目录

实验一　全加器的设计

（一）实验目的

以四位二进制全加器为例熟悉利用 QuartusII 的原理图输入方法和文本输入法设计简单组合电路；学习多层次工程的设计方法。

（二）实验要求

⑴用文本方法实现一位全加器，再采用层次设计法用原理图输入完成 4 位全加器的设计；

⑵给出此项设计的仿真波形；

⑶用发光 LED 指示显示结果。

（三）实验流程

1、创建 1 位全加器工程，新建 verilog 文本文件，编译，转换为.bsf 符号文件。

(1) 顶层文件

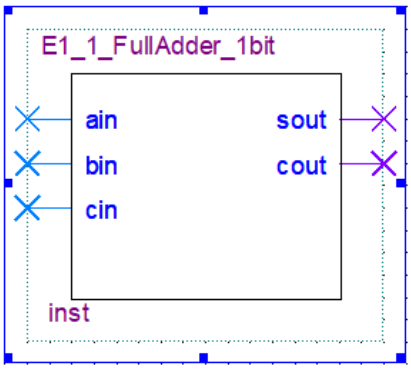
```
module E1_1_FullAdder_1bit (
    ain,
    bin,
    cin,

    sout,   // sum bit
    cout    // carry bit
);

input   ain, bin, cin;
output  sout, cout;

wire    sum_1, carry_1, carry_2;

assign  cout    = carry_1 | carry_2;

Adder Adder_inst_1(
    .ain    (ain),
    .bin    (bin),
    .sout   (sum_1),
    .cout   (carry_1)
);

Adder Adder_inst_2(
    .ain    (sum_1),
    .bin    (cin),
    .sout   (sout),
    .cout   (carry_2)
);

endmodule
```

(2) 半加器模块

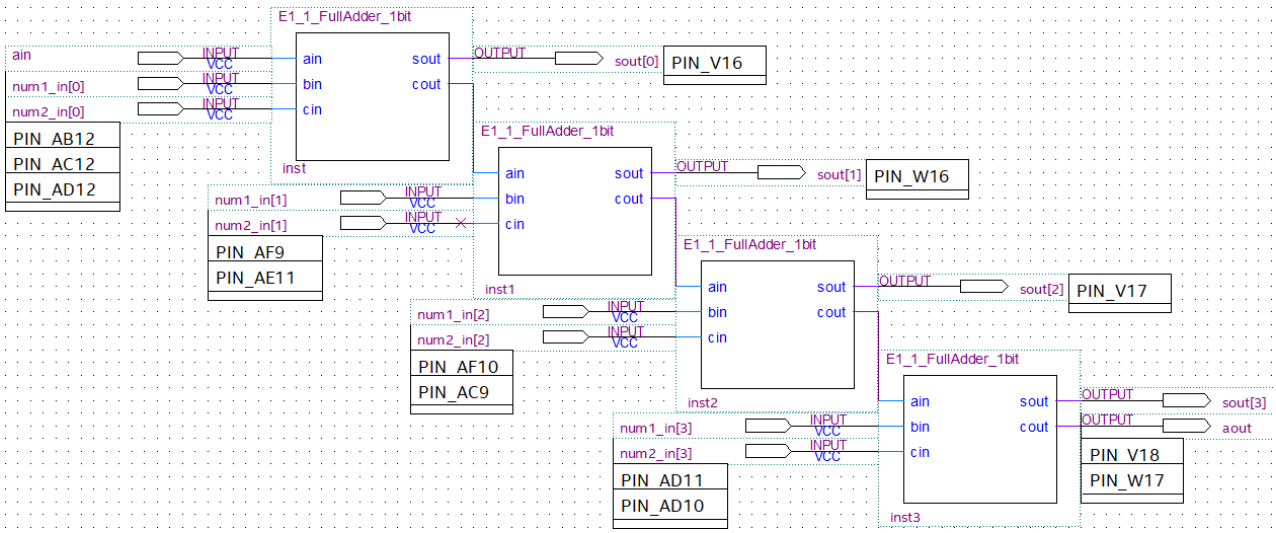
```
module Adder (
    ain,
    bin,
    sout,   // sum bit
    cout    // carry bit
);

input   ain, bin;

output  sout, cout;

assign  sout     = ain ^ bin;
assign  cout     = ain & bin;
```

```
13
14  endmodule
```

