MES2KG 开发板

实验指导

版本

修改日期	修改版本	修改原因
2022/08/12	v1.0	创建文档

Myminieye 微信公众号二维码如下图，欢迎扫码关注，我们将会不定期的发送一些 FPGA 相关的技术推文或者半导体行业时讯的推文：

MYMINIEYE

微信扫描二维码，关注我的公众号

目录

1. 控制 LED 灯 6

1.1 实验目的 6

1.2 实验要求 6

1.3 实验原理 6

1.4 实验源码设计（完整源码查看 DEMO 源文件） 7

1.4.1 文件头设计 7

1.4.2 设计 module 8

1.4.3 完整的 Module（不含注释） 10

1.4.4 硬件管脚分配 10

1.5 实验步骤 12

1.5.1 打开 PDS 软件，创建工程 12

1.5.2 添加设计文件 16

1.5.3 编译 23

1.5.4 工程约束 23

1.6 生成位流文件并下载 26

1.7 实验现象 26

2. LED 流水灯 27

2.1 实验目的 27

2.2 实验要求 27

2.3 实验原理 27

2.4 实验源码设计（完整源码查看 DEMO 源文件） 28

2.5 实验步骤 29

2.6 实验现象 29

3. 键控彩灯 30

3.1 实验目的 30

3.2 实验要求 30

3.3 实验原理 30

3.3.1 按键控制模块功能 31

3.3.2 按键消抖 31

3.3.3 LED 控制模块功能 33
3.4 实验源码设计（完整源码查看 DEMO 源文件） 34
3.4.1 顶层文件源码 34
3.4.2 按键控制模块 34
3.4.3 LED 控制模块 36
3.5 实验现象 37
4. 数码管动态显示 38
4.1 实验目的 38
4.2 实验要求 38
4.3 实验原理 38
4.4 实验源码（完整源码查看 DEMO 源文件） 41
4.4.1 顶层模块 41
4.4.2 按键消抖模块 45
4.4.3 按键计数模块 45
4.4.4 时钟分频模块 46
4.4.5 数码管显示模块 47
4.5 实验现象 47
5. 序列检测器 49
5.1 实验目的 49
5.2 实验要求 49
5.3 实验原理 49
5.4 实验源码（完整源码查看 DEMO 源文件） 49
5.4.1 方案设计 49
5.4.2 顶层模块（含数码管显示模块）设计 50
5.4.3 按键 LED 控制模块 53
5.4.4 序列检测模块设计 56
5.5 实验现象 58
6. 密码锁 59
6.1 实验目的 59
6.2 实验要求 59

6.3 实验原理 59
6.4 实验源码（完整源码查看 DEMO 源文件） 59
6.4.1 顶层模块设计 60
6.4.2 按键控制设计 62
6.4.3 按键消抖设计 64
6.4.4 对比模块设计 64
6.4.5 显示模块设计 65
6.5 实验现象 65
7. 数字钟 67
7.1 实验目的 67
7.2 实验要求 67
7.3 实验原理 67
7.4 实验源码（完整源码查看 DEMO 源文件） 68
7.4.1 顶层设计 68
7.4.2 数字时钟产生与控制模块设计 72
7.4.3 时钟分频模块 81
7.4.4 数码管显示模块设计 82
7.5 实验现象 84

1.控制 LED 灯

1.1实验目的

实现对多 LED 灯的控制;

1.2实验要求

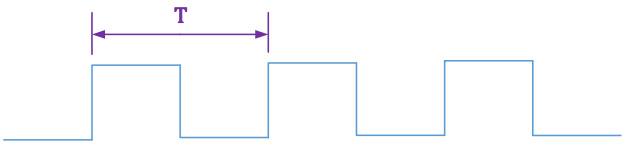
控制 8 个 LED 以 1s 的周期闪烁（0.5s 亮，0.5s 灭）

1.3实验原理

通常的时，分，秒的计时进位大家应该不陌生;

1 小时=60 分钟=3600 秒，当时针转动 1 小时，秒针跳动 3600 次;

那在数字电路中的时钟信号也是有固定的节奏的,这种节奏的开始到结束的时间,我们通常称之为周期（T）。



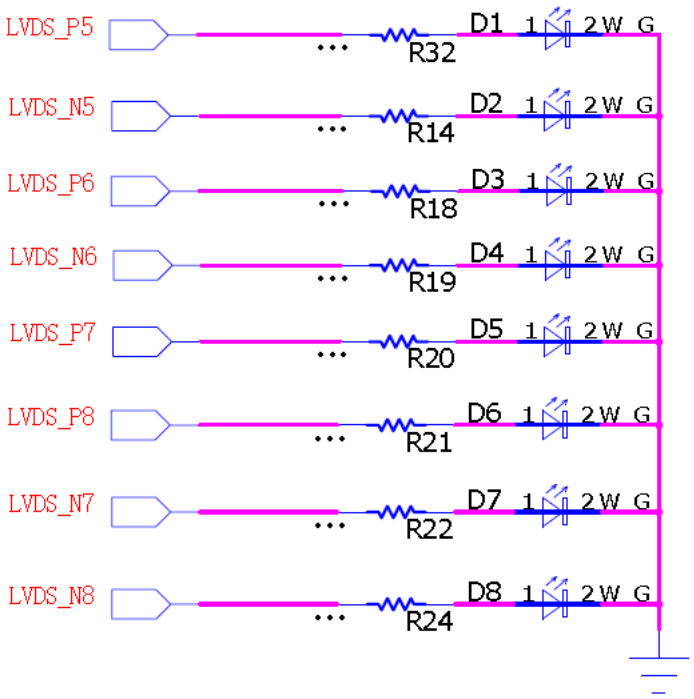
在数字系统中通常关注到时钟的频率,那频率与周期的关系如下：

$$f = \frac{1}{T};$$

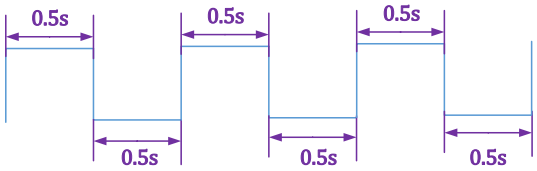
MES2KG 板卡上有一个 40MHz 的晶振提供时钟给到 PGC2KG;

实验分析:

控制 LED 亮灭需要控制 IO 输出的高低电平即可(高电平点亮,低电平熄灭),原理图如下：



控制 LED 周期性的维持 0.5s 亮,0.5s 灭,需要控制 IO 输出 0.5s 高电平,0.5s 低电平周期变化,如下图波形:



外部输入时钟为 40MHz 时钟周期为 25ns（在 verilog 设计中的计数器的计时原理基本上是一致的，确认输入时钟周期，目标计时时间后可得到计数器的计数值到达多少后可得到计时宽度）；

$$0.5s = 20000000*25ns = 20000000 \times T_{40MHz};$$

IO 输出状态只有两种：1 或 0；我们可以使用一个计数器，计数满 20000000 个时钟周期时将 IO 状态进行翻转，即可完成每 0.5S 输出状态跳转，即 LED 灯会以 0.5S 的间隔亮灭变化；

1.4实验源码设计（完整源码查看 demo 源文件）

1.4.1文件头设计

在 module 之前添加文件头，文件头中包含信息有：公司，作者，时间，设计名，工程名，模块名，目标器件，EDA 工具(版本)，模块描述，版本描述（修

改描述）等信息；以及仿真时间单位定义；

```
//////////////////////////////////////////////////////////////////////////////////
// Company:Meyesemi
// Engineer: Will
//
// Create Date: 2022-08-12 20:31
// Design Name:
// Module Name:
// Project Name:
// Target Devices: Pango
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 1.0 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
```

`timescale 1ns / 1ps 表示仿真精度是 1ns，显示精度是 1ps；

`define UD #1 定义 UD 表示#1；#1 仅仿真有效，表示延时一个仿真精度，结合上一条语句表示延时 1ns；

1.4.2设计 module

1.4.2.1 创建 module，确定输入输出信号

```
1  module led_light(
2   input         clk,     // input clock, the frequency is 40MHz
3   input         rstn,    // input reset signal, active at low
4   output [7:0] led      // output LED control signal , lignting at
   high
5  );
6  endmodule
```

此段代码是标准的 module 创建的模型，module 创建时需要确认输入输出信号并定义好位宽，之后在对 module 进行具体的逻辑设计；管脚与管脚之间间隔用“,”,最后一个管脚不用间隔符号；

创建 module 时需要定义输入输出信号;本实验输入时钟和复位即可，输出是

控制 LED 的亮灭，MES2KG 板卡上共有 8 个 LED，故而输出 8bit 位宽的信号；

1.4.2.2设计一个计数器；

单个状态计数 20000000，1 个亮灭周期的计数即为 40000000 = 26'h2625A00；所以计数器的位宽为 26 位即可，此处请结合数字电路中的同步计数器的工作原理分析；

```
reg [25:0] led_light_cnt;
//  time counter
always @(posedge clk) // 触发条件；posedge 为上升沿，negedge 为下降沿
begin
    if(!rstn)
        led_light_cnt <= `UD 26'd0;
    else if(led_light_cnt == 26'd19_999_999)
          led_light_cnt <= `UD 26'd0;
      else
          led_light_cnt <= `UD led_light_cnt + 26'd1;
  end
```

当计数器计数到 26’d19999999 时，计数周期包含了从 0～26’d19999999 的时钟周期，故而总时长时 26‘d20000000 × T_{clk}；硬件输入时钟为 40MHz，所以此计数器的技术周期是 0.5s；

1.4.2.3led 显示状态控制

在指定的时间刻度上对 LED 的状态进行变更，以达到控制 LED 规律的亮灭的目的；

led_light_cnt 的计时周期为 0.5s，故在 led_light_cnt 上取一个点来变更 LED 的显示状态即可完成每隔 0.5s LED 显示发生变化；由于 LED 亮和灭只有两个状态，在赋值处理上将寄存器取反即可得到对应的从亮到灭变化（或从灭到亮的变化）；

```
reg [7:0] led_status;

always @(posedge clk)
begin
    if(!rstn)
        led_status <= `UD 8'd0;
    else if(led_light_cnt == 26'd19_999_999)
        led_status <= `UD ~led_status;
end
```

```
10  assign led = led_status;
```

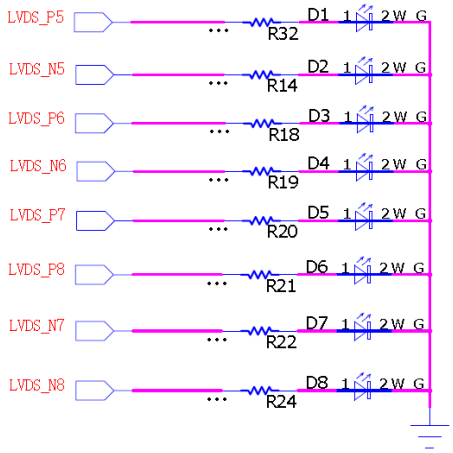
1.4.3完整的 Module（不含注释）

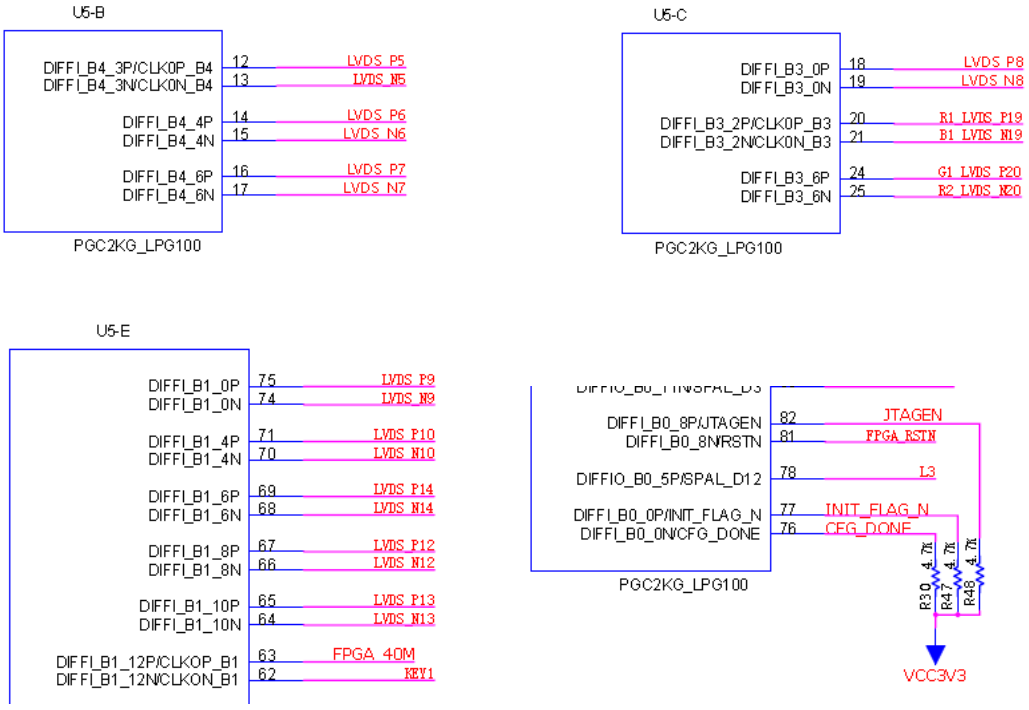
```
1   `timescale 1ns / 1ps
2   `define UD #1
3   module led_light(
4       input            clk,
5       input            rstn,
6       output [7:0]     led
7   );
8   //=================================================
9       //reg and wire
10      reg [25:0] led_light_cnt;
11      reg [ 7:0] led_status;
12      //  time counter
13      always @(posedge clk)
14      begin
15          if(!rstn)
16              led_light_cnt <= `UD 26'd0;
17          else if(led_light_cnt == 26'd19_999_999)
18              led_light_cnt <= `UD 26'd0;
19          else
20              led_light_cnt <= `UD led_light_cnt + 26'd1;
21      end
22
23      // led status change
24      always @(posedge clk)
25      begin
26          if(!rstn)
27              led_status <= `UD 8'd0;
28          else if(led_light_cnt == 25'd19_999_999)
29              led_status <= `UD ~led_status;
30      end
31      assign led = led_status;
32
33  endmodule
```

1.4.4硬件管脚分配

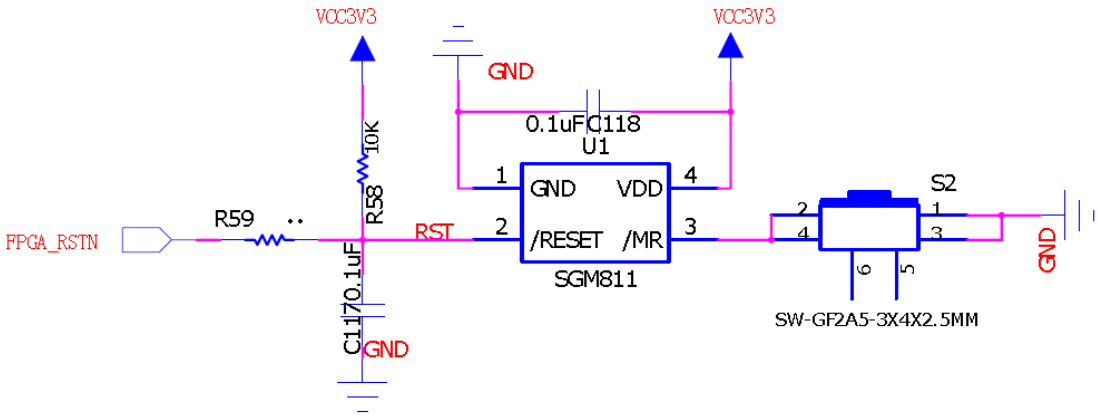
MES2KG 的 LED 和 CLK 与 FPGA 的 IO 连接部分的原理图如下（在工程中做物

理约束时需要结合原理图的 FPGA 管脚分配进行约束）：





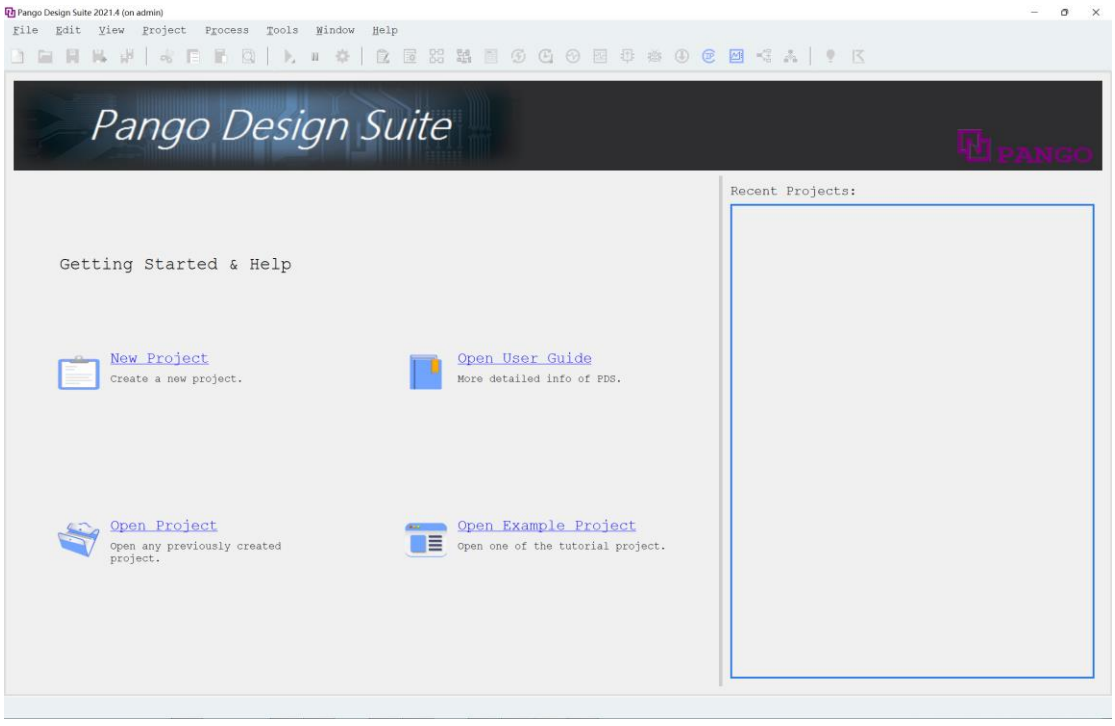
复位设计是低电平有效，而 MES2KG 板卡上的按键按下时为低电平，松开为高电平，可用按键输入来做复位信号；使用 RSET 做为复位按键即可；



