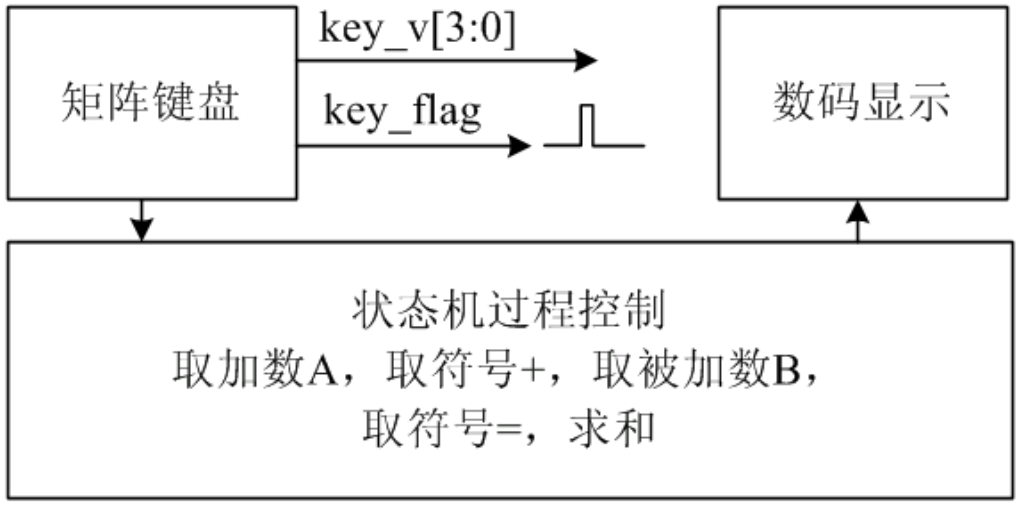
```
    8'b1101_1011:key_value<=6;
    8'b1101_0111:key_value<=7;
    8'b1011_1110:key_value<=8;
    8'b1011_1101:key_value<=9;
    8'b1011_1011:key_value<=10;
    8'b1011_0111:key_value<=11;
    8'b0111_1110:key_value<=12;
    8'b0111_1101:key_value<=13;
    8'b0111_1011:key_value<=14;
    8'b0111_0111:key_value<=15;

    endcase
    end
end
endmodule
```

3	2	1	0
7	6	5	4
=	+	9	8

三．重难点解析

原理框图：



从上图及以上两种按键识别程序可以看出，必须设置一个按键标志位以获取有效按键。

各模块功能分析：

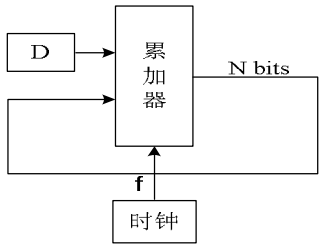
（1）自由分频电路：

```
    module div_clk(clki,f,clko);
    input clki;
    input [25:0]f;
```

```
    output clko;
    reg [25:0]cnt;
    reg clko;
    always @(posedge clki) begin
        if(cnt >= 26'D25_000_000) begin
           cnt=0;
           clko=!clko;
        end
        else cnt=cnt+f;
    end
    endmodule
```