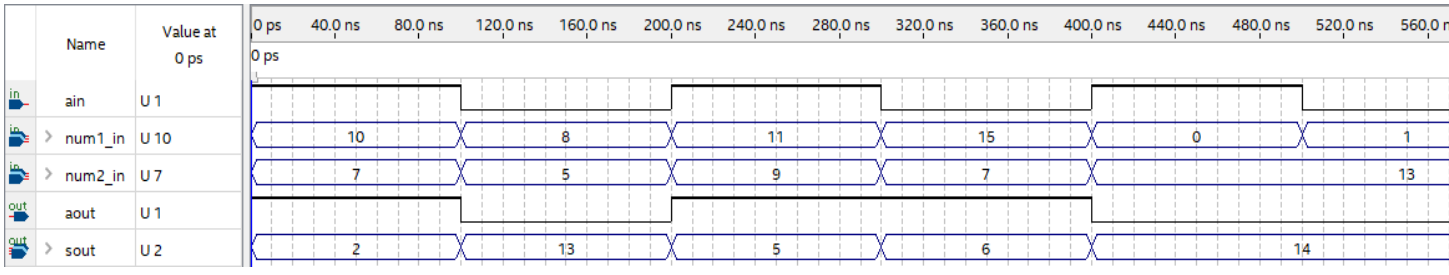
(3) .bsf 文件



2、同一文件夹下创建 4 位全加器工程，新建 bdf 原理图文件并编译。



3、新建 vwf 波形文件，时序仿真验证加法功能。



4、引脚锁定并再次编译。

Node Name	Direction	Location	I/O Standard
ain	Input	PIN_AB12	2.5 V
aout	Output	PIN_W17	2.5 V
num1_in[0]	Input	PIN_AC12	2.5 V
num1_in[1]	Input	PIN_AF9	2.5 V
num1_in[2]	Input	PIN_AF10	2.5 V
num1_in[3]	Input	PIN_AD11	2.5 V
num2_in[0]	Input	PIN_AD12	2.5 V
num2_in[1]	Input	PIN_AE11	2.5 V
num2_in[2]	Input	PIN_AC9	2.5 V
num2_in[3]	Input	PIN_AD10	2.5 V
sout[0]	Output	PIN_V16	2.5 V
sout[1]	Output	PIN_W16	2.5 V
sout[2]	Output	PIN_V17	2.5 V
sout[3]	Output	PIN_V18	2.5 V

5、添加.sof 文件下载测试。

（四）实验效果

最终完整实现目标功能。

实验二　模可变计数器的设计

（一）实验目的

1、进一步熟悉 EDA 开发板和 Quartus II 软件的使用方法；

2、学习静态数码管的使用；

3、学习计数器的设计、仿真和硬件测试；学习 7 段数码显示译码器设计；

（二）实验要求

设计模可变计数器，可任选模的大小（例模 15、模 115），实验要求：

（1）　设置一位控制位 M，要求 M=0：模 X 计数；M=1：模 Y 计数；

（2）　计数结果用 3 位数码管显示，显示 BCD 码；

（3）　给出此项设计的仿真波形；

（4）　选择实验电路验证此计数器的功能。

设置涉及 2 个开关和一个按键，一个开关控制改变模值，另一开关作为使能控制，按键作为异步清 0。

（三）实验程序

(1) 顶层文件

```
1   module E2_Counter (
2       clk,
3       rst_N,
4       en_SW,
5       modulo_SW,
6
7       digital_tube_3b,
8       LED
9   );
10
11  input   clk, rst_N;
12  input   en_SW, modulo_SW;
13
14  output  [20:0]  digital_tube_3b;
15  output          LED;
16
17  wire    [11:0]  number_BCD;
18  wire            clk_1Hz;
19
    ClockDivider ClockDivider_inst (
20      .clkin  (clk),
21      .clkout (clk_1Hz)
22  );
23
    Timer Timer_inst (
24      .clk        (clk_1Hz),
25      .rst_N      (rst_N),
26      .en_SW      (en_SW),
27      .modulo_SW  (modulo_SW),
28      .number_BCD (number_BCD),
29      .LED        (LED)
30  );
31
    DigitalTube_3b DigitalTube_3b_inst (
32      .clk        (clk),
33      .rst_N      (rst_N),
34      .number_BCD (number_BCD),
```

```
    .pin_out    (digital_tube_3b)
);

endmodule
```