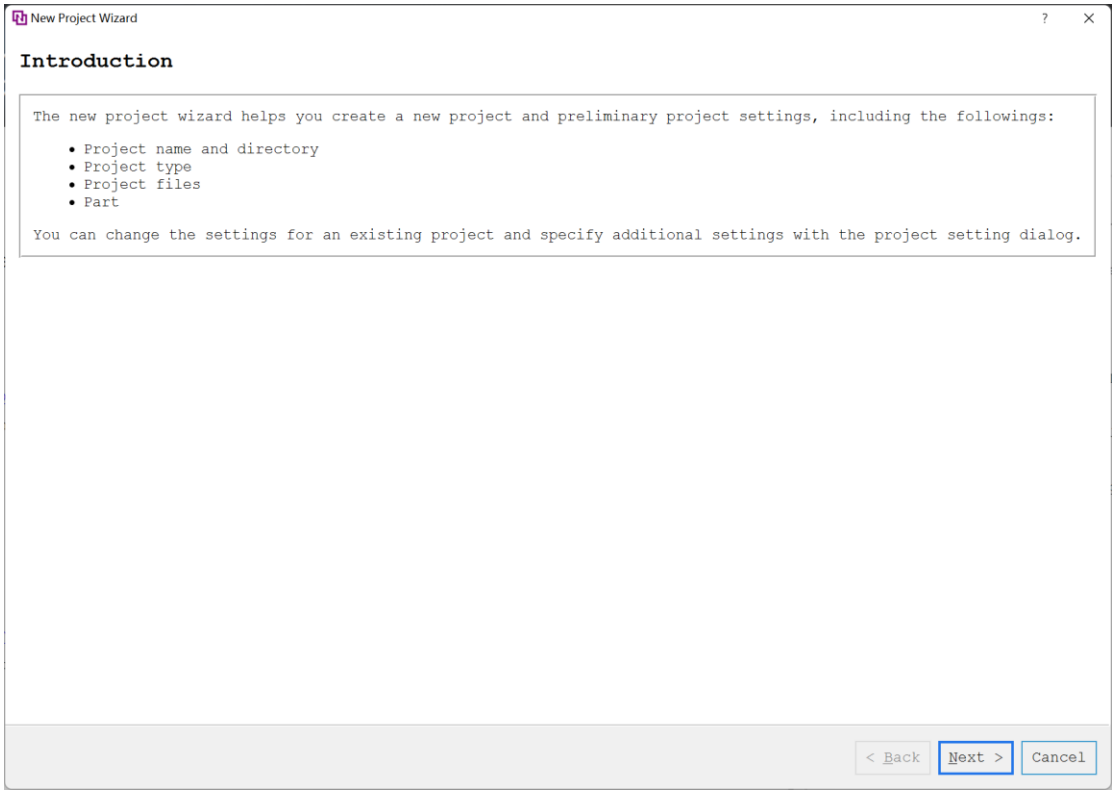
1.5实验步骤

1.5.1打开 PDS 软件，创建工程

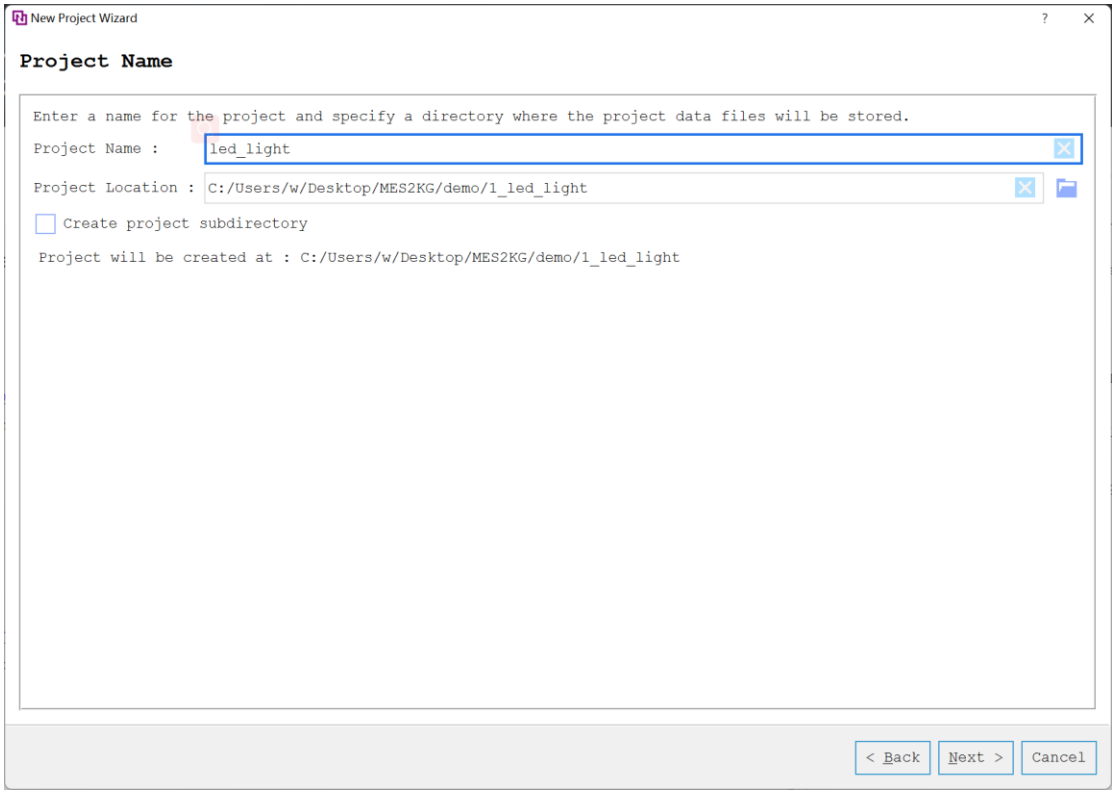
Step1：打开 PDS 软件，单击 New Project



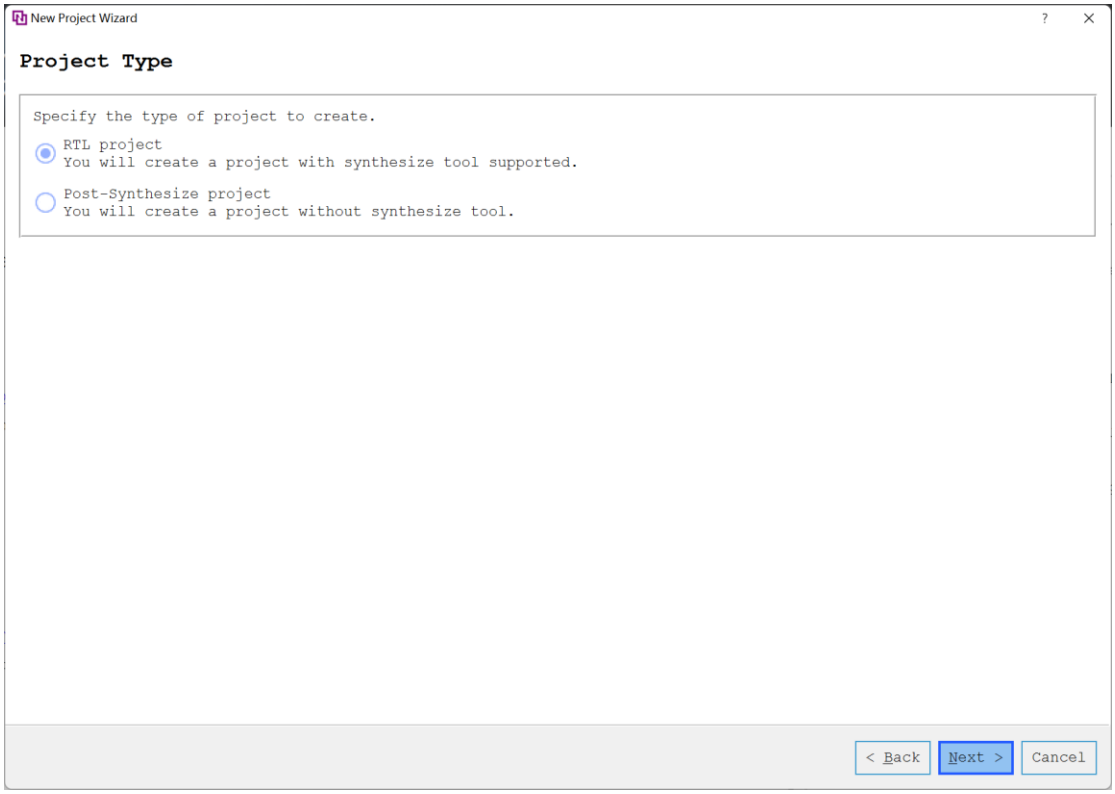
Step2：单击 NEXT



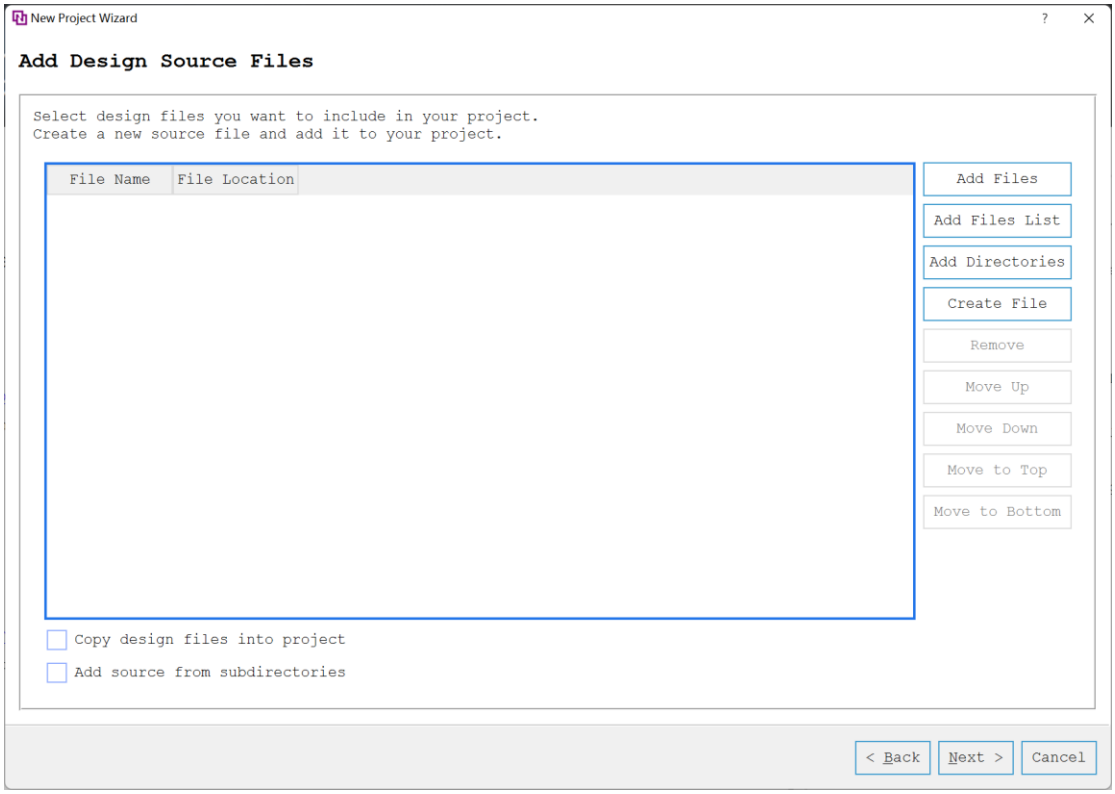
Step3：创建名为 led_light 的工程到对应的文件目录，之后单击 Next



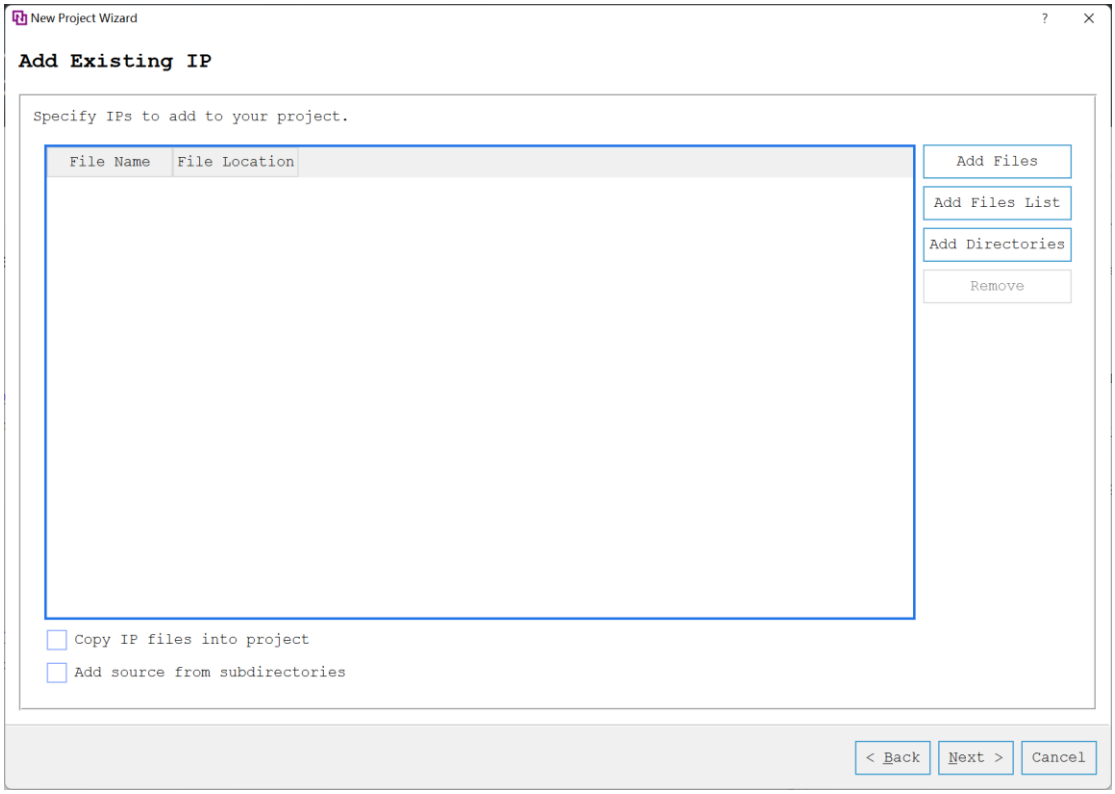
Step4：选择 RTL project，单击 Next



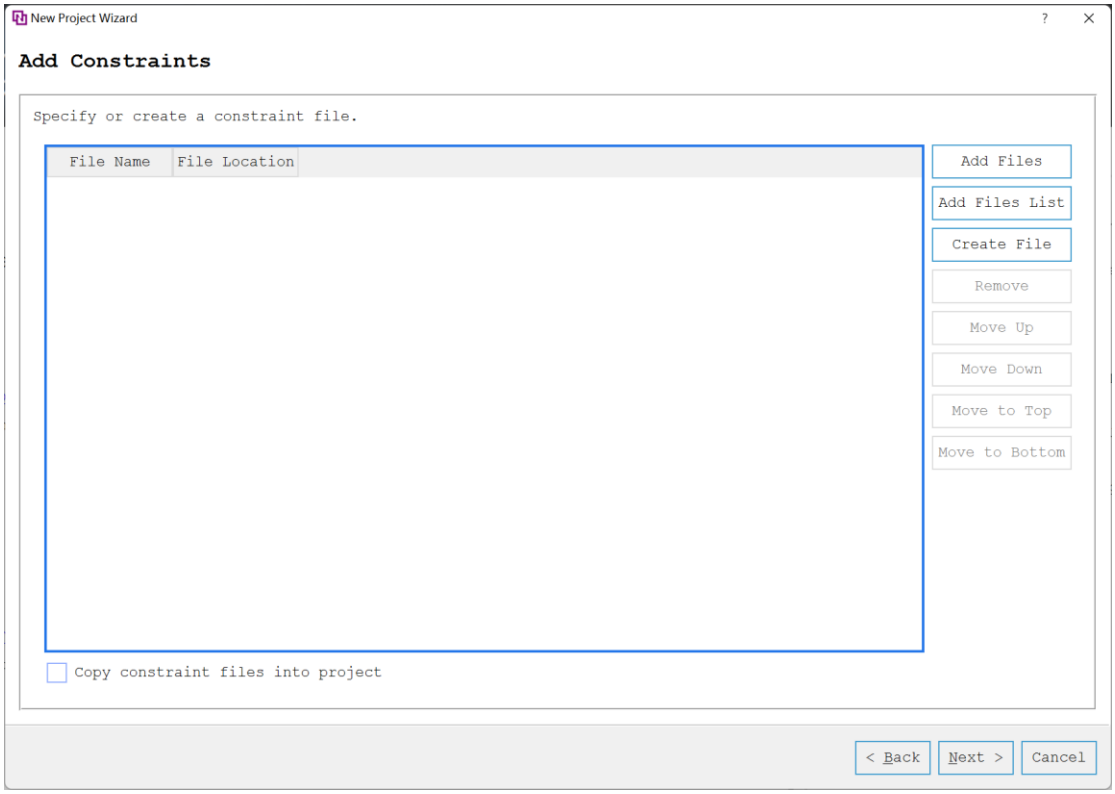
Step5：单击 Next(也可添加.v 文件)