在本实验中，各模块所需频率各不相同且有一定要求，故采用自由分频方便又易懂。

【分频原理】



时钟为f（例如 50M），累加器为 N bits，在时钟驱动下，累加器逐次和 D 不断累加，累加器溢出周期

$$T=\frac{2^N}{D\times f}$$

变换得到累加器溢出的频率

$$f'=D\times\frac{f}{2^N}$$

当$\frac{f}{2^N}=1$时，$f'=D$，实现整数倍分频。

当$\frac{f}{2^N}=0.1$时，$f'=D/10$，实现 0.1 倍分频。

……

（2）通用数码管驱动程序：

```
    module
seg8_disp(clk,d0,d1,d2,d3,d4,d5,d6,d7,dp,a,b,c,d,e,f,g,sel);
    input clk;
    input [3:0]d0,d1,d2,d3,d4,d5,d6,d7;
    output dp,a,b,c,d,e,f,g;
    output [2:0]sel;
    reg [2:0]scan;
    reg [3:0]in;
    reg [7:0]x;
```

```
always @(posedge clk)begin
  scan=scan+3'H1;
  case(scan)
  3'H0: in=d0;
  3'H1: in=d1;
  3'H2: in=d2;
  3'H3: in=d3;
  3'H4: in=d4;
  3'H5: in=d5;
  3'H6: in=d6;
  3'H7: in=d7;
  endcase
end
assign sel=scan;
assign  {dp,a,b,c,d,e,f,g} =
                (in==4'H0)? 8'b01111110 :
                (in==4'H1)? 8'b00110000 :
                (in==4'H2)? 8'b01101101 :
                (in==4'H3)? 8'b01111001 :
                (in==4'H4)? 8'b00110011 :
                (in==4'H5)? 8'b01011011 :
                (in==4'H6)? 8'b01011111 :
                (in==4'H7)? 8'b01110000 :
                (in==4'H8)? 8'b01111111 :
                (in==4'H9)? 8'b01111011 :
                (in==4'HA)? 8'b00110001 :
                (in==4'HB)? 8'b00001001 : 8'b10000000;
endmodule
```

（3）状态机控制的循环：

```
module state(clr,clk,key_flag,keyv,AA,BB,CC,DD,Sum);
input clr,clk,key_flag;
input [3:0]keyv;
output [3:0]AA,BB,CC,DD,Sum;
reg [2:0]s;
reg [3:0]AA,BB,CC,DD,Sum;
always @(posedge clk or negedge clr) begin
if(!clr)begin AA=4'HC;BB=4'HC;Sum=0;s=0;end
else begin
 case (s)
```

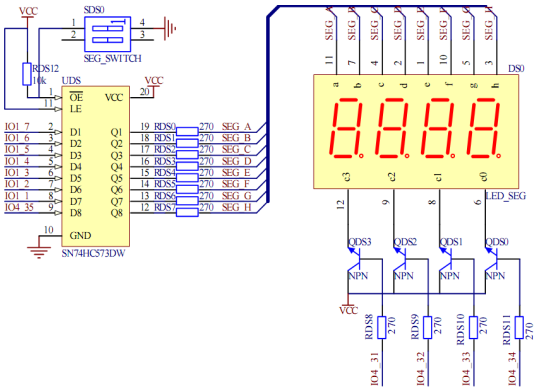
```
      3'H0 : begin AA=4'HC;BB=4'HC;CC=4'HC;DD=4'HC;s=1;end
      3'H1 : begin if(key_flag) begin BB=4'HC;CC=4'HC;DD=4'HC;
                          Sum=4'HC;AA=keyv;s=2;end end
      3'H2 : begin if((key_flag)&&(keyv==4'HA)) s=3; end
      3'H3 : begin if((key_flag)&&(keyv!=4'HA)) s=4;BB=4'HA ;end
      3'H4 : begin if(key_flag) begin CC=keyv;s=5;end end
      3'H5 : begin if((key_flag)&&(keyv==4'HB)) begin DD=4'HB;
                                  Sum=AA+CC;s=6; end end
      3'H6 : begin if((key_flag)&&(keyv!=4'HB)) s=1; end
      default : begin  s=1; end
    endcase
        end
    end
    endmodule
```