(2) 时钟分频模块（ClockDivider）

（与定时器模块原理相同——计数，但相比实现非常容易，此处略去）

(3) 模可变定时器模块

```
module Timer (
    clk,
    rst_N,
    en_SW,
    modulo_SW,
    number_BCD,
    LED
);

input   clk, rst_N;
input   en_SW, modulo_SW;

output  reg [11:0]  number_BCD;
output  reg         LED = 0;

reg     [9:0]   number  = 10'd0;

always @(posedge clk, negedge rst_N)
begin
    if (!rst_N) begin
        number <= 12'b0;
        LED <= 0;
    end
    else begin
        if (!en_SW) begin
            number <= number;
            LED <= 0;
        end
        else begin
            if (!modulo_SW) begin
                if (number >= 10'd14) begin
                    number <= 10'd0;
                    LED <= 1;
                end
                else begin
                    number <= number + 10'd1;
                    LED <= 0;
                end
            end
            else begin
                if (number >= 10'd114) begin
                    number <= 10'd0;
                    LED <= 1;
                end
                else begin
                    number <= number + 10'd1;
                    LED <= 0;
                end
            end
        end
    end

    number_BCD[3:0] <= number % 10;
    number_BCD[7:4] <= number / 10 % 10;
```

```
    number_BCD[11:8] <= number / 100 % 10;
end

endmodule
```