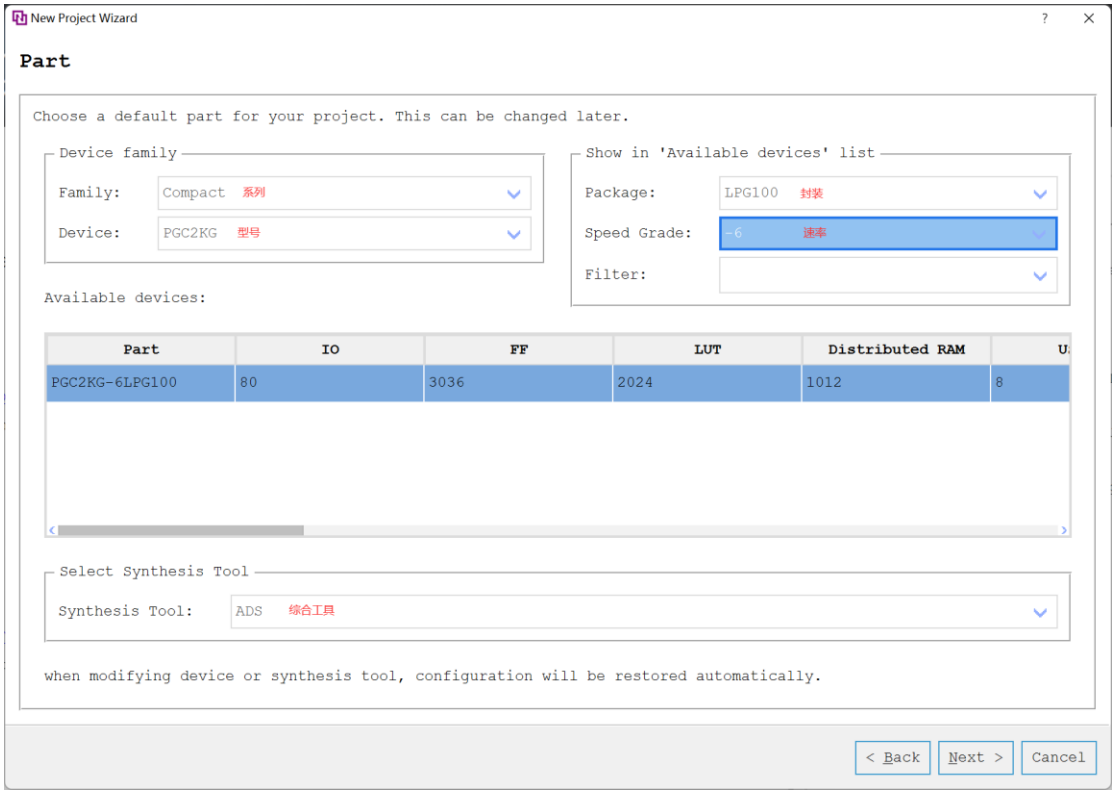
Step6：单击 Next



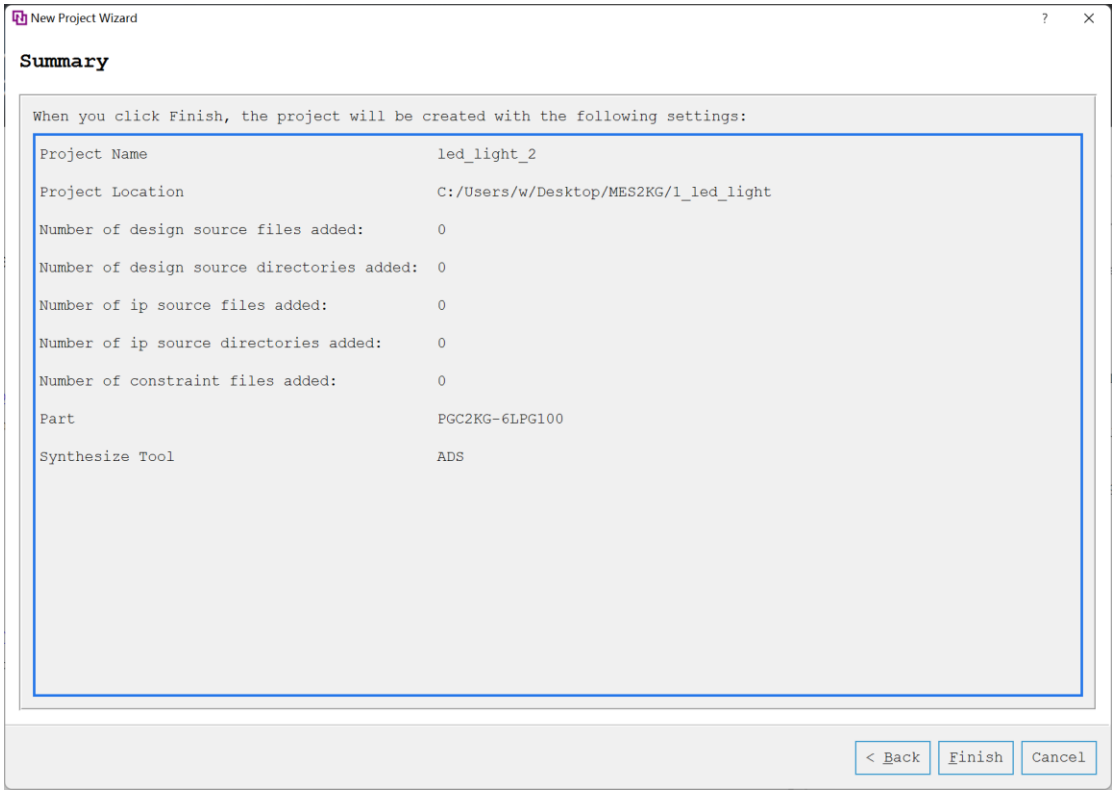
Step7：单击 Next



Step8：选择器件系列、型号、封装、速率、综合工具，之后单击 Next

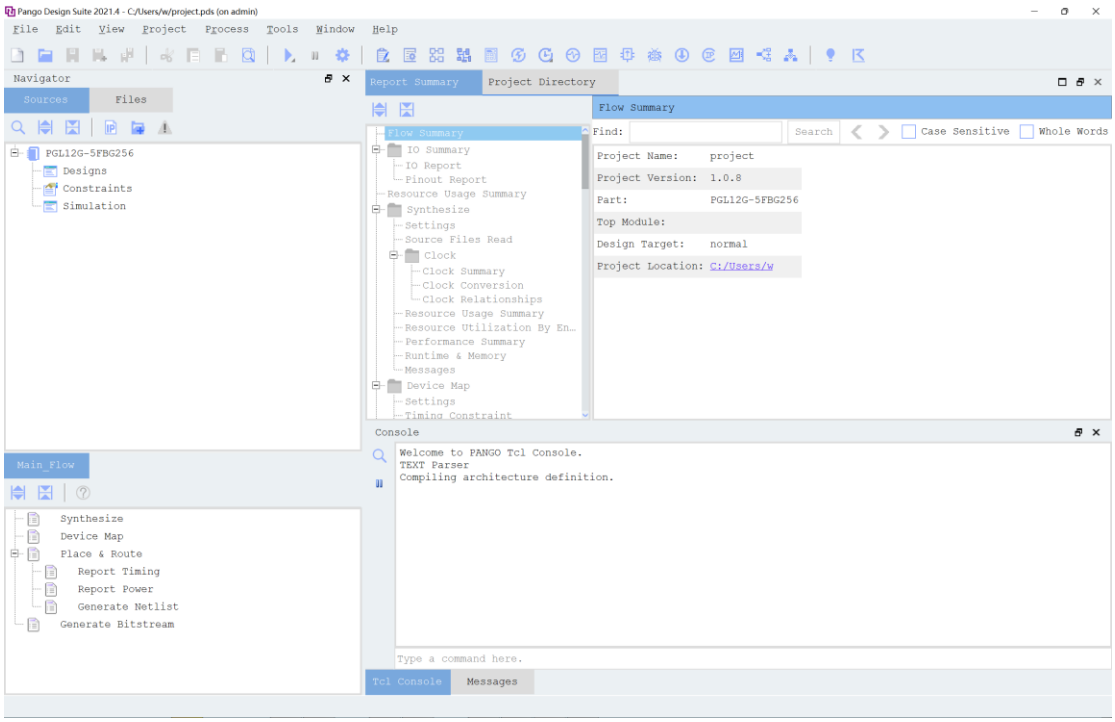


Step9：单击 Finish,完成工程创建

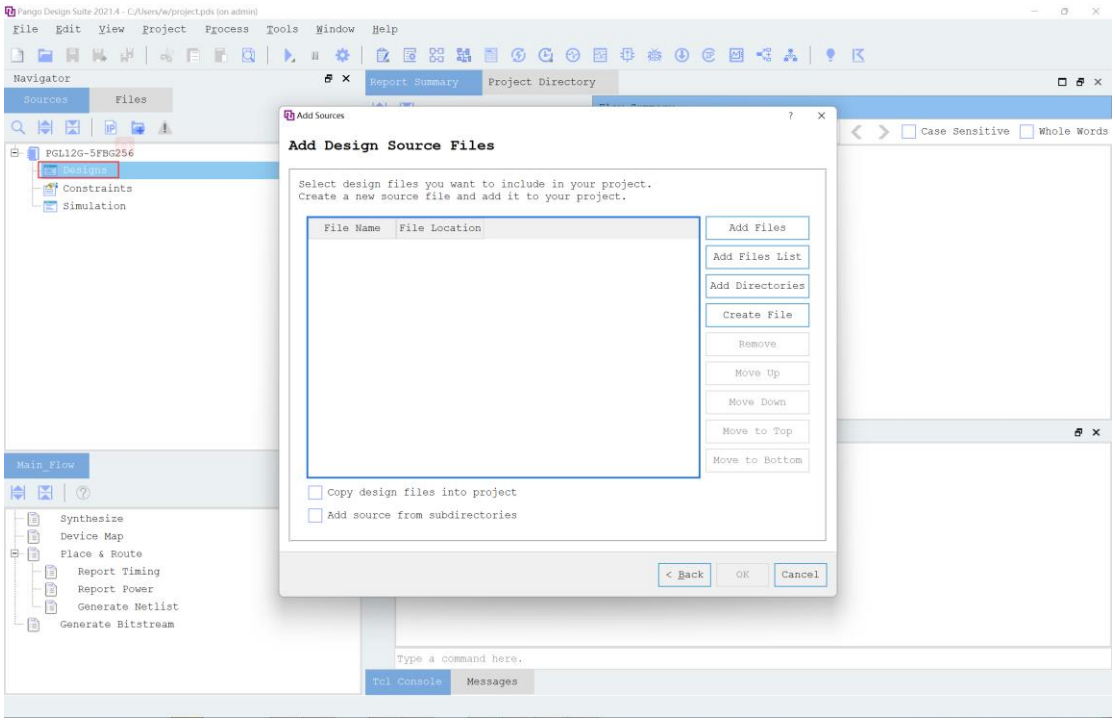


1.5.2添加设计文件

PDS 软件界面如下图：

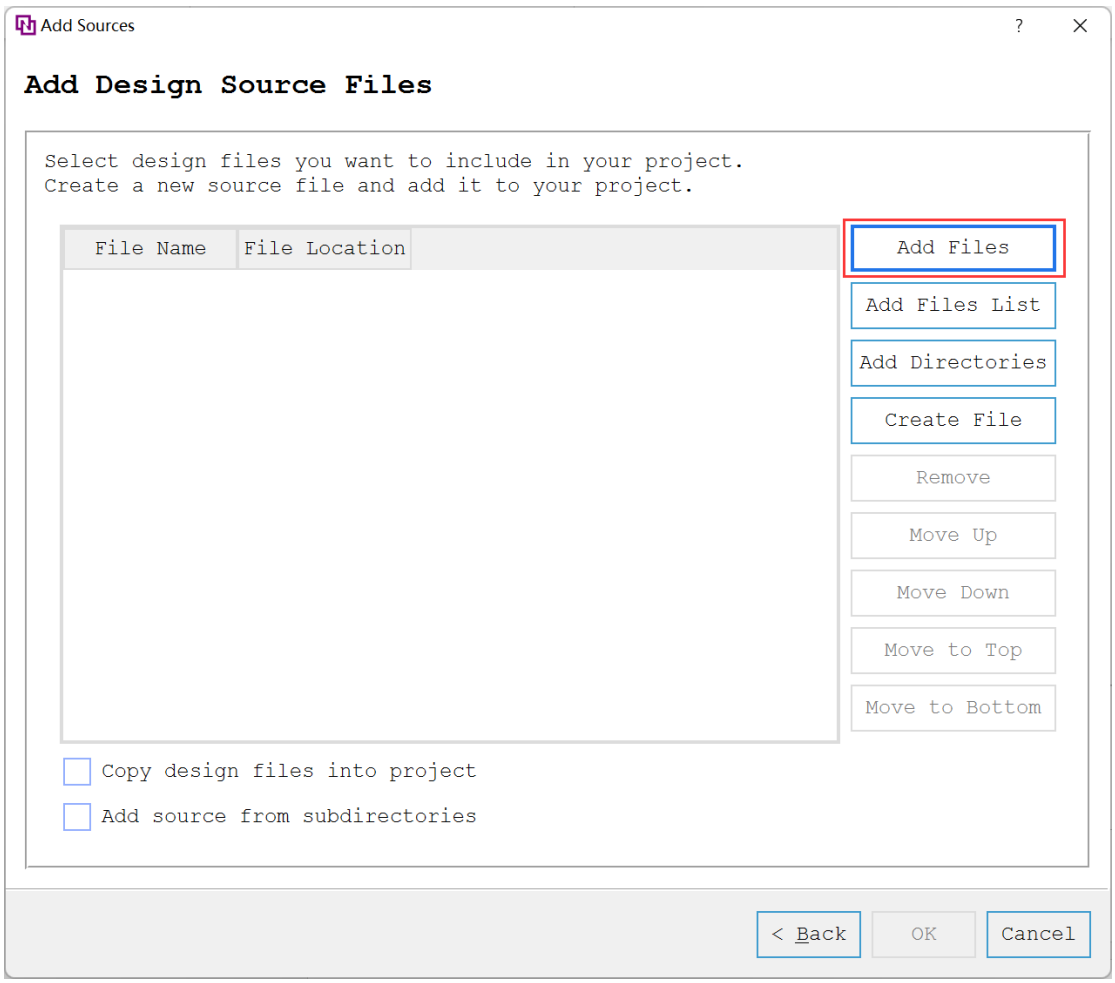


双击 Designs，将前面设计的 module 新建到文件中，或者将前面编辑好的 verilog 文件添加到工程中：



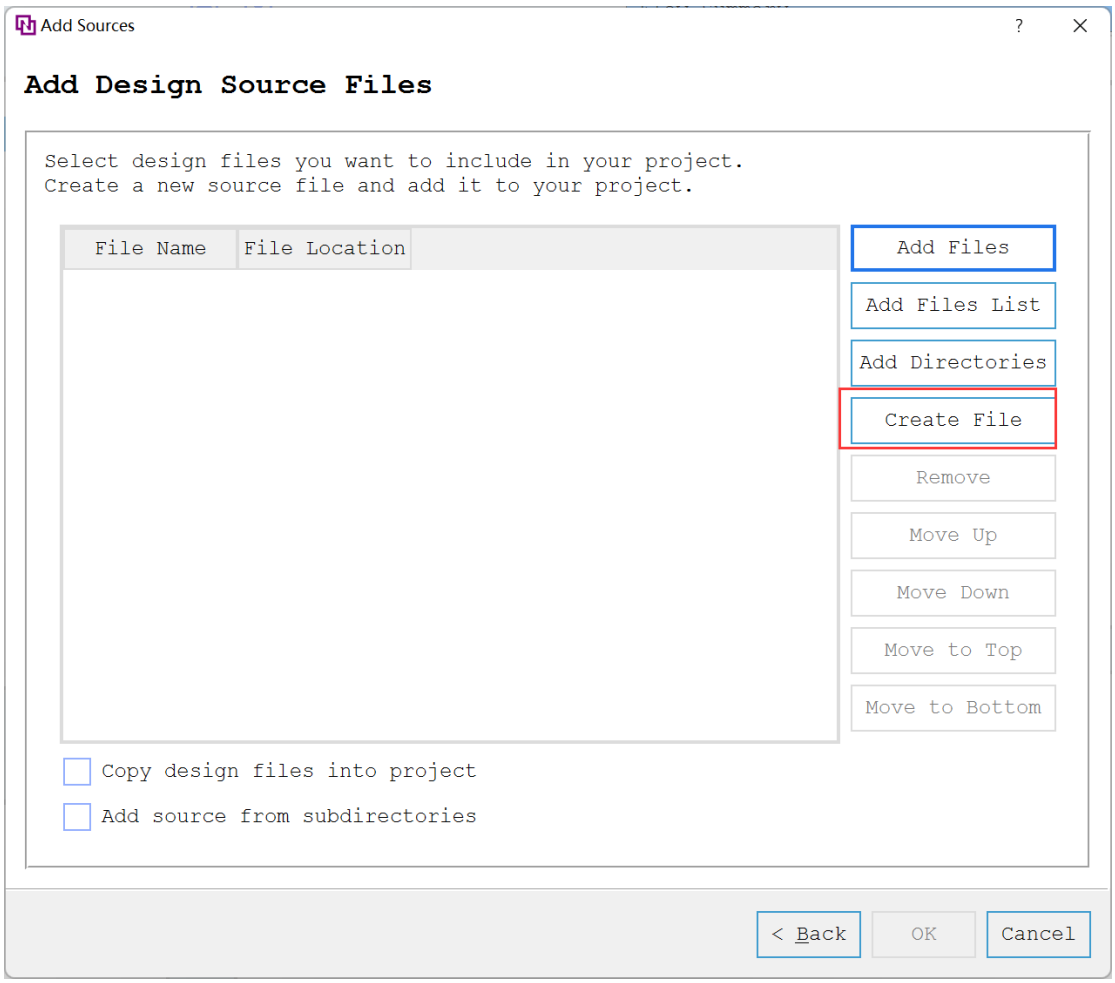
添加文件到工程：

在窗口中点击 Add Files，选择添加文件到工程；

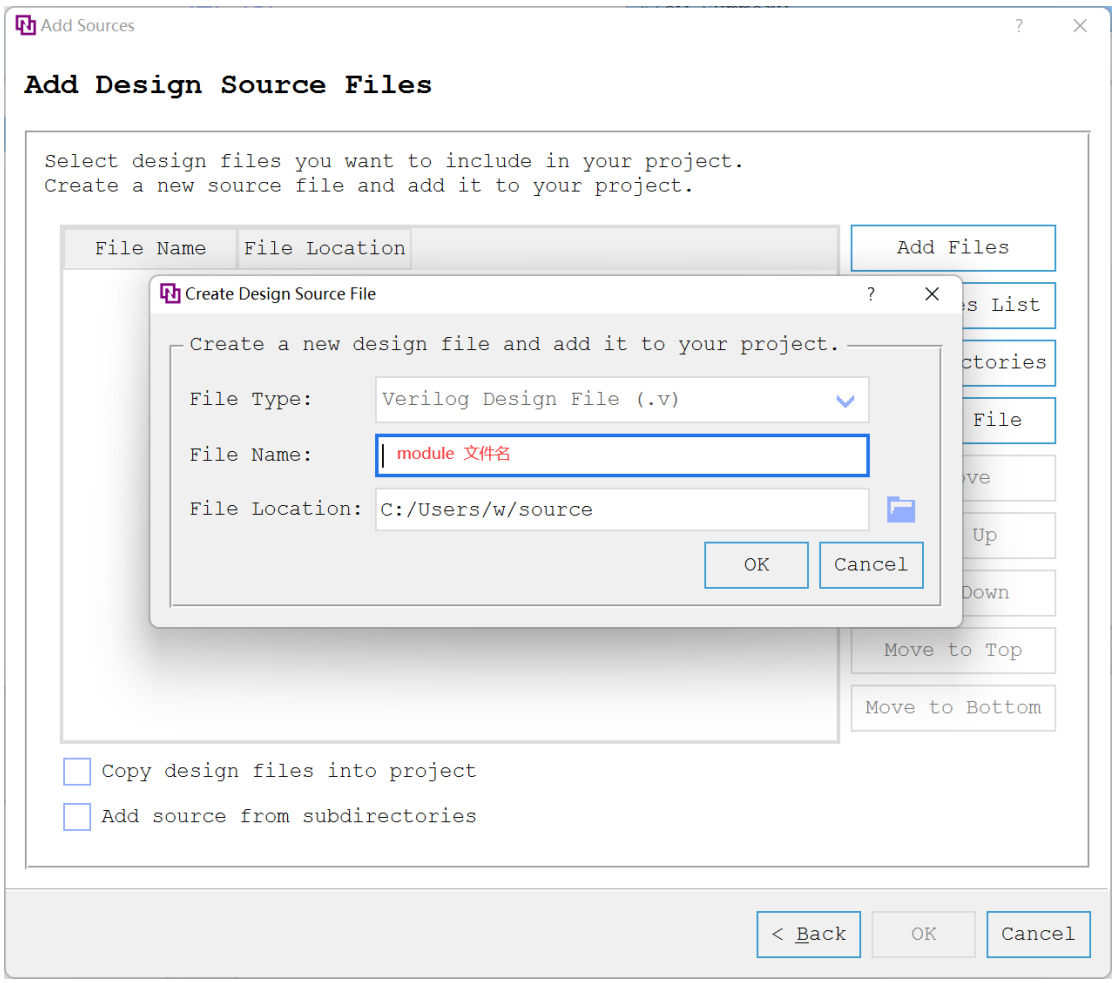


新建文件到工程：

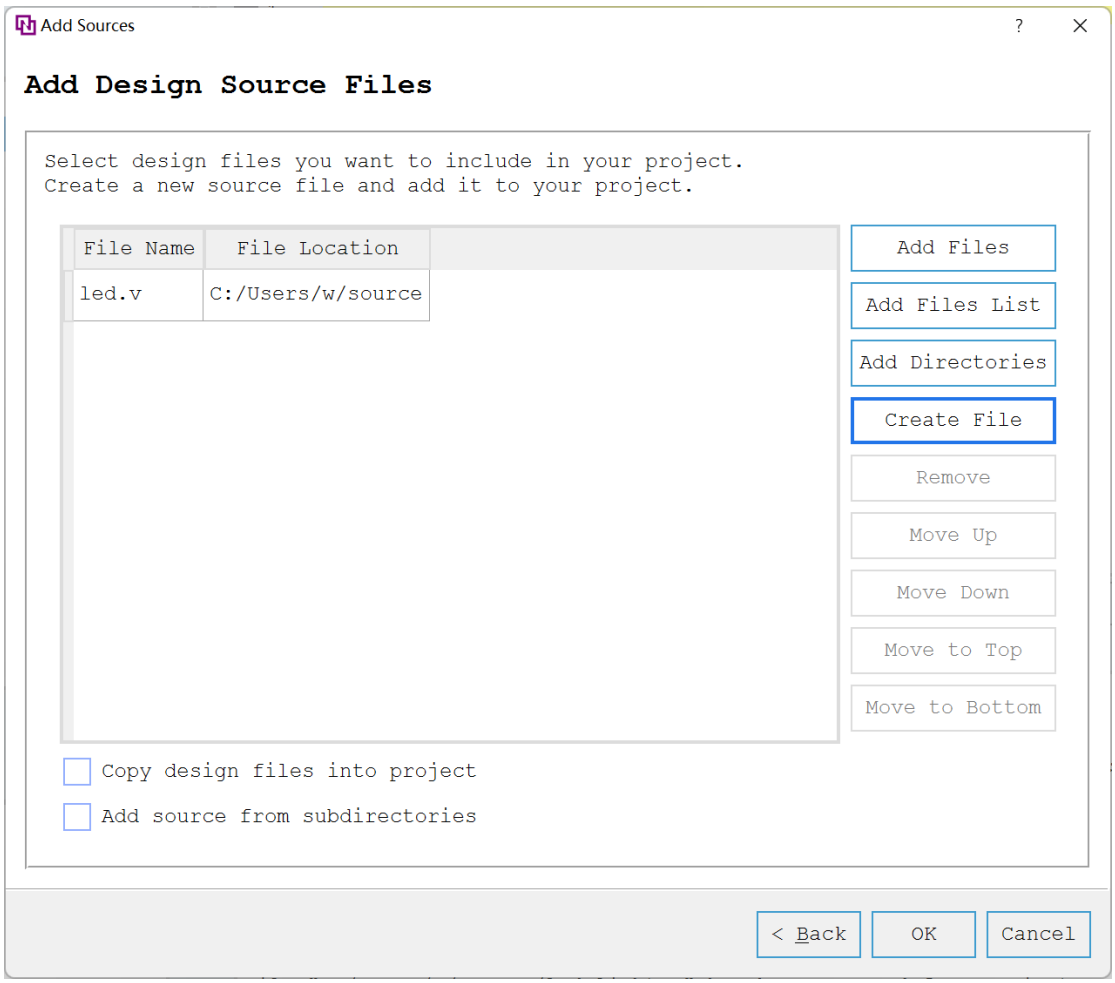
1）在窗口中点击 Create File；



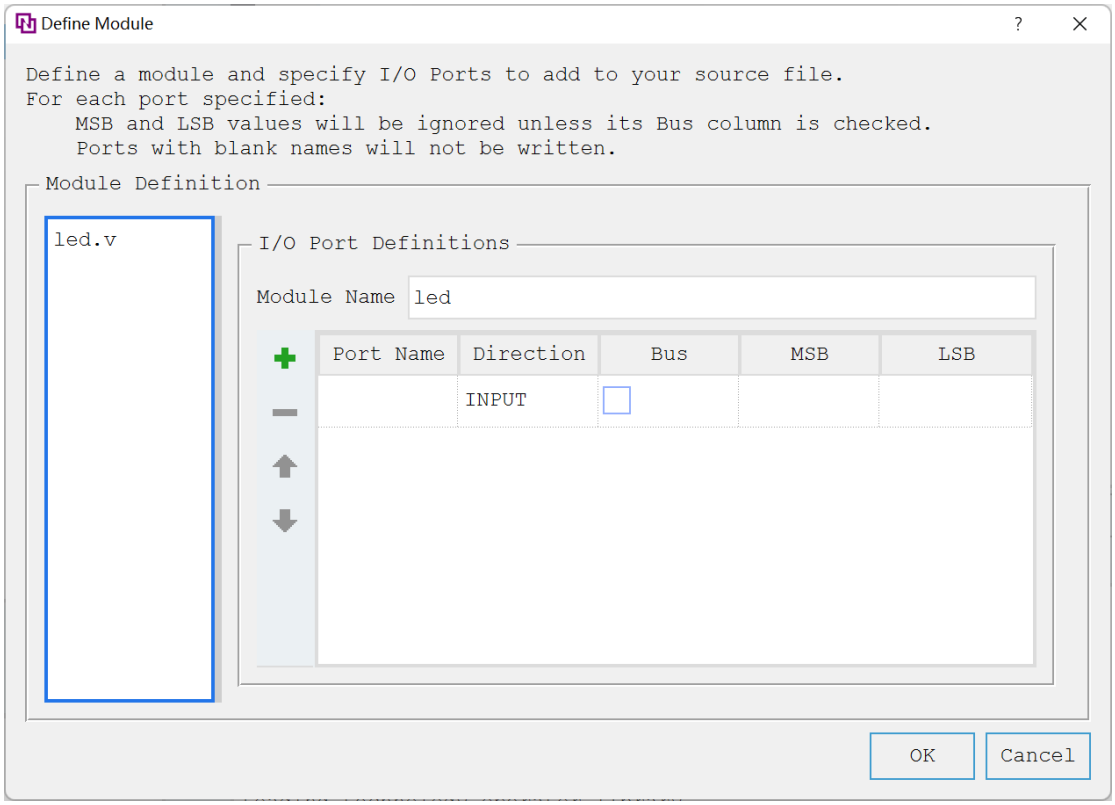
2）选择 Verilog Design File，文件名和 module 名一致，默认路径，点击 OK；


