一. 基础部分

1.整体框架

采用自顶向下的层次化设计思路，将实现功能的各模块放入单独的文件内完成，最后通过 top 文件例化各个模块实现整体功能，再根据引脚约束与硬件对应。在本例的基础部分主要分为计数频率 5hz 的分频，七段数码管的驱动，七段数码管扫描的频率，计数模块，16 进制到 10 进制转换模块。下面逐项介绍：

2.计数频率 5HZ 分频

a.设计思路

由上一个实验，我们知道 1HZ 的分频需要计数到 49999999，因此本例的 5HZ 应该计数到上次计数值的五分之一：9999999，也就是十六进制下的 98967f，将分频后的时钟 div_clk_098 作为输出。为了最终的顶层设计具有清零功能，因此对每一个子模块也加入清零引脚。

b.源代码

```
module divclk_2_098(
input clk_098,
input rst_098,
output reg div_clk_098
    );
    reg[31:0]counter_098;
    always@(posedge clk_098 or posedge rst_098)
    begin
    if(rst_098)
    counter_098<=32'h00000000;
    else
```

```
        if(counter_098==32'h0098967f)//5hz 分频
        begin
        counter_098<=32'h00000000;
        div_clk_098<=~div_clk_098;
        end
        else
        counter_098<=counter_098+1;
        end
    endmodule
```

3.数码管刷新频率分频

a.设计思路

根据人眼的视觉暂留效应，当数码管的刷新频率很快的时候，看上去会像多个数码管通识导通。但原有 100MHZ 的主频太快，以此刷新并不能实现效果，查阅资料并实际测试，让其计数到 60000 左右可以完成任务，最终选取了 61567，即 16 进制下的 f07f，将输出 scan_clk_098 作为顶层文件中的七段数码管的时钟进行驱动刷新。

b.源代码

```
module divclk_098(
input clk_098,
input rst_098,
output reg scan_clk_098
    );
    reg[19:0]counter_098;
```

```
always@(posedge clk_098 or posedge rst_098)
begin
if(rst_098)
counter_098<=20'h00000;
else
if(counter_098==20'h0f07f)
begin
counter_098<=20'h00000;
scan_clk_098<=~scan_clk_098;
end
else
counter_098<=counter_098+1;
end

endmodule
```

4.计数

a.基本思路

为实现从 002 到 254 的计数，将计数满的判断值设为 255，计数满则返回 002，

但在实际测试中，如果将判断值设为 255 因为扫描 8 个数码管仍需要一段时间，导致第一个数码管会出现 255 的数值，显然这不是我们期望的，因此可在代码中加入判断是否小于等于 254 的条件，不满足则不显示。但是这样势必会增加代码量和硬件电路，因此直接将判断值改为 254，利用数码管的扫描时间完成最后一个数的计数，因为跨越时间很小，其影响可忽略不计。

b.源代码

```
module counterb_098(
input clk_098,
input rst_098,
output reg[7:0]counter_098
    );
    always@(posedge clk_098 or posedge rst_098)
    begin
    if(rst_098)
    counter_098<=8'h02;
    else
    if(counter_098==8'hfe)
    counter_098<=8'h02;
    else
    counter_098<=counter_098+1;
    end
    endmodule
```

5.进制转化

a.设计思路

利用已知的余 3 码转化程序，将 16 进制转换为 10 进制送入数码管，其中 counter1_098 为 16 进制数，counter2_098 为 10 进制数。但在实际操作过程中，如果不做转化，也可以利用 16 进制数驱动数码管，完成正常计数，推测因为 vivado 版本更新类似 C 语言一样，会做隐式的转化约束，为了不必要的隐患还是加上了这个程序。

b.源代码

```
module exchange_098(
input wire [7:0] counter1_098,
output reg [9:0] counter2_098 );
reg [17:0] z_098;
integer i_098;
always @(*)
  begin
     for(i_098=0;i_098<=17;i_098=i_098+1)
          z_098[i_098]=0;
          z_098[10:3]=counter1_098;
            repeat(5)
          begin
            if(z_098[11:8]>4)
          z_098[11:8]=z_098[11:8]+3;
          if(z_098[15:12]>4)
            z_098[15:12]=z_098[15:12]+3;
```

```
        z_098[17:1]=z_098[16:0];
        end
        counter2_098=z_098[17:8];
         end
          endmodule
```

