Runber 开发板

实验指导

版本

修改日期	修改版本	修改原因
2020/05/20	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

1.1 实验目的

实现对多 LED 灯的控制;

1.2 实验要求

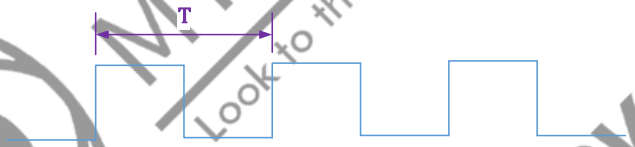
控制 8 个 LED 以 1s 的周期闪烁（0.5s 亮，0.5s 灭）

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 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）。



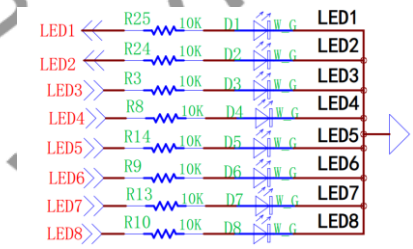
在数字系统中通常关注到时钟的频率,那频率与周期的关系如下：

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

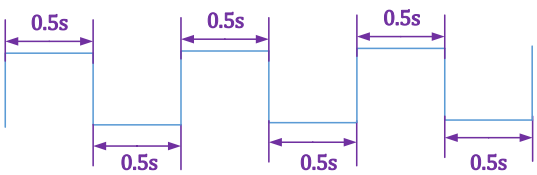
Runber 板卡上有一个 12MHz 的晶振提供时钟给到 GW1N;

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 12MHz 时钟周期为 83.333ns（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）；

$$0.5s = 6000000 * 83.33ns = 6000000 \times T_{12MHz};$$

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 6000000 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

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；

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3   //            Company          : MYMINIEYE Technology CO., Ltd
4   //         Internet site       : www.myminieye.com
5   // WeChat public account       : MYMINIEYE
6   //            Engineer         : Nill
7   //
8   //          Create Date        : 2020-06-04 14:34
9   //          Design Name        : led_light
10  //          Module Name        : led_light
11  //         Project Name        : led_light
12  //       Target Devices        : Gowin@GW1N-UV4LQ144C6/I5
13  //         Tool Versions       : Gowin1.9.2
14  //          Description        :
15  //
16  //         Dependencies        :
17  //
18  //            Revision         : Revision 0.01 - File Created
19  //   Additional Comments       :
20  //
21  //////////////////////////////////////////////////////////////////////////////////
22  `define UD #1
```

`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

```
module led_light(
        input           clk,        // input clock, the frequency is 12MHz
        input           rstn,       // input reset signal, active at low
        output [7:0]    led         // output LED control signal , lignting at high
);
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，Runber 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 6000000，1 个亮灭周期的计数即为 12000000 = 24’hB71B00;所以计数器的位宽为 24 位即可，若兼容更高频率的话需要将位宽设置更大（下方代码中计数器设计为 25bit 位宽是兼容到 50MHz，25‘d25000000 = 25’h17D7840）；此处请结合数字电路中的同步计数器的工作原理分析；

```
1       reg [24:0] led_light_cnt;
2
3       //   time counter
4       always @(posedge clk) // 触发条件；posedge 为上升沿，negedge 为下降沿
5       begin
6           if(!rstn)
7               led_light_cnt <= `UD 25'd0;
8           else if(led_light_cnt == 25'd599_9999)
9               led_light_cnt <= `UD 25'd0;
10          else
11              led_light_cnt <= `UD led_light_cnt + 25'd1;
12      end
```

当计数器计数到 25‘d5999999 时，计数周期包含了从 0 ~ 25’d5999999 的时钟周期，故而总时长时 25‘d6000000 × T_{clk}；硬件输入时钟为 12MHz，所以此计数器的技术周期是 0.5s；

1.4.2.3 led 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）；

```
1    reg [24:0] led_status;
2
3    always @(posedge clk)
4    begin
5        if(!rstn)
6            led_status <= `UD 8'd0;
7        else if(led_light_cnt == 25'd599_9999)
8            led_status <= `UD ~led_status;
9    end
10
11   assign led = led_status;
```

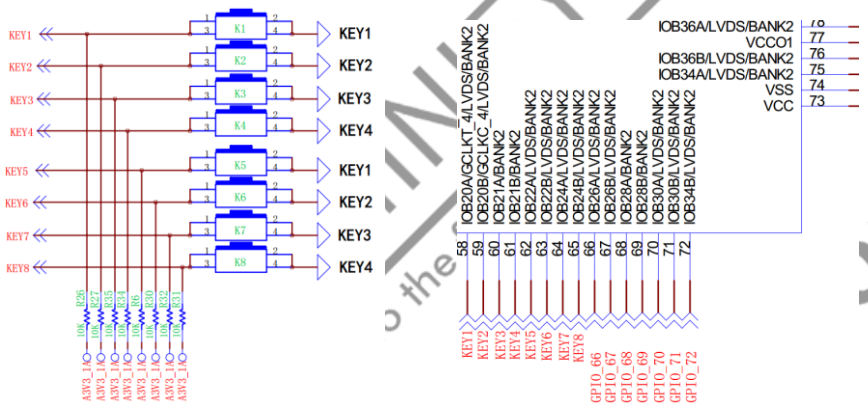
1.4.3 完整的 Module（不含注释头）

```
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    //=======================================================
10       //reg and wire
11       reg [24:0] led_light_cnt;
12       reg [ 7:0] led_status;
13       //   time counter
14       always @(posedge clk)
15       begin
16           if(!rstn)
17               led_light_cnt <= `UD 25'd0;
18           else if(led_light_cnt == 25'd599_9999)
19               led_light_cnt <= `UD 25'd0;
20           else
21               led_light_cnt <= `UD led_light_cnt + 25'd1;
22       end
23
24       // led status change
25       always @(posedge clk)
26       begin
27           if(!rstn)
28               led_status <= `UD 8'd0;
29           else if(led_light_cnt == 25'd599_9999)
30               led_status <= `UD ~led_status;
31       end
32       assign led = led_status;
33
34   endmodule
35
```

1.4.4 硬件管脚分配

Runber 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）：

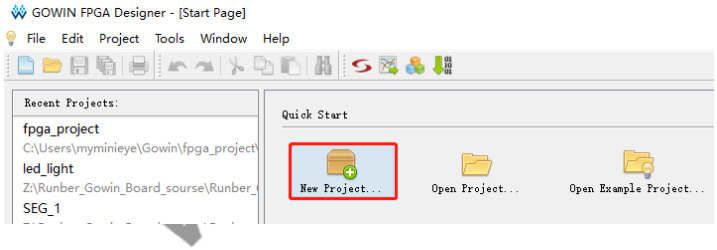
复位设计是低电平有效，而 Runber 板卡上的按键按下时为低电平，松开为高电平，可用按键输入来做复位信号；使用 KEY1 做为复位按键即可；



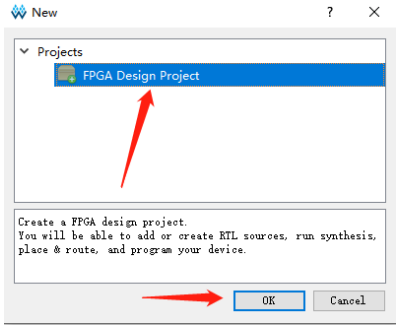
1.5 实验步骤

1.5.1 打开 Gowin 云源软件，创建工程

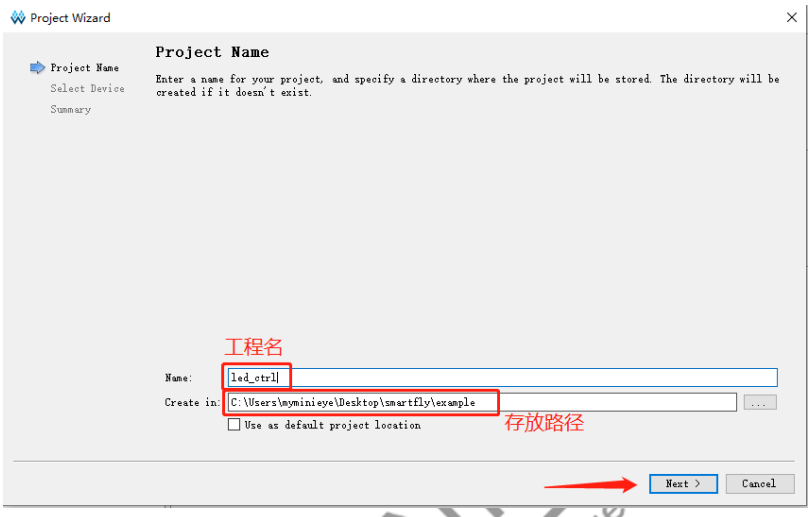
Step1：打开 gowin 软件，单击 New Project....



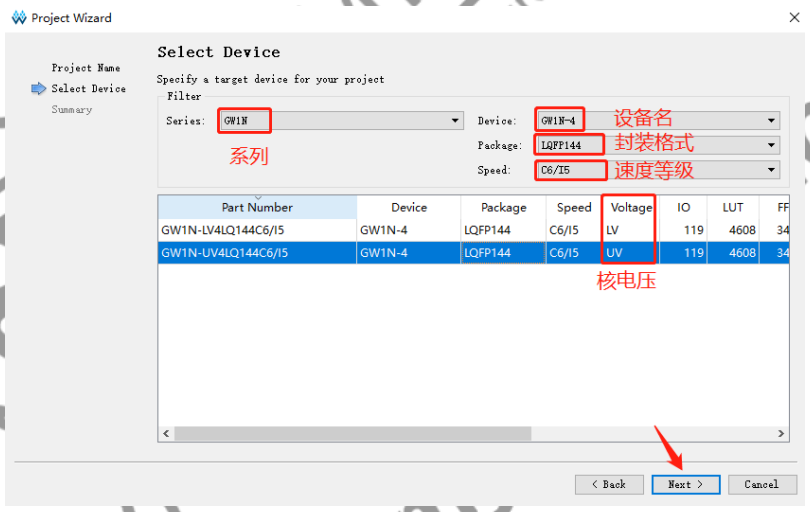
Step2：选择 FPGA Design Project，单击 OK



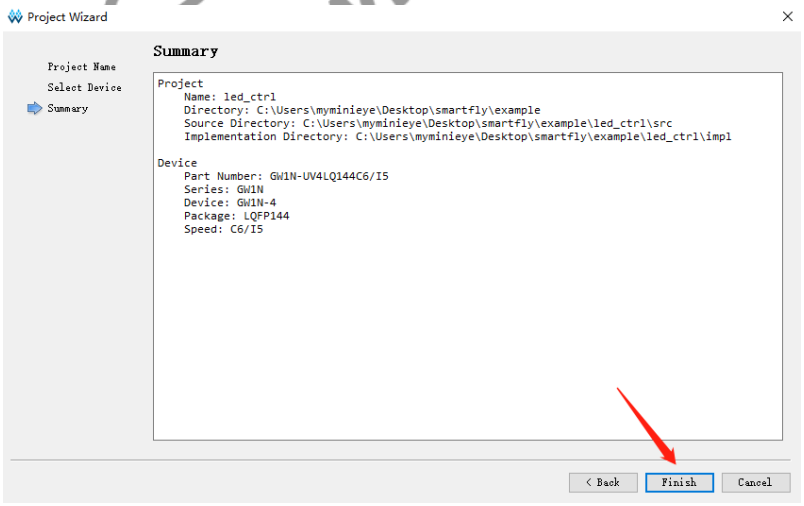
Step3：创建名为 led_ctrl 的工程到对应的文件目录，之后单击 Next



Step4：选择封装格式和速度等级，之后单击 Next

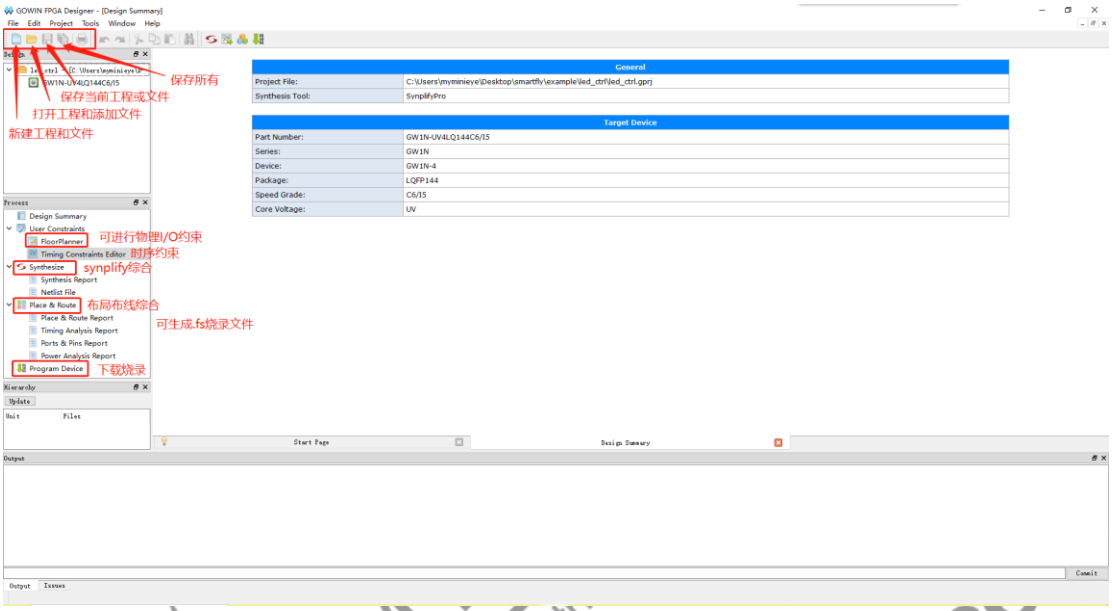


Step5：单击 Finish,完成工程创建

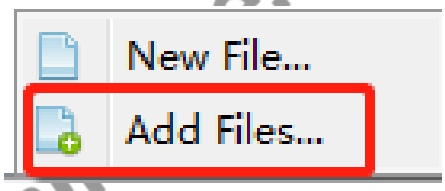


1.5.2 添加设计文件

Gowin 云源软件界面如下图：



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

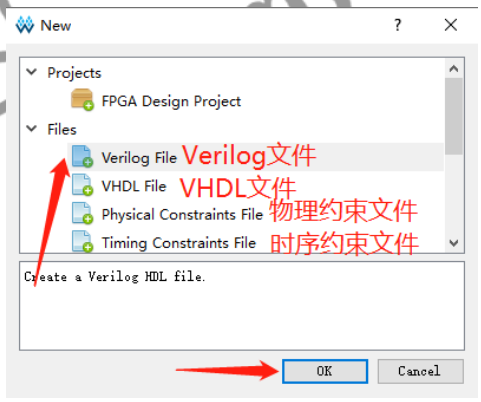


添加文件到工程：

在 Design 窗口中右击，选择 ADD FILE；

新建文件到工程：

单击上图中新建文件图标、或单击 File/New…、或右击 Design 窗口空白处选择 New File…；弹出新建文件窗口单击 Verilog File，之后再单击 OK 确定



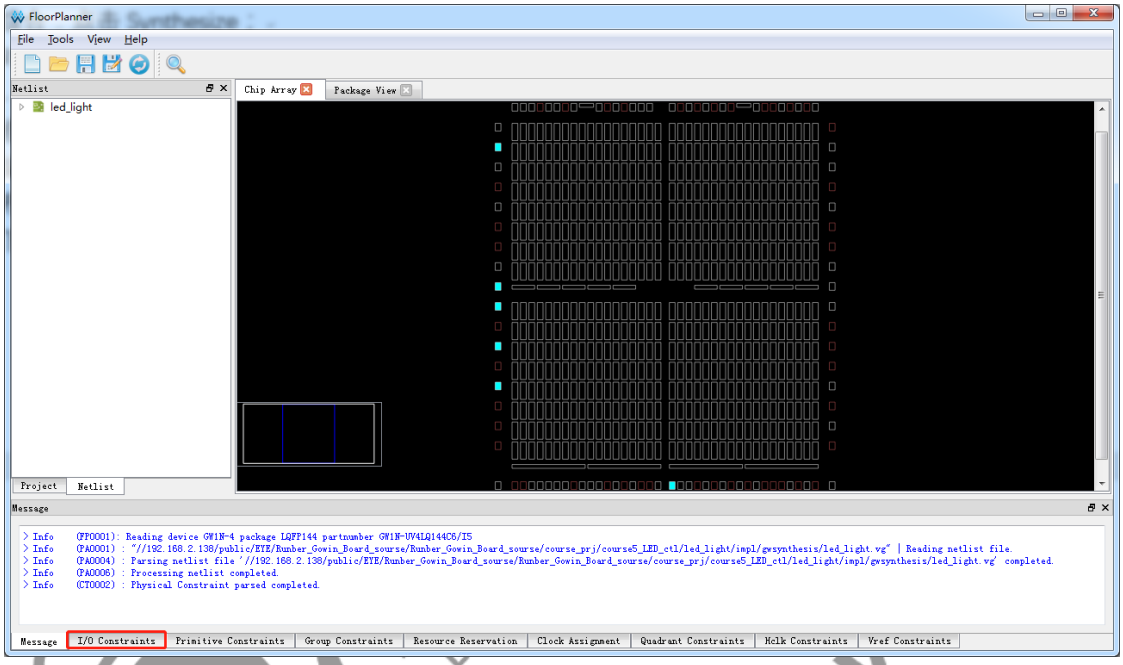
1.5.3 综合

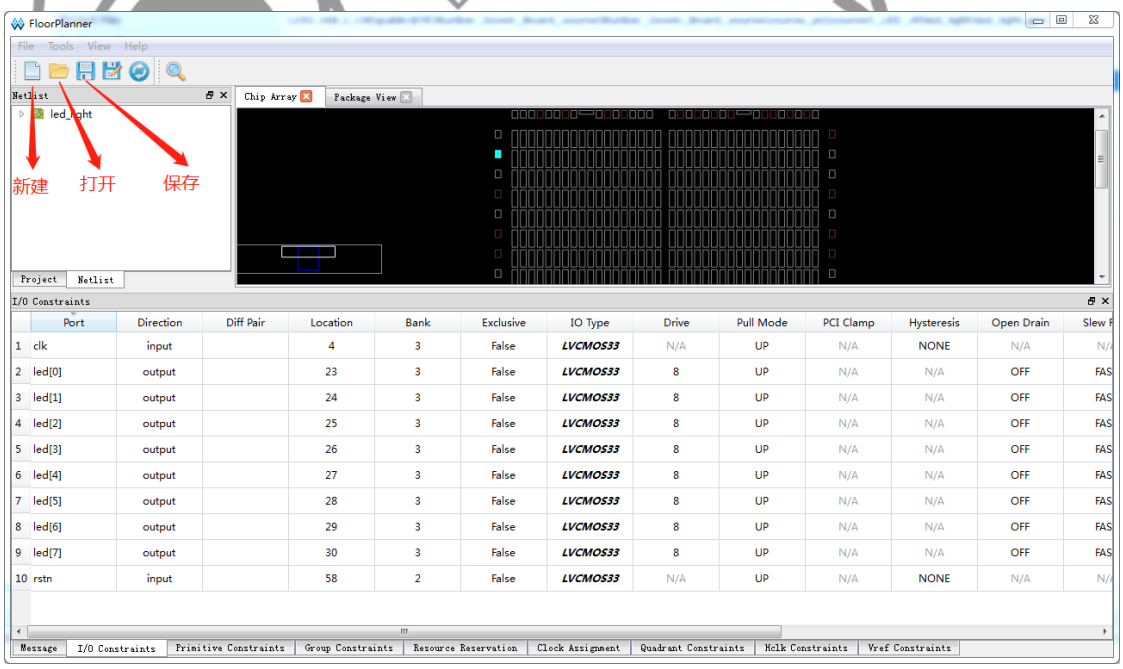
点击 Synthesize；

1.5.4 工程约束

1.5.4.1 对工程做物理约束

双击 FloorPlanner，后点击弹出窗口下方的 I/O Constraints，编辑 IO 的分配，





按照原理图修改好 IO 分配后，点击保存按钮，会保存出一个 cst 文件（约束的语法请参考高云官方提供的文档：Gowin 设计约束指南）。内容如下（也可直接新建一个 cst 文件，编辑器中的内容，约束的语法大家可将工具配置和生成的文件对比查看）：