(4) 3 位数码管驱动模块

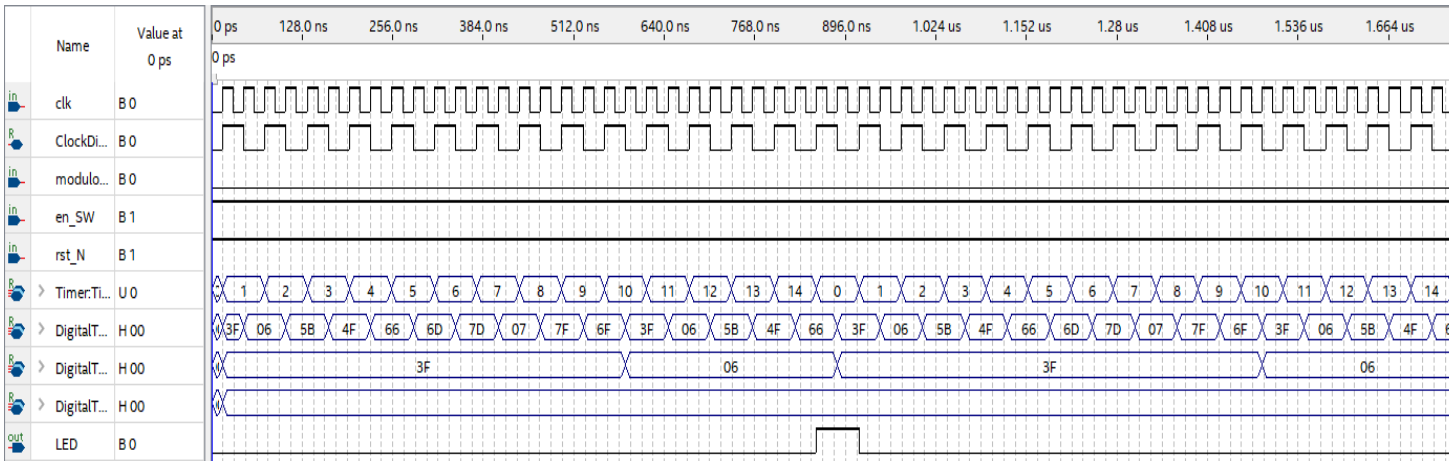
（实例化 3 次 1 位数码管驱动模块即可，此处代码略去）

(5) 1 位数码管驱动模块

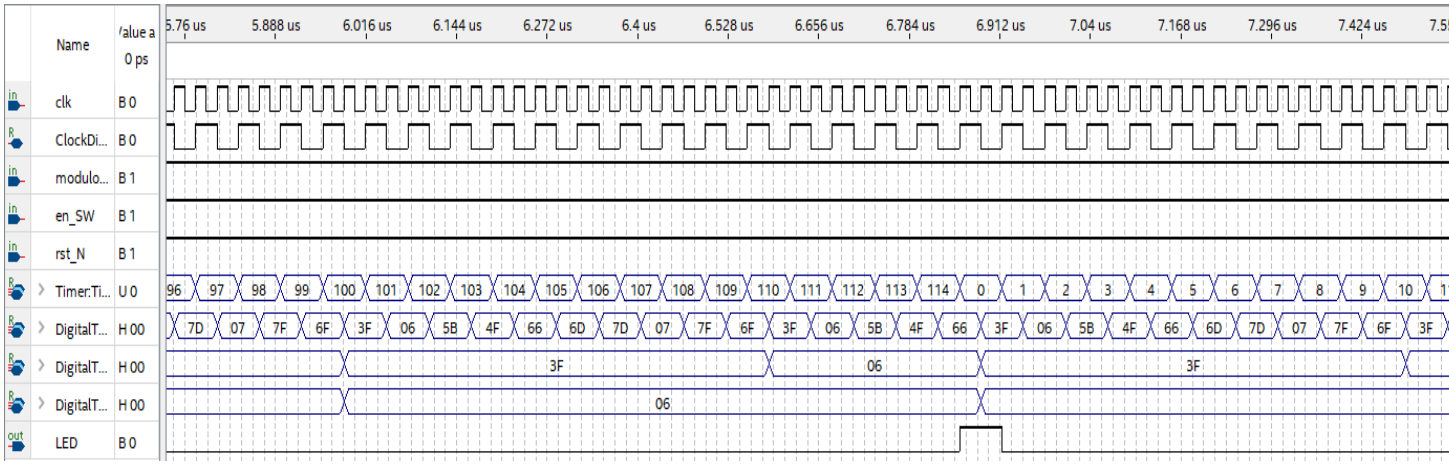
```
module DigitalTube (
    clk,
    rst_N,

    number,
    pin_out
);

input   clk, rst_N;

input           [3:0]   number;

output  reg     [6:0]   pin_out;


always @(posedge clk, negedge rst_N)
begin
    if (!rst_N) begin
        pin_out <= 7'b11111111;
    end
    else begin
        case (number)
            0: pin_out <= 7'b1000000;
            1: pin_out <= 7'b1111001;
            2: pin_out <= 7'b0100100;
            3: pin_out <= 7'b0110000;
            4: pin_out <= 7'b0011001;
            5: pin_out <= 7'b0010010;
            6: pin_out <= 7'b0000010;
            7: pin_out <= 7'b1111000;
            8: pin_out <= 7'b0000000;
            9: pin_out <= 7'b0010000;
            default: pin_out <= 7'b11111111;
        endcase
    end
end

endmodule
```

（四）实验波形

(1) 模 15



（2）模 115（时间轴已往后拖至计数复位处）



（五）实验效果

最终完整实现目标功能。

实验三　序列信号发生和检测器设计

（一）实验目的：学习一般有限状态机的设计，用状态机实现序列发生和检测器的电路设计。

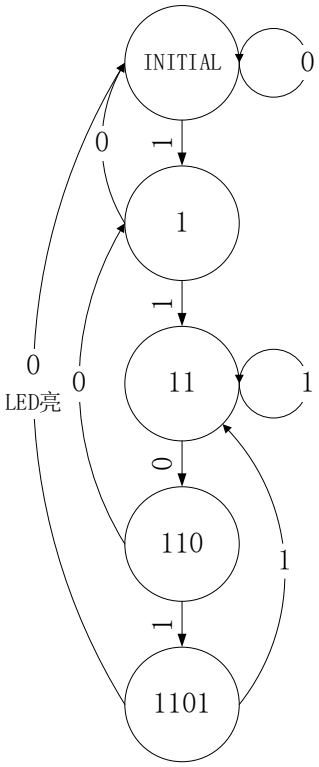
（二）实验要求：先实现串行序列发生器的设计，产生序列 0111010011011010；再设计检测器，若检测到串行序列 11010 则输出为“1”，否则输出为“0”，并对其进行仿真和硬件测试，选择实验电路验证功能。

下载程序后，可通过 led 串行输出序列信号，另用五个 led 灯来观测待检测序列，当 11010 五个全部出现在 led 上时，标识位灯 M 亮起，说明检测到“11010” 的信号，即符合设计要求。