6.数码管驱动

a.设计思路

数码管的输入时钟应该为刷新时钟，依次对 8 个数码管赋值，虽然同时只导通一个，但是刷新频率很快，所以看起来是 8 个同时导通。

对 10 进制数 conter_098 进行分解，又余 3 码转化可知，低 4 位是十进制个位表示，中 4 位是十位，高两位代表百位。因为计数知道 254，所以只需要 3 个数码管，对其他数码管置 0。分别对个位十位百位进行判断，转化为对应的数码管驱动数送入数码管引脚。因为最初要显示 8 个 1，因此加入一个开关，如果开关拨动则全部置 1，否则正常计数。在这里也曾考虑到再开一个时钟做延迟，先显示 1 再计数，但是这样代码量很大而且效果不怎么好，最后还是选择了开关切换作为 11111111 的显示和计数模式的切换。

b.源代码

```
module seg7_098(
input clk_098,
input rst_098,
input sel_098,
input [9:0] counter_098,
```

```
output reg[7:0]an_098,
output reg[6:0]a_g_098
        );
        reg[2:0] counter1_098;
        integer i_098;
        function[6:0]seg7_098;
        input[3:0]x_098;
        begin
        case(x_098)
        0:seg7_098=7'b0000001;
        1:seg7_098=7'b1001111;
        2:seg7_098=7'b0010010;
        3:seg7_098=7'b0000110;
        4:seg7_098=7'b1001100;
        5:seg7_098=7'b0100100;
        6:seg7_098=7'b0100000;
        7:seg7_098=7'b0001111;
        8:seg7_098=7'b0000000;
        9:seg7_098=7'b0000100;
    'hA:seg7_098=7'b0001000;
    'hB:seg7_098=7'b1100000;
    'hC:seg7_098=7'b0110001;
```

```
    'hD:seg7_098=7'b1000010;
    'hE:seg7_098=7'b0110000;
    'hF:seg7_098=7'b0111000;
    default: seg7_098=7'b0000001;
        endcase
        end
        endfunction
        initial
        begin
        counter1_098=2'b00;
        end
always@(posedge clk_098)
begin
counter1_098<=counter1_098+1;
end
    always@(*)
    case(counter1_098)
    3'b000:
    begin
    if(sel_098)
    a_g_098=7'b1001111;
    else
```

```
a_g_098=seg7_098(counter_098[3-:4]);
an_098=8'b10000000;
end
3'b001:
 begin
 if(sel_098)
 a_g_098=7'b1001111;
 else
 a_g_098=seg7_098(counter_098[7-:4]);
 an_098=8'b01000000;
 end
  3'b010:
    begin
    if(sel_098)
    a_g_098=7'b1001111;
    else
    a_g_098=seg7_098(counter_098[9-:2]);
    an_098=8'b00100000;
    end
     3'b011:
       begin
       if(sel_098)
```

```
a_g_098=7'b1001111;
else
a_g_098=seg7_098(0);
an_098=8'b00010000;
end
3'b100:
        begin
        if(sel_098)
        a_g_098=7'b1001111;
        else
        a_g_098=seg7_098(0);
        an_098=8'b00001000;
        end
 3'b101:
                begin
                if(sel_098)
                a_g_098=7'b1001111;
                else
                a_g_098=seg7_098(0);
                an_098=8'b00000100;
                end
    3'b110:
```

```
                                    begin
                                    if(sel_098)
                                    a_g_098=7'b1001111;
                                    else
                                    a_g_098=seg7_098(0);
                                    an_098=8'b00000010;
                                    end
            3'b111:
                                      begin
                                      if(sel_098)
                                      a_g_098=7'b1001111;
                                      else
                                      a_g_098=seg7_098(0);
                                       an_098=8'b00000001;
                                       end
            default:
            begin
            a_g_098=seg7_098(0);
            an_098=8'b00000000;
            end
endcase
```

endmodule

7.顶层驱动

a.整体思路

在 top 文件中，对各模块进行例化和联系，将上述各个子模块串在一起，最终对 top 文件的引脚进行约束，联系实际硬件，完成基础部分功能。

b.源代码

```
module top_098(
input clk_098,
input rst_098,
input sel_098,

output[7:0]an_098,
output[6:0]a_g_098

);
wire div_clk_098;
wire scan_clk_098;
wire band_clk_098;
wire[7:0]counter_098;
wire[9:0]counter2_098;

divclk_2_098 divclk_2_098_inst(
```

```
.clk_098(clk_098),
.rst_098(rst_098),
.div_clk_098(div_clk_098)
);

divclk_098 divclk_098_inst(
.clk_098(clk_098),
.rst_098(rst_098),
.scan_clk_098(scan_clk_098)
);

counterb_098 counterb_098_inst(
.clk_098(div_clk_098),
.rst_098(rst_098),
.counter_098(counter_098)
);

exchange_098 exchange_098_inst(
.counter1_098(counter_098),
.counter2_098(counter2_098)
);
```

```
seg7_098 seg7_098_inst(
.clk_098(scan_clk_098),
.rst_098(rst_098),
.sel_098(sel_098),
.counter_098(counter2_098),
.an_098(an_098),
.a_g_098(a_g_098)
);

endmodule
```