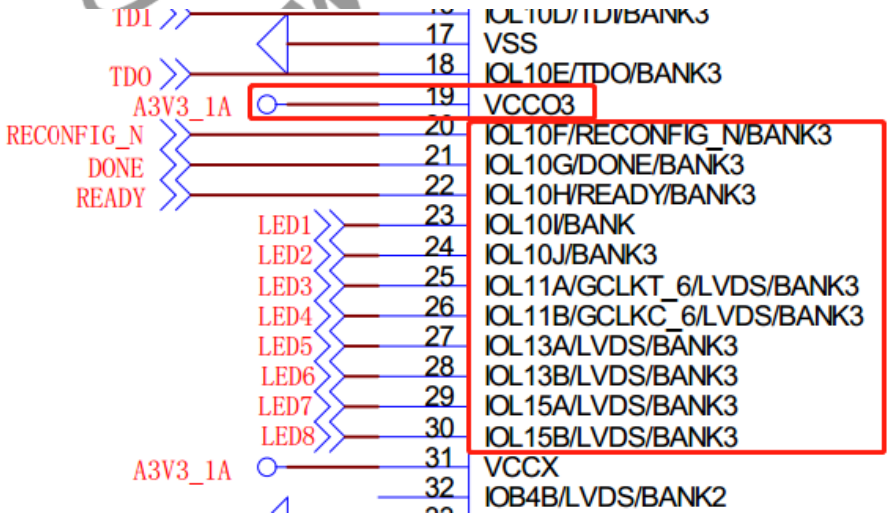
```
1   //Copyright (C)2014-2019 Gowin Semiconductor Corporation.
2   //All rights reserved.
3   //File Title: Physical Constraints file
4   //GOWIN Version: V1.9.1.01Beta
5   //Part Number: GW1N-LV4LQ144C6/I5
6   //Created Time: Wed Sep 04 20:36:36 2019
7   IO_LOC "clk" 4;
8   IO_LOC "led[0]" 23;
9   IO_LOC "led[1]" 24;
10  IO_LOC "led[2]" 25;
11  IO_LOC "led[3]" 26;
12  IO_LOC "led[4]" 27;
13  IO_LOC "led[5]" 28;
14  IO_LOC "led[6]" 29;
15  IO_LOC "led[7]" 30;
16  IO_LOC "rstn" 58;
17  IO_PORT "clk"      IO_TYPE=LVCMOS33;
18  IO_PORT "rstn"     IO_TYPE=LVCMOS33;
19  IO_PORT "led[0]"   IO_TYPE=LVCMOS33;
20  IO_PORT "led[1]"   IO_TYPE=LVCMOS33;
21  IO_PORT "led[2]"   IO_TYPE=LVCMOS33;
22  IO_PORT "led[3]"   IO_TYPE=LVCMOS33;
23  IO_PORT "led[4]"   IO_TYPE=LVCMOS33;
24  IO_PORT "led[5]"   IO_TYPE=LVCMOS33;
25  IO_PORT "led[6]"   IO_TYPE=LVCMOS33;
26  IO_PORT "led[7]"   IO_TYPE=LVCMOS33;
```

IO_LOC "signal" pin ：表示将 signal 关联到管脚 pin；

不同封装的 pin 的命名有区别，注意 pin 表示的是芯片的 PAD（管脚）序号；

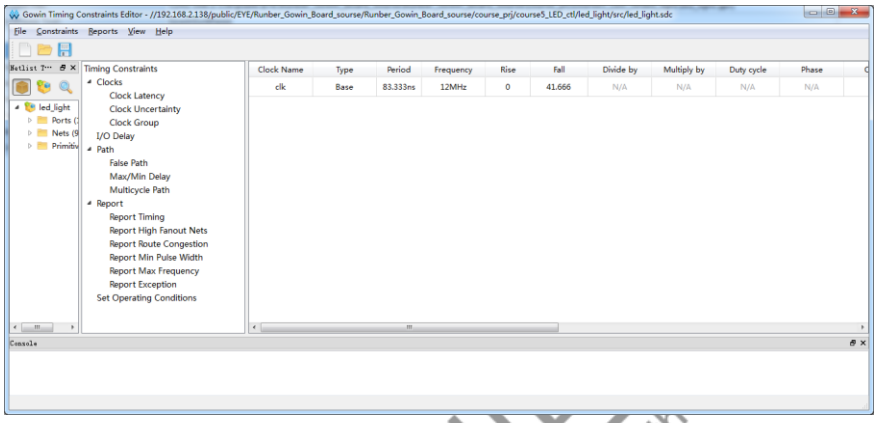
IO_PORT "signal" IO_TYPE=Level ：表示对 signal 的电平标准设置为 Level；

IO_TYPE 为电平标准，电平标准电压值需要结合 bank 电源（如下图所示，Runber 板卡的 LED 灯分配在 Bank3 上；VCCO3 为 Bank3 的 bank 电源 IO，连接的电源为 3.3V），IO 类型根据需要选择 LVDS 或 LVCMOS 或者 LVTTL 等类型；所以我们 LED 的 IO_TYPE 定义为 LVCOMS33；

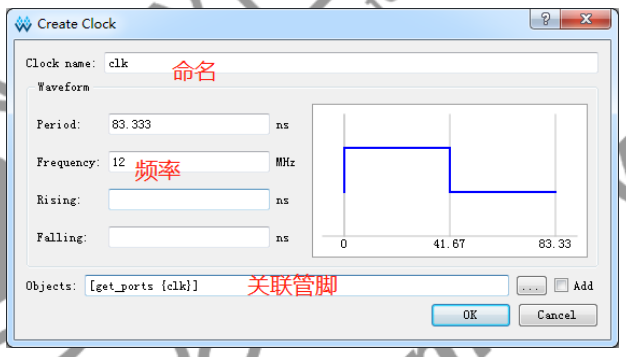


1.5.4.2 时序约束文件编辑

双击 gowin 上 Process 框中的 Timing Constraints Editor；



当前工程相对比较简单，但也需要告诉 EDA 工具我们输入的时钟的频率是多少；在左上方的空白窗口中右击，后选择 creat clock；在弹窗中对时钟命名，编辑时钟频率，关联输入管脚；



点击 OK 后会创建一条时钟约束，在时钟约束主窗口点击保存按钮，将会保存一个 sdc 文件；具体的内容如下（约束的语法请参考高云官方提供的文档：Gowin 设计约束指南）：

```
1  //Copyright (C)2014-2020 GOWIN Semiconductor Corporation.
2  //All rights reserved.
3  //File Title: Timing Constraints file
4  //GOWIN Version: 1.9.2.02 Beta
5  //Created Time: 2020-01-07 11:18:21
6  create_clock -name clk -period 83.333 -waveform {0 41.666} [get_ports {clk}]
```

1） 布局布线：双击 Place & Route；

2） 下载烧录至板卡：双击 Program Device，对应界面在第一星期资料有详细描述，这里就不重复介绍（后续文档对于重复的部分也是点到略过的方式展现）；

1.6 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

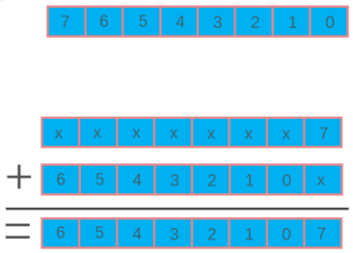
8 个 LED 以 0.5s 间隔流水

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；

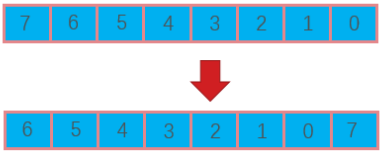
在 C 语言中做流水灯的实验需要用到一个中间变量(代码如下左侧，数据位的搬移如下右图)：

```
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：

```
1    reg [7:0] data;
2
3    always @(posedge clk)
4    data <= {data[6:0],data[7]};
5    //或：
6    wire [7:0] data1;
7    wire [7:0] out;
8    assign out = {data1[6:0],data1[7]};
```



2.4 实验源码设计

Module 的具体内容如下：

```
1   `timescale 1ns / 1ps
2   `define UD #1
3   module water_led  (
4       input               clk,
5       input               rstn,
6       output [7:0]        led
7   );
8
9   //=====================================================
10  //reg and wire
11      reg [24:0]   led_light_cnt;
12      reg [ 7:0]   led_status;
13
14      //   time counter
15      always @(posedge clk)
16      begin
17          if(!rstn)
18              led_light_cnt <= `UD 25'd0;
19          else if(led_light_cnt == 25'd599_9999)
20              led_light_cnt <= `UD 25'd0;
21          else
22              led_light_cnt <= `UD led_light_cnt + 25'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'd599_9999)
31              led_status <= `UD {led_status[6:0], led_status[7]};
32      end
33
34      assign led = led_status;
35
36  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，管脚分配与 led 闪烁实验一致。

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 按键消抖

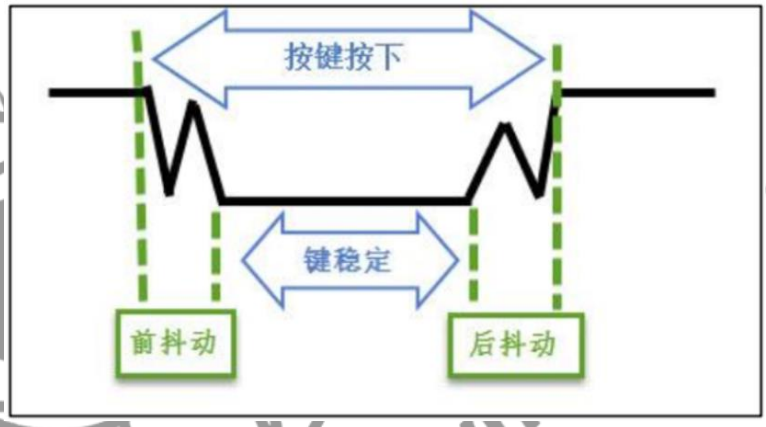
3.1 实验目的

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；

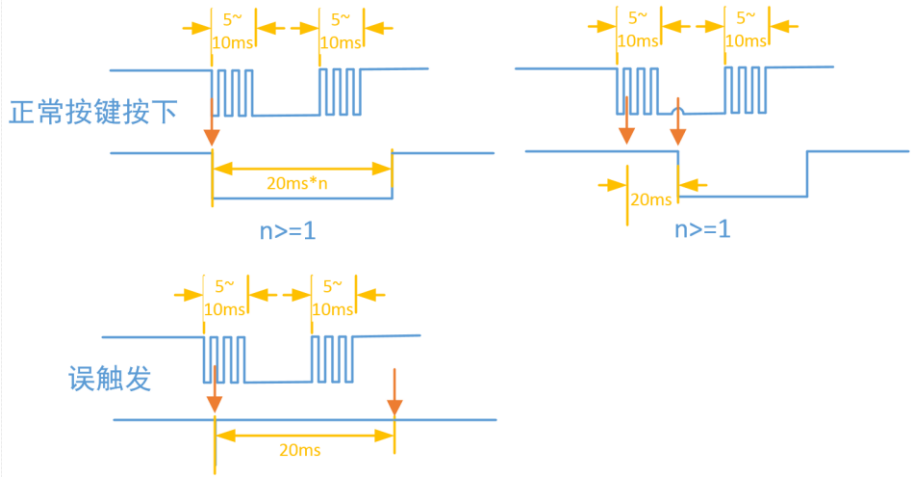
3.2 实验要求

实现按键消抖；

3.3 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 18bit 的计数器，其计数最大值为：N = 18'h3FFFF = 18'd262143

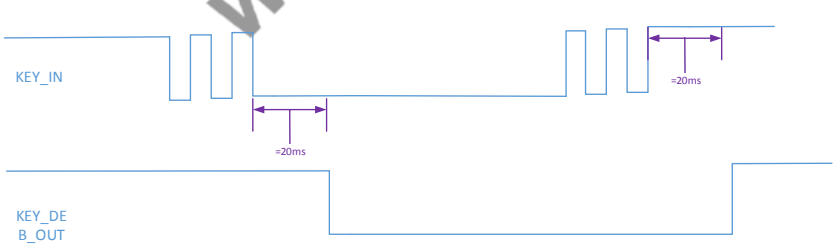
最大计数值时，计时为：t = N*T = N/f = 262143/12M = 21.84525ms > 20ms;

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；

3.4 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   )
6   (
7       input                           clk,        //12MHz
8       Input       [BTN_WIDTH-1:0]     btn_in,     // key input
9       output reg  [BTN_WIDTH-1:0]     btn_deb     // key output
10  );
11
12  //=============================================
13      reg [17:0] time_cnt= 18'd0;
14      always @ (posedge clk)
15      begin
16          if(time_cnt == 18'h3A980)       // reset at 20ms counter over
17              time_cnt <= `UD 20'd0;
18          else
19              time_cnt <= `UD time_cnt + 20'd1;
20      end
21
22      always @(posedge clk)
23      begin
24          if(time_cnt == 20'd0)
25              btn_deb <= `UD btn_in;   // latch the key input every 20ms
26      end
27
28  endmodule
29
```

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；

4 键控流水灯

4.1 实验目的

1 设计 4 种流水灯效果，可通过按键选择控制哪一种。

2 选择一个按键作为控制输入，按下一次换一种显示效果，在四种效果中循环。

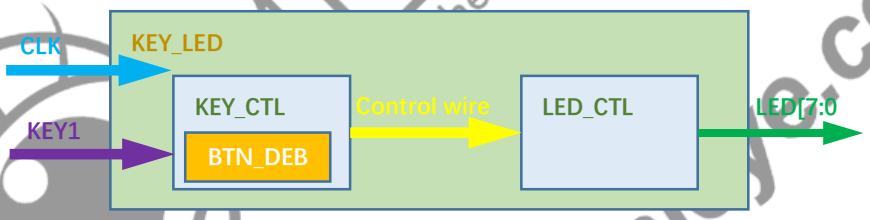
4.2 实验要求

1、实验平台：蜂鸟系列开发板；

2、按键输入由 K1 输入，LED 输出为 D1 ~ D8。

4.3 实验原理

实现框架如下：



1、顶层实现按键切换 LED 的流水灯状态；

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
);
```

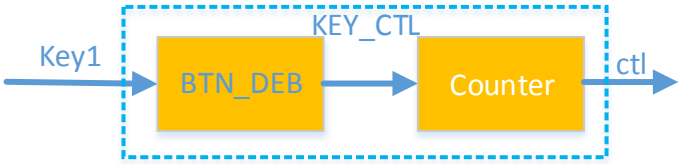
4.3.1 按键控制模块功能

接收按键输入信号。统计按键按下次数，由于流水灯模式是 4 种，计数统计范围是 0 ~ 3 循环，将计数结果传递给 LED 控制模块；

根据需求输入信号有：时钟，按键；输出信号有：流水灯控制信号；

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）改变继而调整流水灯的状态；



4.3.2 LED 控制模块功能

4 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，流水灯模式控制信号； 出信号：8bit 位宽的 LED 控制信号；

功能处理注意事项：流水灯状态切换点，不同状态的切换时如何初始化；

4.4 实验源码设计

4.4.1 顶层文件源码

```
1   `timescale 1ns / 1ps
2   `define UD #1
3   module key_led_top(
4       input       clk,
5       input       key,
6       output [7:0] led
7   );
8
9       wire [1:0] ctrl;
10
11      key_ctl key_ctl(
12          .clk       (  clk  ),//input       clk,
13          .key       (  key  ),//input       key,
14          .ctrl      (  ctrl  )//output [1:0]ctrl
15      );
16
17      led u_led(
18          .clk   (  clk   ),//input       clk,
19          .ctrl  (  ctrl  ),//input [1:0] ctrl,
20
21          .led   (  led   ) //output[7:0] led
22      );
23
24  endmodule
25
```

4.4.2 按键控制模块

```
1   `timescale 1ns / 1ps
2   `define UD #1
3   module key_ctl(
4       input           clk,
5       input           key,
6
7       output     [1:0] ctrl
8   );
9
10      wire btn_deb;
11      // 按键消抖
12      btn_deb#(
13          .BTN_WIDTH   (  4'd1     ) //parameter                BTN_WIDTH = 4'd8
14      ) U_btn_deb
15      (
16          .clk         (  clk      ),//input                       clk,
17          .btn_in      (  key      ),//input      [BTN_WIDTH-1:0] btn_in,
18
19          .btn_deb     (  btn_deb  ) //output reg [BTN_WIDTH-1:0]btn_deb
20      );
21
22      reg btn_deb_1d;
23      always @(posedge clk)
24      begin
25          btn_deb_1d <= `UD btn_deb; //get the btn_deb delay one clock cycle
26      End
27
28  //下降沿获取方式：前一个时钟周期为高电平，当前时钟周期为低电平；
29  // 故而将按键消抖后的信号打一拍（保持上一时钟周期的状态）
30  //
31  //  sig          ____________|____________
32  //
33  //sig_reg        _____________|_________
34  //
35  //falling        ____________|  |_________
36
37      reg [1:0]  key_push_cnt=2'd0;
38      always @(posedge clk)
39      begin
40          if(~btn_deb & btn_deb_1d) //get he falling edge of btn_deb
41          begin
42              key_push_cnt <= `UD key_push_cnt + 2'd1;
43               end
44      end
45
46      assign ctrl = key_push_cnt;
47
48  endmodule
49
```