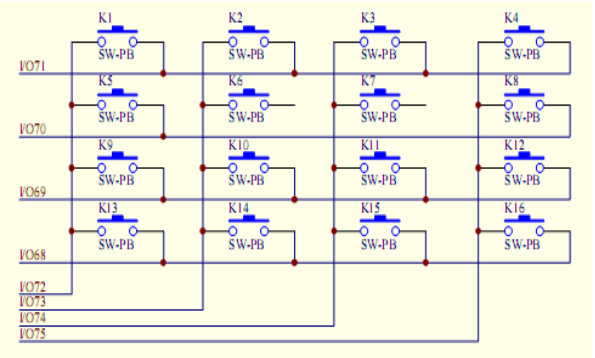
（4）顶层调用模块：

```
    module add_top(clk,clr,row,col,a,b,c,d,e,f,g,sel);
    input clk,clr;
    input [3:0]row;
    output [3:0]col;
    output [2:0]sel;
    output a,b,c,d,e,f,g;
    wire [2:0]s;
    reg [3:0]qq,A0,B0,C0,D0,Sum;
    reg [3:0]q;
    wire cko0,cko1;
    wire key_flag;
    wire [2:0]sel;
    div_clk clkA(.clki(clk),.f(26'D10000),.clko(cko0));
    key_recognize recognize(.clk(cko0),.reset(clr),.row(row),
                          .col(col),.key_value(q),.key_flag(key_flag
));
    //或者用 keyboard recognize(.clk(cko0),.clr(clr),
                              .L(row),.keyv(q),.C(col),.k1(key_flag)
);
    div_clk clkB(.clki(clk),.f(26'D2000),.clko(cko1));
    state  cycle(.clr(clr),.clk(cko1),.key_flag(key_flag),
              .keyv(q),.AA(A0),.BB(B0),.CC(C0),.DD(D0),.Sum(Sum));
    div_clk clkC(.clki(clk),.f(26'D300),.clko(cko2));
    seg8_disp
```

```
display(.clk(cko2),.sel(sel),.d0(Sum),.d1(D0),.d2(C0),.d3(B0),
        .d4(A0),.d5(12),.d6(12),.d7(12),.a(a),.b(b),.c(c),.d(d),.e(e
),.f(f),.g(g));
endmodule
```

四．硬件资源及引脚分配





五．实验结果

被加数 \ 加数	0	1	2	3	4	5	6	7	8	9
0										
1										———
2									———	———
3								———	———	———
4							———	———	———	———
5						———	———	———	———	———
6					———	———	———	———	———	———
7				———	———	———	———	———	———	———
8			———	———	———	———	———	———	———	———
9		———	———	———	———	———	———	———	———	———