Add Sources
Add Design Source Files
Select design files you want to include in your project.
Create a new source file and add it to your project.
File Name
File Location
Add Files
Create Design Source File
Create a new design file and add it to your project.
File Type:
Verilog Design File (.v)
File Name:
module 文件名
File Location:
C:/Users/w/source
OK
Cancel
Move to Top
Move to Bottom
Copy design files into project
Add source from subdirectories
< Back
OK
Cancel

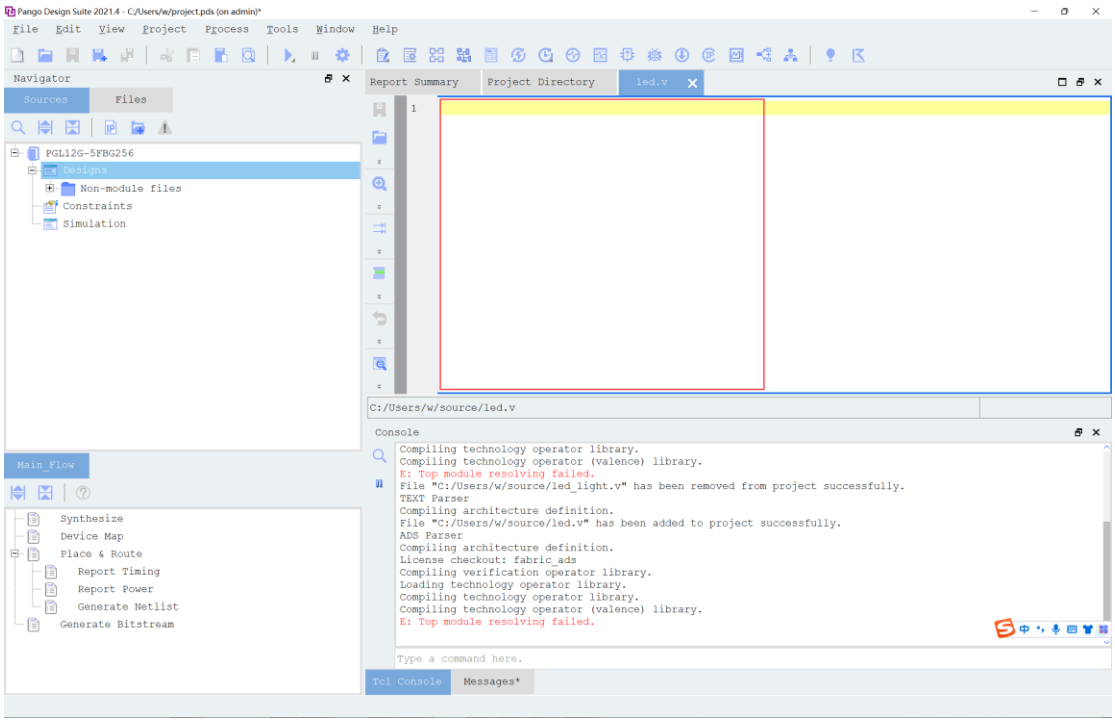
3）点击 OK；


Add Sources
Add Design Source Files
Select design files you want to include in your project.
Create a new source file and add it to your project.
File Name
File Location
led.v
C:/Users/w/source
Add Files
Add Files List
Add Directories
Create File
Remove
Move Up
Move Down
Move to Top
Move to Bottom
Copy design files into project
Add source from subdirectories
< Back
OK
Cancel

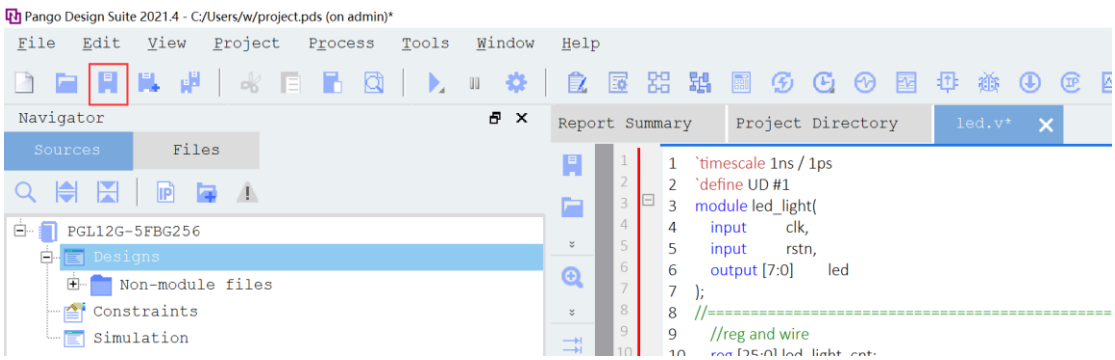
3）点击 Cancel；



4）默认打开新建文件，将前面设计的 module 复制进去，

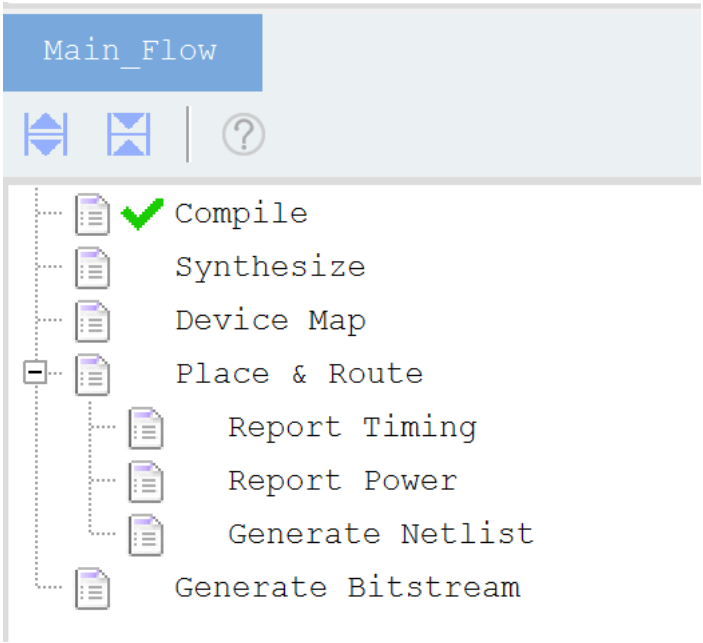


5）点击保存，新建文件完成



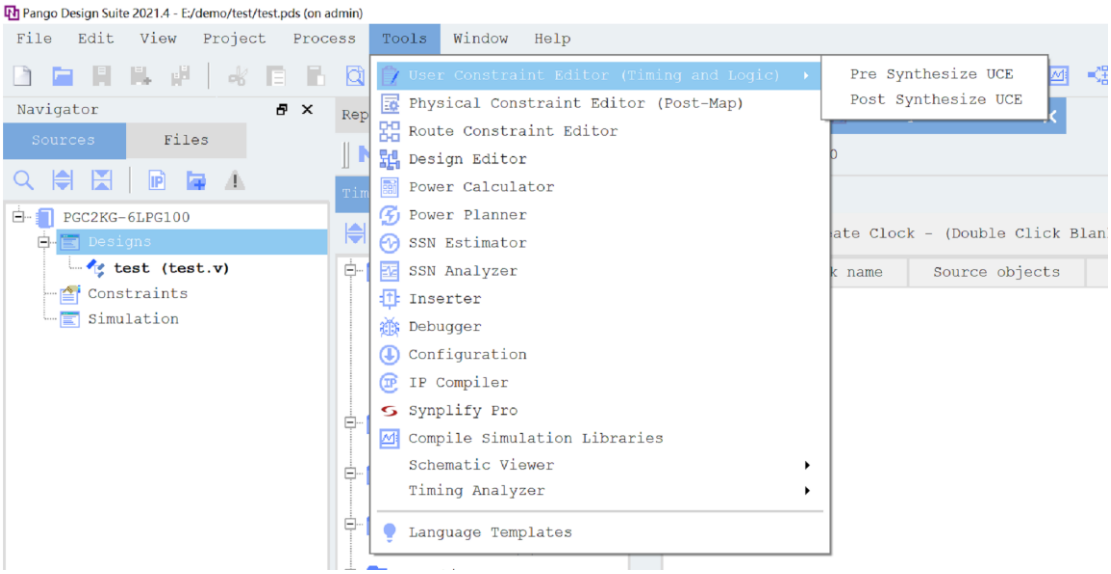
1.5.3编译

双击 Compile 或右击选择 Run，编译完成后如下；

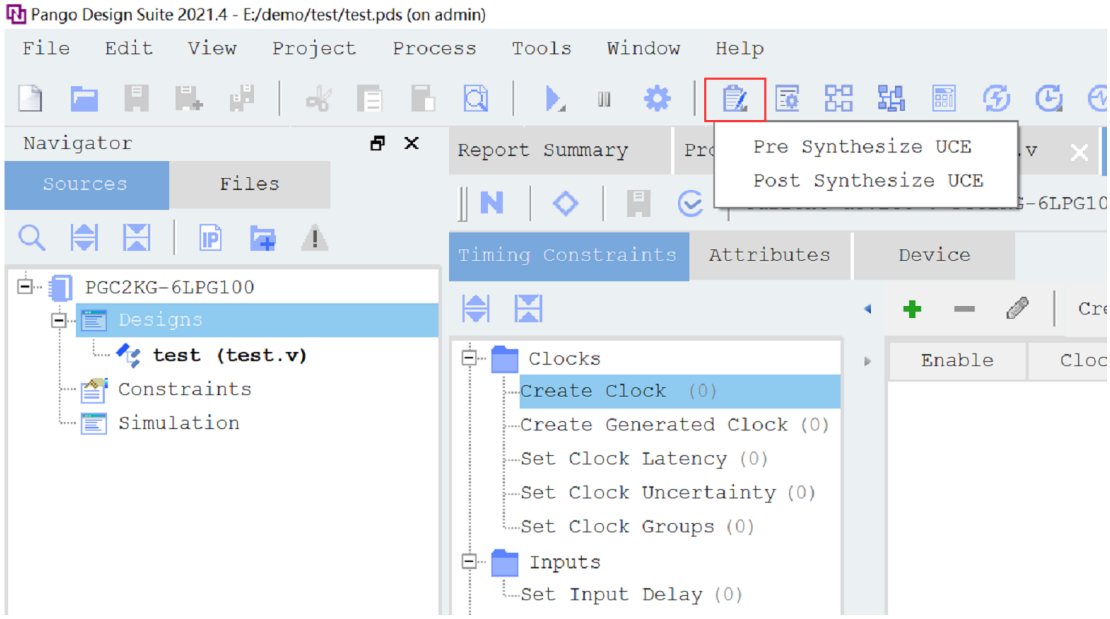


1.5.4工程约束

点击 Tools 选择 User Constraint Editor(Timing and Logic)或者点击工具栏图标 User Constraint Editor(Timing and Logic) 选择 Pre Synthesize UCE，如下图所示。



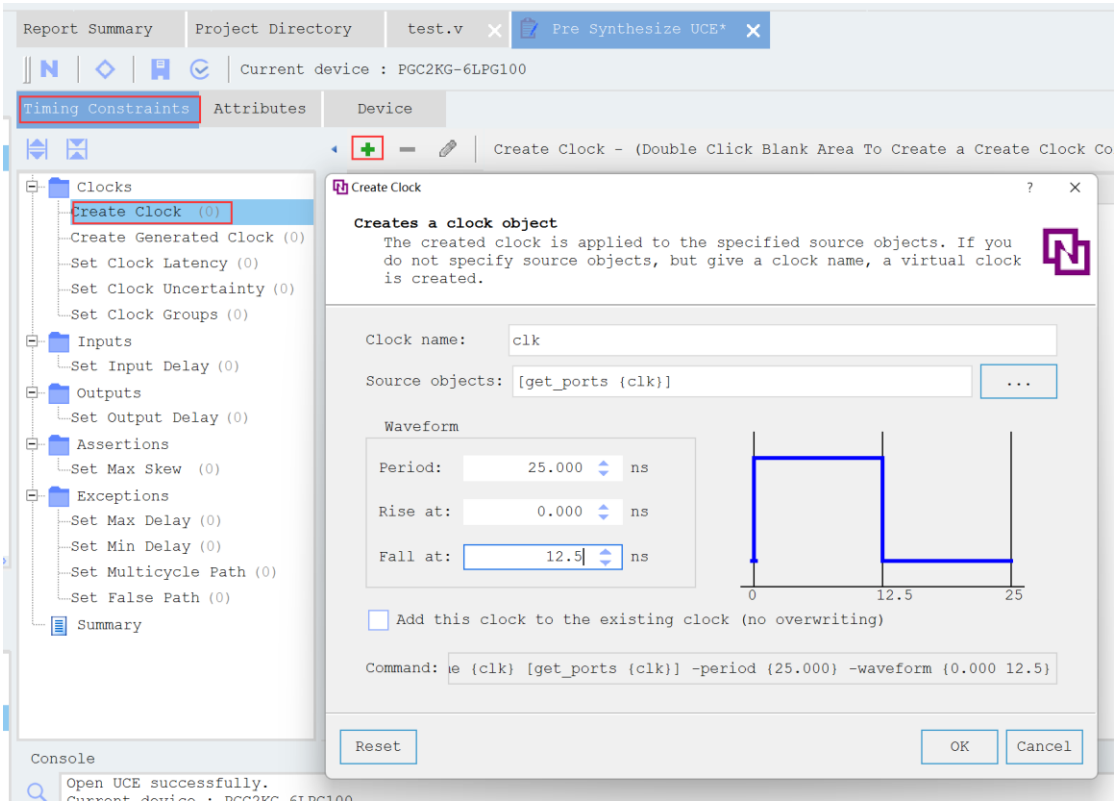
Tools 下的 User Constraint Editor(Timing and Logic)



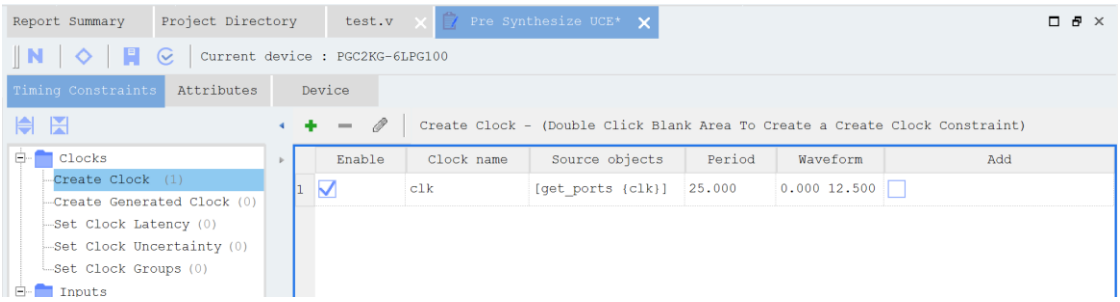
工具栏 User Constraint Editor(Timing and Logic)图标

1.5.4.1 时钟约束

打开 UCE 后，选择 Timing Constraints 后选择 Create Clock 添加基准时钟，基准时钟一般是通过输入 port 输入用户涉及的板上时钟。

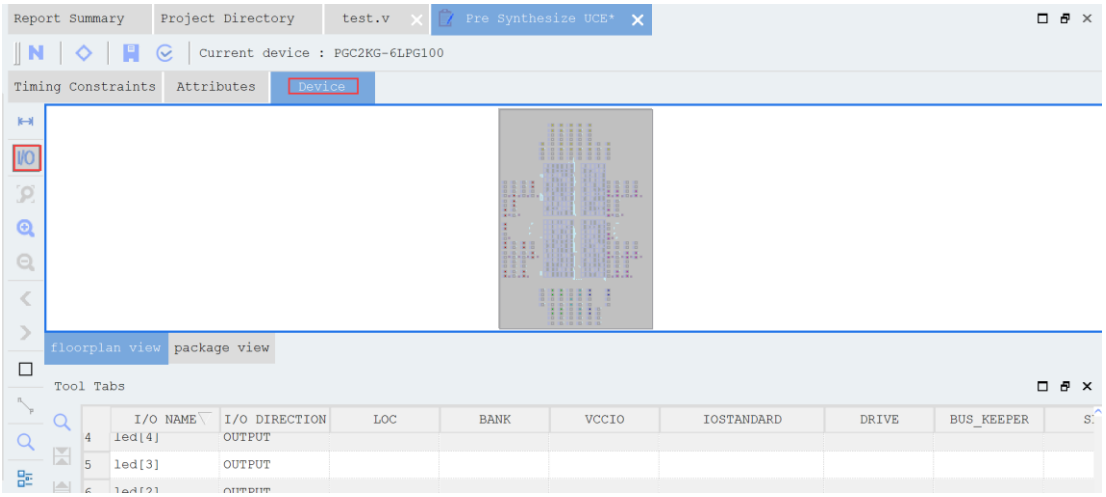


在弹窗中对时钟命名，关联时钟管脚，添加时钟参数，点击 OK 会创建一条时钟约束，Reset 重置该页面。创建完成如下图所示：



1.5.4.2 物理约束

打开 UCE 后，选择 Device 后选择 I/O，根据原理图编辑 IO 的分配。



Tool Tabs

	I/O NAME	I/O DIRECTION	LOC	BANK	VCCIO	IOSTANDARD
1	led[7]	OUTPUT	19	BANK3	1.2	LVCMOS12
2	led[6]	OUTPUT	17	BANK4	1.2	LVCMOS12
3	led[5]	OUTPUT	18	BANK3	1.2	LVCMOS12
4	led[4]	OUTPUT	16	BANK4	1.2	LVCMOS12
5	led[3]	OUTPUT	15	BANK4	1.2	LVCMOS12
6	led[2]	OUTPUT	14	BANK4	1.2	LVCMOS12
7	led[1]	OUTPUT	13	BANK4	1.2	LVCMOS12
8	led[0]	OUTPUT	12	BANK4	1.2	LVCMOS12
9	clk	INPUT	63	BANK1	1.2	LVCMOS12
10	rstn	INPUT	81	BANK0	1.2	LVCMOS12