对于按键消抖模块这里就不重复介绍了，请查看第 3 章节内容了解按键消抖模块的工作原理；

4.4.3 LED 控制模块

```
1   `timescale 1ns / 1ps
2   `define UD #1
3   module led(
4       input         clk,
5       input  [1:0]  ctrl,
6       output [7:0]  led
7   );
8
9       reg [24:0] led_light_cnt = 25'd0;
10      reg [ 7:0] led_status = 8'b1000_0000;
11
12      //  time counter
13      always @(posedge clk)
14      begin
15          if(led_light_cnt == 25'd19_999_999)
16              led_light_cnt <= `UD 25'd0;
17          else
18              led_light_cnt <= `UD led_light_cnt + 25'd1;
19      end
20
21      reg [1:0] ctrl_1d;     //保存上一个 led 状态周期的 ctrl 值
22      always @(posedge clk)
23      begin
24          if(led_light_cnt == 25'd19_999_999)
25              ctrl_1d <= ctrl;//此处设计能保证状态切换时，从 0 时刻开始下一次流水状态
26      end
27
28      // led status change
29      always @(posedge clk)
30      begin
31          if(led_light_cnt == 25'd19_999_999)//0.5s 周期
32          begin
33              case(ctrl)
34                  2'd0 :  //从高位到低位的 led 流水灯
35                  begin
36                      if(ctrl_1d != ctrl)
37                          led_status <= `UD 8'b1000_0000;
38                      else
39                          led_status <= `UD {led_status[0],led_status[7:1]};
40                  end
41                  2'd1 :  //从地位到高位的 led 流水灯
42                  begin
43                      if(ctrl_1d != ctrl)
44                          led_status <= `UD 8'b0000_0001;
45                      else
46                          led_status <= `UD {led_status[6:0],led_status[7]};
47                  end
```

```
48                  2'd2 :  //从低位到高位增加亮灯的个数
49                  begin
50                      if(ctrl_1d != ctrl || led_status == 8'b1111_1111)
51                          led_status <= `UD 8'b0000_0000;
52                      else
53                          led_status <= `UD {led_status[6:0],1'b1};
54                  end
55                  2'd3 :  //从高位到低位增加灭灯的个数
56                  begin
57                      if(ctrl_1d != ctrl || led_status == 8'b0000_0000)
58                          led_status <= `UD 8'b1111_1111;
59                      else
60                          led_status <= `UD {1'b0,led_status[7:1]};
61                  end
62              endcase
63          end
64      end
65
66      assign led = led_status;
67
68  endmodule
69
70  endmodule
71
```

4.5 实验现象

上电后下载完固件，默认 LED8 ~ LED1 流水，每按下一次 KEY1，流水灯状态切换一次，总共四种状态可供循环切换；

5 数码管静态

5.1 实验目的

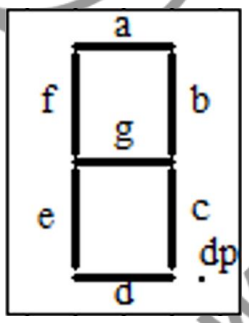
5.2 实验要求

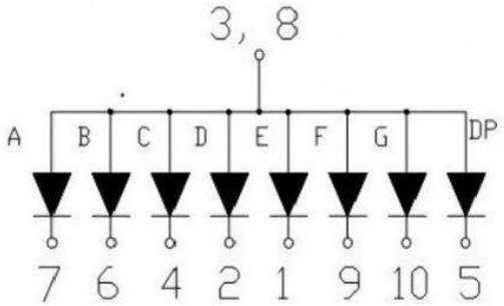
四个数码管同时显示 0-9，每隔 1s 数字变一次。

5.3 实验原理

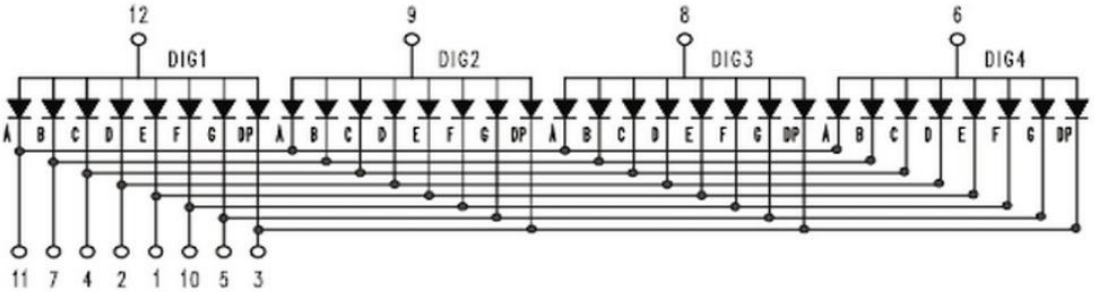
5.3.1 数码管工作原理

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：

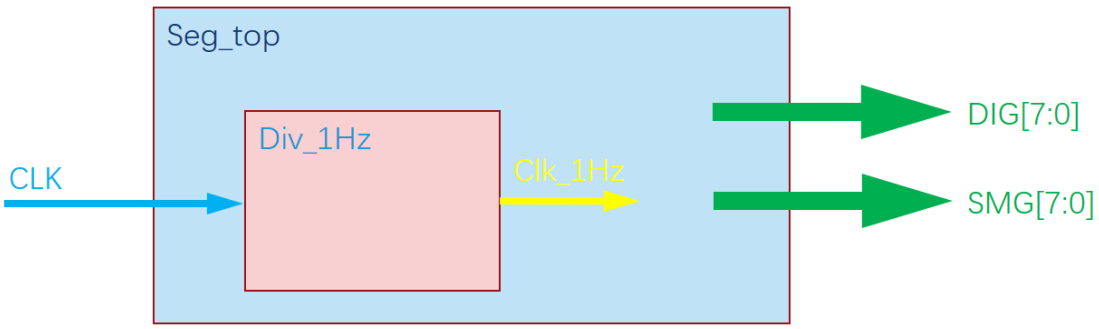
输入相应的电平点亮一根根小火柴 a-b-c-d-e-f-g-dp。如果数码管是共阴极，给高电平 1 即可相应点亮，反之如果是共阳极，给低电平 0 即可相应点亮。Runber 板的数码管为共阳极数码管，所以才有如下配置显示出 0～9：

```
assign dig = 4'b0000;
    // 0 1 2 3 4 5 6 7
    // G F E D C B A P
case(counter)
    4'd0:smg = 8'b0111_1110;//"0"   8'b1000_0001
    4'd1:smg = 8'b0011_0000;//"1"   8'b1100_1111
    4'd2:smg = 8'b0110_1101;//"2"   8'b1001_0010
    4'd3:smg = 8'b0111_1001;//"3"   8'b1000_0110
    4'd4:smg = 8'b0011_0011;//"4"   8'b1100_1100
    4'd5:smg = 8'b0101_1011;//"5"   8'b1010_0100
    4'd6:smg = 8'b0101_1111;//"6"   8'b1010_0000
    4'd7:smg = 8'b0111_0000;//"7"   8'b1000_1111
    4'd8:smg = 8'b0111_1111;//"8"   8'b1000_0000
    4'd9:smg = 8'b0111_1011;//"9"   8'b1000_0100
    default:smg = 8'b0111_1110;
endcase
```

5.3.2 方案设计

1、顶层实现数码管显示切换；

2、需要设计一个 1S 钟的计时器；



1Hz 的时钟产生请参照 LED 闪烁实验，输出的 LED 控制信号就是 8 个 1Hz 的信号；

5.4 实验源码

```
1   `timescale 1ns / 1ps
2   `define UD #1
3   module top_seq
4   (
5       input               clk,//12MHZ 83.33ns
6       output [3:0]        dig,
7       output reg [7:0]    smg
8   );
9
10      wire clk_1hz;
11      div_clk u_div_clk
12       (
13          .clk(clk),
14          .clk_1hz(clk_1hz)
15      );
16
17      reg [3:0]counter=0;
18      always @(posedge clk_1hz)
19      begin
20          if(counter==4'd9)
21              counter <= `UD 4'd0;
22          else
23              counter <= `UD counter + 1'b1;
24      end
25
26      assign dig = 4'b0000;
27      always @(*)
28      begin
29          case(counter)
30              4'd0:smg = 8'b0111_1110;//"0"  8'b1000_0001
31              4'd1:smg = 8'b0011_0000;//"1"  8'b1100_1111
32              4'd2:smg = 8'b0110_1101;//"2"  8'b1001_0010
33              4'd3:smg = 8'b0111_1001;//"3"  8'b1000_0110
34              4'd4:smg = 8'b0011_0011;//"4"  8'b1100_1100
35              4'd5:smg = 8'b0101_1011;//"5"  8'b1010_0100
36              4'd6:smg = 8'b0101_1111;//"6"  8'b1010_0000
37              4'd7:smg = 8'b0111_0000;//"7"  8'b1000_1111
38              4'd8:smg = 8'b0111_1111;//"8"  8'b1000_0000
39              4'd9:smg = 8'b0111_1011;//"9"  8'b1000_0100
40              default:smg = 8'b0111_1110;
41          endcase
42      end
43  endmodule
```

5.5 实验现象

上电烧录固件后，4 位数码管显示数值相同，并且每秒钟变化一次，从 0～9 依次循环跳转显示。

6 数码管动态显示

6.1 实验目的

动态控制 4 位数码管显示不同的数值；

6.2 实验要求

四个数码管显示不同的数字，按键 K0 控制第一个数码管，按一下数字加 1，从 0 到 9，按键 K1 控制第二个数码管，按一下数字加 1，从 0 到 9，按键 K2 控制第三个数码管，按键 K3 控制第四个数码管。

6.3 实验原理

通过前面的章节大家也了解到数码管其实是由多个 LED 灯组合而成，通过不同的组合显示出不同的结果，上一个实验完成了 4 个数码管同时点亮显示同一个数值，那对与显示不同的数值该如何处理呢？

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。

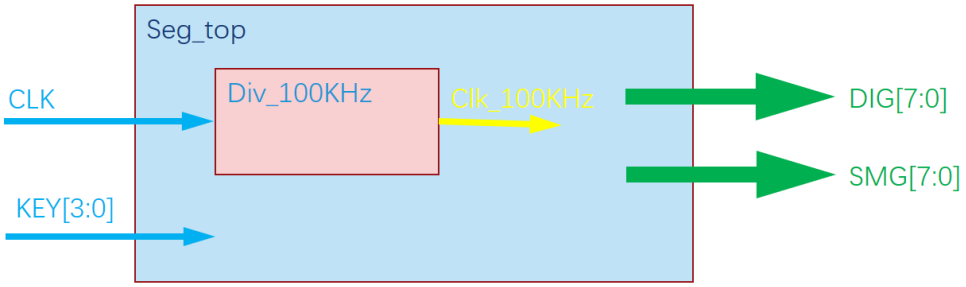
在设计数码管闪烁式显示时，对于人眼观测来说，频率越高越好，但是数码管中的 LED 灯珠点亮对于高电平（关注发光响应时间，Runber 板卡上的数码管发光响应时间是 0.1us 级的）是有要求的，故而不是越高越好，取一个适当的刷新频率即可，实验中我们取刷新率为 100KHz。

方案设计：

1、按键消抖；

2、按键计数；

3、数码管的分时显示；



6.4 实验源码

6.4.1 顶层模块

```
1   `timescale 1ns / 1ps
2   `define UD #1
3   module top_seq2
4   (
5       input clk,//50MHZ 20ns
6       input [3:0]button,
7       output reg [3:0]dig,
8       output reg [7:0]smg
9   );
10  /*=============================================================
11                          按键消斗
12  =============================================================*/
13  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 key0
29  (
30      .clk(clk),
31      .key(key[0]),
32      .key_times(key0_cnt)
33  );
```

```
34  wire [3:0] key1_cnt;
35  key_cnt key1
36  (
37      .clk(clk),
38      .key(key[1]),
39      .key_times(key1_cnt)
40  );
41
42  wire [3:0] key2_cnt;
43  key_cnt key2
44  (
45      .clk(clk),
46      .key(key[2]),
47      .key_times(key2_cnt)
48  );
49
50  wire [3:0] key3_cnt;
51  key_cnt key3
52  (
53      .clk(clk),
54      .key(key[3]),
55      .key_times(key3_cnt)
56  );
57  /*=====================================================
58                      时钟分频
59  =====================================================*/
60  wire clk_100khz;
61  div_clk div_clk
62  (
63      .clk       (clk),
64      .clk_100khz (clk_100khz)
65  );
66  /*=====================================================
67                      数码管显示
68  =====================================================*/
69  reg  [1:0]sel=0;
70  wire [3:0]dig0;
71  wire [7:0]smg0;
72
73  always @(posedge clk_100khz)
74  begin
75      sel <= `UD sel+1'b1;
76  end
77
78  seq_control seq_control_0
79  (
80      .sel(2'd3),
81      .key(key0_cnt),
82      .dig(dig0),
83      .smg(smg0)
84  );
85
```

```
86   wire [3:0]dig2;
87   wire [7:0]smg2;
88
89   seq_control seq_control_2
90   (
91       .sel(2'd1),
92       .key(key2_cnt),
93       .dig(dig2),
94       .smg(smg2)
95   );
96
97   wire [3:0]dig3;
98   wire [7:0]smg3;
99
100  seq_control seq_control_3
101  (
102      .sel(2'd0),
103      .key(key3_cnt),
104      .dig(dig3),
105      .smg(smg3)
106  );
107
108  // Select display seq:
109  always @(posedge clk_100khz)
110  begin
111      if(sel==2'b00)
112          dig <= `UD dig0;
113      else if(sel==2'b01)
114          dig <= `UD dig1;
115      else if(sel==2'b10)
116          dig <= `UD dig2;
117      else if(sel==2'b11)
118          dig <= `UD dig3;
119  end
120
121  always @(posedge clk_100khz)
122  begin
123      if(sel==2'b00)
124          smg <= `UD smg0;
125      else if(sel==2'b01)
126          smg <= `UD smg1;
127      else if(sel==2'b10)
128          smg <= `UD smg2;
129      else if(sel==2'b11)
130          smg <= `UD smg3;
131  end
132
133  endmodule
134
```

6.4.2 按键控制模块

```
1   `timescale 1ns / 1ps
2   `define UD #1
3   module key_cnt
4   (
5       input                   clk,
6       input                   rstn_key,
7       input                   key,
8       output reg [3:0]        key_times
9   );
10
11      reg key_reg;
12      always @(posedge clk)
13      begin
14          key_reg <= `UD key;     //get key one clock delay value
15      end
16
17      always @(posedge clk )
18      begin
19          if(key_reg&&~key)       //falling edge
20          begin
21              if(key_times==4'd9)
22                  key_times <=`UD 4'd0;
23              else
24                  key_times <=`UD key_times + 1'b1;
25          end
26          else
27              key_times <=`UD key_times;
28      end
29
30  endmodule
31
```

6.4.3 数码管控制模块

```
1   `timescale 1ns / 1ps
2   `define UD #1
3   module seq_control
4   (
5       input [1:0]sel,
6       input [3:0]key,
7       output reg [3:0]dig,
8       output reg [7:0]smg
9   );
10
```

```
/*=============================================================
                        位选择映射
=============================================================*/
always @(*)
begin
    case(sel)
        2'd0:dig = 4'b1110;
        2'd1:dig = 4'b1101;
        2'd2:dig = 4'b1011;
        2'd3:dig = 4'b0111;
        default:dig = 4'b1111;
    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'b0111_1110;//"0"   8'b1000_0001
        4'd1:smg = 8'b0011_0000;//"1"   8'b1100_1111
        4'd2:smg = 8'b0110_1101;//"2"   8'b1001_0010
        4'd3:smg = 8'b0111_1001;//"3"   8'b1000_0110
        4'd4:smg = 8'b0011_0011;//"4"   8'b1100_1100
        4'd5:smg = 8'b0101_1011;//"5"   8'b1010_0100
        4'd6:smg = 8'b0101_1111;//"6"   8'b1010_0000
        4'd7:smg = 8'b0111_0000;//"7"   8'b1000_1111
        4'd8:smg = 8'b0111_1111;//"8"   8'b1000_0000
        4'd9:smg = 8'b0111_1011;//"9"   8'b1000_0100
        default:smg = 8'b1111_1111;
    endcase
end

endmodule

```

6.5 实验现象

KEY0 ~ 3 分别控制数码管从左到右的数码管显示，按键 K0 控制第一个数码管，按一下数字加 1，从 0 到 9，按键 K1 控制第二个数码管，按一下数字加 1，从 0 到 9，按键 K2 控制第三个数码管，按键 K3 控制第四个数码管。

7 UART 串口通信

7.1 实验目的

使用 RUNBER 板卡通过串口与 PC 进行数据交互，并做简单的验证。

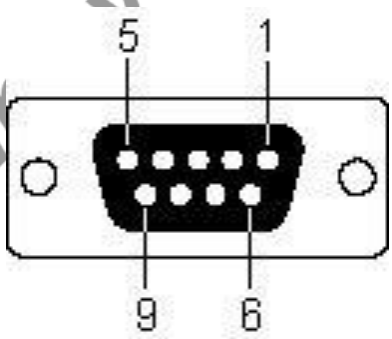
7.2 实验要求

串口通信时波特率设置为 115200bps，数据格式为 1 位起始位、8 位数据位、无校验位、1 位结束位。板子 1s 向串口助手发送一次十进制显示的 “===HELLO WORD===” ，通过串口助手向板子以十六进制形式发送数字，用数码管显示收到的十六进制数，LED 以十六进制显示亮起。

7.3 实验原理

7.3.1 串口原理

从右图我们可以看到标准串口接口是 9 根线，具体含义如下：



数据线：

TXD（pin 3）：串口数据输出(Transmit Data)

RXD（pin 2）：串口数据输入(Receive Data)

握手：

RTS（pin 7）：发送数据请求(Request to Send)

CTS（pin 8）：清除发送(Clear to Send)

DSR（pin 6）：数据发送就绪(Data Send Ready)

DCD（pin 1）：数据载波检测(Data Carrier Detect)

DTR（pin 4）：数据终端就绪(Data Terminal Ready)