8.实验结果

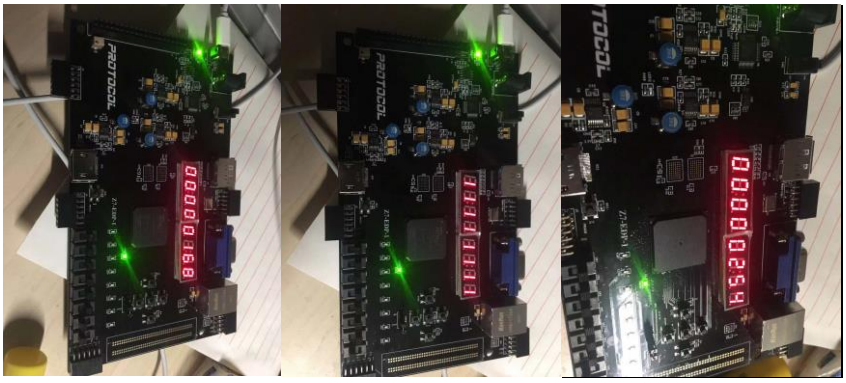

图 1 基础部分实验现象

二．提高部分

(1)LED 指示溢出

1.整体思路

因为原本计数器没有溢出检查和判断，使用过程中可能因为无法判断溢出和造成困扰，因此用 LED 指示溢出。最初 LED 全灭，溢出一次第一个灯亮，一直到溢出 8 次第 8 个 LED 灯亮。之后全部灯灭重新开始判断。这样 8

个灯完成一次流水操作，也给使用者充分的时间判断溢出了多少次。因此加入一个新的子模块：timeled_098，用来对 LED 驱动，同时为原来的计数模块引入 4 位的寄存器 time_098 传递溢出次数，将此传入 timeled_098 进行控制和显示。

2.timeled_098 文件代码

```
module timeled_098(
input clk_098,
input rst_098,
input[3:0] time_098,
output reg [7:0]led_098
    );
always@(posedge clk_098 or posedge rst_098)
begin
if(rst_098)
led_098<=8'b00000000;
else
begin
case(time_098)
0: led_098<=8'b00000000;
1: led_098<=8'b00000001;
2: led_098<=8'b00000010;
3: led_098<=8'b00000100;
4: led_098<=8'b00001000;
```

```
5: led_098<=8'b00010000;
6: led_098<=8'b00100000;
7: led_098<=8'b01000000;
8: led_098<=8'b10000000;
default: led_098<=8'b11111111;
endcase
end
end

endmodule
```

(2)开关模值改变

为了实现更灵活的计数，用三个开关控制改变计数的上限和下限，SW2 打开时可以改变模值，即使能模值更改功能，此时如果 SW3 为 1 则计数最大值加 10，如果 SW3 为 0 则计数最大值减 10，如果 SW4 为 1 则计数最小加 10，如果 SW4 为 0 则计数最小值减 10。为此增添如下模块，第一，分频时钟 3 用于判断开关，为不占用 CPU 不时时判断是否改变模值，以固定的频率判断是否要改变模值，节省 CPU 资源；第二，模值改变子文件：count_change_098。下面分步说明：

1. 判断模值改变分频

源代码：

```
module div_clk_3_098(
input clk_098,
```

```
input rst_098,
output reg max_clk_098
    );
    reg[31:0]counter_098;
    always@(posedge clk_098 or posedge rst_098)
    begin
    if(rst_098)
    counter_098<=32'h00000000;
    else
    if(counter_098==32'h8f0d1710)//5hz 分频
    begin
    counter_098<=32'h00000000;
    max_clk_098<=~max_clk_098;
    end
    else
    counter_098<=counter_098+1;
    end
endmodule
```

2.模值改变

a. 设计思路

通过设置三个输入管脚 cmd_098、change_098 和 change_low_098 接在三个开关上，cmd_098 打开时表示可以改变模值，即使能模值改变选项，change_098 为 1 则增加计数最大值，change_098 为 0 则减小计数最大值，

change_low_098 为 1 则增加计数最小值，change_low_098 为 0 时则减小计数最小值。为观察方便，以 10 为单位改变模值。通过这轮操作，即可改变计数的上限和下限，实现更为灵活的计数。

b.源代码

```
module count_change_098(
input clk_098,
input cmd_098,
input change_098,
input change_low_098,
output reg[7:0]maxcount_098,
output reg[7:0]mincount_098
      );
      initial
      begin
        maxcount_098=8'hfe;
          mincount_098=8'h02;
      end

      always@(posedge clk_098)
      begin

      if(cmd_098)
      begin
```

```
        if(change_098)
        maxcount_098=maxcount_098+8'h0a;
        else
          maxcount_098=maxcount_098-8'h0a;
          if(change_low_098)
                mincount_098=mincount_098+8'h0a;
                else
                  mincount_098=mincount_098-8'h0a;
        end


        end
    endmodule
```