编辑好 IO 分配后，点击保存，完成约束。

1.6生成位流文件并下载

双击 Generate Bitstream 或右击选择 Run，生成二进制位流文件。

下载位流文件请参考《PDS 快速使用手册》2.8 下载位流文件

1.7实验现象

8 个 LED 灯同时亮和灭，亮和灭之间间隔时间为 0.5s；

2. LED 流水灯

2.1实验目的

掌握流水灯原理并实现流水灯

2.2实验要求

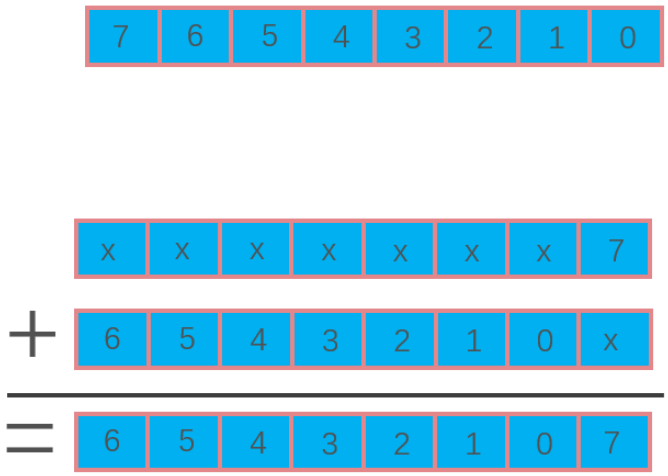
流水灯：8 个 LED 以 0.5s 间隔接替闪烁

2.3实验原理

相比上一个 LED 闪烁的实现，只需要改变 LED 的状态。将 8 个 LED 灯流水式的点亮；

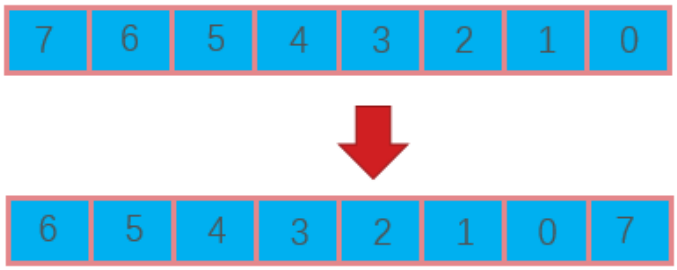
在 C 语言中做流水灯的实验需要用到一个中间变量(代码如下左侧，数据位的搬移如下右图)：

```
1   int data;
2   int temp1,temp2;
3
4   data = 0x01;
5   temp1 = data >> 7;
6   temp2 = data <<1;
7   data   =   temp1   |
    temp2;
```



在 FPGA 的开发中是基于硬件，语言也是硬件描述语言，verilog 的处理单位就是 1bit；8bit 的位宽数据可看作 8 个独立的信号线，这 8 个信号线之间的排序及相互之间的赋值可以随意组合；代码如下：

```
reg [7:0] data;
always @(posedge clk)
data <= {data[6:0],data[7]};
//或：
wire [7:0] data1;
wire [7:0] out;
assign out = {data1[6:0],data1[7]};
```



2.4实验源码设计（完整源码查看 demo 源文件）

Module 的具体内容如下：

```
`timescale 1ns / 1ps
`define UD #1
module water_led (
    input         clk,
    input         rstn,
    output [7:0]  led
);

//=================================================
//reg and wire
    reg [25:0]    led_light_cnt;
    reg [ 7: 0]   led_status;

    //  time counter
    always @(posedge clk)
    begin
```

```
17          if(!rstn)
18              led_light_cnt <= `UD 26'd0;
19          else if(led_light_cnt == 26'd19_999_999)
20              led_light_cnt <= `UD 26'd0;
21          else
22              led_light_cnt <= `UD led_light_cnt + 26'd1;
23      end
24
25      // led status change
26      always @(posedge clk)
27      begin
28          if(!rstn)
29              led_status <= `UD 8'b0000_0001;
30          else if(led_light_cnt == 25'd19_999_999)
31              led_status <= `UD {led_status[6:0], led_status[7]};
32      end
33      assign led = led_status;
34      endmodule
```

2.5实验步骤

工程创建及编译流程与前面 Led 闪烁实验一致，在添加文件的步骤，添加本实验的 water_led 的 verilog 文件即可，管脚分配如下：

Tool Tabs

	I/O NAME	I/O DIRECTION	LOC	BANK	VCCIO	IOSTANDARD
1	led[7]	OUTPUT	19	BANK3	1.2	LVCMOS12
2	led[6]	OUTPUT	17	BANK4	1.2	LVCMOS12
3	led[5]	OUTPUT	18	BANK3	1.2	LVCMOS12
4	led[4]	OUTPUT	16	BANK4	1.2	LVCMOS12
5	led[3]	OUTPUT	15	BANK4	1.2	LVCMOS12
6	led[2]	OUTPUT	14	BANK4	1.2	LVCMOS12
7	led[1]	OUTPUT	13	BANK4	1.2	LVCMOS12
8	led[0]	OUTPUT	12	BANK4	1.2	LVCMOS12
9	clk	INPUT	63	BANK1	1.2	LVCMOS12
10	rstn	INPUT	81	BANK0	1.2	LVCMOS12

2.6实验现象

8 个 led 依次被点亮，后一个灯被点亮时前一个灯熄灭，依次往返，让亮起来的 led 灯像是在 8 个 led 灯上流动起来一样，故而此实验称之为流水灯。

3.键控彩灯

3.1实验目的

1　设计 8 种彩灯效果，可通过按键切换。

2　选择一个按键作为控制输入，按下一次换一种显示效果，在 8 种效果中循环。

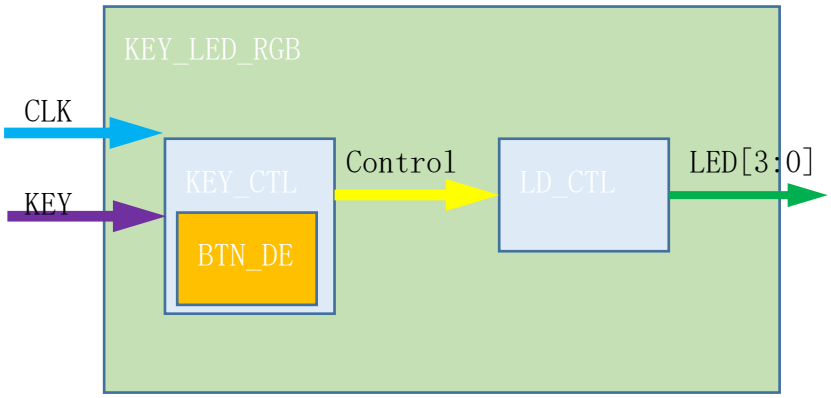
3.2实验要求

1、实验平台：MES2KG 开发板；

2、按键输入由 K1 输入，LED 输出为 LD1～LD4。

3.3实验原理

实现框架如下：



1、顶层实现按键切换 LED 的彩灯状态；

2、需要设计一个输入控制模块及一个输出控制模块；

这个实验带大家将多个模块整合成为一个工程，涉及到的知识点有子模块设计、模块例化；子模块的设计主要是依据功能定位，确定输入输出，再做具体的设计；

模块例化方式如下：

```
module_name #(
    .PARAM      (  PARAM_SET )      // PARAM 为例化模块的常量接口；PARAM_SET 为常量赋值内容
) unint_name(          // module_name 为例化 module 名；unint_name 为例化后单元名称
    .port       (  signal    )    // port 为例化模块中的管脚；signal 为当前模块的信号
);
```

3.3.1按键控制模块功能

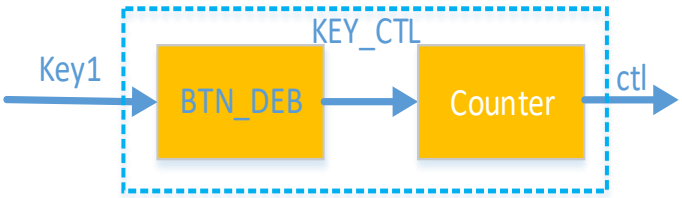
接收按键输入信号。统计按键按下次数，由于彩灯模式是 8 种，计数统计范围是 0～7 循环，将计数结果传递给 LD 控制模块；

根据需求输入信号有：时钟，按键；输出信号有：彩灯控制信号；

内部功能处理：

<1>内部需要对按键信号做消抖处理；

<2>按键触发计数器（计数值输出）改变继而调整彩灯的状态；

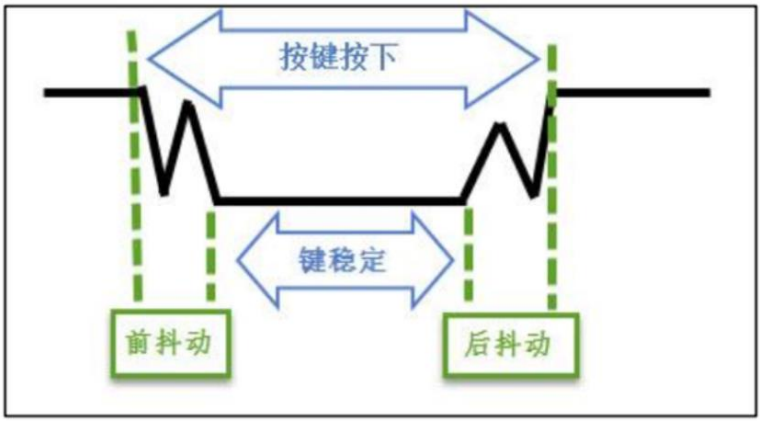


3.3.2按键消抖

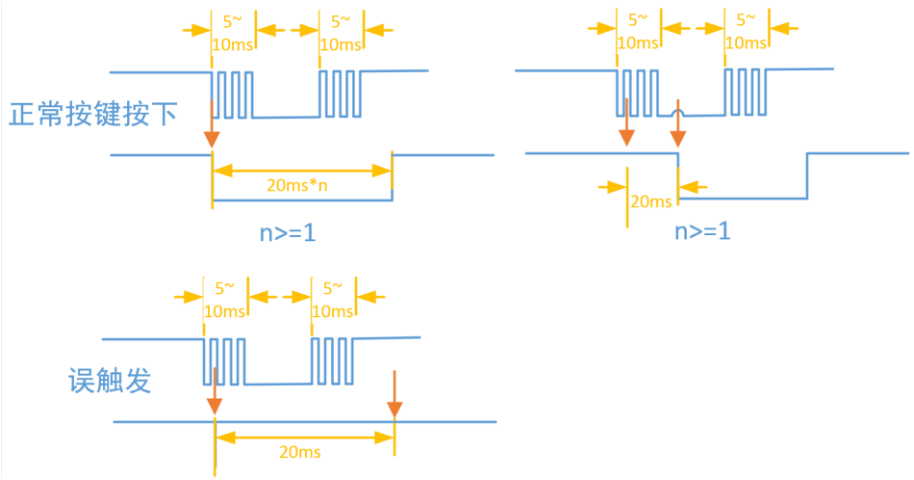
3.3.2.1 消抖目的

机械式弹片按键，在按下或松开时会有机械抖动，导致在按下或松开时按键的状态不稳定，在快速的变化，在使用按键输入信号时如果采集了抖动时的状态，会导致工程运行出现不可控的变化，故而我们需要将这段时间的抖动信号给滤除掉，故此实验称之为按键消抖；

3.3.2.2 实验原理



前后抖动时间约为 5～10ms，前后抖动共在 20ms，以最大 20ms 做设计，使用计数到 N 归零的计数器来做时间刻度计时；以 20ms 的间歇对按键输入信号进行采集，从而避开按键的抖动引起的信号快速变化；



设计 1 个 20bit 的计数器,其计数最大值为:N = 20’hC3500 = 20’d800000 最大计数值时，计时为：t= N*T = N/f = 800000/40M = 20ms;

注：对于计数器完成计时功能在 LED 灯控制中已有详细讲解，需要关注输入时钟频率以及目标计时时长，从而得到计数器的计数范围；

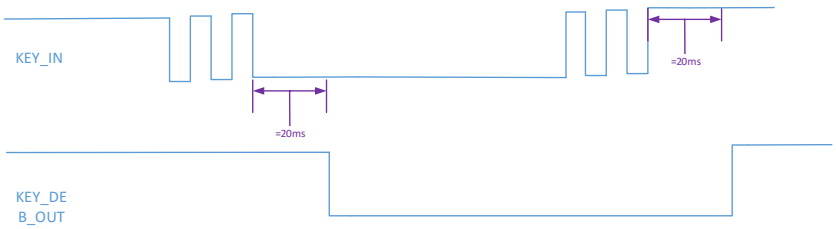
3.3.2.3 实验源码设计

```
1  `timescale 1ns / 1ps
2  `define UD #1
3   module btn_deb#(
4      parameter                    BTN_WIDTH = 4'd8 // key width
   常量
5      parameter                MS_20 = 20'd800_000
6  )
```

```
(
    input                               clk,    //40MHz
    Input             [BTN_WIDTH-1:0]     btn_in, // key input
    output reg      [BTN_WIDTH-1:0]       btn_deb // key output
);

//=========================================
    reg [19:0] time_cnt= 20'd0;
    always @ (posedge clk)
    begin
        if(time_cnt == MS_20-1' b1)    // reset at 20ms counter
over
            time_cnt <= `UD 20'd0;
        else
            time_cnt <= `UD time_cnt + 1'd1;
    end
        always @(posedge clk)
    begin
        if(time_cnt == MS_20-1' b1)
            btn_deb <= `UD btn_in;  // latch the key input every
20ms
    end
endmodule
```

此种方法有一定误触发概率出现，大家可在此基础上做补充完善；思路如下（扩展实现）：



这个 module 的设计中新增加一种语法：parameter ；在 verilog 中 parameter 是对常量进行定义,将 parameter 定义放在 module 的接口中是可进行模块传递，传递方式请看后面模块例化；

3.3.3LED 控制模块功能

8 种流水灯模式有按键传递过来的计数控制切换，每一个 LED 的显示状态完整后进入下一模式初始化。根据需求可得到如下信息：

输入信号：时钟，彩灯模式控制信号； 出信号：12bit 位宽的 LED 控制信号；

功能处理注意事项：彩灯状态切换点，不同状态的切换时如何初始化；

3.4实验源码设计（完整源码查看 demo 源文件）

3.4.1顶层文件源码

```
`timescale 1ns / 1ps
`define UD #1
module key_led_rgb_top(
    input           clk,//40MHz
    input           key,

    output [11:0]    led_rgb
);

   wire [2:0] ctrl;

   key_ctl key_ctl(
       .clk        (  clk  ),//input           clk,
       .key        (  key  ),//input           key,

       .ctrl       (  ctrl  )//output     [1:0] ctrl
   );

   led_rgb  u_led_rgb(
       .clk   (  clk   ),//input         clk,
       .ctrl  (  ctrl  ),//input  [1:0]  ctrl,

       .led_rgb   (  led_rgb   ) //output [11:0]  led
   );
endmodule
```

3.4.2按键控制模块

```
`timescale 1ns / 1ps
`define UD #1
module key_ctl(
    input           clk,
    input           key,
```

```
39
40      output     [3:0] ctrl
41  );
42
43      wire btn_deb;
44      // 按键消抖
45      btn_deb#(
46          .BTN_WIDTH      (    4'd1                ),  //parameter
    BTN_WIDTH = 4'd8
47          .MS_20       (  20'd800_000  )
48      ) U_btn_deb
49      (
50          .clk                  (    clk                 ),//input
    clk,
51          .btn_in       (  key          ),//input       [BTN_WIDTH-
    1:0] btn_in,
52
53          .btn_deb     (  btn_deb     ) //output reg [BTN_WIDTH-1:0]
    btn_deb
54      );
55
56      reg btn_deb_1d;
57      always @(posedge clk)
58      begin
59          btn_deb_1d <= `UD btn_deb;
60      end
61
62      reg [2:0]  key_push_cnt=3'd0;
63      always @(posedge clk)
64      begin
65          if(~btn_deb & btn_deb_1d)
66          begin
67              if(key_push_cnt == 3'd7)
68                  key_push_cnt <= `UD 2'd0;
69              else
70                  key_push_cnt <= `UD key_push_cnt + 3'd1;
71          end
72      end
73
74      assign ctrl = key_push_cnt;
75  endmodule
```

按键消抖

```
1  `timescale 1ns / 1ps
2  `define UD #1
3  module btn_deb#(
4      parameter BTN_WIDTH = 4'd8,
5      parameter MS_20 = 20'd800_000
6  )
7  (
8      input      clk,//40MHz
9      input      [BTN_WIDTH-1:0] btn_in,
10     Output reg [BTN_WIDTH-1:0] btn_deb
11 );
12 //================================================================
   ==================
13     reg [19:0] time_cnt= 20'd0;
14 always@(posedge clk)
15   begin
16      if(time_cnt == MS_20 - 1'b1)
17          time_cnt <= `UD 20'd0;
18      else
19          time_cnt <= `UD time_cnt + 1'd1;
20   end
21 always @(posedge clk)
22    begin
23      if(time_cnt == MS_20 - 1'b1)
24           btn_deb <= `UD btn_in;
25    End
26 endmodule
27
```

3.4.3LED 控制模块

```
28 `timescale 1ns / 1ps
29 `define UD #1
30 module led_rgb(
31     input         clk,//40MHz
32     input  [2:0]  ctrl,
33
34     output  reg [11:0]  led_rgb
35 );
36
37     // led_rgb status change
38     always @(posedge clk)
39        begin
```

```
40              case(ctrl)
41                    3'd0 :  //R
42                         led_rgb<=`UD 12'b0000_1111_1111;
43                    3'd1 :  //G
44                        led_rgb<=`UD 12'b1111_0000_1111;
45                    3'd2 :  //B
46                        led_rgb<=`UD 12'b1111_1111_0000;
47                    3'd3 :  //RG
48                        led_rgb<=`UD 12'b0000_0000_1111;
49                    3'd4 :  //RB
50                        led_rgb<=`UD 12'b0000_1111_0000;
51                    3'd5 :  //GB
52                        led_rgb<=`UD 12'b1111_0000_0000;
53                    3'd6 :  //RGB
54                        led_rgb<=`UD 12'b0000_0000_0000;
55                    default:led_rgb<=`UD 12'b1111_1111_1111;
56                endcase
57            end
58  endmodule
```

3.5实验现象

上电后下载完固件，默认 LD1～LD4 流水，每按下一次 KEY1，彩灯状态切换一次，总共 8 种状态可供循环切换；

4.数码管动态显示

4.1实验目的

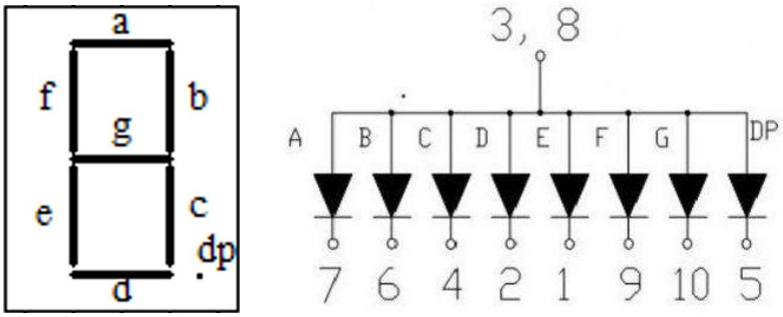
动态控制 4 位数码管显示不同的数值；

4.2实验要求

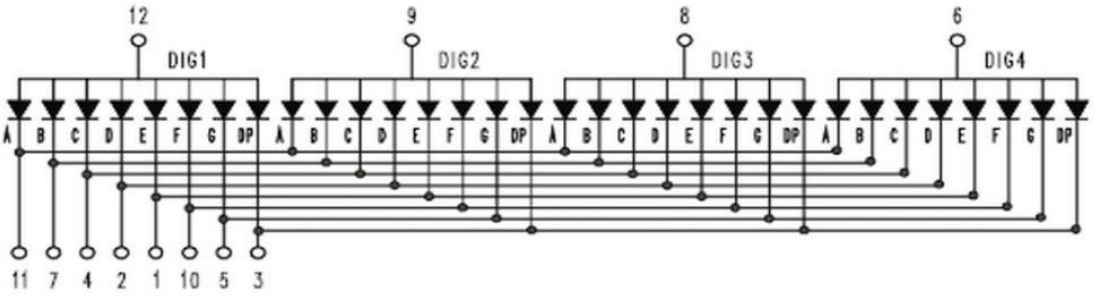
四个数码管显示不同的数字，按键 K1 控制第一个数码管，按一下数字加 1，从 0 到 9，按键 K2 控制第二个数码管，按一下数字加 1，从 0 到 9，按键 K3 控制第三个数码管，按键 K4 控制第四个数码管。

4.3实验原理

数码管是一种半导体发光器件，其基本单元是发光二极管。能显示 4 个数码管叫四位数码管。数码管按段数分为七段数码管和八段数码管，八段数码管比七段数码管多一个发光二极管单元（多一个小数点显示）；按发光二极管单元连接方式分为共阳极数码管和共阴极数码管。共阳数码管是指将所有发光二极管的阳极接到一起形成公共阳极(COM)的数码管。共阳数码管在应用时应将公共极 COM 接到+5V，当某一字段发光二极管的阴极为低电平时，相应字段就点亮。当某一字段的阴极为高电平时，相应字段就不亮。共阴数码管是指将所有发光二极管的阴极接到一起形成公共阴极(COM)的数码管。共阴数码管在应用时应将公共极 COM 接到地线 GND 上，当某一字段发光二极管的阳极为高电平时，相应字段就点亮。当某一字段的阳极为低电平时，相应字段就不亮。