地线：

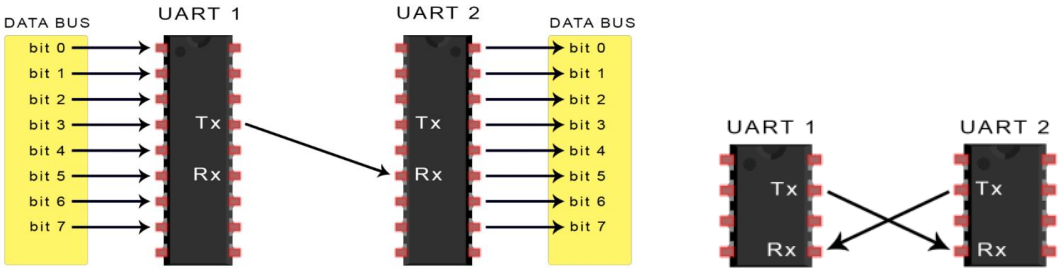
GND（pin 5）：地线

其它

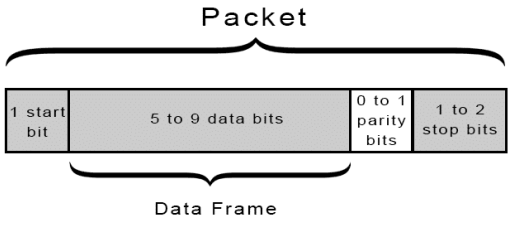
RI（pin 9）：铃声指示

通常我们用 RS232 串口仅用到了 9 根传输线中的三根：TXD，RXD，GND。但是对于数据传输，双方必须对数据传输采用使用相同的波特率，约定同样的传输模式（传输架构，握手条件等）。尽管这种方法对于大多数应用已经足够，但是对于接收方过载的情况这种使用受到限制。

RS232 的串口连接方式：



串口传输协议如下：



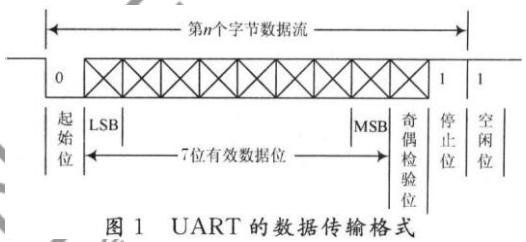


图 1 UART 的数据传输格式

起始位：先发出一个逻辑”0”信号，表示传输字符的开始。

数据位：可以是 5~8 位逻辑”0”或”1”。如 ASCII 码（7 位），扩展 BCD 码（8 位）。

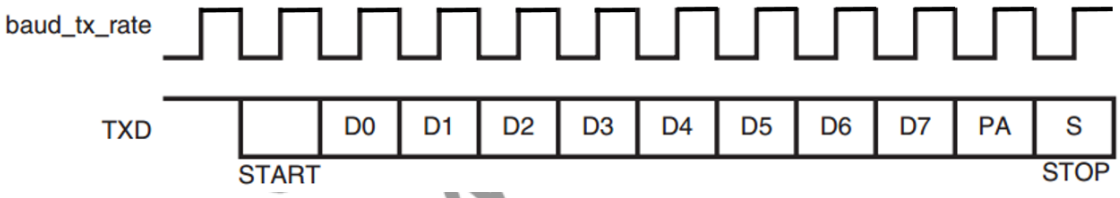
校验位：数据位加上这一位后，使得“1”的位数应为偶数(偶校验)或奇数(奇校验)

停止位：它是一个字符数据的结束标志。可以是 1 位、1.5 位、2 位的高电平。

空闲位：处于逻辑“1”状态，表示当前线路上没有资料传送。

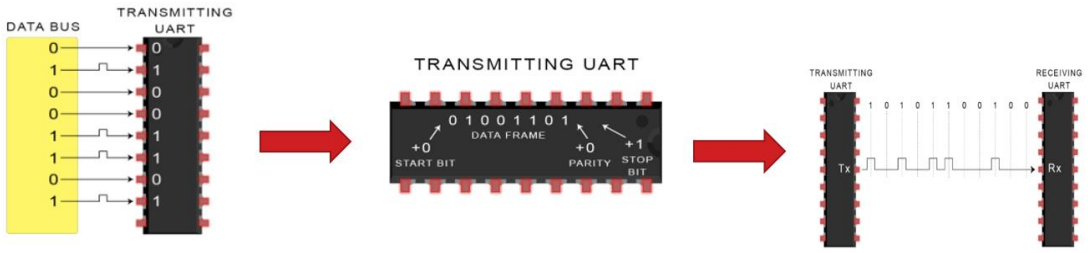
波特率：uart 中的波特率就可以认为是比特率，即每秒传输的位数(bit)。一般选波特率都会有 9600,19200,115200 等选项。其实意思就是每秒传输这么多个比特位数(bit)。

引入波特率的概念后可得到串口的传输节奏如下：

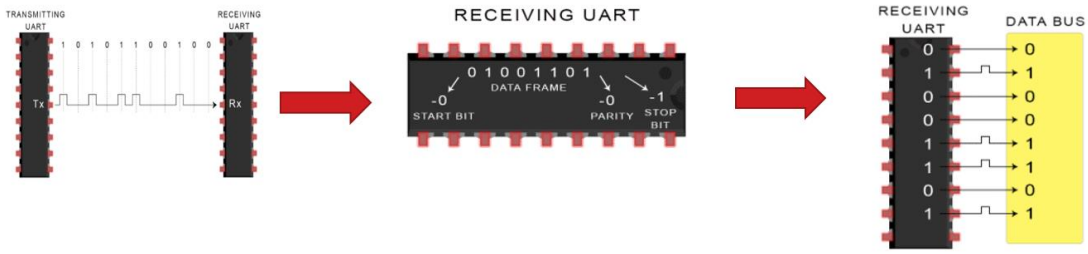


7.3.2 串口传输步骤

7.3.2.1 串口发送流程



7.3.2.2 串口接收流程



7.3.3 串口发送字符

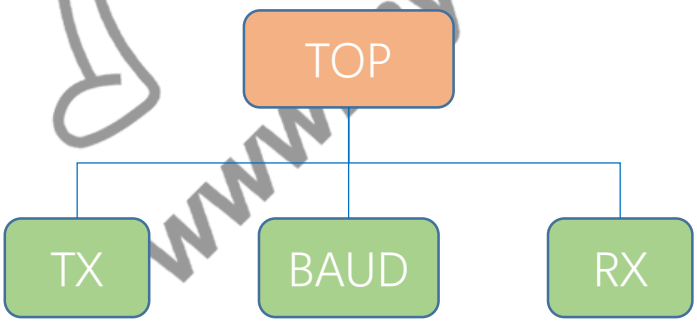
从前面串口协议中可以了解到串口每次传输可以以有 5 ~ 8bit 数据，在计算机中字符通常用 ASCII 码（7bit）表示，所以字符的发送可以用 ASCII 码发送。

查询 ASCII 码表格可得到： “===HELLO WORD===”用到的字符对应 ASCII 码；

字符	ASCII 码（hex）	字符	ASCII 码（hex）	字符	ASCII 码（hex）	字符	ASCII 码（hex）
=	3D	H	48	E	45	L	4C
O	4F	W	57	R	52	D	44
回车	0D	换行	0A	空格	20		

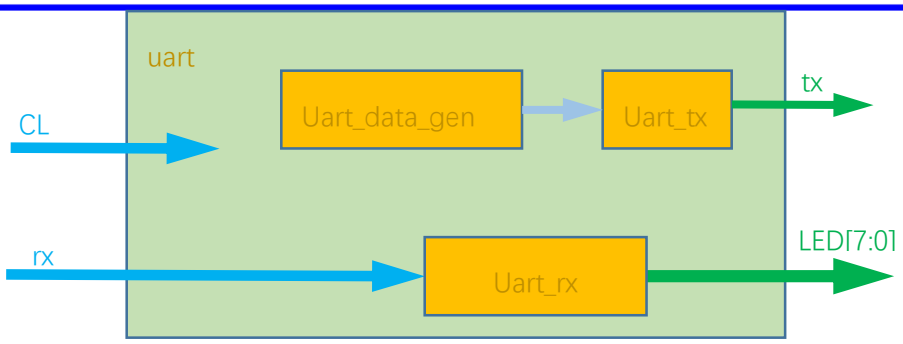
7.4 实验源码设计

从实验目的分析可将实验做如下划分：



从原理上分析波特率的计算是一个计数器，发射和接收可复用，我们在设计时为保持 TX，或 RX 的完整性，故将波特周期计数器集成在各自模块内部；

上述分析仅仅搭建好 RUNBER 的与 PC 通信的桥梁 UART，传输的数据没有体现。故而需要增加发送数据模块，与接收数据模块；



对于数码管显示部分留给大家课后自行集成，对数码管显示还不太明白的查阅第 5、6 章节内容。

7.4.1 串口发射模块设计

目标：接收到一个发送命令信号时，将 data[7:0] -> 依次发出{start,data[0:7],stop}共 10bit 数据（无校验位，停止位 1bit）；

有两种方法可以将一个并行数据串行化；

方法一：通过 bit 计数与 baud 计数控制移位输出；

```
1   // transmit bit
2       always@(posedge clk)
3       begin
4           if(!rstn)
5               txd <= `UD 1'b1;
6           else
7           begin
8               if(trans_en)
9               Begin
10  //  将开始标志和停止标志以及传输数据集成放到 trans_data 中可用下方语句
11  //              txd <= `UD trans_data[trans_bit];
12  //  单 bit 控制用下方语句
13                  case(trans_bit)
14                      4'h0    :txd <= `UD 1'b0;
15                      4'h1    :txd <= `UD tx_data_reg[0];
16                      4'h2    :txd <= `UD tx_data_reg[1];
17                      4'h3    :txd <= `UD tx_data_reg[2];
18                      4'h4    :txd <= `UD tx_data_reg[3];
19                      4'h5    :txd <= `UD tx_data_reg[4];
20                      4'h6    :txd <= `UD tx_data_reg[5];
21                      4'h7    :txd <= `UD tx_data_reg[6];
22                      4'h8    :txd <= `UD tx_data_reg[7];
23                      4'h9    :txd <= `UD 1'b1;
24                      default :txd <= `UD 1'b1;
25                  endcase
26              end
27              else
28                  txd <= `UD 1'b1;
29          end
30      end
31
```

方法二：通过 bit 计数与 baud 计数控制状态跳转，在状态机中输出；

```
//   logical ouput  状态机输出
always @ (posedge clk)
begin
  if(tx_en)
  begin
      case(tx_state)
          IDLE       :  uart_tx  <= `UD 1'h1;  //空闲状态输出高电平
          SEND_START :  uart_tx  <= `UD 1'h0; //start 状态发送一个波特周期的低电平
          SEND_DATA  :                        //发送状态每个波特周期发送一个 bit；
          begin
              case(tx_bit_cnt)
                  3'h0  :  uart_tx  <= `UD trans_data[0];
                  3'h1  :  uart_tx  <= `UD trans_data[1];
                  3'h2  :  uart_tx  <= `UD trans_data[2];
                  3'h3  :  uart_tx  <= `UD trans_data[3];
                  3'h4  :  uart_tx  <= `UD trans_data[4];
                  3'h5  :  uart_tx  <= `UD trans_data[5];
                  3'h6  :  uart_tx  <= `UD trans_data[6];
                  3'h7  :  uart_tx  <= `UD trans_data[7];
                  default: uart_tx  <= `UD 1'h1;
              endcase
          end
          SEND_STOP  :  uart_tx  <= `UD 1'h1; //发送停止状态 输出 1 个波特周期高电平
          default    :  uart_tx  <= `UD 1'h1; // 其他状态默认与空闲状态一致，保持高电平输出
      endcase
  end
  else
      uart_tx <= `UD 1'h1;
end

```

方法一的 module 如下：

```
`timescale 1ns / 1ps
`define UD #1
module uart_tx #(
    parameter  BAUND_RATE_CNT = 12'd1250
    //115200 : 12MHz,  12000000/115200 = 10'd104
    //9600   :        12000000/9600 = 11'd1250
)
(
   input      clk,
   input      rstn,
   input      trig,      // active posedge

   input [7:0] tx_data,

   output reg  txd,
   output     tx_busy
);

```

```
19  //=============================================================================
20  // baund rate set
21      reg [11:0] baund_cnt;
22      always @(posedge clk)
23      begin
24          if(!rstn)
25              baund_cnt <= `UD 12'd0;
26          else
27          begin
28              if(baund_cnt == BAUND_RATE_CNT - 1'b1)
29                  baund_cnt <= `UD 12'd0;
30              else
31                  baund_cnt <= `UD baund_cnt + 12'd1;
32          end
33      end
34
35      wire baund_over = (baund_cnt == BAUND_RATE_CNT - 1'b1) ? 1'b1 : 1'b0;
36
37  //=============================================================================
38  //transmit start
39      reg trig_1d;
40      reg [7:0] tx_data_reg;
41      always @(posedge clk)
42      begin
43          trig_1d <= `UD trig;
44      end
45
46      reg start_en;
47      wire start;
48      always @(posedge clk)
49      begin
50          if(!rstn)
51              start_en <= `UD 1'b0;
52          else if(~trig_1d & trig & ~start_en)
53              start_en <= `UD 1'b1;
54          else if(baund_over)
55              start_en <= `UD 1'b0;
56      end
57      assign start = ~trig_1d & trig;//start_en & baund_over;
58
59      //将 data 在触发发送时进行锁存
60      always @(posedge clk)
61      begin
62          if(!rstn)
63              tx_data_reg <= `UD 8'h3f;
64          else if(~trig_1d & trig)
65              tx_data_reg <= `UD tx_data;
66      end
67
```

```
68  //=============================================================================
69  // trasmit data   将 start stop data 锁存在一个锁存器中。
70      reg [9:0] trans_data;
71
72      always @(posedge clk)
73      begin
74          if(!rstn)
75              trans_data <= `UD 10'h3f;
76          else if(~trig_1d & trig)
77              trans_data <= `UD {1'b1,tx_data,1'b0};
78      end
79
80  //=============================================================================
81  // transmit control
82      reg      trans_en;
83      reg [3:0] trans_bit;
84      always @(posedge clk)
85      begin
86          if(!rstn)
87              trans_en <= `UD 1'b0;
88          else if(~trig_1d & trig)
89              trans_en <= `UD 1'b1;
90          else if(trans_bit == 4'd9 && baund_over)
91              trans_en <= `UD 1'b0;
92          else
93              trans_en <= `UD trans_en;
94      end
95
96      assign tx_busy = ~trans_en;
97
98      always @(posedge clk)
99      begin
100         if(!rstn)
101             trans_bit <= `UD 4'd0;
102         else
103         begin
104             if(trans_en && baund_over)
105             begin
106                 if(trans_bit == 4'd9)
107                     trans_bit <= `UD 4'd0;
108                 else
109                     trans_bit <= `UD trans_bit + 4'd1;
110             end
111             else if(!trans_en)
112                 trans_bit <= `UD 4'd0;
113         end
114     end
115
```

```
116 //=============================================================================
117 // transmit bit
118     always@(posedge clk)
119     begin
120         if(!rstn)
121             txd <= `UD 1'b1;
122         else
123         begin
124             if(trans_en)
125             begin
126 //  将开始标志和停止标志以及传输数据集成放到 trans_data 中可用下方语句
127 //                txd <= `UD trans_data[trans_bit];
128 //  单 bit 控制用下方语句
129                 case(trans_bit)
130                     4'h0    :txd <= `UD 1'b0;
131                     4'h1    :txd <= `UD tx_data_reg[0];
132                     4'h2    :txd <= `UD tx_data_reg[1];
133                     4'h3    :txd <= `UD tx_data_reg[2];
134                     4'h4    :txd <= `UD tx_data_reg[3];
135                     4'h5    :txd <= `UD tx_data_reg[4];
136                     4'h6    :txd <= `UD tx_data_reg[5];
137                     4'h7    :txd <= `UD tx_data_reg[6];
138                     4'h8    :txd <= `UD tx_data_reg[7];
139                     4'h9    :txd <= `UD 1'b1;
140                     default :txd <= `UD 1'b1;
141                 endcase
142             end
143             else
144                 txd <= `UD 1'b1;
145         end
146     end
147
```