某一结果如：3 -| 4=7 。

若两数之和超过 9 则数码管和位不显示任何数字。

六．经验总结

矩阵键盘的识别原理——扫描法。

如何设置键盘扫描频率，提高灵敏度。多少 Hz 为最合适？

对于控制循环的状态机，它的状态扫描速度必须快于按键扫描速度，还有每个状态要完成的任务。

简单状态机

一、任务要求：利用状态机在数码管上显示相应数字。

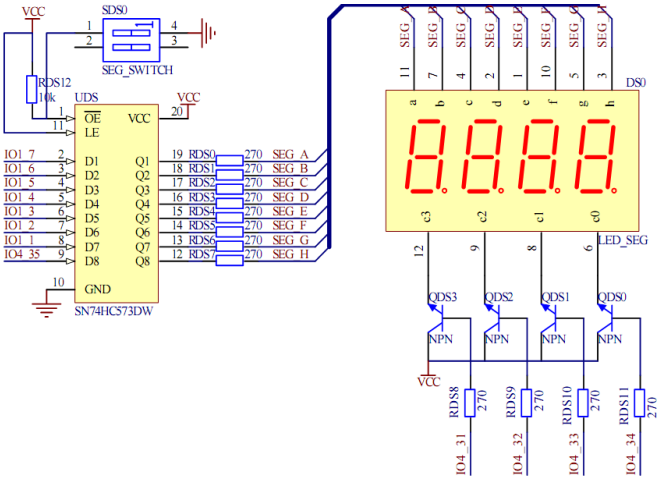
二、实验环境

Cyclone II FPGA 开发板

Quartus II

PC 机

三、电路原理图：



四、任务实施步骤：

1、新建工程，输入以下实验参考代码；

```
module state_machine(clk,rst,c,en);
input clk,rst;
output[7:0] c;
reg[7:0] c;
output[7:0] en;
parameter state0=3'b000,
          state1=3'b001,
          state2=3'b010,
          state3=3'b011,
          state4=3'b100,
          state5=3'b101,
          state6=3'b110,
          state7=3'b111;
reg[2:0] state;
reg[23:0] cnt;
assign en=0;
always@(posedge clk or negedge rst)
begin
    if(!rst) begin
        state<=state0;
        cnt<=0;
     end
    else begin
        cnt<=cnt+1;
        if(cnt==24'hffffff) begin
            case(state)
                state0:
                    state<=state1;
                state1:
                    state<=state2;
                state2:
                    state<=state3;
                state3:
                    state<=state4;
```

```
                state4:
                    state<=state5;
                state5:
                    state<=state6;
                state6:
                    state<=state7;
                state7:
                    state<=state0;
            endcase
        end
    end
end
always@(state)
begin
    case(state)
        state0:
            c=8'b1100_0000;
        state1:
            c=8'b1111_1001;
        state2:
            c=8'b1010_0100;
        state3:
            c=8'b1011_0000;
        state4:
            c=8'b1001_1001;
        state5:
            c=8'b1001_0010;
        state6:
            c=8'b1000_0010;
        state7:
            c=8'b1111_1000;
    endcase
end
endmodule
```