4 位共阴数码管内部管脚连接图如下：



段选：段选由 8 根 led 灯组成，分别为 a，b，c，d，e，f，g，dp；

由段选信号控制某段数码管点亮；

位选：位选由 4 组 8 个段选 LED 组成，分别为 seg1，seg2，seg3，seg4；

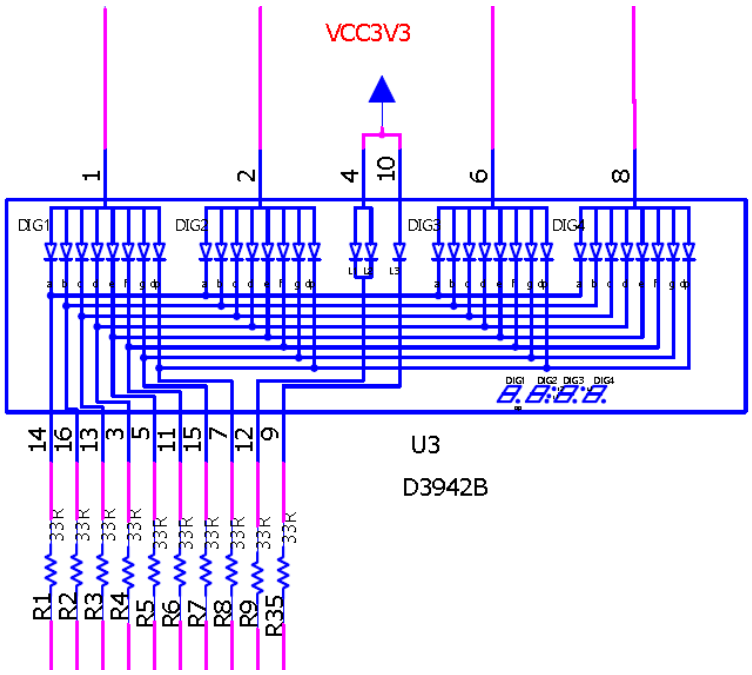
由选通信号控制第几块数码管点亮；

例：如果我们只点亮第一位的 A：需要将 11 脚配置高电平，其他段选（1-5，7，10，11）配置低电平；将 12 脚配置低电平，其他位选脚配置（6，8，9）高电平；

点亮数码管原理：

输入相应的电平点亮一根根小火柴 a-b-c-d-e-f-g-dp。如果数码管是共阴极，给高电平 1 即可相应点亮，反之如果是共阳极，给低电平 0 即可相应点亮。

MES2KG 数码管底板的数码管使用共阳数码管；



数码管显示出 0～9，代码如下，通过传递要显示的数值给到 key 上，可显

示对应数值，sel 选择对应的数码管，如需 4 个如果要显示同样的字符，仅需将 dig 的 4 位全部置 1，需要做好对应编码；

```
module seq_control
(
    input [1:0]sel,
    input [3:0]key,
    output reg [3:0]dig,
    output reg [7:0]smg
);
    /*==================================================
                        位选择映射
    ==================================================*/
    always @(*)
    begin
        case(sel)
            2'd0:dig = 4'b0001;
            2'd1:dig = 4'b0010;
            2'd2:dig = 4'b0100;
            2'd3:dig = 4'b1000;
            default:dig = 4'b0000;
        endcase
    end

    //=========================================
    // 0 1 2 3 4 5 6 7
    // G F E D C B A P
    //共阳极数码管，为 0 有效，即点亮
    //=========================================
    always @(*)
    begin
        case(key)
            4'd0:smg = 8'b1000_0001;//"0"
            4'd1:smg = 8'b1100_1111;//"1"
            4'd2:smg = 8'b1001_0010;//"2"
            4'd3:smg = 8'b1000_0110;//"3"
            4'd4:smg = 8'b1100_1100;//"4"
            4'd5:smg = 8'b1010_0100;//"5"
            4'd6:smg = 8'b1010_0000;//"6"
            4'd7:smg = 8'b1000_1111;//"7"
            4'd8:smg = 8'b1000_0000;//"8"
            4'd9:smg = 8'b1000_0100;//"9"
            default:smg = 8'b1111_1111;
        endcase
    end

endmodule

```

硬件连接上无法同一个时间点显示出不同的数值，我们可以通过刷新显示的方式造成视觉上同时显示了不同的数值，依据如下：

人眼对于时间频率的响应近似一个滤波器，在一般室内强光下，对 15~20Hz

信号最敏感，有很强闪烁感(flick)，大于 75Hz 响应为 0，闪烁感消失。刚到达闪烁感消失的频率叫做临界融合频率(CFF)。在较暗的环境下，呈低通特性，且 CFF 会降低，这时对 5Hz 信号最敏感，大于 25Hz 闪烁基本消失。电影院环境很暗，放映机的刷新率为 24Hz 也不感到闪烁；这种特性也可以解析为视觉暂留特性，即当影像消失/变化时，大脑的影像不会立刻消失，而是保留一个短暂时间。

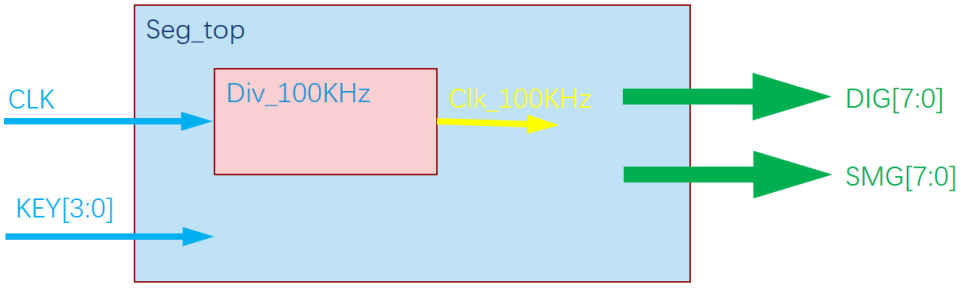
在设计数码管闪烁式显示时，对于人眼观测来说，频率越高越好，但是数码管中的 LED 灯珠点亮对于高电平（关注发光响应时间）是有要求的，故而不是越高越好，取一个适当的刷新频率即可，实验中我们取刷新率为 100KHz。

方案设计：

1、按键消抖：参考按键流水灯实验

2、按键计数：参考按键流水灯实验

3、数码管的分时显示；



4.4实验源码（完整源码查看 demo 源文件）

4.4.1顶层模块

```
`timescale 1ns / 1ps
`define UD #1
module top_seq
(
    input clk,//40MHZ 25ns
    input [3:0]button,
    output reg [3:0]dig,
    output reg [7:0]smg
);
/*================================================
                    按键消抖
================================================*/
wire [3:0]key;
```

```
14  btn_deb
15  #(
16     .BT_WIDTH(4'd4)
17  )
18  u_btn_deb
19  (
20      .clk(clk),
21      .btn_in(button),
22      .btn_out(key)
23  );
24  /*==================================================
25                          4 个按键的计数
26  ==================================================*/
27  wire [3:0] key0_cnt;
28  key_cnt key1
29  (
30      .clk(clk),
31      .key(key[0]),
32      .key_times(key0_cnt)
33  );
34
35  wire [3:0] key1_cnt;
36  key_cnt key2
37  (
38      .clk(clk),
39      .key(key[1]),
40      .key_times(key1_cnt)
41  );
42
43  wire [3:0] key2_cnt;
44  key_cnt key3
45  (
46      .clk(clk),
47      .key(key[2]),
48      .key_times(key2_cnt)
49  );
50
51  wire [3:0] key3_cnt;
52  key_cnt key4
53  (
54      .clk(clk),
55      .key(key[3]),
56      .key_times(key3_cnt)
```

```
57  );
58  /*=================================================
59                        时钟分频
60  =================================================*/
61  wire clk_100khz;
62  div_clk div_clk
63  (
64      .clk           (clk),
65      .clk_100khz (clk_100khz)
66  );
67  /*=================================================
68                        数码管显示
69  =================================================*/
70  reg  [1:0]sel=0;
71  wire [3:0]dig0;
72  wire [7:0]smg0;
73
74  always @(posedge clk_100khz)
75  begin
76      sel <= `UD sel+1'b1;
77  end
78
79  seq_control seq_control_0
80  (
81      .sel(2'd3),
82      .key(key0_cnt),
83      .dig(dig0),
84      .smg(smg0)
85  );
86
87  wire [3:0]dig1;
88  wire [7:0]smg1;
89
90  seq_control seq_control_1
91  (
92      .sel(2'd2),
93      .key(key1_cnt),
94      .dig(dig1),
95      .smg(smg1)
96  );
97
98  wire [3:0]dig2;
99  wire [7:0]smg2;
```

```
100
101 seq_control seq_control_2
102 (
103     .sel(2'd1),
104     .key(key2_cnt),
105     .dig(dig2),
106     .smg(smg2)
107 );
108
109 wire [3:0]dig3;
110 wire [7:0]smg3;
111
112 seq_control seq_control_3
113 (
114     .sel(2'd0),
115     .key(key3_cnt),
116     .dig(dig3),
117     .smg(smg3)
118 );
119
120
121 always @(posedge clk_100khz)
122 begin
123     if(sel==2'b00)
124             dig <= `UD dig0;
125     else if(sel==2'b01)
126             dig <= `UD dig1;
127     else if(sel==2'b10)
128             dig <= `UD dig2;
129     else if(sel==2'b11)
130             dig <= `UD dig3;
131 end
132
133
134 always @(posedge clk_100khz)
135 begin
136     if(sel==2'b00)
137             smg <= `UD smg0;
138     else if(sel==2'b01)
139             smg <= `UD smg1;
140     else if(sel==2'b10)
141             smg <= `UD smg2;
142     else if(sel==2'b11)
```

```
143             smg <= `UD smg3;
144 end
145 endmodule
```

4.4.2按键消抖模块

```
1   `timescale 1ns / 1ps
2   `define     UD #1
3   module btn_deb #(
4       parameter     BT_WIDTH = 4'd8
5   )
6   (
7       input                               clk,     //40M.25ns
8       input          [BT_WIDTH-1:0]       btn_in,
9
10      output reg [BT_WIDTH-1:0]       btn_out
11  );
12
13      reg [19:0] time_cnt=20'd0;//20ms 计时计数寄存器;
14
15      always @(posedge clk)
16      begin
17          time_cnt <= `UD time_cnt + 20'd1;  //计数器到 20'hf_ffff
    时周期约为 21ms
18      end
19
20      always @(posedge clk)
21      begin
22          if(time_cnt == 0)    //每隔 20ms 取一次按键状态值;
23              btn_out <= `UD btn_in;
24      end
25  endmodule
```

4.4.3按键计数模块

```
1   `timescale 1ns / 1ps
2   `define UD #1
3   module key_cnt
4   (
5       input clk,//40M,25ns
6       input rstn_key,
7       input key,
8       output reg [3:0]key_times
9   );
10
```

```
reg key_reg;
always @(posedge clk)
begin
    key_reg <= `UD key;
end

always @(posedge clk )
begin
    if(key_reg&&~key&&key_times==4'd9)
            key_times <=`UD 4'd0;
    else if(key_reg&&~key)
            key_times <=`UD key_times + 1'b1;
end

endmodule
```

4.4.4时钟分频模块

```
`timescale 1ns / 1ps
`define UD #1
module div_clk
(
    input clk,//40M
    output  clk_100khz//0.1M
);
reg [8:0]cnt;
always @(posedge clk)
begin
    if(cnt == 9'd399)
            cnt<= `UD 9'd0;
    else
            cnt <= `UD cnt + 1'b1;
end
reg flag=1'b0;
always @(posedge clk)
begin
    if(cnt == 9'd199)
            flag <= `UD 1'b1;
    else if(cnt == 9'd399)
            flag <= `UD 1'b0;
end
assign clk_100khz = flag;
endmodule
```

4.4.5数码管显示模块

```
`timescale 1ns / 1ps
`define UD #1
module seq_control
(
    input [1:0]sel,
    input [3:0]key,
    output reg [3:0]dig,
    output reg [7:0]smg
);
/*=================================================
                        位选择映射
=================================================*/
always @(*)
begin
    case(sel)
            2'd0:dig = 4'b0001;
            2'd1:dig = 4'b0010;
            2'd2:dig = 4'b0100;
            2'd3:dig = 4'b1000;
            default:dig = 4'b0000;
    endcase
end
//共阳极数码管，为0有效，即点亮
always @(*)
begin
    case(key)
            4'd0:smg = 8'b1000_0001;//"0"
            4'd1:smg = 8'b1100_1111;//"1"
            4'd2:smg = 8'b1001_0010;//"2"
            4'd3:smg = 8'b1000_0110;//"3"
            4'd4:smg = 8'b1100_1100;//"4"
            4'd5:smg = 8'b1010_0100;//"5"
            4'd6:smg = 8'b1010_0000;//"6"
            4'd7:smg = 8'b1000_1111;//"7"
            4'd8:smg = 8'b1000_0000;//"8"
            4'd9:smg = 8'b1000_0100;//"9"
            default:smg = 8'b1111_1111;
    endcase
end
endmodule
```

4.5实验现象

KEY1～4 分别控制数码管从左到右的数码管显示，按键 K1 控制第一个数码管，按一下数字加 1，从 0 到 9，按键 K2 控制第二个数码管，按一下数字加 1，从 0 到 9，按键 K3 控制第三个数码管，按键 K4 控制第四个数码管。

5.序列检测器

5.1实验目的

在连续信号中,检测是否包含特定序列,例如检测“1101”中是否包含“01”

5.2实验要求

1、 拨码开关 SW1-SW4 作为序列信号输入；

2、 KEY1-KEY2 作为特定信号输入序列，KEY 按下后对应的 LED 灯会亮起，表示对应位为 1，再按一下会熄灭，表示对应位为 0；

3、 K4 为序列检测开始和序列检测结束按键，初次按下 KEY4，开始检测，此时 LED4 也会被点亮，显示当前状态，再按一下停止检测，LED4 熄灭；结束后序列串中出现特定序列的次数显示在数码管上。

5.3实验原理

SW1～SW4 的状态为检测序列；

LED1～LED2 为特定序列；

数码管显示的结果为 LED[2:1]在 SW[4:1]中出现的次数；

5.4实验源码（完整源码查看 demo 源文件）

5.4.1方案设计

从实验目的分析此实验的实现需要有三个功能模块：

1、按键 LED 模块；

按键调整特定序列，由 KEY[1:0]控制特定序列值；KEY4 控制是否检测；输出用 LED 来显示及保存特定序列,同时也将特定序列与检测使能信号传递给检测模块；

2、序列对比模块；

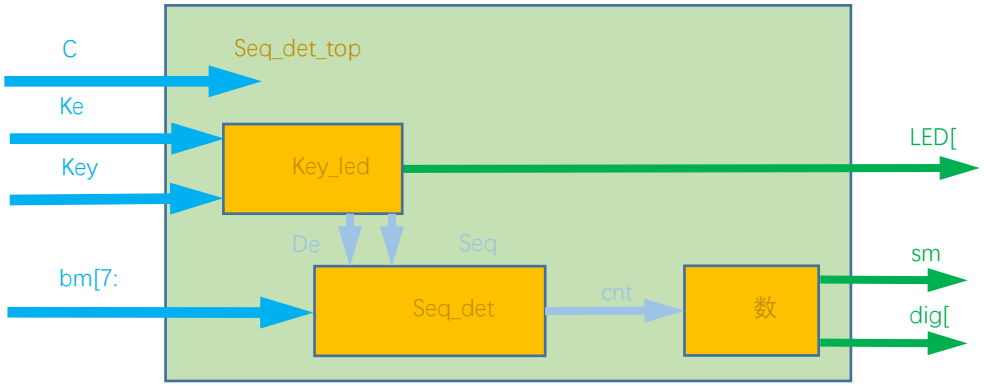
由拨码开关提供待检测序列,接收按键控制模块传递过来的特定序列与检测使能信号控制与待检测序列进行比较;比较结果输出给到数码管显示模块进行显示；

3、数码管控制模块

数码管显示模块的目标是将统计结果显示出来,用动态数码管显示的方式即

可；

对应模块之间的连线如下框图：



5.4.2顶层模块（含数码管显示模块）设计

```
1   `timescale 1ns / 1ps
2   `define UD #1
3   module top_seq_det
4   (
5       input           clk,    // input clock  40M
6       input           key_det,        // KEY 4 input control detected
    enable
7       input [1:0]     key_in, // KEY[1:0] input control detected
    sequence
8       input [3:0]     bm,     // DIP Switch[3:0] input used to
    detecting
9       output          key_det_led, // display the detecte status
10      output [1:0]    key_in_led,     // display the detected
    sequence
11      output reg [3:0]        dig,    // output Digital tube Bit
    selection
12      output reg [7:0]        smg     // output Digital tube segment
    selection
13  );
14
15      /*=================================================
16                              按键消抖及状态标记
17      =================================================*/
18      wire [1:0]seq_data;
19      key_control key_control
20      (
```

```
21          .clk                    (  clk              ),
22          .key_det            (  key_det              ),
23          .key_in             (  key_in           ),
24          .key_det_led        (  key_det_led          ),
25          .key_in_led         (  key_in_led[1:0]      ),
26          .seq_data           (  seq_data         )
27      );
28
29      /*=================================================
30                          序列检测
31      =================================================*/
32      wire [3:0]data;
33
34      seq_det seq_det (
35          .clk                (  clk              ),
36          .key_det_led        (  ~key_det_led),//检测状态标记
37          .key_in_led         (  seq_data         ),//待检测序列
38          .bm                 (  bm               ),//输入序列
39          .data               (  data             )
40      );
41      /*=================================================
42                          时钟分频
43      =================================================*/
44      wire clk_1khz;
45      div_clk div_clk(
46          .clk                (  clk          ),
47          .clk_1khz       (  clk_1khz     )
48      );
49      /*=================================================
50                          数码管显示
51      =================================================*/
52      reg  [1:0]sel=0;
53      wire [3:0]dig0;
54      wire [7:0]smg0;
55
56      always @(posedge clk_1khz)
57      begin
```

```
58          sel <= `UD sel+1'b1;
59      end
60      // Digital Tube 0 output
61      seq_control seq_control_0(
62          .sel(sel),
63          .key(data),
64          .dig(dig0),
65          .smg(smg0)
66      );
67      // Digital Tube 1 output
68      wire [3:0]dig1;
69      wire [7:0]smg1;
70      seq_control seq_control_1
71      (
72          .sel(sel),
73          .key(4'd0),
74          .dig(dig1),
75          .smg(smg1)
76      );
77      // Digital Tube 2 output
78      wire [3:0]dig2;
79      wire [7:0]smg2;
80      seq_control seq_control_2
81      (
82          .sel(sel),
83          .key(4'd0),
84          .dig(dig2),
85          .smg(smg2)
86      );
87      wire [3:0]dig3;
88      wire [7:0]smg3; // Digital Tube 3 output
89      seq_control seq_control_3
90      (
91          .sel(sel),
92          .key(4'd0),
93          .dig(dig3),
94          .smg(smg3)
95      );
96      // display by Digital Tube
97      always @(posedge clk_1khz)
98      begin
99          if(sel==2'b00)
100             dig <= `UD dig0;
```

```
        else if(sel==2'b01)
            dig <= `UD dig1;
        else if(sel==2'b10)
            dig <= `UD dig2;
        else if(sel==2'b11)
            dig <= `UD dig3;
    end

    always @(posedge clk_1khz)
    begin
        if(sel==2'b00)
            smg <= `UD smg0;
        else if(sel==2'b01)
            smg <= `UD smg1;
        else if(sel==2'b10)
            smg <= `UD smg2;
        else if(sel==2'b11)
            smg <= `UD smg3;
    end
    endmodule
```