方法二的 module 设计如下：

```
1   `timescale 1ns / 1ps
2   `define UD #1
3   module uart_tx #(
4       parameter           BPS_NUM  =    16'd434
5   //  设置波特率为 4800 时，bit 位宽时钟周期个数:50MHz set 10417  40MHz set 8333
6   //  设置波特率为 9600 时，bit 位宽时钟周期个数:50MHz set 5208   40MHz set 4167
7   //  设置波特率为 115200 时，bit 位宽时钟周期个数:50MHz set 434  40MHz set 347 12M set 104
8   )
9   (
10      input         clk,       // clock                                时钟信号
11      input [7:0]    tx_data,  // uart tx data signal byte;            等待发送的字节数据
12      input         tx_pluse, // uart tx enable signal,rising is active; 发送模块发送触发信号
13
14      output reg     uart_tx,  // uart tx transmit data line          发送模块串口发送信号线
15      output        tx_busy   // uart tx module work states,high is busy;发送模块忙状态指示
16  );
17
```

```
18      //=============================================================================
19      //wire and reg in the module
20      //=============================================================================
21      reg            tx_pluse_reg =0;
22      reg [7:0]       trans_data=0;
23
24      reg [2:0]       tx_bit_cnt=0;   //the bits number has transmited.
25
26      reg [2:0]       tx_state=0;     //current state of tx state machine.
27      reg [2:0]       tx_state_n=0;   //next state of tx state machine.
28
29      reg [3:0]       pluse_delay_cnt=0;
30      reg            tx_en = 0;
31
32      // uart tx state machine's state
33      localparam  IDLE       = 4'h0;  //tx state machine's state.空闲状态
34      localparam  SEND_START = 4'h1;  //tx state machine's state.发送 start 状态
35      localparam  SEND_DATA  = 4'h2;  //tx state machine's state.发送数据状态
36      localparam  SEND_STOP  = 4'h3;  //tx state machine's state.发送 stop 状态
37      localparam  SEND_END   = 4'h4;  //tx state machine's state.发送结束状态
38
39      // uart bps set  the clk's frequency is 50MHz
40      reg [15:0]    clk_div_cnt=0;    //count for division the clock.
41
42      //=============================================================================
43      //logic
44      //=============================================================================
45      assign tx_busy = (tx_state != IDLE);
46      //some control single.
47
48      always @(posedge clk)
49      begin
50          tx_pluse_reg <= `UD tx_pluse;
51      end
52
53      // uart 锁存待发射数据
54
55      always @(posedge clk)
56      begin
57          if(~tx_pluse_reg & tx_pluse)
58              trans_data <= `UD tx_data;
59      end
60
61      // uart 模块发送工作使能标志信号
62      always @(posedge clk)
63      begin
64          if(~tx_pluse_reg & tx_pluse)
65              tx_en <= `UD 1'b1;
66          else if(tx_state == SEND_END)
67              tx_en <= `UD 1'b0;
68      end
69
```

```
70      //division the clock to satisfy baud rate.波特周期计数器
71      always @ (posedge clk)
72      begin
73          if(clk_div_cnt == BPS_NUM || (~tx_pluse_reg & tx_pluse))
74              clk_div_cnt   <= `UD 16'h0;
75          else
76              clk_div_cnt   <= `UD clk_div_cnt + 16'h1;
77      end
78
79      //count the number has transmited.发送数据状态中，发送 bit 位计数，以波特周期累加
80      always @ (posedge clk)
81      begin
82          if(!tx_en)
83              tx_bit_cnt    <= `UD 3'h0;
84          else if((tx_bit_cnt == 3'h7) && (clk_div_cnt == BPS_NUM))
85              tx_bit_cnt    <= `UD 3'h0;
86          else if((tx_state == SEND_DATA) && (clk_div_cnt == BPS_NUM))
87              tx_bit_cnt    <= `UD tx_bit_cnt + 3'h1;
88          else
89              tx_bit_cnt    <= `UD tx_bit_cnt;
90      end
91
92      //=====================================================================
93      //transmitter state machine
94      //=====================================================================
95      //   state change 状态跳转
96      always @(posedge clk)
97      begin
98          tx_state <= tx_state_n;
99      end
100
101     //   state change condition 状态跳转条件及规律
102     always @ (*)
103     begin
104       case(tx_state)
105         IDLE    :
106         begin
107             if(~tx_pluse_reg & tx_pluse) //触发发送做 16 个 clock 延时后跳到发送 start 状态
108                 tx_state_n = SEND_START;
109             else
110                 tx_state_n = tx_state;
111         end
112         SEND_START  :
113         begin
114             if(clk_div_cnt == BPS_NUM) //发送一个波特周期的低电平后进入，发送数据状态
115                 tx_state_n = SEND_DATA;
116             else
117                 tx_state_n = tx_state;
118         end
```

```
119         SEND_DATA   :
120         begin
121             if(tx_bit_cnt == 3'h7 && clk_div_cnt == BPS_NUM)
122                 //计时 8 个波特周期后（发送了 8bit 数据），跳转到发送 stop 状态
123                 tx_state_n = SEND_STOP;
124             else
125                 tx_state_n = tx_state;
126         end
127         SEND_STOP   :
128         begin
129             if(clk_div_cnt == BPS_NUM)
130                //设置停止位宽为 1 个波特周期，计数发送一个波特周期的高电平，之后跳转到发送结束状态
131                 tx_state_n = SEND_END;
132             else
133                 tx_state_n = tx_state;
134         end
135         SEND_END    :  tx_state_n = IDLE;
136         default :  tx_state_n = IDLE;
137       endcase
138     end
139
140     //   logical ouput  状态机输出
141     always @ (posedge clk)
142     begin
143       if(tx_en)
144       begin
145           case(tx_state)
146               IDLE       :  uart_tx  <= `UD 1'h1;  //空闲状态输出高电平
147               SEND_START :  uart_tx  <= `UD 1'h0; //start 状态发送一个波特周期的低电平
148               SEND_DATA  :                        //发送状态每个波特周期发送一个 bit，
149               begin
150                   case(tx_bit_cnt)
151                       3'h0  :  uart_tx  <= `UD trans_data[0];
152                       3'h1  :  uart_tx  <= `UD trans_data[1];
153                       3'h2  :  uart_tx  <= `UD trans_data[2];
154                       3'h3  :  uart_tx  <= `UD trans_data[3];
155                       3'h4  :  uart_tx  <= `UD trans_data[4];
156                       3'h5  :  uart_tx  <= `UD trans_data[5];
157                       3'h6  :  uart_tx  <= `UD trans_data[6];
158                       3'h7  :  uart_tx  <= `UD trans_data[7];
159                       default: uart_tx  <= `UD 1'h1;
160                   endcase
161               end
162               SEND_STOP  :  uart_tx  <= `UD 1'h1; //发送停止状态 输出 1 个波特周期高电平
163               default    :  uart_tx  <= `UD 1'h1;
164                    // 其他状态默认与空闲状态一致，保持高电平输出
165           endcase
166       end
167       else
168           uart_tx <= `UD 1'h1;
169     end
170
171     endmodule
172
```

7.4.2 串口接收模块设计

串口接收模块是发射模块的逆过程，设计思路区别不大，但是有如下几点需要注意：

1、 接收开始信号，当 rx 下降沿到来后保持几个时钟周期的低电平，表明进入接收 start；

2、 接收数据提取位置，前面讲发射的时候都是在波特周期开始的位置变更数据，接收数据提取时需要在 rx 稳定时刻取数，去波特周期的中间位置取数；

3、 最终输出数据锁存，在最后 1bit 存入寄存器后需要对接收数据锁存，并在之后需要给出数据使能信号，表示输出数据有效；

Module 设计如下：

```
1   `timescale 1ns / 1ps
2   `define UD #1
3
4   module uart_rx # (
5       parameter           BPS_NUM     =    16'd433
6   //  设置波特率为 4800 时，   bit 位宽时钟周期个数:50MHz set 10417  40MHz set 8333
7   //  设置波特率为 9600 时，   bit 位宽时钟周期个数:50MHz set 5208   40MHz set 4167
8   //  设置波特率为 115200 时，bit 位宽时钟周期个数:50MHz set 434    40MHz set 347
9   )
10  (
11        //input ports
12        input             clk,
13        input             rstn,
14        input             uart_rx,
15
16        //output ports
17        output reg [7:0]  rx_data,
18        output reg        rx_en,
19        output            rx_finish
20  );
21
22      // uart rx state machine's state
23      localparam  IDLE        = 4'h0;    //空闲状态，等待开始信号到来.
24      localparam  RECEIV_START = 4'h1;    //接收 Uart 开始信号，低电平一个波特周期.
25      localparam  RECEIV_DATA  = 4'h2;    //接收 Uart 传输数据信号.
26      localparam  RECEIV_STOP  = 4'h3;    //停止状态数据线是高电平，
27      localparam  RECEIV_END   = 4'h4;    //结束中转状态.
28
29      //====================================================================
30      //wire and reg in the module
31      //====================================================================
32      reg    [2:0]        rx_state=0;  //current state of tx state machine. 当前状态
33      reg    [2:0]        rx_state_n=0;//next state of tx state machine.下一个状态
34      reg    [7:0]        rx_data_reg; //接收数据缓冲寄存器
35      reg                 uart_rx_1d;   //save uart_rx one cycle.保存 uart_rx 一个时钟周期
36      reg                 uart_rx_2d;   //save uart_rx one cycle.保存 uart_rx 前两个时钟周期
37      wire                start;     //active when start a byte receive. 检测到 start 信号标志
38      reg    [15:0]       clk_div_cnt; //count for division the clock.波特周期计数器
39
```

```
40      //=============================================================================
41      //logic
42      //=============================================================================
43      //some control single.
44      always @ (posedge clk)
45      begin
46          uart_rx_1d <= `UD uart_rx;
47          uart_rx_2d <= `UD  uart_rx_1d;
48      end
49
50      assign start     = (!uart_rx) && (uart_rx_1d || uart_rx_2d);
51      assign rx_finish = (rx_state == RECEIV_END);
52
53      //division the clock to satisfy baud rate.波特周期计数器
54      always @ (posedge clk)
55      begin
56          if(rx_state == IDLE || clk_div_cnt == BPS_NUM)
57              clk_div_cnt   <= `UD 16'h0;
58          else
59              clk_div_cnt   <= `UD clk_div_cnt + 16'h1;
60      end
61
62      // receive bit data numbers
63      //在接收数据状态中，接收的 bit 位计数，每一个波特周期计数加 1
64      reg    [2:0]      rx_bit_cnt=0;    //the bits number has transmited.
65      always @ (posedge clk)
66      begin
67          if(rx_state == IDLE)
68              rx_bit_cnt <= `UD 3'h0;
69          else if((rx_bit_cnt == 3'h7) && (clk_div_cnt == BPS_NUM))
70              rx_bit_cnt <= `UD 3'h0;
71          else if((rx_state == RECEIV_DATA) && (clk_div_cnt == BPS_NUM))
72              rx_bit_cnt <= `UD rx_bit_cnt + 3'h1;
73          else
74              rx_bit_cnt <= `UD rx_bit_cnt;
75      end
76  //=============================================================================
77  //receive state machine
78  //=============================================================================
79      //状态机状态跳转
80      always @(posedge clk)
81      begin
82          rx_state <= rx_state_n;
83      end
84
85      //状态机状态跳转条件及跳转规律
86      always @ (*)
87      begin
88        case(rx_state)
89            IDLE      :
90            begin
91                if(start) //监测到 start 信号到来，下一状态跳转到 start 状态
92                    rx_state_n = RECEIV_START;
93                else
94                    rx_state_n = rx_state;
95            end
```

```
96          RECEIV_START    :
97          begin
98              if(clk_div_cnt == BPS_NUM)        //已完成接收 start 标志信号
99                  rx_state_n = RECEIV_DATA;
100             else
101                 rx_state_n = rx_state;
102         end
103         RECEIV_DATA    :
104         begin
105             if(rx_bit_cnt == 3'h7 && clk_div_cnt == BPS_NUM)
106                 //已完成 8bit 数据的传输
107                 rx_state_n = RECEIV_STOP;
108             else
109                 rx_state_n = rx_state;
110         end
111         RECEIV_STOP    :
112         begin
113             if(clk_div_cnt == BPS_NUM)  //已完成接收 stop 标志信号
114                 rx_state_n = RECEIV_END;
115             else
116                 rx_state_n = rx_state;
117         end
118         RECEIV_END    :
119         begin
120             if(!uart_rx_1d)
121       //数据线重新被拉低，表示新数据传输又发送 start 标志信号，需要跳转到 start 状态
122                 rx_state_n = RECEIV_START;
123             else
124                 //没有其他状况出现时，回到空闲状态，等待 start 信号的到来
125                 rx_state_n = IDLE;
126         end
127         default    :  rx_state_n = IDLE;
128       endcase
129     end
130
131     // 状态机输出
132     always @ (posedge clk)
133     begin
134         case(rx_state)
135             IDLE        ,
136             RECEIV_START :
137             //在空闲和 start 状态时将接收数据缓冲寄存器和数据使能置位；
138             begin
139                 rx_en <= `UD 1'b0;
140                 rx_data_reg <= `UD 8'h0;
141             end
142             RECEIV_DATA  :
143             begin
144                 if(clk_div_cnt == BPS_NUM[15:1])
145                     //在一个波特周期的中间位置取数据线上传输的数据；
146                     rx_data_reg  <= `UD {uart_rx , rx_data_reg[7:1]};  //以循环
    右移的方式将 uart_rx 数据填入缓冲寄存器的最高位（Uart 传输低位在前，最后一个 bit 刚好是最高位）
147             end
```

```
148             RECEIV_STOP  :
149             begin
150                 rx_en   <= `UD 1'b1;      // 输出使能信号，表示最新的数据输出有效
151                 rx_data <= `UD rx_data_reg; // 将缓冲寄存器的值赋值给输出寄存器
152             end
153             RECEIV_END    :
154             begin
155                 rx_data_reg <= `UD 8'h0;
156             end
157             default:    rx_en <= `UD 1'b0;
158         endcase
159     end
160
161 endmodule
162
```

7.4.3 串口发射控制模块设计

目标 :产生 1S 间隔的触发信号并输出第一个发送字节， busy 的下降沿时输出下一个字节；

Module 如下：

```
1   `timescale 1ns / 1ps
2   `define UD #1
3   module uart_data_gen(
4       input              clk,
5       input              rstn,
6       input      [7:0]    read_data,
7       input              tx_busy,
8       input      [7:0]    write_max_num,
9       output reg [7:0]     write_data,
10      output reg          write_en
11  );
12      // set every second send a string,"====HELLO WORLD==="
13      // 设置约每秒发送一个字符串
14      reg [23:0] time_cnt=0;
15      reg [ 7:0] data_num;
16      always @(posedge clk)
17      begin
18          time_cnt <= `UD time_cnt + 24'd1;
19      end
20
21      // 设置串口发射工作区间
22      reg        work_en=0;
23      reg        work_en_1d=0;
24      always @(posedge clk)
25      begin
26          if(time_cnt == 25'd2048)
27              work_en <= `UD 1'b1;
28          else if(data_num == write_max_num-1'b1)
29              work_en <= `UD 1'b0;
30      end
```

```
32      always @(posedge clk)
33      begin
34          work_en_1d <= `UD work_en;
35      end
36
37      // get the tx_busy's falling edge   获取 tx_busy 的下降沿
38      reg           tx_busy_reg=0;
39      wire          tx_busy_f;
40      always @ (posedge clk) tx_busy_reg <= `UD tx_busy;
41
42      assign tx_busy_f = (!tx_busy) && (tx_busy_reg);
43
44      // 串口发射数据触发信号
45      reg write_pluse;
46      always @ (posedge clk)
47      begin
48          if(!rstn)
49              write_pluse <= `UD 1'b0;
50          else if(work_en)
51          begin
52              if(~work_en_1d || tx_busy_f)
53                  write_pluse <= `UD 1'b1;
54              else
55                  write_pluse <= `UD 1'b0;
56          end
57          else
58              write_pluse <= `UD 1'b0;
59      end
60
61      always @ (posedge clk)
62      begin
63          if(!rstn)
64              data_num    <= `UD 8'h0;
65          else if(~work_en & tx_busy_f)
66              data_num   <= 7'h0;
67          else if(write_pluse)
68              data_num   <= data_num + 8'h1;
69      end
70
71      always @(posedge clk)
72      begin
73          write_en <= `UD write_pluse;
74      end
75
76  //  字符的对应 ASCII 码
77  // H:0x48      E:0x45     L:0x4C    O:0x4F   W:0x57    R:0x52    D:0x44
78      always @ (posedge clk)
79      begin
80          case(data_num)
81              8'h0  ,
82              8'h1  ,
83              8'h2  ,
84              8'h3  : write_data <= `UD 8'h3D;// ASCII code is =
85              8'h4  : write_data <= `UD 8'h48;// ASCII code is H
86              8'h5  : write_data <= `UD 8'h45;// ASCII code is E
```

```
87              8'h6  : write_data <= `UD 8'h4C;// ASCII code is L
88              8'h7  : write_data <= `UD 8'h4C;// ASCII code is L
89              8'h8  : write_data <= `UD 8'h4F;// ASCII code is 0
90              8'h9  : write_data <= `UD 8'h20;// ASCII code is
91              8'ha  : write_data <= `UD 8'h57;// ASCII code is W
92              8'hb  : write_data <= `UD 8'h4F;// ASCII code is O
93              8'hc  : write_data <= `UD 8'h52;// ASCII code is R
94              8'hd  : write_data <= `UD 8'h4C;// ASCII code is L
95              8'he  : write_data <= `UD 8'h44;// ASCII code is D
96              8'hf  ,
97              8'h10 ,
98              8'h11 : write_data <= `UD 8'h3D;// ASCII code is =
99              8'h12 : write_data <= `UD 8'h0d;
100             8'h13 : write_data <= `UD 8'h0a;
101             default :   write_data <= `UD read_data;
102         endcase
103     end
104
105 endmodule
106
```

7.4.4 串口实验顶层模块设计

目标：板子 1s 向串口助手发送一次十进制显示的“===HELLO WORD===”，通过串口助手向板子以十六进制形式发送数字， LED 以十六进制显示亮起。

Uart_data_gen 模块产生一个间隔 1S 钟的触发信号，同时输出第一个发送字节，等待 uart_tx 输出的 busy 下降沿到来，获知 uart_tx 进入空闲状态可发送下一个 byte 时，再次给出串口发送的触发脉冲，并输出下一个字节；

Uart_rx 模块接收到数据后输出一个 rx_en 信号（接收数据使能信号）、一组接收数据信号；接收的数据信号是锁存的，可直接点亮 LED 灯；

具体的 module 实现如下:

```
1   `timescale 1ns / 1ps
2   `define    UD #1
3
4   module uart_top(
5       //input ports
6       input         clk,
7       input         rstn,
8       input         uart_rx,
9
10      //output ports
11      output  [7:0] led,
12      output        uart_tx
13  );
14
```

```
15      parameter       BPS_NUM = 16'd104;
16      //  设置波特率为4800时，  bit位宽时钟周期个数:50MHz set 10417  40MHz set 8333
17      //  设置波特率为9600时，  bit位宽时钟周期个数:50MHz set 5208   40MHz set 4167
18      //  设置波特率为115200时，bit位宽时钟周期个数:50MHz set 434    40MHz set 347 12M set 104
19  //=====================================================================
20  //wire and reg in the module
21  //=====================================================================
22      wire            tx_busy;        //transmitter is free.
23      wire            rx_finish;      //receiver is free.
24      wire    [7:0]   rx_data;        //the data receive from uart_rx.
25      wire    [7:0]   tx_data;
26      wire            tx_en;          //enable transmit.
27      wire            rx_en;
28  //=====================================================================
29  //instance
30  //=====================================================================
31      reg  [7:0] receive_data;
32      always @(posedge clk)  receive_data <= led;
33      uart_data_gen uart_data_gen(
34          .clk                (  clk          ),//input             clk,
35          .rstn               (  rstn         ),//input             rstn,
36          .read_data          (  receive_data ),//input      [7:0]  read_data,
37          .tx_busy            (  tx_busy      ),//input             tx_busy,
38          .write_max_num      (  8'h14        ),//input  [7:0] write_max_num,
39          .write_data         (  tx_data      ),//output reg [7:0]  write_data
40          .write_en           (  tx_en        ) //output reg        write_en
41      );
42
43      //uart transmit data module.
44      uart_tx #(
45           .BPS_NUM           (  BPS_NUM      ) //parameter BPS_NUM  = 16'd434
46       )
47       u_uart_tx(
48          .clk                (  clk          ),// input           clk,
49          .tx_data            (  tx_data      ),// input [7:0]     tx_data,
50          .tx_pluse           (  tx_en        ),// input           tx_pluse,
51          .uart_tx            (  uart_tx      ),// output reg      uart_tx,
52          .tx_busy            (  tx_busy      ) // output          tx_busy
53      );
54
55      //Uart receive data module.
56      uart_rx #(
57           .BPS_NUM           (  BPS_NUM      ) //parameter BPS_NUM  = 16'd434
58       )
59       u_uart_rx (
60          .clk                (  clk          ),// input           clk,
61          .rstn               (  rstn         ),// input           rstn,
62          .uart_rx            (  uart_rx      ),// input           uart_rx,
63          .rx_data            (  rx_data      ),// output reg [7:0]  rx_data,
64          .rx_en              (  rx_en        ),// output reg        rx_en,
65          .rx_finish          (  rx_finish    ) // output          rx_finish
66      );
67      assign led = rx_data;
68
69  endmodule
70
```