产生的序列和检测的序列值可任选。

发生器和检测器最好异步，以确保能检测到，可以将时钟经非门后再接入检测器。

（三）序列检测状态转移图



（四）实验程序

(1) 顶层文件（包含时钟分频、序列发生、序列检测共三个模块实例化）

```
1   module E3_Sequence (
2       input                       clk                     ,
3       input                       rst_n                   ,
4
5       // input                        seq_in                  ,
6       // output                       seq_out                 ,
7
8       output                      LED_seq_out             ,
9
10      output      [4:0]           LED_seq_in              ,
11      output                      LED_seq_equal
12      );
13
```

```
    wire                        s_clk_1Hz                                       ;
    wire                        s_clk_1Hz_N                                     ;

    wire                        s_data                                          ;

    wire        [15:0]          sequence_generator                              ;

    assign                      sequence_generator  = 16'b0111_0100_1101_1010   ;

    ClockDivider ClockDivider_inst (
        .clkin              ( clk                   ),
        .clkout             ( s_clk_1Hz             ),
        .clkout_N           ( s_clk_1Hz_N           )
    );

    SequenceGenerator SequenceGenerator_inst (
        .clk                ( s_clk_1Hz             ),
        .rst_n              ( rst_n                 ),
        .seqence            ( sequence_generator    ),
        .seq_out            ( s_data                ),
        .LED_seq_out        ( LED_seq_out           )
    );

    SequenceDetector SequenceDetector_inst (
        .clk                ( s_clk_1Hz_N           ),
        .rst_n              ( rst_n                 ),
        .seq_in             ( s_data                ),
        .LED_seq_in         ( LED_seq_in            ),
        .LED_seq_equal      ( LED_seq_equal         )
    );

endmodule
```

(2) 时钟分频模块（ClockDivider）

（基本沿用实验 2 模块，此处略去）

(3) 序列发生模块

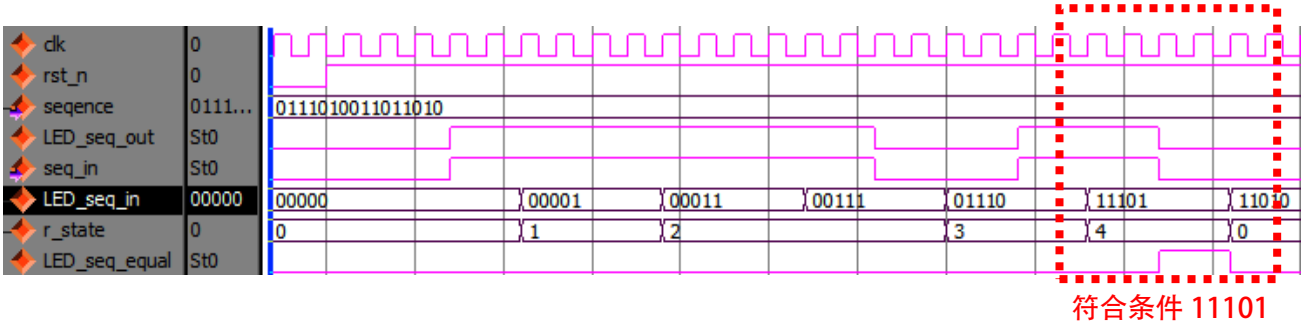
```
module SequenceGenerator (
    input                       clk                     ,
    input                       rst_n                   ,

    input       [15:0]          seqence                 ,

    output                      seq_out                 ,
    output                      LED_seq_out
    );

    reg         [15:0]          r_seq_out                               ;

    assign  seq_out             = r_seq_out[15]                         ;
    assign  LED_seq_out         = r_seq_out[15]                         ;

    always @(posedge clk, negedge rst_n) begin
        if (!rst_n) begin
            r_seq_out   <= seqence;
        end else begin
            r_seq_out   <= {r_seq_out[14:0], r_seq_out[15]};
        end
    end

endmodule
```