5.4.3按键 LED 控制模块

```
`timescale 1ns / 1ps
`define UD #1
module key_control
(
    input          clk,    // input clock
    input          key_det,     // KEY 8 input
    input [1:0]    key_in,    // KEY[2:0] input
    output reg     key_det_led,  // LED3 control sigal
    output reg [1:0]       key_in_led,// LED[1:0] control signal
    output reg [1:0]       seq_data   // Sequence to be detected
);

//==按键消斗=======================================

    wire [1:0] key_out;
    wire       key_det_out;
    btn_deb#(
        .BTN_WIDTH   (  4'd2               ) , //parameter
BTN_WIDTH = 4'd8
        .MS_20       (  20'd800_000         )
```

```
19      ) btn_deb_key
20      (
21          .clk          (  clk                    ),//input
        clk,
22          .btn_in       (  key_in                 ),//input [BTN_WIDTH-
    1:0] btn_in,
23
24          .btn_deb      (  key_out                ) //output reg
    [BTN_WIDTH-1:0]btn_deb
25      );
26
27      btn_deb#(
28          .BTN_WIDTH    (  4'd1                   ) , //parameter
    BTN_WIDTH = 4'd8
29          .MS_20        (  20'd800_000             )
30      ) btn_deb_det
31      (
32          .clk          (  clk                    ),//input
        clk,
33          .btn_in       (  key_det                ),//input
    [BTN_WIDTH-1:0] btn_in,
34
35          .btn_deb      (  key_det_out            ) //output reg
    [BTN_WIDTH-1:0]btn_deb
36      );
37
38      reg [1:0]key_out_reg;
39      reg key_det_out_reg;
40
41      always @(posedge clk)
42      begin
43          key_out_reg <= `UD key_out;
44          key_det_out_reg<= `UD key_det_out;
45          key_det_led <= `UD ~key_8_flag;//LED 高电平熄灭
46      end
47
48      reg key_8_flag = 1'b0;
49      always @(posedge clk)
50      begin
51          if(!key_det_out && key_det_out_reg)
52              key_8_flag <= `UD ~key_8_flag;
53          else
```

```
54              key_8_flag <= `UD key_8_flag;
55      end
56
57      reg [1:0]key_flag=2'b00;
58      always @(posedge clk)
59      begin
60          if(~(key_det_led || key_8_flag)) //检测结束，序列复位
61              key_flag[0] <= `UD 1'b0;
62          else if(!key_out[0] && key_out_reg[0])
63              key_flag[0] <= `UD ~key_flag[0];
64          else
65              key_flag[0] <= `UD key_flag[0];
66      end
67
68      always @(posedge clk)
69      begin
70          if(~(key_det_led || key_8_flag))//检测结束，序列复位
71              key_flag[1] <= `UD 1'b0;
72          else if(!key_out[1] && key_out_reg[1])
73              key_flag[1] <= `UD ~key_flag[1];
74          else
75              key_flag[1] <= `UD key_flag[1];
76      end
77
78      always @(posedge clk)
79      begin
80          key_in_led <= `UD ~key_flag;//LED高电平熄灭
81      end
82
83      //seq_data
84      always @(posedge clk)
85      begin
86          if(key_8_flag)
87              seq_data <= `UD ~key_in_led;
88      end
89  endmodule
```

按键消抖

```
1   `timescale 1ns / 1ps
2   `define UD #1
3   module btn_deb#(
```

```
parameter BTN_WIDTH = 4'd8,
parameter MS_20 = 20'd800_000
)
(
input clk,//40MHz
input [BTN_WIDTH-1:0] btn_in,
output reg [BTN_WIDTH-1:0] btn_deb
);

//=============================================================
==================
reg [19:0] time_cnt= 20'd0;
always@(posedge clk)
begin
if(time_cnt == MS_20 - 1'b1)
time_cnt <= 20'd0;
else
time_cnt <= time_cnt + 1'd1;
end
always @(posedge clk)
begin
if(time_cnt == MS_20 - 1'b1)
btn_deb <= btn_in;
end
endmodule
```

5.4.4序列检测模块设计

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////
////////////////
// Company:Meyesemi
// Engineer: Will
//
// Create Date: 2022-08-12 20:31
// Design Name:
// Module Name:
// Project Name:
// Target Devices: Pango
// Tool Versions:
// Description:
//
// Dependencies:
```

```
15  //
16  // Revision:
17  // Revision 1.0 - File Created
18  // Additional Comments:
19  //
20  //////////////////////////////////////////////////////////////////////////////////
21  `define UD #1
22  module seq_det
23  (
24      input             clk,
25      input             key_det_led,//检测状态标记
26      input [1:0]       key_in_led,//待检测序列
27      input [3:0]       bm,//输入序列
28      output reg [3:0]  data
29  );
30
31      // 4bit data detecte 2 bit sequence, we need compare three numbers
32      reg [2:0]flag=0;
33      reg      det_en=0;
34
35      reg key_det_led_1d=0;
36      always @(posedge clk)
37      begin
38          key_det_led_1d <= `UD key_det_led;
39      end
40
41      always @(posedge clk)
42      begin
43          if(~key_det_led_1d && key_det_led) //检测按键触发后才进入
44              det_en <= `UD 1'b1;
45      end
46
47      always @(posedge clk)
48      begin
49          if(key_det_led && det_en)
50          begin
```

```
51              flag[0] <= `UD (bm[3:2]==key_in_led);
52              flag[1] <= `UD (bm[2:1]==key_in_led);
53              flag[2] <= `UD (bm[1:0]==key_in_led);
54          end
55      end
56
57      always @(posedge clk)
58      begin
59          if(~key_det_led && key_det_led_1d)//检测结束统计结果
60              data <= `UD flag[2] + flag[1] + flag[0];
61      end
62  endmodule
```

5.5实验现象

实验步骤：

1、调整输入序列，更改拨码开关的输入值（SW[3：0]）；

2、调整固定序列，通过轻触按键改变 LED 状态（LED[1:0]）；

3、按下轻触按键 KEY4，进入检测，查看数码管显示的统计结果；

4、按下轻触按键 KEY4，退出检测，重新执行前面三个步骤；

实验现象举例：

当 SW[3:0]=4’b1010;LED[2:0]=2’b01 时，按下 Key4 后数码管显示数字 1；

当 SW[3:0]=4’b1010;LED[2:0]=3’b10 时，按下 Key4 后数码管显示数字 2；

6.密码锁

6.1实验目的

利用 MES2K 板卡上的按键，拨码开关以及数码管实现一种简单的密码锁；

6.2实验要求

利用拨码开关设置密码，使用按键输入开锁密码。当开锁密码与设定密码相同时开锁成功，数码管显示 8888，密码错误时显示 7777。

SW1- SW4 设置 2 位数密码，每两位设置一位密码，BM[0:1]设置第一位对应 BM1 和 BM2，BM[2:3]设置第二位。所以密码是由 0，1，2，3 组成的四位数。

KEY1-KEY2 作为密码输入，按键按一下数字加 1，数字由数码管显示，数字在 0，1，2，3 中循环。

K4 作为确认按键，按下 K4，输入的密码与设置的密码比对，如相同则显示 8888，若不同则显示 7777。 按下 K3 清零，按下后数码管显示 0000，可以重新输密码。

6.3实验原理

原理上与前一个章节的序列检测是类似的，在前一个实验的基础上有了一些延伸；

序列对比的位宽发生改变，单个数据占 2bit，一个按键控制输入密码数据设置为 2bit 即可；对比与重新开始在此实验用两个按键实现，一个确认对比，一个清空结果；

6.4实验源码（完整源码查看 demo 源文件）

根据需求我们需要如下三个子模块：

①按键控制模块；

1、对 4 个按键输入信号均做消抖处理，2、KEY4 和 KEY3 取下降沿输出，3、KEY[2：1]以下降沿来变更各自的输入密码，每次数字加 1（0～3 循环，2bit 即可）

②数码管显示模块；

显示状态有两种：

密码输入状态：

1、上电默认状态； 2、KEY3 下降沿触发进入重置状态；3、实时显示 2 位输入密码；

密码验证状态：

1、KEY4 下降沿触发进入；

2、显示密码验证结果，正确则显示 8888，错误则显示 7777；

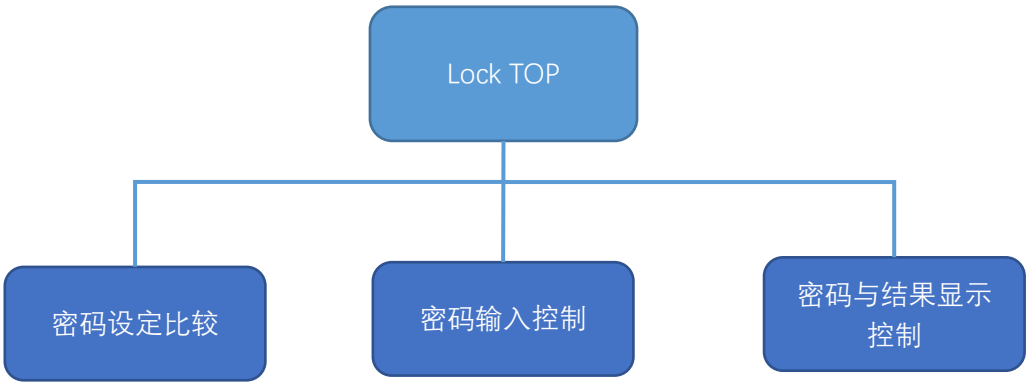
③密码验证模块；

KEY4 下降沿触发使能工作； KEY4 下降沿触发所存输入密码，并与拨码开关设置的密码进行比较；

输出密码比较结果，提供个数码管显示模块。

6.4.1顶层模块设计

顶层模块与上述三个模块之间的关系如下图：



输入输出信号如下表：

信号	位宽	方向	描述
clk	1	输入	外部输入时钟，输入时钟为 40MHz
key	2	输入	轻触按键输入信号， K1~K2 输入
enter	1	输入	密码确认比对信号， K4 输入
init	1	输入	密码确认重新输入信号， K3 输入
sw	4	输入	密码设置输入信号， SW1~4 输入
smg	8	输出	密码对比结果显示数码管段选信号输出
dig	4	输出	密码对比结果显示数码管位选信号输出

Module 设计如下：

```
`timescale 1ns / 1ps
`define UD #1
module lock_top(
    input              clk,
    input  [1:0]    key,
    input              enter,
    input              init,
    input  [3:0]    sw,

    output [7:0]   smg,
    output [3:0]   dig
);

    wire               enter_trig;
    wire               init_trig;
    wire [3:0]        ctrl;
    wire               com_result;

    key_ctl key_ctl(
        .clk             (  clk         ),//input
clk,
        .key             (  key         ),//input      [1:0] key,
        .enter         (  enter       ),//input           enter,
        .init            (  init        ),//input
init,

        .enter_trig    (  enter_trig  ),//output
enter_trig,
        .init_trig       (  init_trig   ),//output
init_trig,
        .ctrl              (  ctrl        ) //output     [3:0]
ctrl
    );

    compare compare(
        .clk           (  clk         ),//input        clk,
        .sw            (  sw          ),//input [3:0]  sw,
        .ctrl          (  ctrl        ),//input [3:0]  ctrl,
        .enter_trig      (  enter_trig  ),//input
enter_trig,
        .com_result    (  com_result  ) //output       com_result
    );

```

```
38      seq_display seq_display(
39          .clk                ( clk         ),//input
    clk,
40          .enter_trig     ( enter_trig  ),//input
    enter_trig,
41          .init_trig        ( init_trig   ),//input
    init_trig,
42          .com_result     ( com_result  ),//input
    com_result,
43          .ctrl                 ( ctrl        ),//input
    [3:0]   ctrl,
44
45          .smg            ( smg         ),//output reg  [7:0]
    smg,
46          .dig              ( dig         ) //output reg  [3:0]
    dig
47      );
48
49  endmodule
```

6.4.2按键控制设计

Module 设计如下：

```
1   `timescale 1ns / 1ps
2   `define UD #1
3   module key_ctl(
4       input           clk,
5       input     [1:0] key,
6       input           enter,
7       input           init,
8
9       output          enter_trig,
10      output          init_trig,
11      output    [3:0] ctrl
12  );
13
14      wire [3:0] btn_deb;
15      // 按键消抖
16      btn_deb#(
17          .BTN_WIDTH   ( 4'd4               ) , //parameter
    BTN_WIDTH = 4'd8
18          .MS_20           ( 20'd800_000          )
19      ) btn_deb_key
```

```
20      (
21          .clk             (  clk                  ),//input
        clk,
22          .btn_in      (  {enter,init,key}     ),//input
    [BTN_WIDTH-1:0] btn_in,
23          .btn_deb   (  btn_deb              ) //output reg
    [BTN_WIDTH-1:0]btn_deb
24      );
25
26      reg [1:0]  key1_push_cnt=2'd0;
27      reg [1:0]  key2_push_cnt=2'd0;
28
29      reg btn1_deb_1d,btn2_deb_1d;
30      reg enter_deb_1d,init_deb_1d;
31
32      assign enter_trig = ~btn_deb[3] & enter_deb_1d;
33      assign init_trig = ~btn_deb[2] & init_deb_1d;
34
35      always @(posedge clk)
36      begin
37          btn1_deb_1d  <= `UD btn_deb[0];
38          btn2_deb_1d  <= `UD btn_deb[1];
39          init_deb_1d  <= `UD btn_deb[2];
40          enter_deb_1d <= `UD btn_deb[3];
41      end
42
43      always @(posedge clk)
44      begin
45          if(~btn_deb[2] & init_deb_1d)
46              key1_push_cnt <= `UD 2'd0;
47          else if(~btn_deb[0] & btn1_deb_1d)
48          begin
49              key1_push_cnt <= `UD key1_push_cnt + 2'd1;
50          end
51      end
52
53      always @(posedge clk)
54      begin
55          if(~btn_deb[2] & init_deb_1d)
56              key2_push_cnt <= `UD 2'd0;
57          else if(~btn_deb[1] & btn2_deb_1d)
58          begin
59              key2_push_cnt <= `UD key2_push_cnt + 2'd1;
```

```
60          end
61      end
62
63      assign ctrl = {key2_push_cnt,key1_push_cnt};
64
65  endmodule
```

6.4.3按键消抖设计

```
63  `timescale 1ns / 1ps
64  `define UD #1
65  module btn_deb#(
66      parameter                   BTN_WIDTH = 4'd8,
67      parameter                   MS_20 = 20'd800_000
68  )
69  (
70      input                       clk,//40MHz
71      input       [BTN_WIDTH-1:0] btn_in,
72
73      output reg [BTN_WIDTH-1:0] btn_deb
74  );
75
76  //==========================================================
    ==================
77      reg [19:0] time_cnt= 20'd0;
78      always@(posedge clk)
79      begin
80          if(time_cnt == MS_20 - 1'b1)
81              time_cnt <= 20'd0;
82          else
83              time_cnt <= time_cnt + 1'd1;
84      end
85
86      always @(posedge clk)
87      begin
88          if(time_cnt == MS_20 - 1'b1)
89              btn_deb <= btn_in;
90      end
91
92  endmodule
```

6.4.4对比模块设计

Module 设计

```
`timescale 1ns / 1ps
`define UD #1
module compare(
    input        clk,
    input [3:0]  sw,
    input [3:0]  ctrl,
    input        enter_trig,

    output       com_result
);

//=============================================================
==============

    //锁存当前的输入密码;
    reg [3:0] ctrl_1d;
    always @(posedge clk)
    begin
        if(enter_trig)
            ctrl_1d <= `UD ctrl;
    end

    assign com_result = (ctrl_1d == sw);

endmodule
```

6.4.5显示模块设计

此模块设计需要注意数码管显示的两种模式：密码输入模式与密码对比结果显示模式；两种模式的切换由 enter_trig 与 init_trig 触发进入；

对于数码管的显示控制模块这里就不重复描述了；

6.5实验现象

验证步骤：

1、 调整输入序列，更改拨码开关的输入值（SW[3：0]）；

2、 调整固定序列，通过轻触按键调整输入密码，数码管实时显示输入密码；

3、 按下轻触按键 KEY3，触发进行密码比对，并且数码管显示比对结果；

4、 按下轻触按键 KEY4，进入重新输入密码状态，重新执行前面三个步骤；

实验现象

当 SW[3:0]=8’b1010;当输入密码状态时显示 0022 时，按下 Key3 后数码管显示数字 8888；当输入密码状态时显示不是 4 个 2 时 2，按下 Key3 后数码管显示数字 7777；按下 Key4 后重新调整密码，进入输入密码状态；

7.数字钟

7.1实验目的

设计一个具有计时功能和校时功能的数字时钟。

7.2实验要求

数码管显示小时和分钟，秒钟用 LED 闪烁标识。

三个按键用于时钟校准。

K1 用于切换正常计时，校准小时和分钟

K2 用于时钟的“+”

K3 用于时钟的“-”

校准相应的刻度，该数码管闪烁。

7.3实验原理

从上述的实验要求分析可得到此数字钟我们实现过程中要注意两个功能点：

1、计时显示功能：LED 闪烁显示秒钟读秒，数码管右侧两位显示分钟计时，数码管左侧两位显示时钟计时；

此功能的实现由两个细节功能实现：①1S 计时控制，与前面的实验中需要计时功能模块实现方式一致，注意此处计时的周期为 1S 即可；②计时过程中进位控制；进位控制有四处需要进位：

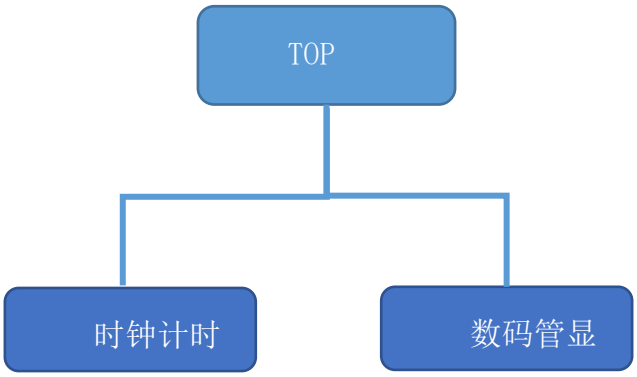
秒 进 分	LED 灯亮灭一次计数为 1，亮灭的一个周期为 1S，当 LED 亮灭 60 次时分钟计数个位加 1；
分$_{个位}$ 进 分$_{十位}$	当个位计数到 9 时，秒到分再次进位时，十位需要加一，个位置位为 0；
分进时	1 小时=60 分钟；当分钟计时为 59，并且秒到分再次进位时，时钟计数个位加 1；分钟计数置位为 0；
时$_{个位}$ 进 时$_{十位}$	显示为 24 小时制，当时钟计数十位小于 2 时，时钟个位个位计数到 9 时，分到时再次进位时，十位加 1；
24 小时制计满时归零	当时钟计数十位等于 2 时，时钟个位个位计数到 3 时，分到时再次进位时，时，分，秒三部分计数均归零；

2、计时校准功能：通过对应按键控制调整分钟计时与时钟计时，调整的过程中对应位需要闪烁；

此项功能中注意两点：①调整对应位是，数码管该位进行闪烁；②调整时注意进位；

基于上述分析我们将项目分成两个部分：

1. 时钟计时与控制。
2. 数码管显示控制。



7.4实验源码（完整源码查看 demo 源文件）

7.4.1顶层设计

输入输出信号如下表：

信号	位宽	方向	描述
clk	1	输入	外部输入时钟，MES2KG 板卡输入时钟为 40MHz
key	3	输入	时钟校准信号输入（轻触按键）
led	1	输出	时钟秒钟跳动显示（LED 灯闪烁一次，为 1S）
smg	8	输出	数码管段选输出
dig	4	输出	数码管位选输出

```
1  `timescale 1ns / 1ps
2  `define UD #1
3  module top_watch
4  (
5      input clk,
6      input [2:0]key,
7      output led,
```

```
8       output reg [3:0]dig,
9       output reg [7:0] smg
10  );
11
12    parameter CLK_FRE = 26'd40_000_000;
13  /*================================================
14                          复位信号的产生
15  ================================================*/
16  reg [4:0] rstn_cnt=0;
17  always @(posedge clk)
18  begin
19      if(rstn_cnt==5'h1f)
20              rstn_cnt <= `UD rstn_cnt;
21      else
22              rstn_cnt <= `UD rstn_cnt + 1'b1;
23  end
24
25  wire rstn;
26  assign rstn = rstn_cnt[4];
27  /*================================================
28                      数字时钟的产生和控制
29  ================================================*/
30  wire [3:0] hour_h_o,hour_l_o,minutes_h_o,minutes_l_o;
31  wire [2:0] state_flag;
32  watch_data_gen #(
33      .CLK_FRE    (CLK_FRE    )
34  ) u_watch_data_gen
35  (
36      .clk            (clk            ), //input clk,
37      .rstn           (rstn           ), //input rstn,
38      .key            (key            ), //input [2:0]key,
39      .hour_h_o       (hour_h_o       ), //output reg [3:0]hour_h_o,
40      .hour_l_o       (hour_l_o       ), //output reg [3:0]hour_l_o,
41      .minutes_h_o(minutes_h_o), //output reg [3:0]minutes_h_o,
42      .minutes_l_o(minutes_l_o), //output reg [3:0]minutes_l_o,
43      .second_led     (led            ), //output reg second_led,
44      .state_flag (state_flag )  //output reg [2:0]state_flag
45  );
46  /*================================================
47                                  时钟分频
```