7.5 实验现象

用 SSCOM 设置好串口配置，连接对应的串口后有如下现象：

实验现象一：在串口工具中每隔 1S 中打印一次： “===HELLO WORD===” 并回车换行；

实验现象二：在串口工具上以 Hex 格式发送 55；我们可看到 RUNBER 板卡上的 D1,D3,D5,D7 被点亮，D2,D4,D6,D8 为熄灭状态；发送 AA；我们可看到 RUNBER 板卡上的 D2,D4,D6,D8 被点亮， D1,D3,D5,D7 为熄灭状态；

实验现象三：数码管显示串口接收数据（此现象为大家课后自行扩展查看，对模块例化与模块连接做扩展练习）；

8 序列检测器

8.1 实验目的

在连续信号中，检测是否包含特定序列，例如检测“11011000 “中是否包含” 101”

8.2 实验要求

1、 拨码开关 SW0-SW7 作为序列信号输入；

2、 KEY1-KEY3 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、 K8 为序列检测开始和序列检测结束按键，初次按下 KEY8，开始检测，此时 LED8 也会被点亮，显示当前状态，再按一下停止检测，LED8 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。

8.3 实验原理

SW0 ~ SW7 的状态为检测序列；

LED1 ~ LED3 为特定序列；

数码管显示的结果为 LED[3:1]在 SW[7:0]中出现的次数；

8.4 实验源码设计

8.4.1 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[2:0]控制特定序列值；KEY8 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；

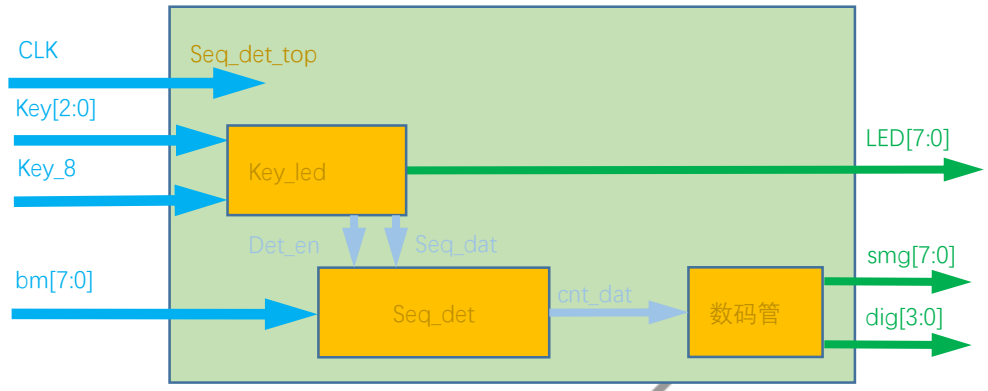
2、 序列对比模块；

由拨码开关提供代检测序列，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代检测序列进行比较；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：



8.4.2 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
6       input            key_8,       // KEY 8 input control detected enable
7       input [2:0]      key_in,      // KEY[2:0] input control detected sequence
8       input [7:0]      bm,          // DIP Switch[7:0] input used to detecting
9       output           key8_led,    // display the detecte status
10      output [6:0]     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                      按键消斗及状态标记
17  =================================================*/
18  wire [2:0]seq_data;
19  key_control key_control
20  (
21      .clk             (  clk             ),
22      .key_8           (  key_8           ),
23      .key_in          (  key_in          ),
24      .key8_led        (  key8_led        ),
25      .key_in_led      (  key_in_led[2:0] ),
26      .seq_data        (  seq_data        )
27  );
28
29  assign key_in_led[6:3] = 4'd0;
30
31  /*=================================================
32                      序列检测
33  =================================================*/
34  wire [3:0]data;
35
```

```
36  seq_det seq_det
37  (
38      .clk              (  clk        ),
39      .key8_led         (  key8_led   ),//检测状态标记
40      .key_in_led       (  seq_data   ),//待检测序列
41      .bm               (  bm         ),//输入序列
42      .data             (  data       )
43  );
44  /*=================================================
45                      时钟分频
46  =================================================*/
47  wire clk_100khz;
48  div_clk div_clk
49  (
50      .clk            (  clk       ),
51      .clk_100khz     (  clk_100khz)
52  );
53  /*=================================================
54                      数码管显示
55  =================================================*/
56  reg  [1:0]sel=0;
57  wire [3:0]dig0;
58  wire [7:0]smg0;
59
60  always @(posedge clk_100khz)
61  begin
62      sel <= `UD sel+1'b1;
63  end
64  // Digital Tube 0 output
65  seq_control seq_control_0
66  (
67      .sel(sel),
68      .key(data),
69      .dig(dig0),
70      .smg(smg0)
71  );
72  // Digital Tube 1 output
73  wire [3:0]dig1;
74  wire [7:0]smg1;
75  seq_control seq_control_1
76  (
77      .sel(sel),
78      .key(4'd0),
79      .dig(dig1),
80      .smg(smg1)
81  );
82  // Digital Tube 2 output
83  wire [3:0]dig2;
84  wire [7:0]smg2;
85  seq_control seq_control_2
86  (
87      .sel(sel),
88      .key(4'd0),
89      .dig(dig2),
90      .smg(smg2)
91  );
```

```
93   wire [3:0]dig3;
94   wire [7:0]smg3; // Digital Tube 3 output
95   seq_control seq_control_3
96   (
97       .sel(sel),
98       .key(4'd0),
99       .dig(dig3),
100      .smg(smg3)
101  );
102  // display by Digital Tube
103  always @(posedge clk_100khz)
104  begin
105      if(sel==2'b00)
106          dig <= `UD dig0;
107      else if(sel==2'b01)
108          dig <= `UD dig1;
109      else if(sel==2'b10)
110          dig <= `UD dig2;
111      else if(sel==2'b11)
112          dig <= `UD dig3;
113  end
114
115  always @(posedge clk_100khz)
116  begin
117      if(sel==2'b00)
118          smg <= `UD smg0;
119      else if(sel==2'b01)
120          smg <= `UD smg1;
121      else if(sel==2'b10)
122          smg <= `UD smg2;
123      else if(sel==2'b11)
124          smg <= `UD smg3;
125  end
126
127  endmodule
128
```

8.4.3 按键 LED 控制模块

```
1    `timescale 1ns / 1ps
2    `define UD #1
3    module key_control
4    (
5        input               clk,       // input clock
6        input               key_8,     // KEY 8 input
7        input [2:0]         key_in,    // KEY[2:0] input
8        output reg          key8_led,  // LED8 control sigal
9        output reg [2:0]    key_in_led,// LED[2:0] control signal
10       output reg [2:0]    seq_data   // Sequence to be detected
11   );
12
```

```
13  /*=================================================
14                      按键消斗
15  =================================================*/
16  wire [2:0]key_out;
17  btn_deb #(
18     .BT_WIDTH(4'd3)
19  ) u_btn_deb_key1 (
20      .clk(clk),
21      .btn_in(key_in),
22      .btn_out(key_out)
23  );
24
25  btn_deb #(
26     .BT_WIDTH(4'd1)
27  ) u_btn_deb_key8 (
28      .clk(clk),
29      .btn_in(key_8),
30      .btn_out(key_8_out)
31  );
32
33  /*=================================================
34  =================================================*/
35  reg [2:0]key_out_reg;
36  reg key_8_out_reg;
37
38  always @(posedge clk)
39  begin
40      key_out_reg <= `UD key_out;
41      key_8_out_reg<= `UD key_8_out;
42  end
43
44  reg key_8_flag=0;
45  always @(posedge clk)
46  begin
47      if(!key_8_out && key_8_out_reg)
48          key_8_flag <= `UD ~key_8_flag;
49      else
50          key_8_flag <= `UD key_8_flag;
51  end
52
53  always @(posedge clk)
54  begin
55      key8_led <= `UD key_8_flag;
56  end
57
58  reg [2:0]key_flag=3'b000;
59  always @(posedge clk)
60  begin
61      if(key_8_flag==1'b0)
62          key_flag[0] <= `UD 1'b0;
63      else if(!key_out[0] && key_out_reg[0])
64          key_flag[0] <= `UD ~key_flag[0];
65      else
66          key_flag[0] <= `UD key_flag[0];
67  end
68
```

```
69  always @(posedge clk)
70  begin
71      if(key_8_flag==1'b0)
72          key_flag[1] <= `UD 1'b0;
73      else if(!key_out[1] && key_out_reg[1])
74          key_flag[1] <= `UD ~key_flag[1];
75      else
76          key_flag[1] <= `UD key_flag[1];
77  end
78
79  always @(posedge clk)
80  begin
81      if(key_8_flag==1'b0)
82          key_flag[2] <= `UD 1'b0;
83      else if(!key_out[2] && key_out_reg[2])
84          key_flag[2] <= `UD ~key_flag[2];
85      else
86          key_flag[2] <= `UD key_flag[2];
87  end
88  always @(posedge clk)
89  begin
90      key_in_led <= `UD key_flag;
91  end
92
93  //seq_data
94  always @(posedge clk)
95  begin
96      if(key_8_flag)
97          seq_data <= `UD key_in_led;
98  end
99
100 endmodule
101
```

8.4.4 序列检测模块设计

```
1   `timescale 1ns / 1ps
2   `define UD #1
3   module seq_det
4   (
5       input            clk,
6       input            key8_led,//检测状态标记
7       input [2:0]       key_in_led,//待检测序列
8       input [7:0]       bm,//输入序列
9       output reg [3:0]  data
10  );
11
12  // 8bit data detecte 3 bit sequence, we need compare six numbers
13  reg [5:0]flag;
14
```

```
15  always @(posedge clk)
16  begin
17      if(!key8_led&&bm[7:5]==key_in_led)
18          flag[0] <= `UD 1'b1;
19      else
20          flag[0] <= `UD 1'b0;
21  end
22
23  always @(posedge clk)
24  begin
25      if(!key8_led&&bm[6:4]==key_in_led)
26          flag[1] <= `UD 1'b1;
27      else
28          flag[1] <= `UD 1'b0;
29  end
30
31  always @(posedge clk)
32  begin
33      if(!key8_led&&bm[5:3]==key_in_led)
34          flag[2] <= `UD 1'b1;
35      else
36          flag[2] <= `UD 1'b0;
37  end
38
39  always @(posedge clk)
40  begin
41      if(!key8_led&&bm[4:2]==key_in_led)
42          flag[3] <= `UD 1'b1;
43      else
44          flag[3] <= `UD 1'b0;
45  end
46
47  always @(posedge clk)
48  begin
49      if(!key8_led&&bm[3:1]==key_in_led)
50          flag[4] <= `UD 1'b1;
51      else
52          flag[4] <= `UD 1'b0;
53  end
54
55  always @(posedge clk)
56  begin
57      if(!key8_led&&bm[2:0]==key_in_led)
58          flag[5] <= `UD 1'b1;
59      else
60          flag[5] <= `UD 1'b0;
61  end
62
63  always @(posedge clk)
64  begin
65      data <= `UD flag[5] + flag[4] + flag[3] + flag[2] + flag[1] + flag[0];
66  end
67
68  endmodule
69
```

8.5 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[7：0]）；

2、调整固定序列，通过轻触按键改变 LED 状态（LED[2:0]）；

3、按下轻触按键 KEY8，进入检测，查看数码管显示的统计结果；

4、按下轻触按键 KEY8，退出检测，重新执行前面三个步骤；

实验现象

当 SW[7:0]=8'b10101010;LED[2:0]=3'b101 时，按下 Key8 后数码管显示数字 3；

当 SW[7:0]=8'b10101010;LED[2:0]=3'b100 时，按下 Key8 后数码管显示数字 0；

9 密码锁

9.1 实验目的

利用 RUNBER 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；

9.2 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。

BM1- BM8 设置 4 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，BM[4:5]设置第三位，BM[6:7]设置第四位。所以密码是由 0，1，2，3 组成的四位数。

KEY0-KEY3 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。

K7 作为确认按键，按下 K7，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 K6 清零，按下后数码管显示 0000，可以重新输密码。

9.3 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；

9.4 实验源码

根据需求我们需要如下三个子模块：

①按键控制模块；

1、对 6 个按键输入信号均做消抖处理，2、KEY7 和 KEY6 取下降沿输出，3、KEY[3：0]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、KEY6 下降沿触发进入重置状态；3、实时显示 4 位输入密码；

密码验证状态：

1、KEY7 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；

KEY7 下降沿触发使能工作； KEY7 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。

9.4.1 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，Runber 板卡输入时钟为 12MHz
key	4	输入	轻触按键输入信号，Runber 板卡上 K1~K4 输入
enter	1	输入	密码确认比对信号，Runber 板卡上 K7 输入
init	1	输入	密码确认重新输入信号，Runber 板卡上 K6 输入
sw	8	输入	密码设置输入信号，Runber 板卡上 SW1~8 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：

```
1   `timescale 1ns / 1ps
2   `define UD #1
3   module lock_top(
4       input          clk,
5       input  [3:0]   key,
6       input          enter,
7       input          init,
8       input  [7:0]   sw,
9
10      output [7:0]   smg,
11      output [3:0]   dig
12  );
13
```

```
14      wire            enter_trig;
15      wire            init_trig;
16      wire [7:0]      ctrl;
17      wire            com_result;
18
19      key_ctl key_ctl(
20          .clk          (  clk         ),//input           clk,
21          .key          (  key         ),//input     [3:0] key,
22          .enter        (  enter       ),//input           enter,
23          .init         (  init        ),//input           init,
24
25          .enter_trig   (  enter_trig  ),//output          enter_trig,
26          .init_trig    (  init_trig   ),//output          init_trig,
27          .ctrl         (  ctrl        ) //output    [7:0] ctrl
28      );
29
30      compare compare(
31          .clk          (  clk         ),//input       clk,
32          .sw           (  sw          ),//input [7:0] sw,
33          .ctrl         (  ctrl        ),//input [7:0] ctrl,
34          .enter_trig   (  enter_trig  ),//input       enter_trig,
35
36          .com_result   (  com_result  ) //output      com_result
37      );
38
39      seq_display(
40          .clk          (  clk         ),//input             clk,
41          .enter_trig   (  enter_trig  ),//input             enter_trig,
42          .init_trig    (  init_trig   ),//input             init_trig,
43          .com_result   (  com_result  ),//input             com_result,
44          .ctrl         (  ctrl        ),//input      [7:0]  ctrl,
45
46          .smg          (  smg         ),//output reg [7:0]  smg,
47          .dig          (  dig         ) //output reg [3:0]  dig
48      );
49
50  endmodule
51
```

9.4.2 按键控制设计

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，Runber 板卡输入时钟为 12MHz
key	4	输入	轻触按键输入信号，Runber 板卡上 K1~K4 输入
enter	1	输入	密码确认比对信号，Runber 板卡上 K7 输入
init	1	输入	密码重新输入信号，Runber 板卡上 K6 输入
enter_trig	1	输出	确认对比触发脉冲信号输出
init_trig	1	输出	进入重新输入密码触发脉冲信号输出

ctrl	8	输出	输入的 4 个密码输出，与设置密码的位定义一致

Module 设计如下：

```
1   `timescale 1ns / 1ps
2   `define UD #1
3   module key_ctl(
4       input               clk,
5       input      [3:0]    key,
6       input               enter,
7       input               init,
8
9       output              enter_trig,
10      output              init_trig,
11      output     [7:0]    ctrl
12  );
13
14      wire [5:0] btn_deb;
15      // 按键消抖
16      btn_deb#(
17          .BTN_WIDTH   (  4'd6             ) //parameter  BTN_WIDTH = 4'd8
18      ) U_btn_deb
19      (
20          .clk         (  clk              ),//input                   clk,
21          .btn_in      (  {enter,init,key} ),//input [BTN_WIDTH-1:0] btn_in,
22
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      reg [1:0]  key3_push_cnt=2'd0;
29      reg [1:0]  key4_push_cnt=2'd0;
30
31      reg btn1_deb_1d,btn2_deb_1d,btn3_deb_1d,btn4_deb_1d;
32      reg enter_deb_1d,init_deb_1d;
33
34      //确认和重新输入密码触发脉冲信号获取
35      assign enter_trig = ~btn_deb[5] & enter_deb_1d;
36      assign init_trig = ~btn_deb[4] & init_deb_1d;
37
38      always @(posedge clk)
39      begin
40          btn1_deb_1d  <= `UD btn_deb[0];
41          btn2_deb_1d  <= `UD btn_deb[1];
42          btn3_deb_1d  <= `UD btn_deb[2];
43          btn4_deb_1d  <= `UD btn_deb[3];
44          init_deb_1d  <= `UD btn_deb[4];
45          enter_deb_1d <= `UD btn_deb[5];
46      end
47
```

```
48      always @(posedge clk)
49      begin
50          if(~btn_deb[4] & init_deb_1d)
51              key1_push_cnt <= `UD 2'd0;
52          else if(~btn_deb[0] & btn1_deb_1d)
53          begin
54              key1_push_cnt <= `UD key1_push_cnt + 2'd1;
55          end
56      end
57
58      always @(posedge clk)
59      begin
60          if(~btn_deb[4] & init_deb_1d)
61              key2_push_cnt <= `UD 2'd0;
62          else if(~btn_deb[1] & btn2_deb_1d)
63          begin
64              key2_push_cnt <= `UD key2_push_cnt + 2'd1;
65          end
66      end
67
68      always @(posedge clk)
69      begin
70          if(~btn_deb[4] & init_deb_1d)
71              key3_push_cnt <= `UD 2'd0;
72          else if(~btn_deb[2] & btn3_deb_1d)
73          begin
74              key3_push_cnt <= `UD key3_push_cnt + 2'd1;
75          end
76      end
77
78      always @(posedge clk)
79      begin
80          if(~btn_deb[4] & init_deb_1d)
81              key4_push_cnt <= `UD 2'd0;
82          else if(~btn_deb[3] & btn4_deb_1d)
83          begin
84              key4_push_cnt <= `UD key4_push_cnt + 2'd1;
85          end
86      end
87      //密码输出：{第四位密码，第三位密码，第二位密码，第一位密码}，每位密码占 2bit
88      assign ctrl = {key4_push_cnt,key3_push_cnt,key2_push_cnt,key1_push_cnt};
89
90  endmodule
91
```

9.4.3 对比模块设计

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，Runber 板卡输入时钟为 12MHz
sw	8	输入	设置 4 位密码

enter_trig	1	输入	确认对比触发脉冲信号输入
ctrl	8	输入	输入的 4 位密码,
com_result	1	输出	输出对比结果

Module 设计