2、引脚绑定记录：

请截图说明引脚绑定详细情况。

3、综合编译报告：

完成以上步骤后，请编译工程，并把编译报告截图。

4、系统调试

程序下载，下载完成后，观察程序运行效果。

五、任务拓展

1、画出上面程序代码状态转移图。

2、下载并验证课本 P218-219 例子。

项目名称：矩阵按键与数码管显示

一、任务要求：判断矩阵按键值并显示在数码管上。

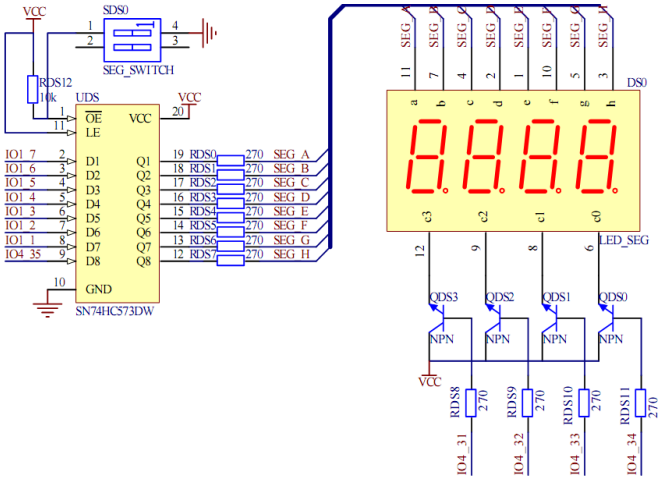
二、实验环境

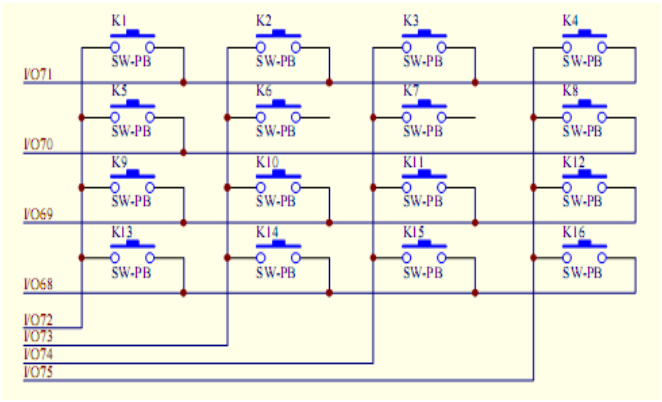
Cyclone II FPGA 开发板

Quartus II

PC 机

三、电路原理图：





四、任务实施步骤：

1、新建工程，输入以下实验参考代码；

2、引脚绑定记录：

请截图说明引脚绑定详细情况。

3、综合编译报告：

完成以上步骤后，请编译工程，并把编译报告截图。

4、系统调试

程序下载，下载完成后，S1为复位键，复位后程序自动运行。

五、任务拓展

参考代码：

```
module key1(clk,rst,row,column,dataout,en) ;
input clk,rst;
input[3:0] column;//列线
output[3:0] row;//行线
output[7:0] dataout;//数码管显示数据
reg[7:0] dataout;
output[7:0] en;//数码管显示使能
reg[3:0] row;

reg[3:0] scan_key; //扫描码寄存器
reg[15:0] cnt_scan;//扫描频率计数器
```

```
assign en=0;

always@(posedge clk or negedge rst)
begin
    if(!rst) begin
        row<=4'b1110;
        cnt_scan<=0;
     end
    else begin
        cnt_scan<=cnt_scan+1;
        if(cnt_scan==16'hffff) begin
            row[3:1]<=row[2:0];
            row[0]<=row[3];   //4 根行线循环送出低电平
         end
     end
end

always@(posedge clk or negedge rst)
begin
    if(!rst) begin
        scan_key<=0;
     end
    else begin
        case(row)   //该 case 结果检测何处有键按下
            4'b1110:
                case(column)
                    4'b1110: begin
                        scan_key<=0;
                     end
                    4'b1101: begin
                        scan_key<=1;
                     end
                    4'b1011: begin
                        scan_key<=2;
                     end
                    4'b0111: begin
                        scan_key<=3;
                     end
                 endcase
            4'b1101:
                case(column)
                    4'b1110: begin
                        scan_key<=4;
                     end
```

```
                    4'b1101: begin
                        scan_key<=5;
                     end
                    4'b1011: begin
                        scan_key<=6;
                     end
                    4'b0111: begin
                        scan_key<=7;
                     end
                endcase
            4'b1011:
                case(column)
                    4'b1110: begin
                        scan_key<=8;
                     end
                    4'b1101: begin
                        scan_key<=9;
                     end
                    4'b1011: begin
                        scan_key<=10;
                     end
                    4'b0111: begin
                        scan_key<=11;
                     end
                endcase
            4'b0111:
                case(column)
                    4'b1110: begin
                        scan_key<=12;
                     end
                    4'b1101: begin
                        scan_key<=13;
                     end
                    4'b1011: begin
                        scan_key<=14;
                     end
                    4'b0111: begin
                        scan_key<=15;
                     end
                endcase
             default:
                scan_key<=15;
        endcase
    end
```

```
end



always@(scan_key)
begin
    case(scan_key)
        4'b0000:
            dataout<=8'b11000000;
        4'b0001:
            dataout<=8'b11111001;
        4'b0010:
            dataout<=8'b10100100;
        4'b0011:
            dataout<=8'b10110000;
        4'b0100:
            dataout<=8'b10011001;
        4'b0101:
            dataout<=8'b10010010;
        4'b0110:
            dataout<=8'b10000010;
        4'b0111:
            dataout<=8'b11111000;
        4'b1000:
            dataout<=8'b10000000;
        4'b1001:
            dataout<=8'b10010000;
        4'b1010:
            dataout<=8'b10001000;
        4'b1011:
            dataout<=8'b10000011;
        4'b1100:
            dataout<=8'b11000110;
        4'b1101:
            dataout<=8'b10100001;
        4'b1110:
            dataout<=8'b10000110;
        4'b1111:
            dataout<=8'b10001110;
     endcase
end

endmodule
```