(4) 序列检测模块（依照上述状态机实现）

```
module SequenceDetector (
    input                       clk                 ,
    input                       rst_n               ,

    input                       seq_in              ,

    output      [4:0]           LED_seq_in          ,
    output                      LED_seq_equal
    );

    parameter                   p_STATE_0           = 3'd0      ;
    parameter                   p_STATE_1           = 3'd1      ;
    parameter                   p_STATE_2           = 3'd2      ;
    parameter                   p_STATE_3           = 3'd3      ;
    parameter                   p_STATE_4           = 3'd4      ;

    reg         [ 2:0]          r_state             = 3'd0      ;
    reg         [ 4:0]          r_seq_in                        ;
    reg                         r_seq_equal                     ;

    assign  LED_seq_in          = r_seq_in                      ;
    assign  LED_seq_equal       = (r_state == p_STATE_4 && seq_in == 1'b0) ? 1'b1 : 1'b0    ;

    always @(posedge clk, negedge rst_n) begin
        if (!rst_n) begin
            r_state     <= p_STATE_0;
        end else begin
            case (r_state)
                p_STATE_0:  begin                   // INITIAL
                    if (seq_in == 1'b0) begin
                        r_state     <= p_STATE_0;
                    end else begin
                        r_state     <= p_STATE_1;
                    end
                end

                p_STATE_1:  begin                   // 1
                    if (seq_in == 1'b0) begin
                        r_state     <= p_STATE_1;
                    end else begin
                        r_state     <= p_STATE_2;
                    end
                end

                p_STATE_2:  begin                   // 11
                    if (seq_in == 1'b0) begin
                        r_state     <= p_STATE_3;
                    end else begin
                        r_state     <= p_STATE_2;
                    end
                end

                p_STATE_3:  begin                   // 110
                    if (seq_in == 1'b0) begin
                        r_state     <= p_STATE_1;
                    end else begin
                        r_state     <= p_STATE_4;
                    end
                end

                p_STATE_4:  begin                   // 1101
                    if (seq_in == 1'b0) begin
                        r_state     <= p_STATE_0;
                    end else begin
```

```
65                          r_state     <= p_STATE_2;
66                      end
67                  end
68
69                  default:    begin
70                      r_state     <= p_STATE_0;
71                  end
72              endcase
73          end
74      end
75
76
77      always @(posedge clk, negedge rst_n) begin
78          if (!rst_n) begin
79              r_seq_in    <= 5'b0;
80          end else begin
81              r_seq_in    <= {r_seq_in[4:0], seq_in};
82          end
83      end
84
85  endmodule
```