```
1   `timescale 1ns / 1ps
2   `define UD #1
3   module compare(
4       input       clk,
5       input [7:0]  sw,
6       input [7:0]  ctrl,
7       input       enter_trig,
8
9       output      com_result
10  );
11
12  //=============================================================================
13      //锁存当前的输入密码;
14      reg [7:0] ctrl_1d;
15      always @(posedge clk)
16      begin
17          if(enter_trig)
18              ctrl_1d <= `UD ctrl;
19      end
20
21      assign com_result = (ctrl_1d == sw);
22  endmodule
23
```

9.4.4 显示模块设计

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，Runber 板卡输入时钟为 12MHz
enter_trig	1	输入	确认对比触发脉冲信号输入
init_trig	1	输入	重新输入密码触发脉冲信号输入
com_result	1	输入	输入比对模块的对比结果
ctrl	8	输入	输入的 4 位密码,
smg	8	输出	数码管段选输出
dig	4	输出	数码管位选输出

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；

```
1   `timescale 1ns / 1ps
2   `define UD #1
3   module seq_display(
4       input              clk,
5       input              enter_trig,
6       input              init_trig,
7       input              com_result,
8       input      [7:0]   ctrl,
9
10      output reg  [7:0]   smg,
11      output reg  [3:0]   dig
12  );
13
14      //=============================================================================
15      //显示状态区分
16      reg      seq_status= 1'b0;
17      always @(posedge clk)
18      begin
19          if(enter_trig)
20              seq_status <= `UD 1'b1;
21          else if(init_trig)
22              seq_status <= `UD 1'b0;
23      end
24
25      //=============================================================================
26      //数码管显示荣控制
27      reg [3:0] key0_cnt=4'd0,key1_cnt=4'd0,key2_cnt=4'd0,key3_cnt=4'd0;
28      always @(posedge clk)
29      begin
30          if(seq_status)
31          begin
32              if(com_result)
33              begin
34                  key0_cnt <= `UD 4'd8;
35                  key1_cnt <= `UD 4'd8;
36                  key2_cnt <= `UD 4'd8;
37                  key3_cnt <= `UD 4'd8;
38              end
39              else
40              begin
41                  key0_cnt <= `UD 4'd7;
42                  key1_cnt <= `UD 4'd7;
43                  key2_cnt <= `UD 4'd7;
44                  key3_cnt <= `UD 4'd7;
45              end
46          end
47          else
48          begin
49              key0_cnt <= `UD {2'd0,ctrl[1:0]};
50              key1_cnt <= `UD {2'd0,ctrl[3:2]};
51              key2_cnt <= `UD {2'd0,ctrl[5:4]};
52              key3_cnt <= `UD {2'd0,ctrl[7:6]};
53          end
54      end
55
```

```
56      //=================================================
57      //                  时钟分频
58      wire clk_100khz;
59      div_clk div_clk (
60          .clk       (clk),
61          .clk_100khz (clk_100khz)
62      );
63      //=================================================
64      //                  数码管显示
65      reg  [1:0]sel=0;
66      always @(posedge clk_100khz)
67      begin
68          sel <= `UD sel+1'b1;
69      end
70
71      wire [3:0]dig0;
72      wire [7:0]smg0;
73      seq_control seq_control_0
74      (
75          .sel(2'd3),
76          .key(key0_cnt),
77          .dig(dig0),
78          .smg(smg0)
79      );
80
81      wire [3:0]dig1;
82      wire [7:0]smg1;
83      seq_control seq_control_1
84      (
85          .sel(2'd2),
86          .key(key1_cnt),
87          .dig(dig1),
88          .smg(smg1)
89      );
90
91      wire [3:0]dig2;
92      wire [7:0]smg2;
93      seq_control seq_control_2
94      (
95          .sel(2'd1),
96          .key(key2_cnt),
97          .dig(dig2),
98          .smg(smg2)
99      );
100
101     wire [3:0]dig3;
102     wire [7:0]smg3;
103     seq_control seq_control_3
104     (
105         .sel(2'd0),
106         .key(key3_cnt),
107         .dig(dig3),
108         .smg(smg3)
109     );
110
```

```
111     always @(posedge clk_100khz)
112     begin
113         if(sel==2'b00)
114             dig <= `UD dig0;
115         else if(sel==2'b01)
116             dig <= `UD dig1;
117         else if(sel==2'b10)
118             dig <= `UD dig2;
119         else if(sel==2'b11)
120             dig <= `UD dig3;
121     end
122 
123     always @(posedge clk_100khz)
124     begin
125         if(sel==2'b00)
126             smg <= `UD smg0;
127         else if(sel==2'b01)
128             smg <= `UD smg1;
129         else if(sel==2'b10)
130             smg <= `UD smg2;
131         else if(sel==2'b11)
132             smg <= `UD smg3;
133     end
134 
135 endmodule
136 
```

9.5 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[7：0]）；

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 KEY7，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 KEY6，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[7:0]=8’b10101010;当输入密码状态时显示 4 个 2，按下 Key7 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 Key7 后数码管显示数字 7777；按下 Key6 后重新调整密码，进入输入密码状态；

10 数字钟

10.1 实验目的

设计一个具有计时功能和校时功能的数字时钟。

10.2 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。

10.3 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：

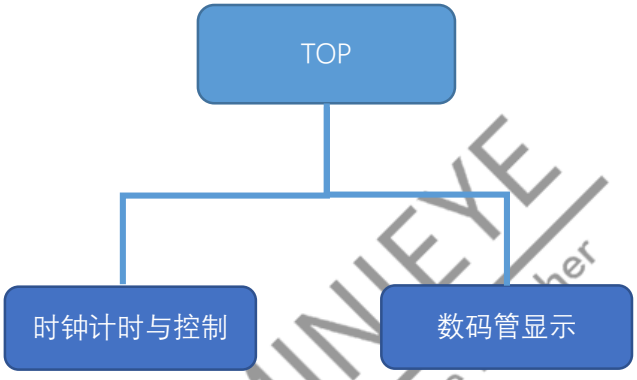
秒 进 分	LED 灯亮灭一次计数为 1，亮灭的一个周期为 1S,当 LED 亮灭 60 次时分钟计数个位加 1；
分个位 进 分十位	当个位计数到 9 时，秒到分再次进位时，十位需要加一，个位置位为 0；
分进时	1 小时=60 分钟；当分钟计时为 59，并且秒到分再次进位时，时钟计数个位加 1；分钟计数置位为 0；
时个位 进 时十位	显示为 24 小时制，当时钟计数十位小于 2 时，时钟个位个位计数到 9 时，分到时再次进位时，十位加 1；
24 小时制计满时归零	当时钟计数十位等于 2 时，时钟个位个位计数到 3 时，分到时再次进位时，时，分，秒三部分计数均归零；

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。



10.4 实验源码

10.4.1 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，Runber 板卡输入时钟为 12MHz
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[3:0]  dig,
9       output[7:0]  smg
10  );
11
12      parameter CLK_FRE = 26'd12_000_000;
13
```

```
14      /*=====================================================
15                      复位信号的产生
16      =====================================================*/
17      reg [4:0] rstn_cnt=0;
18      always @(posedge clk)
19      begin
20          if(rstn_cnt==5'h1f)
21              rstn_cnt <= `UD rstn_cnt;
22          else
23              rstn_cnt <= `UD rstn_cnt + 1'b1;
24      end
25
26      wire rstn;
27      assign rstn = rstn_cnt[4];
28      /*=====================================================
29                          数字时钟的产生和控制
30      =====================================================*/
31      wire [3:0] hour_h,hour_l,minutes_h,minutes_l;
32      wire [2:0] dig_ctl;
33      watch_data_gen #(
34          .CLK_FRE        (  CLK_FRE  )//parameter  CLK_FRE=26'd12_000_000
35      ) u_watch_data_gen
36      (
37          .clk            (  clk            ), //input clk,
38          .rstn           (  rstn           ), //input rstn,
39          .key            (  key            ), //input [2:0]key,
40          .hour_h_o       (  hour_h         ), //output reg [3:0]hour_h_o,
41          .hour_l_o       (  hour_l         ), //output reg [3:0]hour_l_o,
42          .minutes_h_o    (  minutes_h      ), //output reg [3:0]minutes_h_o,
43          .minutes_l_o    (  minutes_l      ), //output reg [3:0]minutes_l_o,
44          .second_led     (  led            ), //output reg second_led,
45          .state_flag     (  dig_ctl        )  //output reg [2:0]state_flag
46      );
47
48      display_ctl #(
49           .CLK_FRE       (  CLK_FRE  )//parameter CLK_FRE = 26'd12_000_000
50      )
51      (
52           .clk           (  clk            ),//input            clk,
53           .dig_ctl       (  dig_ctl        ),//input      [2:0]  dig_ctl,
54           .hour_h        (  hour_h         ),//input      [3:0]  hour_h,
55           .hour_l        (  hour_l         ),//input      [3:0]  hour_l,
56           .minutes_h     (  minutes_h      ),//input      [3:0]  minutes_h,
57           .minutes_l     (  minutes_l      ),//input [3:0]       minutes_l,
58           .sec_en        (  led            ),//input             sec_en,
59
60           .dig           (  dig            ),//output reg [3:0]  dig,
61           .smg           (  smg            ) //output reg [7:0]  smg
62      );
63
64  endmodule
65
```

10.4.2 时钟计时与控制模块设计

10.4.2.1 计时控制顶层设计

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，Runber 板卡输入时钟为 12MHz
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、时钟校准控制（分钟低位示例，其他为调整类似，进位值需要注意变化）

```
always @(posedge clk)
begin
    if(key_cnt!=3'd1)                //校准前将分钟低位和输出值保持一致
        minutes_l_fix <= `UD minutes_l;
    else if(key_cnt==3'd1)           //当处于分钟低位校准状态时，执行对应调整
    begin
        if(up_pluse)                 //按下"+"按键,校准数值加 1;
        begin
            if(minutes_l_fix == 4'd9)//若此时校准值已经为 9 时，则校准值变为 0
                minutes_l_fix <= `UD 4'd0;//"+"
            else
                minutes_l_fix <= `UD minutes_l_fix + 1'b1;//"+"
        end
        else if(down_pluse)           //按下"-"按键,校准数值减 1;
        begin
            if(minutes_l_fix == 4'd0)//若此时校准值已经为 0 时，则校准值变为 9
                minutes_l_fix <= `UD 4'd9;
            else
                minutes_l_fix <= `UD minutes_l_fix - 1'b1;
        end
        else
            minutes_l_fix <= `UD minutes_l_fix;
    end
    else
        minutes_l_fix <= `UD minutes_l_fix;
end
```

2、 时钟进位控制（分钟低位为例，前一级进位信号触发累加，自身进位信号触发归零）

```
1       wire min_l_carry;
2       assign min_l_carry = (sec_carry == 1'b1) && (minutes_l==4'd9);
3       //minutes_l gen
4       always @(posedge clk)
5       begin
6           if(!rstn)//初始值为 0
7               minutes_l <= `UD 4'd0;
8           else if(key_cnt==3'd1)//校准时，分钟低位为校准值
9               minutes_l <= `UD minutes_l_fix;
10          else if(min_l_carry)   //9 分 59 秒产生进位，低位赋值为 0
11              minutes_l <= `UD 4'd0;
12          else if(sec_carry)     //60 秒产生分钟的低位进位
13              minutes_l <= `UD minutes_l +1'b1;
14      end
15
```

10.4.2.2 按键控制模块设计

按键控制模块功能主要是接收按键输入信号触发对时钟计数进行校准调整；通过 KEY0 调整校准位，KEY1 控制加，KEY2 控制减；module 关键点如下（key_out 为按键消抖输出）：

```
1       /*================================================
2           //key[0] -> k1 ;用于校准标记 key_cnt = 3'd0 用于正常显示；
3           key_cnt = 3'd1 用于分钟低位校准； 3'd2 用于分钟高位校准；
4           key_cnt = 3'd3 用于时钟低位校准； 3'd4 用于时钟高位校准；
5       ================================================*/
6       reg [2:0]key_out_reg=3'd0;
7       always @(posedge clk)
8       begin
9           key_out_reg <= `UD key_out;
10      end
11
12      reg [2:0]key_cnt=3'd0;
13      always @(posedge clk)
14      begin
15          if(key_cnt==3'd4 && (!key_out[0] && key_out_reg[0]))
16              key_cnt <= `UD 3'd0;
17          else if(!key_out[0] && key_out_reg[0])
18              key_cnt <= `UD key_cnt + 1'b1;
19      end
20
21      assign dig_ctl = key_cnt;  //输出到其他模块
22      /*================================================
23                  key[1] 用于"+"；key[2] 用"-"
24      ================================================*/
25      always @(posedge clk)
26      begin
27          up_pluse   <= `UD !key_out[1] && key_out_reg[1];
28          down_pluse <= `UD !key_out[2] && key_out_reg[2];
29      end
30
```

10.4.3 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);
                2'd3:begin if(sec_en) dig = ~(4'b1000 );else dig = ~(4'b0000); end
            default:dig = ~(4'b0000);
            endcase
        default:dig = ~(4'b0000);
    endcase
```

10.5 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 KEY0，进入校准模式，第一次按下 KEY0，进入分钟低位计数校准调节，之后再次按下 KEY0，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 KEY0 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 KEY1 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 KEY2 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；

11 频率计

11.1 实验目的

对一个方波信号进行频率测量；

11.2 实验要求

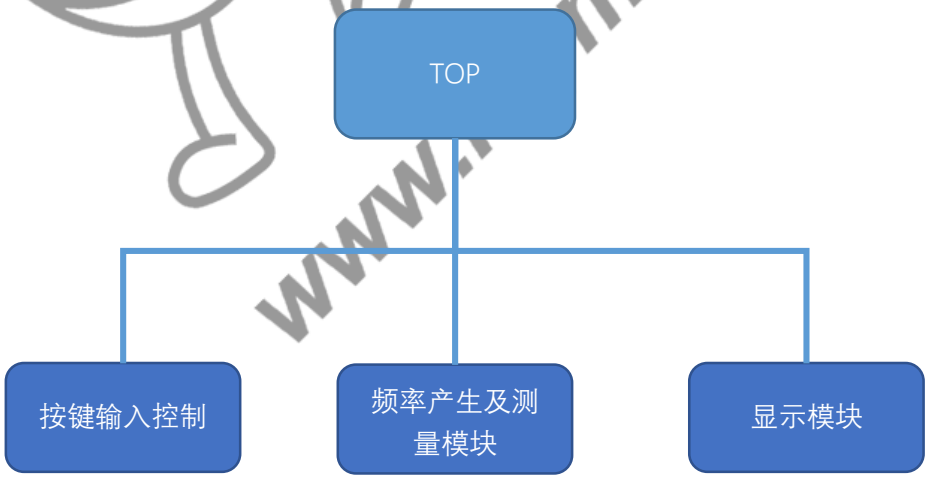
将 50MHZ 的系统时钟信号分频，得到一个低频信号，该信号可以通过按键 KEY0 设定 16 种不同的频率值,并用所设计的频率计测量所产生的低频信号的频率。每 4s 测量 1 次。其中,1s 用于测量，3s 用于显示。测量时，读数变化;测量结束后，结果显示 3s,之后重新测量。当测量频率大于 9999Hz 时,显示 9999。

11.3 实验原理

根据上述需求可总结出顶层模块的输入输出信号如下：

信号	位宽	方向	描述
clk	1	输入	外部输入时钟，Runber 板卡输入时钟为 12MHz
key	1	输入	信号源切换信号输入（轻触按键）
dig	4	输出	数码管位选输出
smg	8	输出	数码管段选输出

整体的功能模块划分可按下图进行设计：



11.3.1 按键输入控制模块

按键消抖，按键记数，通过按键的输入次数调整控制待测试时钟频率，计数范围为 0～15，预设 15 种频率信号待测试。

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，Runber 板卡输入时钟为 12MHz
key	1	输入	信号源切换信号输入（轻触按键）
key_times	4	输出	待测试信号选择序号输出

11.3.2 频率产生及测量模块

频率测量方法：在 1s 时间内统计时钟上升沿个数（注意十进制溢出情况）。统计好结果后有 3S 钟的时间保持显示测量结果，3s 后才可测试下一组频率；

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，Runber 板卡输入时钟为 12MHz
key_times	4	输入	待测试信号选择序号输入
seg0	4	输出	频率计时（Hz）统计个位输出
seg1	4	输出	频率计时（Hz）统计十位输出
seg2	4	输出	频率计时（Hz）统计百位输出
seg3	4	输出	频率计时（Hz）统计千位输出

11.3.3 数码管显示控制。

与前面数码管控制基本相同，将每位要显示的数值传输到模块即可；

11.4 实验源码

11.4.1 顶层模块

顶层模块主要是将三个模块关联起来；信号连接关系如下（代码详情请看源码文件）：

其中 key_times 表示按键按下次数，Fre_Hz 表示测试的时钟频率，单位是 Hz，总共有 4 组信号分别是个位，十位，百位，千位；

11.4.2 按键输入控制模块

此模块的设计主要是针对按键按下的次数做统计，并传递给频率测量模块，代码相对较简单，这里就不再解析，详情请看源码文件；

11.4.3 频率产生及测量模块

11.4.3.1 低频时钟产生

我们只有 4 位数码管，显示频率的最大值为 9999Hz；产生的频率值有 16 种；Runber 板卡的输入时钟为 12MHz，对时钟周期计数达到 24'd12000000 归零时可产生一个 1S 周期（1Hz）的技数器；对时钟周期计数达到 11'd1200 归零时可产生一个 100uS 周期（10KHz）的技数器；低频时钟产生我们采用一个 24bit 位宽的计数器，取计数器中的某一位来做待测试时钟，当取第 24bit 时，频率为 0.715Hz；当取第 23bit 时，频率为 1.43Hz；当取第 12bit 时，频率为 2929.69Hz；当取第 10bit 时，频率为 11718.75Hz；这种取值的方式将会有两组频率超出范围；信号获取方式如下：

```
reg [23:0] clk_gen;
always @(posedge clk)
begin
    clk_gen <=`UD clk_gen + 1'b1;
end

wire    freq_gen;
assign    freq_gen = clk_gen[23-key_times];
```

11.4.3.2 频率测量计数

在原理部分已描述我们是通过 1S 的时间段统计待测试信号的上升沿数量来得到信号的频率值；故而要先得到待测信号的上升沿，1S 的时间宽度的脉冲信号；

待测信号的上升沿获取方式如下：