```
48  ================================================*/
49  wire clk_100khz;
50  div_clk #(
51      .CLK_FRE    (CLK_FRE    )
52  )  div_clk
53  (
54      .clk           (clk),
55      .clk_100khz (clk_100khz)
56  );
57  /*================================================
58                      数码管显示
59  ================================================*/
60  reg  [1:0]sel=0;
61  wire [3:0]dig0;
62  wire [7:0]smg0;
63
64  always @(posedge clk_100khz)
65  begin
66      sel <= `UD sel+1'b1;
67  end
68
69  seq_control seq_control_0
70  (
71      .clk(clk),
72      .sec_en(led),
73      .control_dig(state_flag),
74      .sel(2'd0),
75      .key(hour_h_o),
76      .dig(dig0),
77      .smg(smg0)
78  );
79
80  wire [3:0]dig1;
81  wire [7:0]smg1;
82
83  seq_control seq_control_1
84  (
85      .clk(clk),
86      .sec_en(led),
87      .control_dig(state_flag),
88      .sel(2'd1),
89      .key(hour_l_o),
```

```
90      .dig(dig1),
91      .smg(smg1)
92  );
93
94  wire [3:0]dig2;
95  wire [7:0]smg2;
96
97  seq_control seq_control_2
98  (
99      .clk(clk),
100     .sec_en(led),
101     .control_dig(state_flag),
102     .sel(2'd2),
103     .key(minutes_h_o),
104     .dig(dig2),
105     .smg(smg2)
106 );
107
108 wire [3:0]dig3;
109 wire [7:0]smg3;
110
111 seq_control seq_control_3
112 (
113     .clk(clk),
114     .sec_en(led),
115     .control_dig(state_flag),
116     .sel(2'd3),
117     .key(minutes_l_o),
118     .dig(dig3),
119     .smg(smg3)
120 );
121
122
123 always @(posedge clk_100khz)
124 begin
125     if(sel==2'b00)
126             dig <= `UD dig0;
127     else if(sel==2'b01)
128             dig <= `UD dig1;
129     else if(sel==2'b10)
130             dig <= `UD dig2;
131     else if(sel==2'b11)
132             dig <= `UD dig3;
```

```
133 end
134 always @(posedge clk_100khz)
135 begin
136     if(sel==2'b00)
137             smg <= `UD smg0;
138     else if(sel==2'b01)
139             smg <= `UD smg1;
140     else if(sel==2'b10)
141             smg <= `UD smg2;
142     else if(sel==2'b11)
143             smg <= `UD smg3;
144 end
145 endmodule
146
```

7.4.2数字时钟产生与控制模块设计

在此模块中我们要实现前面描述的两个主要的功能点：计时与控制；

输入输出信号如下表：

信号	位宽	方向	描述
clk	1	输入	外部输入时钟,MES2KG 板卡输入时钟为 40MHz
rstn	1	输入	外部输入复位信号,
key	3	输入	时钟校准信号输入（轻触按键）
hour_h_o	4	输出	时钟高位计数
hour_l_o	4	输出	时钟低位计数
minutes_h_o	4	输出	分钟高位计数
minutes_l_o	4	输出	分钟低位计数
second_led	1	输出	时钟秒钟跳动显示（LED 灯闪烁一次，为 1S）
state_flag	3	输出	数码管段选输出

Module 设计的关键点如下（完整 module 查看源文件）：

```
1  `timescale 1ns / 1ps
2  `define UD #1
3  module watch_data_gen #(
4      parameter CLK_FRE = 26'd40_000_000
5  )
6  (
7      input clk,
```

```
8       input rstn,
9       input [2:0]key,
10      output reg [3:0]hour_h_o,
11      output reg [3:0]hour_l_o,
12      output reg [3:0]minutes_h_o,
13      output reg [3:0]minutes_l_o,
14      output reg second_led,
15      output reg [2:0]state_flag
16  );
17
18  /*================================================
19                          基准的产生
20  ================================================*/
21  // 1s= 25ns * 40_000_000;
22  reg [25:0]second_cnt;
23  always @(posedge clk)
24  begin
25      if(second_cnt==CLK_FRE-1'b1)
26              second_cnt <= `UD 26'd0;
27      else
28              second_cnt <= `UD second_cnt + 1'b1;
29  end
30  //每个 1s 闪烁一次
31  always @(posedge clk)
32  begin
33      if(!rstn)
34              second_led <= `UD 1'b0;
35      if(second_cnt==(CLK_FRE>>1)-1'b1)
36              second_led <= `UD 1'b1;
37      else if(second_cnt==CLK_FRE-1'b1)
38              second_led <= `UD 1'b0;
39  end
40
41  reg [3:0]minutes_h;
42  reg [3:0]minutes_l;
43  reg [3:0]hour_h;
44  reg [3:0]hour_l;
45
46  reg [3:0]minutes_l_fix=0;
47  reg [3:0]minutes_h_fix=0;
48  reg [3:0]hour_l_fix=0;
```

```
reg [3:0]hour_h_fix=0;
/*==================================================
                        校准逻辑产生
==================================================*/
wire [2:0]key_out;

btn_deb #(
   .BT_WIDTH(4'd3),
   .CLK_FRE (CLK_FRE)
)
u_btn_deb
(
    .clk    (clk),
    .btn_in (key),
    .btn_out(key_out)
);
/*==================================================
    //key[0] -> k1 ;用于校准标记

    key_cnt = 3'd0 用于正常显示;

    key_cnt = 3'd1 用于分钟低位校准;

    key_cnt = 3'd2 用于分钟高位校准;

    key_cnt = 3'd3 用于时钟低位校准;

    key_cnt = 3'd4 用于时钟高位校准;
==================================================*/
reg [2:0]key_out_reg=3'd0;
always @(posedge clk)
begin
    key_out_reg <= `UD key_out;
end

reg [2:0]key_cnt=3'd0;
always @(posedge clk)
begin
    if(key_cnt==3'd4 && (!key_out[0] && key_out_reg[0]))
           key_cnt <= `UD 3'd0;
    else if(!key_out[0] && key_out_reg[0])
```

```
85              key_cnt <= `UD key_cnt + 1'b1;
86      end
87      /*================================================
88                          key[1] 用于"+"; key[2] 用"-"
89      ================================================*/
90      always @(posedge clk)
91      begin
92          if(key_cnt==3'd0)//校准前将分钟低位和输出值保持一致
93                  minutes_1_fix <= `UD minutes_1;
94          else if(!key_out[1] && key_out_reg[1] && key_cnt==3'd1 &&
        minutes_1_fix == 4'd9)//当处于分钟校准状态时，按下"+"按键,且此时
        校准值已经为 9 时，再按下按键，则校准值变为 0
95                  minutes_1_fix <= `UD 4'd0;//"+"
96          else if(!key_out[2] && key_out_reg[2] && key_cnt==3'd1 &&
        minutes_1_fix == 4'd0)//当处于分钟校准状态时，按下"-"按键,且此时
        校准值已经为 0 时，再按下按键，则校准值变为 9
97                  minutes_1_fix <= `UD 4'd9;//"-"
98          else if(!key_out[1] && key_out_reg[1] && key_cnt==3'd1)//当处
        于分钟低位校准状态时，按下"+"按键,校准数值加 1;
99                  minutes_1_fix <= `UD minutes_1_fix + 1'b1;//"+"
100         else if(!key_out[2] && key_out_reg[2] && key_cnt==3'd1)//当处
        于分钟低位校准状态时，按下"-"按键,校准数值减 1;
101                 minutes_1_fix <= `UD minutes_1_fix - 1'b1;    //"-"
102     end
103
104
105     always @(posedge clk)
106     begin
107         if(key_cnt!=3'd2)//校准前数据和输出值保持一致
108                 minutes_h_fix <= `UD minutes_h;
```

```
109     else if(!key_out[1] && key_out_reg[1] && key_cnt==3'd2 &&
    minutes_h_fix == 4'd5)//当处于校准状态时，按下"+"按键,且此时校准
    值已经为 5 时，再按下按键，则校准值变为 0
110             minutes_h_fix <= `UD 4'd0;//"+"
111     else if(!key_out[2] && key_out_reg[2] && key_cnt==3'd2 &&
    minutes_h_fix == 4'd0)//当处于分钟校准状态时，按下"-"按键,且此时
    校准值已经为 0 时，再按下按键，则校准值变为 5
112             minutes_h_fix <= `UD 4'd5;//"-"
113     else if(!key_out[1] && key_out_reg[1] && key_cnt==3'd2)//当处
    于校准状态时，按下"+"按键,按下"+"按键,校准数值加 1;
114             minutes_h_fix <= `UD minutes_h_fix + 1'b1;//"+"
115     else if(!key_out[2] && key_out_reg[2] && key_cnt==3'd2)//当处
    于校准状态时，按下"-"按键,校准数值减 1;
116             minutes_h_fix <= `UD minutes_h_fix - 1'b1;//"-"
117 end
118
119
120 always @(posedge clk)
121 begin
122     if(key_cnt!=3'd3 )//校准前数据赋值为 0
123             hour_l_fix <= `UD 4'd0;
124     else if(!key_out[1] && key_out_reg[1] && key_cnt==3'd3 &&
    hour_h_o != 4'd2 && hour_l_fix==4'd9)//当处于校准状态时，按下"+"
    按键,且此时小时的高位不为 2 且校准值已经为 9 时，再按下按键，则校准
    值变为 0
125             hour_l_fix <= `UD 4'd0;//"+"
126     else if(!key_out[1] && key_out_reg[1] && key_cnt==3'd3 &&
    hour_h_o == 4'd2 && hour_l_fix==4'd3)//当处于校准状态时，按下"+"
```

按键,且此时小时的高位为 2 且校准值已经为 3 时，再按下按键，则校准值变为 0

```
127             hour_l_fix <= `UD 4'd0;//"+"
128     else if(!key_out[2] && key_out_reg[2] && key_cnt==3'd3 &&
```

hour_h_o != 4'd2 && hour_l_fix==4'd0)//当处于校准状态时，按下"-"按键,且此时小时的高位不为 2 时，校准位为 0，再按下按键，则校准值变为,9

```
129             hour_l_fix <= `UD 4'd9;//"-"
130     else if(!key_out[2] && key_out_reg[2] && key_cnt==3'd3 &&
```

hour_h_o == 4'd2 && hour_l_fix==4'd0)//当处于校准状态时，按下"-"按键,且此时小时的高位为 2 时，校准位为 0，再按下按键，则校准值变为 3

```
131             hour_l_fix <= `UD 4'd3;//"-"
132     else if(!key_out[1] && key_out_reg[1] && key_cnt==3'd3)//当处
```

于校准状态时，按下"+"按键,按下"+"按键,校准数值加 1;

```
133             hour_l_fix <= `UD hour_l_fix + 1'b1;//"+"
134     else if(!key_out[2] && key_out_reg[2] && key_cnt==3'd3)//当处
```

于校准状态时，按下"-"按键,校准数值减 1;

```
135             hour_l_fix <= `UD hour_l_fix - 1'b1;
136 end
137
138
139 always @(posedge clk)
140 begin
141     if(key_cnt!=3'd4)//校准前数据和输出值保持一致
142             hour_h_fix <= `UD hour_h;
143     else if(!key_out[1] && key_out_reg[1] && key_cnt==3'd4 &&
```

hour_h_fix == 4'd2)//当处于校准状态时，按下"+"按键,且此时小时的高位为 2，再按下按键，则校准值变为 0

```
144             hour_h_fix <= `UD 4'd0;//"+"
145     else if(!key_out[2] && key_out_reg[2] && key_cnt==3'd4 &&
    hour_h_fix == 4'd0)//当处于校准状态时，按下"-"按键,且此时小时的高
    位为 0，再按下按键，则校准值变为 2
146             hour_h_fix <= `UD 4'd2;//"-"
147     else if(!key_out[1] && key_out_reg[1] && key_cnt==3'd4 )//当
    处于校准状态时，按下"+"按键,按下"+"按键,校准数值加 1;
148             hour_h_fix <= `UD hour_h_fix + 1'b1;//"+"
149     else if(!key_out[2] && key_out_reg[2] && key_cnt==3'd4)//当处
    于校准状态时，按下"-"按键,校准数值减 1;
150             hour_h_fix <= `UD hour_h_fix - 1'b1;//"-"

151 end
152 /*==================================================
153                     秒钟的产生:中间值
154 ==================================================*/
155 //秒钟计 60 次 (0~59) 为 1 分钟
156 reg [5:0]second_flag=0;
157 always @(posedge clk)
158 begin
159     if(second_cnt==CLK_FRE-1'b1 && second_flag==6'd59)
160             second_flag <=`UD 6'd0;
161     else if(second_cnt==CLK_FRE-1'b1)
162             second_flag <=`UD second_flag + 1'b1;
163 end
164 /*==================================================
165                     分钟的产生:中间值
166 ==================================================*/
167 //minutes_l gen
168 always @(posedge clk)
169 begin
170     if(!rstn)//初始值为 0
171             minutes_l <= `UD 4'd0;
```

```
172     else if(key_cnt==3'd1)//校准时，分钟低位为校准值
173             minutes_l <= `UD minutes_l_fix;
174     else if(second_cnt==CLK_FRE-1'b1 && second_flag==6'd59 &&
    minutes_l==4'd9)//9分59秒产生进位，低位赋值为0
175             minutes_l <= `UD 4'd0;
176     else if(second_cnt==CLK_FRE-1'b1 && second_flag==6'd59)//60秒
    产生分钟的低位进位
177             minutes_l <= `UD minutes_l +1'b1;
178 end
179 //minutes_h gen
180 always @(posedge clk)
181 begin
182     if(!rstn)
183             minutes_h <= `UD 4'd0;
184     else if(key_cnt==3'd2)
185             minutes_h <= `UD minutes_h_fix;
186     else if(second_cnt==CLK_FRE-1'b1 && second_flag==6'd59 &&
    minutes_h==4'd5  && minutes_l==4'd9)//当为59分59秒的时候产生进
    位，分钟的高位赋值为0;
187             minutes_h <= `UD 4'd0;
188     else if(second_cnt==CLK_FRE-1'b1 && second_flag==6'd59 &&
    minutes_l==4'd9 )//9分59秒的时候产生进位;
189             minutes_h <= `UD minutes_h +1'b1;
190 end
191 /*================================================
192                     小时的产生:中间值
193 ================================================*/
194 always @(posedge clk)
195 begin
196     if(!rstn)
197             hour_l <= `UD 4'd0;
198     else if(key_cnt==3'd3)
199             hour_l <= `UD hour_l_fix;
```

```
200     else if(hour_h!=4'd2 && hour_l==4'd9 && second_cnt==CLK_FRE-
    1'b1 && second_flag==6'd59 && minutes_h==4'd5 &&
    minutes_l==4'd9 )//9:59:59 或者 19:59:59, 下一秒 hour_l 为 0;
201             hour_l <= `UD 4'd0;
202     else if(hour_h==4'd2 && hour_l==4'd3 && second_cnt==CLK_FRE-
    1'b1 && second_flag==6'd59  && minutes_h==4'd5 &&
    minutes_l==4'd9 )//23:59:59;hour_l 为 0;
203             hour_l <= `UD 4'd0;
204     else if(second_cnt==CLK_FRE-1'b1 && minutes_h==4'd5 &&
    minutes_l==4'd9 && second_flag==6'd59)//XX:59:59;hour_l 加 1;
205             hour_l <= `UD hour_l +1'b1;
206 end
207
208 always @(posedge clk)
209 begin
210     if(!rstn)
211             hour_h <= `UD 4'd0;
212     else if(key_cnt==3'd4)
213             hour_h <= `UD hour_h_fix;
214     else if(hour_h==4'd2 && hour_l==4'd3 && second_cnt==CLK_FRE-
    1'b1 && minutes_h==4'd5 && minutes_l==4'd9 &&
    second_flag==6'd59)//当时间刻度为: 23:59:59 ,hour_h 为 0
215             hour_h <= `UD 4'd0;
216     else if(hour_l==4'd9 && second_cnt==CLK_FRE-1'b1 &&
    minutes_h==4'd5 && minutes_l==4'd9 && second_flag==6'd59)//当时间
    刻度为: 09:59:59 or 19:59:59,hour_h 加 1
217             hour_h <= `UD hour_h + 1'b1;
218 end
219
220 /*==================================================
221                             output
222 ==================================================*/
223 /*==================================================
224     //key[0] -> k1 ;用于校准标记
225     key_cnt = 3'd0 用于正常显示;
```

```
226     key_cnt = 3'd1 用于分钟低位校准;

227     key_cnt = 3'd2 用于分钟高位校准;

228     key_cnt = 3'd3 用于时钟低位校准;

229     key_cnt = 3'd4 用于时钟高位校准;
230 ================================================*/
231 always @(posedge clk)
232 begin
233     minutes_1_o <=`UD minutes_1;
234 end
235
236 always @(posedge clk)
237 begin
238     minutes_h_o <=`UD minutes_h;
239 end
240
241 always @(posedge clk)
242 begin
243     hour_1_o <=`UD hour_1;
244 end
245
246 always @(posedge clk)
247 begin
248     hour_h_o <=`UD hour_h;
249 end
250
251 always @(posedge clk)
252 begin
253     state_flag <= `UD key_cnt;
254 end
255
256 endmodule
```

7.4.3时钟分频模块

```
1  `timescale 1ns / 1ps
2  `define UD #1
3  module div_clk #(
4      parameter     CLK_FRE  = 26'd40_000_000
5  )
6  (
```

```
    input clk,
    output  clk_100khz
);
//times =25ns*400=10us=100khz;
parameter CLK_DIV_10US = CLK_FRE/100_000;
reg [8:0]cnt;
always @(posedge clk)
begin
    if(cnt == CLK_DIV_10US - 1'b1)
            cnt<= `UD 9'd0;
    else
            cnt <= `UD cnt + 1'b1;
end

reg flag=1'b0;
always @(posedge clk)
begin
    if(cnt == (CLK_DIV_10US>>1) - 1'b1)
            flag <= `UD 1'b1;
    else if(cnt == CLK_DIV_10US - 1'b1)
            flag <= `UD 1'b0;
end
assign clk_100khz = flag;

endmodule
```

7.4.4数码管显示模块设计

数码管显示模块相比前一个实验需要增加一个功能：当进入校准模式时数码管的校准位需要进行闪烁，故而引入一个 1S 的周期信号，在 1S 时间内 0.5s 正常点亮，0.5s 不点亮使得数码管闪烁；闪烁对应位需要引入按键控制输出的 dig_ctl 信号（前面代码中有描述）；

闪烁控制的模块设计如下：

```
always @(*)
begin
    case(control_dig) // control_dig 为按键控制的校准位信号，对应按键部分的 dig_ctl
        4'd0: //正常显示
            case(sel)
```

```
                2'd0:dig = ~(4'b0001);
                2'd1:dig = ~(4'b0010);
                2'd2:dig = ~(4'b0100);
                2'd3:dig = ~(4'b1000);
            default:dig = ~(4'b0000);
            endcase
        4'd4://时钟高位校准

            case(sel)  // sec_en 为 1s 周期、50%占空比的方波
                2'd0:begin if(sec_en) dig = ~(4'b0001 );else dig
= ~(4'b0000); end
                2'd1:dig = ~(4'b0010);
                2'd2:dig = ~(4'b0100);
                2'd3:dig = ~(4'b1000);
            default:dig = ~(4'b0000);
            endcase
        4'd3://时钟低位校准

            case(sel)
                2'd0:dig = ~(4'b0001);
                2'd1:begin if(sec_en) dig = ~(4'b0010 );else dig
= ~(4'b0000); end
                2'd2:dig = ~(4'b0100);
                2'd3:dig = ~(4'b1000);
            default:dig = ~(4'b0000);
            endcase
        4'd2: //分钟高位校准

            case(sel)
                2'd0:dig = ~(4'b0001);
                2'd1:dig = ~(4'b0010);
                2'd2:begin if(sec_en) dig = ~(4'b0100 );else dig
= ~(4'b0000); end
                2'd3:dig = ~(4'b1000);
            default:dig = ~(4'b0000);
            endcase
        4'd1: //分钟低位校准

            case(sel)
                2'd0:dig = ~(4'b0001);
                2'd1:dig = ~(4'b0010);
                2'd2:dig = ~(4'b0100);
```

```
41                  2'd3:begin if(sec_en) dig = ~(4'b1000 );else dig
   = ~(4'b0000); end
42              default:dig = ~(4'b0000);
43              endcase
44          default:dig = ~(4'b0000);
45      endcase
46  end
```

7.5实验现象

加载后的显示结果为：数码管显示从 00：00 开始，LED1 闪烁（1 次/s）；

按轻触按键 KEY1，进入校准模式，第一次按下 KEY1，进入分钟低位计数校准调节，之后再次按下 KEY1，校准位将会往左移动 1 位，直到校准位为时钟计数高位时，按下 KEY1 将推出校准模式，进入正常计数模式；

在校准模式中按下轻触按键 KEY2 一次，对应校准位加 1，在可计数的最大值时会归 0；

在校准模式中按下轻触按键 KEY3 一次，对应校准位减 1，在减到 0 时会置位为可计数的最大值；