（3）最终设计

a.最终成品

最终的作品可以根据开关改变计数的最大值和最小值，为方便观察一次更改的单位我 10，同时 led 可以指示溢出了多少次，关断 sw2 开关后即关闭更改模值的功能，计数器上限和下限将保持不变。

b.最终顶层文件代码

```
module top_098(
input clk_098,
input rst_098,
input sel_098,
input cmd_098,
```

```
input change_098,
input change_low_098,
output[7:0]led_098,
output[7:0]an_098,
output[6:0]a_g_098

);
wire div_clk_098;
wire scan_clk_098;
wire band_clk_098;
wire max_clk_098;

wire[7:0]counter_098;
wire[9:0]counter2_098;
wire[3:0] time_098;
wire[7:0]maxcount_098;
wire[7:0]mincount_098;
divclk_2_098 divclk_2_098_inst(
.clk_098(clk_098),
.rst_098(rst_098),
.div_clk_098(div_clk_098)
);
```

```
divclk_098 divclk_098_inst(
.clk_098(clk_098),
.rst_098(rst_098),
.scan_clk_098(scan_clk_098)
);

div_clk_3_098(
.clk_098(clk_098),
.rst_098(rst_098),
.max_clk_098(max_clk_098)
);

count_change_098 count_change_098_inst(
.cmd_098(cmd_098),
.change_098(change_098),
.change_low_098(change_low_098),
.clk_098(max_clk_098),
.maxcount_098(maxcount_098),
.mincount_098(mincount_098)
);
```

```
counterb_098 counterb_098_inst(
.clk_098(div_clk_098),
.rst_098(rst_098),
.time_098(time_098),
.maxcount_098(maxcount_098),
.mincount_098(mincount_098),
.counter_098(counter_098)

);

exchange_098 exchange_098_inst(
.counter1_098(counter_098),
.counter2_098(counter2_098)
);

seg7_098 seg7_098_inst(
.clk_098(scan_clk_098),
.rst_098(rst_098),
.sel_098(sel_098),
.counter_098(counter2_098),
.an_098(an_098),
.a_g_098(a_g_098)
```

```
);
timeled_098 timeled_098_inst(
.clk_098(div_clk_098),
.rst_098(rst_098),
.time_098(time_098),
.led_098(led_098)
);

endmodule
```

c.计数部分最终代码

```
module counterb_098(
input clk_098,
input rst_098,
input [7:0] maxcount_098,
input [7:0] mincount_098,
output reg[3:0] time_098,
output reg[7:0]counter_098
    );

    always@(posedge clk_098 or posedge rst_098)
    begin
    if(rst_098)
```

```
    begin
    counter_098<=mincount_098;
      time_098<=0;
    end
    else
    if(counter_098==maxcount_098)
    begin
   time_098=time_098+1;

    if( time_098>=9)
      time_098=0;
    counter_098<=mincount_098;

    end
    else
    begin
    counter_098<=counter_098+1;

   end
    end
    endmodule
```

d.结果图

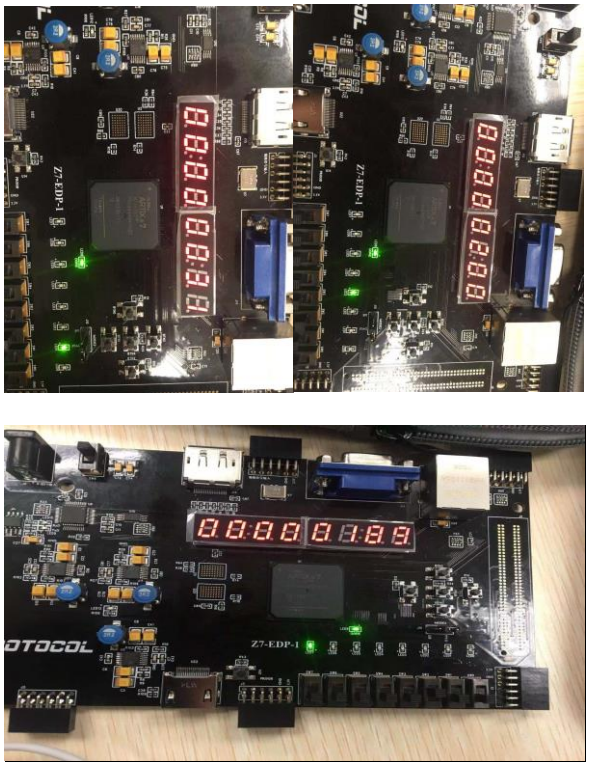

图 2 提高部分效果图