```
reg freq_gen_reg;
always @(posedge clk)
begin
    freq_gen_reg <= `UD freq_gen;
end

wire freq_risedge;
assign freq_risedge = !freq_gen_reg && freq_gen;
```

1S 时间脉宽信号产生如下（1S 的周期信号产生在前面实验中有很多类似的实现方法，这里就不单独点出了）：

```
wire test_flag;
reg [1:0]flag_cnt=2'd0;
always @(posedge clk_1hz)
begin
    flag_cnt <=`UD flag_cnt + 1'b1;
end

assign test_flag = (flag_cnt==2'd0);
```

统计计数的方式如下代码(个位与十位示例，百位与千位基本类似，详情请查看源码文件):

```
1   wire seg0_carry;  //个位溢出
2   wire seg1_carry;  //十位溢出
3   wire seg2_carry;  //百位溢出
4   wire seg3_carry;  //千位溢出
5   //test_flag 是一个脉冲为 1s 的测试使能信号，freq_risedge 为待测试信号上升沿
6   assign seg0_carry = (seg0 == 4'd9) && freq_risedge && test_flag;
7   assign seg1_carry = (seg1 == 4'd9) && seg0_carry;
8   assign seg2_carry = (seg2 == 4'd9) && seg1_carry;
9   assign seg3_carry = (seg3 == 4'd9) && seg2_carry;
10
11  always @(posedge clk)     //频率的个位
12  begin
13      if(seg3_carry)         //当我们测量达到最大值时，将赋值为 9
14          seg0 <= `UD 4'd9;
15      else if(seg0_carry)   //溢出
16          seg0 <= `UD 4'd0;
17      else if(freq_risedge && test_flag)//
18          seg0 <= `UD seg0 + 1'b1;
19      else if(test_start)    //每次测量前将数码管赋值为 0
20          seg0 <= `UD 4'd0;
21  end
22
23  always @(posedge clk)   //频率的十位
24  begin
25      if(seg3_carry)      //当我们测量达到最大值时，将赋值为 9
26          seg1 <= `UD 4'd9;
27      else if(seg1_carry) //当前位计数到 9，溢出处理
28          seg1 <= `UD 4'd0;
29      else if(seg0_carry) //低位进位信号触发当前位计数加 1
30          seg1 <= `UD seg1 + 1'b1;
31      else if(test_start)//每次测量前将数码管赋值为 0
32          seg1 <= `UD 4'd0;
33  end
34
```

11.5 实验现象

加载后的显示结果为：数码管显示从 0000,之后进入 1s 钟的测量时间，3s 钟的显示时间周期性变化；

按轻触按键 KEY0，调整待测试信号的频率，随着按下次数增多，待测试信号的频率上升，当按下次数到 16 次后又回到最初的低频信号；

12 反应测试器

12.1 实验目的

设计一个反应测量仪，用于测量人体反应时间。

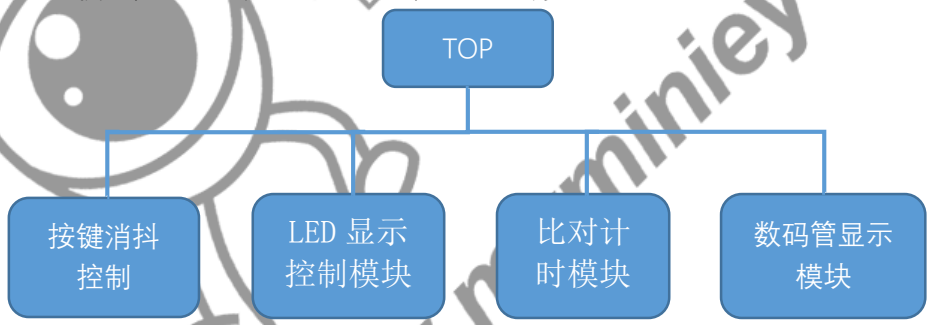
12.2 实验要求

LED0-LED7 八个 LED 灯随机点亮一个，当看到灯亮后立刻按下相应的 KEY。测量灯亮起到按键按下这段时间，然后将该时间以十进制的形式显示在数码管上，以 ms 为单位。

12.3 实验原理

设计思路：

1、 产生一个随机的 led 状态，定义一个移位流水灯。触发开始后 3 秒取出一个固定的亮灯情况，并锁存住一组 led 显示状态，触发开始计时。
2、 以 ms 为单位进行计时，以触发开始信号复位计时器，计时器分个位，十位，百位，千位分别累加进行计数。
3、 数码管显示，重新测量信号触发归零，
4、 比对模块，对按键触发进行比较，比较正确退出。



12.4 实验源码

12.4.1 顶层模块

顶层模块设计关联输入输出信号，以及模块之间的信号互联，此实验对应外部的输出输出信号有：时钟，按键，led 和数码管；

前面已分析到顶层例化有 4 个模块，按键消抖这里不再重复介绍，按键输入信息传递到比对计时模块进行比对，LED 模块接受触发信号得到一组 LED 控制信号（点亮一个 LED 灯），并在 LED 灯点亮时输出一个计时开始信号以及 LED 控制信号给比对计时模块进行计时以及

检测按键是否按对，比对计时模块计时完成后将计时信息以及计时使能信号传递给数码管显示模块进行显示；由于板上仅有 8 个按键，刚好映射 8 个 LED，故而检测开始信号采用组合的方式产生，用 KEY0 与 KEY7 同时按下为检测开始信号；

Module 设计如下：

```
`timescale 1ns / 1ps
`define UD #1
module lock_top(
    input           clk,        //输入时钟 50MHz
    input   [7:0]   key,        //8 个按键输入

    output [7:0]    led,        //8 个 led 输出
    output [7:0]    smg,        //8 位数码管段选
    output [3:0]    dig         //4 位数码管位选
);

    //=============================================================
    wire   [7:0]  btn_deb;
    wire          restart;
    wire          det_start;
    wire          det_end;

    wire [15:0]   ctrl;
    //=============================================================
    btn_deb#(           // 按键消抖
        .BTN_WIDTH    (  4'd8         ) //parameter   BTN_WIDTH = 4'd8 //按键位宽
    ) U_btn_deb
    (
        .clk            (  clk          ),//input           clk,//输入处理时钟 50MHz
        .btn_in         (  key          ),//input   [BTN_WIDTH-1:0] btn_in,//输入按键信号
        .btn_deb        (  btn_deb   ) //output reg [BTN_WIDTH-1:0] btn_deb //输出按键消抖信号
    );

    assign restart = (~btn_deb[0])&(~btn_deb[7]);

    led_ctl led_ctl(
        .clk            (  clk          ),//input clk,          //输入处理时钟 50MHz
        .restart        (  restart      ),//input   restart,    //输入重新开始信号，高点平有效
        .det_start      (  det_start    ),//output reg          det_start,//输出检测开始信号
        .led            (  led          ) //output reg      [7:0] led //输出检测条件，以 LED 视觉触发
    );

    compare compare(
        .clk            (  clk          ),//input   clk,        //输入处理时钟 50MHz
        .det_start      (  det_start    ),//input       det_start,//输入计时开始脉冲信号，高电平有效
        .restart        (  restart      ),//input     restart,  //输入重新开始信号，高点平有效
        .btn_deb        (  btn_deb      ),//input [ 7:0]      btn_deb,  //输入按键信号
        .bit_sel        (  led          ),//input [ 7:0]      bit_sel,  //输入检测条件
        .det_end        (  det_end      ),//output    det_end,  //输出计时结束信号
        .ctrl           (  ctrl         ) //output reg [15:0]     ctrl  //输出计时统计结果
    );

```

```
67      seq_display seq_display(
68          .clk            (  clk        ),//input clk,     //输入处理时钟 50MHz
69          .restart        (  restart    ),//input restart, //输入重新开始信号，高点平有效
70          .det_end        (  det_end    ),//input det_end, //输入计时结束信号
71          .ctrl           (  ctrl       ),//input [15:0] ctrl, //输入计时统计结果
72
73          .smg            (  smg        ),//output reg [7:0]smg, //输出 8 位数码管段选
74          .dig            (  dig        ) //output reg [3:0]dig //输出 4 位数码管位选
75      );
76
77  endmodule
78
```

12.4.2 LED 显示控制模块

此模块的难点在于如何产生随机点亮一个 LED 灯，方法基于流水灯的点亮 LED 的流动，用另外一个计数器产生一个取数使能信号，若流水灯的单个状态维持时钟周期为 N，使能信号计数器计数周期的时钟周期个数为 M，整个出现重叠的周期为 M*N，再加上外部输入触发开始信号，每次检测产生出现点亮同样的 LED 灯的概率就很低了，可忽略不计；

Runber 板卡上 LED 灯高定平控制点亮，流水初始状态设置为 8'b0000_0001；

在触发开始等待 3S，当再一次出现两个计数器符合我们设置的阀值时触发进入开始检测状态。

Module 设计如下:

```
1   `timescale 1ns / 1ps
2   `define UD #1
3   module led_ctl(
4       input                   clk,        //输入处理时钟 12MHz
5       input                   restart,    //输入重新开始信号，高点平有效
6       output reg              det_start,//输出检测开始信号
7       output reg    [7:0]     led       //输出检测条件，以 LED 视觉触发
8   );
9
10      //=============================================================
11      reg [25:0] time_1s_cnt = 26'd0;   //0~11_999_999 1s 周期
12      always @(posedge clk)
13      begin
14          if(time_1s_cnt == 26'd1199_9999)
15              time_1s_cnt <= `UD 26'd0;
16          else
17              time_1s_cnt <= `UD time_1s_cnt + 26'd1;
18      end
19
```

```
79      reg [1:0] second_cnt = 2'd0;
80      always @(posedge clk)
81      begin
82          if(restart)
83              second_cnt <= `UD 2'd0;
84          else if(time_1s_cnt == 26'd1199_9999)
85          begin
86              if(second_cnt == 2'd3)
87                  second_cnt <= `UD second_cnt;
88              else
89                  second_cnt <= `UD second_cnt + 2'd1;
90          end
91      end
92
93      //=============================================================
94      // 循环移位寄存器
95      reg [7:0]   led_temp = 8'b0000_0001;
96      reg [9:0]   time_led_cnt=10'd0;
97      always @(posedge clk)
98      begin
99          if(time_led_cnt == 10'd579)
100             time_led_cnt <= `UD 10'd0;
101         else
102             time_led_cnt <= `UD time_led_cnt + 10'd1;
103     end
104
105     always @(posedge clk)
106     begin
107         if(time_led_cnt == 10'd579)
108             led_temp <= `UD {led_temp[0],led_temp[7:1]};
109     end
110
111     //=============================================================
112     reg [8:0] time_cnt=9'd0;     //取出随机 led 状态
113     always @(posedge clk)
114     begin
115         time_cnt <= `UD time_cnt + 10'd1;
116     end
117
118     //=============================================================
119     reg     start_cnt=1'b0;   //start 计数
120     always @ (posedge clk)
121     begin
122         if(second_cnt == 2'd3 && start_cnt == 1'b0 && time_cnt == 10'd375)
123             det_start <= `UD 1'b1;
124         else
125             det_start <= `UD 1'b0;
126     end
127
128     always @(posedge clk)
129     begin
130         if(restart)
131             start_cnt <= `UD 1'b0;
132         else if(second_cnt == 2'd3 && start_cnt == 1'b0 && time_cnt == 10'd375)
133             start_cnt <= `UD start_cnt + 1'b1;
134     end
```

```
76      always @(posedge clk)
77      begin
78          if(restart)
79              led <= `UD 8'b0000_0000;
80          else if(second_cnt == 2'd3 && start_cnt == 1'b0 && time_cnt == 10'd375)
81              led <= `UD led_temp;
82      end
83
84  endmodule
85
```

12.4.3 比对计时模块

此模块中需注意两个功能：1、计时开始与结束；2、计时如何进位，超时如何统计；

Module 设计如下:

```
1   `timescale 1ns / 1ps
2   `define UD #1
3   module compare(
4       input                   clk,        //输入处理时钟 50MHz
5       input                   det_start,//输入计时开始信号，1 个时钟周期，高电平有效
6       input                   restart,    //重新开始信号
7       input       [7:0]       btn_deb,    //输入按键信号
8       input       [7:0]       bit_sel,    //输入检测条件
9
10      output                  det_end,    //输出计时结束信号
11      output reg [15:0]       ctrl        //输出计时统计结果
12  );
13
14  //=============================================================
15      //毫秒计数器
16      reg [15:0] time_ms_cnt= 16'd0;//0~11999     1ms 周期
17      always@(posedge clk)
18      begin
19          if(det_start)
20              time_ms_cnt <= `UD 16'd0;
21          else if(time_ms_cnt == 16'd11999)
22              time_ms_cnt <= `UD 16'd0;
23          else
24              time_ms_cnt <= `UD time_ms_cnt + 16'd1;
25      end
26
27      //=============================================================
28      reg     counter_en = 1'b0;
29      reg     flow = 1'b0;
30      reg     counter_en_1d = 1'b0;
31
```

```
32      always @(posedge clk)
33      begin
34          if(det_start)
35              counter_en <= `UD 1'b1;
36          else if((btn_deb == (~bit_sel)) || flow || restart)   // 按键按下低电平有效，LED 高电平点亮
37              counter_en <= `UD 1'b0;
38      end
39
40      always @(posedge clk)
41      begin
42          counter_en_1d <= `UD counter_en;
43      end
44
45      assign det_end = (~counter_en & counter_en_1d) & (~restart);
46
47  //=============================================================
48      //统计反应时间
49      //              个位        十位          百位          千位
50      wire [3:0] dec_single,dec_ten,dec_hundred,dec_thousand;
51      wire           dec_sigle_trg; //1ms 计时触发信号
52      wire           carry1,carry2,carry3,carry4;//4 位进位
53
54      assign dec_sigle_trg = counter_en & (time_ms_cnt == 16'd11999);
55
56      always @(posedge clk)
57      begin
58          if(det_start)
59              flow <= `UD 1'b0;
60          else if(carry4)       // 溢出
61              flow <= `UD 1'b1;
62      end
63
64      assign ctrl ={dec_thousand,dec_hundred,dec_ten,dec_single};
65
66      dec_counter single(
67          .clk                (   clk             ),//input                 clk,
68          .det_start          (   det_start       ),//input                 det_start,
69          .trig               (   dec_sigle_trg   ),//input                 trig,
70          .flow               (   flow            ),//input                 flow,
71
72          .carry              (   carry1          ),//output reg            carry,
73          .dec                (   dec_single      ) //output reg [3:0] dec
74      );
75
76      dec_counter ten(
77          .clk                (   clk             ),//input                 clk,
78          .det_start          (   det_start       ),//input                 det_start,
79          .trig               (   carry1          ),//input                 trig,
80          .flow               (   flow            ),//input                 flow,
81
82          .carry              (   carry2          ),//output reg            carry,
83          .dec                (   dec_ten         ) //output reg [3:0] dec
84      );
85
```

```
86      dec_counter hundred(
87          .clk            (   clk         ),//input           clk,
88          .det_start      (   det_start   ),//input           det_start,
89          .trig           (   carry2      ),//input           trig,
90          .flow           (   flow        ),//input           flow,
91
92          .carry          (   carry3      ),//output reg      carry,
93          .dec            (   dec_hundred ) //output reg [3:0] dec
94      );
95
96      dec_counter thousand(
97          .clk            (   clk         ),//input           clk,
98          .det_start      (   det_start   ),//input           det_start,
99          .trig           (   carry3      ),//input           trig,
100         .flow           (   flow        ),//input           flow,
101
102         .carry          (   carry4      ),//output reg      carry,
103         .dec            (   dec_thousand ) //output reg [3:0] dec
104     );
105
106 endmodule
107
```

计时统计模块 module 设计如下：

每一位计时范围 0～9，输入触发信号触发计数加一，当计数到 9 时再输入触发信号则归位到 0 并且输出进位信号（carry），同时接收全溢出信号，当 4 位全部溢出时固定输出 9；

```
1   `timescale 1ns / 1ps
2   `define UD #1
3   module dec_counter(
4       input           clk,
5       input           det_start,
6       input           trig,
7       input           flow,
8
9       output reg      carry,
10      output reg [3:0] dec
11  );
12
13      always @(posedge clk)
14      begin
15          if(trig & dec == 4'd9)
16              carry <= `UD 1'b1;
17          else
18              carry <= `UD 1'b0;
19      end
20
```

```
21      always @(posedge clk)
22      begin
23          if(det_start)    //复位
24              dec <= `UD 4'd0;
25          else if(flow)    //溢出处理
26              dec <= `UD 4'd10;
27          else if(trig)
28          begin
29              if(dec == 4'd9)
30                  dec <= `UD 4'd0;
31              else if(dec != 4'd10)
32                  dec <= `UD dec + 1'b1;
33          end
34      end
35
36  endmodule
37
```

12.4.4 数码管显示模块

数码管显示模块到这个阶段相信大家都不陌生了，这里就给大家介绍一下输入输出信号，以及显示的使能控制；

输入输出信号如下：

```
1   module seq_display(
2       input                   clk,        //输入处理时钟 50MHz
3       input                   restart,    //输入重新开始信号，高点平有效
4       input                   det_end,    //输入计时结束信号
5       input       [15:0]      ctrl,       //输入计时统计结果
6
7       output reg  [7:0]       smg,        //输出 8 位数码管段选
8       output reg  [3:0]       dig         //输出 4 位数码管位选
9   );
10
```

显示数值控制如下：

```
//==========================================================
//数码管显示控制
reg [3:0] key0_cnt=4'd0,key1_cnt=4'd0,key2_cnt=4'd0,key3_cnt=4'd0;
always @(posedge clk)
begin
    if(restart)
    begin
        key0_cnt <= `UD 4'd0;
        key1_cnt <= `UD 4'd0;
        key2_cnt <= `UD 4'd0;
        key3_cnt <= `UD 4'd0;
    end
    else if(det_end)
    begin
        key0_cnt <= `UD ctrl[ 3: 0];
        key1_cnt <= `UD ctrl[ 7: 4];
        key2_cnt <= `UD ctrl[11: 8];
        key3_cnt <= `UD ctrl[15:12];
    end
end

```

12.5 实验现象

烧录固件后，第一次进入不需要触发，第二次以后进入检测需要同时按下 KEY0 和 KEY7 触发开始，开始后三秒 LED0-LED7 八个 LED 灯随机点亮一个，当看到灯亮后立刻按下相应的 KEY。如果匹配 LED 灯的位置，将该时间以十进制的形式显示在数码管上，以 ms 为单位；若不匹配则不结束检测，直到计时超出 9999ms 时自动退出，数码管显示时间 9999；