（五）仿真波形



（六）实验效果

最终完整实现目标功能。

实验四　交通灯控制

（一）实验目的：学习设计优化和状态机的设计。学习较复杂数字系统设计；

（二）设计要求

实现一个由一条主干道和一条乡间公路形成的十字路口的交通灯控制器功能：

1、 有 MR（主红）、MY（主黄）、MG（主绿）、CR（乡红）、CY（乡黄）、CG（乡绿）六盏交通灯需要控制；

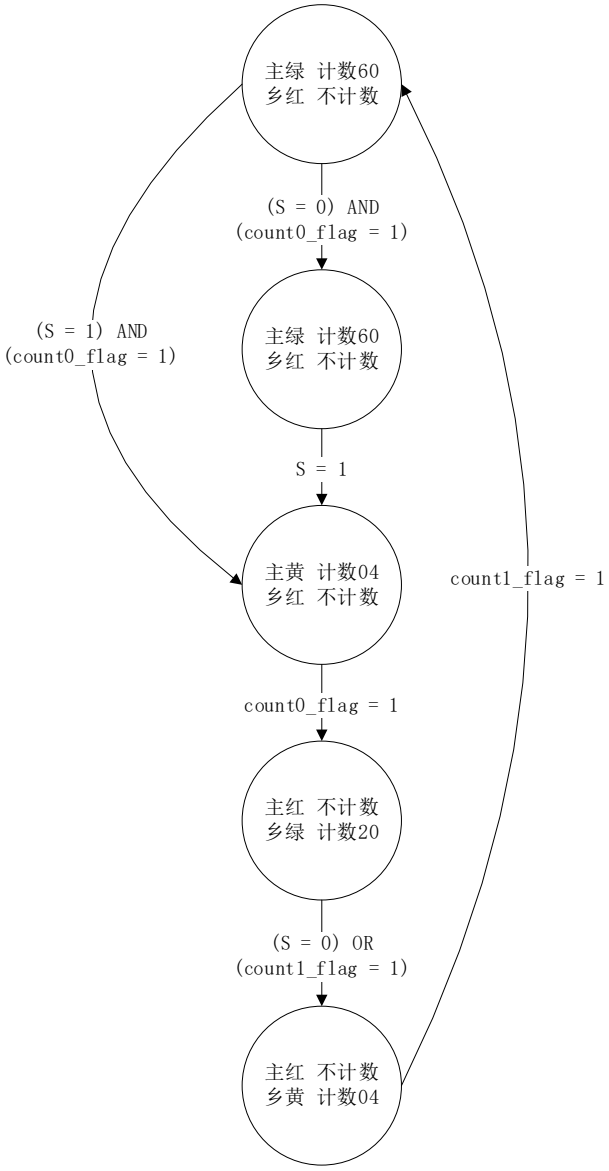
2、 交通灯由绿转红前有 4 秒亮黄灯的间隔时间，由红转绿没有间隔时间；

3、 乡间公路右侧各埋有一个串连传感器，当有车辆准备通过乡间公路时，发出请求信号 S=1，其余时间 S=0；

4、 平时系统停留在主干道通行（MGCR）状态，一旦 S 信号有效，经主道黄灯 4 秒（MYCR）状态后转入乡间公路通行（MRCG）状态，但要保证主干 60s 后才能转换；

5、 一旦 S 信号消失，系统脱离 MRCG 状态，即经乡道黄灯 4 秒（MRCY）状态进入 MGCR 状态，即使 S 信号一直有效，MRCG 状态也不得长于 20 秒钟；

（三）序列检测状态转移图



（四）实验程序

(1) 顶层文件（直接包含状态机逻辑部分）

```
module E4_TrafficLights (
    input                               clk                 ,
    input                               rst_N               ,

    input                               s                   ,

    output              [2:0]           ryg0                ,
    output              [2:0]           ryg1                ,

    output              [13:0]          digital_tube_0      ,
    output              [13:0]          digital_tube_1
);

    parameter                           State_0             = 3'd0      ,
                                        State_1             = 3'd1      ,
                                        State_2             = 3'd2      ,
                                        State_3             = 3'd3      ,
                                        State_4             = 3'd4      ;


    reg                 [2:0]           r_ryg0              ;
    reg                 [2:0]           r_ryg1              ;

    wire                                s_clk_1Hz           ;

    wire                                s_timer0_flag       ;
    wire                                s_timer1_flag       ;

    wire                [7:0]           s_number0_BCD       ;
    wire                [7:0]           s_number1_BCD       ;

    reg                                 r_dt0_enable        ;
    reg                 [5:0]           r_dt0_cnt_num       ;
    wire                                s_dt0_enable        ;
    wire                [5:0]           s_dt0_cnt_num       ;

    reg                                 r_dt1_enable        ;
    reg                 [5:0]           r_dt1_cnt_num       ;
    wire                                s_dt1_enable        ;
    wire                [5:0]           s_dt1_cnt_num       ;

    reg                 [2:0]           r_state             ;

    assign              s_dt0_enable    = r_dt0_enable      ;
    assign              s_dt0_cnt_num   = r_dt0_cnt_num     ;

    assign              s_dt1_enable    = r_dt1_enable      ;
    assign              s_dt1_cnt_num   = r_dt1_cnt_num     ;

    assign              ryg0            = r_ryg0            ;
    assign              ryg1            = r_ryg1            ;


    ClockDivider ClockDivider_inst (
        .clkin              ( clk                   ),
        .rst_N              ( rst_N                 ),
        .clkout             ( s_clk_1Hz             )
    );

    Timer Timer0_inst (
        .clk                ( s_clk_1Hz             ),
        .rst_N              ( rst_N                 ),
```

```
62          .enable             ( s_dt0_enable          ),
63          .count_num          ( s_dt0_cnt_num         ),
64          .flag_re            ( s_timer0_flag         ),
65          .number_BCD         ( s_number0_BCD         )
66      );
67
68      Timer Timer1_inst (
69          .clk                ( s_clk_1Hz             ),
70          .rst_N              ( rst_N                 ),
71          .enable             ( s_dt1_enable          ),
72          .count_num          ( s_dt1_cnt_num         ),
73          .flag_re            ( s_timer1_flag         ),
74          .number_BCD         ( s_number1_BCD         )
75      );
76
77      DigitalTube_2b DigitalTube_2b_0_inst (
78          .clk                ( clk                   ),
79          .rst_N              ( rst_N                 ),
80          .enable             ( r_dt0_enable          ),
81          .number_BCD         ( s_number0_BCD         ),
82          .pin_out            ( digital_tube_0        )
83      );
84
85      DigitalTube_2b DigitalTube_2b_1_inst (
86          .clk                ( clk                   ),
87          .rst_N              ( rst_N                 ),
88          .enable             ( r_dt1_enable          ),
89          .number_BCD         ( s_number1_BCD         ),
90          .pin_out            ( digital_tube_1        )
91      );
92

93      always @(posedge s_clk_1Hz or negedge rst_N) begin
94          if (!rst_N) begin
95              r_state         <= 3'd0;
96
97              r_ryg0          <= 3'b111;
98              r_ryg1          <= 3'b111;
99
100             r_dt0_enable    <= 0;
101             r_dt1_enable    <= 0;
102
103             r_dt0_cnt_num   <= 6'd60;
104             r_dt1_cnt_num   <= 6'd0;
105         end else begin
106             case (r_state)
107                 /* 主绿乡红 */
108                 3'd0 :      begin
109                     r_ryg0          <= 3'b001;
110                     r_ryg1          <= 3'b100;
111
112                     r_dt0_enable    <= 1;
113                     r_dt1_enable    <= 0;
114
115                     if (s_timer0_flag) begin
116                         if (s) begin
117                             r_state         <= 3'd2;
118
119                             r_dt0_cnt_num   <= 6'd4;
120                             r_dt1_cnt_num   <= 6'd0;
121                         end else begin
122                             r_state         <= 3'd1;
123
124                             r_dt0_cnt_num   <= 6'd60;
125                             r_dt1_cnt_num   <= 6'd0;
126                         end
```

```
127                     end else begin
128                         r_state         <= r_state;
129 
130                         r_dt0_cnt_num   <= 6'd60;
131                         r_dt1_cnt_num   <= 6'd0;
132                     end
133                 end
134 
135                 /* 主绿乡红 */
136                 3'd1 :      begin
137                     r_ryg0          <= 3'b001;
138                     r_ryg1          <= 3'b100;
139 
140                     r_dt0_enable    <= 1;
141                     r_dt1_enable    <= 0;
142 
143                     if (s) begin
144                         r_state         <= 3'd2;
145 
146                         r_dt0_cnt_num   <= 6'd4;
147                         r_dt1_cnt_num   <= 6'd0;
148                     end else begin
149                         r_state         <= r_state;
150 
151                         r_dt0_cnt_num   <= 6'd60;
152                         r_dt1_cnt_num   <= 6'd0;
153                     end
154                 end
155 
156                 /* 主黄乡红 */
157                 3'd2 :      begin
158                     r_ryg0          <= 3'b010;
159                     r_ryg1          <= 3'b100;
160 
161                     r_dt0_enable    <= 1;
162                     r_dt1_enable    <= 0;
163 
164                     if (s_timer0_flag) begin
165                         r_state         <= 3'd3;
166 
167                         r_dt0_cnt_num   <= 6'd0;
168                         r_dt1_cnt_num   <= 6'd20;
169                     end else begin
170                         r_state         <= r_state;
171 
172                         r_dt0_cnt_num   <= 6'd4;
173                         r_dt1_cnt_num   <= 6'd0;
174                     end
175                 end
176 
177                 /* 主红乡绿 */
178                 3'd3 :      begin
179                     r_ryg0          <= 3'b100;
180                     r_ryg1          <= 3'b001;
181 
182                     r_dt0_enable    <= 0;
183                     r_dt1_enable    <= 1;
184 
185                     if (!s || s_timer1_flag) begin
186                         r_state         <= 3'd4;
187 
188                         r_dt0_cnt_num   <= 6'd0;
189                         r_dt1_cnt_num   <= 6'd4;
190                     end else begin
191                         r_state         <= r_state;
```

```
192
193                         r_dt0_cnt_num   <= 6'd0;
194                         r_dt1_cnt_num   <= 6'd20;
195                     end
196                 end
197
198                 /* 主红乡黄 */
199                 3'd4 :      begin
200                     r_ryg0          <= 3'b100;
201                     r_ryg1          <= 3'b010;
202
203                     r_dt0_enable    <= 0;
204                     r_dt1_enable    <= 1;
205
206                     if (s_timer1_flag) begin
207                         r_state         <= 3'd0;
208
209                         r_dt0_cnt_num   <= 6'd60;
210                         r_dt1_cnt_num   <= 6'd0;
211                     end else begin
212                         r_state         <= r_state;
213
214                         r_dt0_cnt_num   <= 6'd0;
215                         r_dt1_cnt_num   <= 6'd4;
216                     end
217                 end
218
219                 default :   begin
220                     r_state         <= 3'd0;
221
222                     r_ryg0          <= 3'b111;
223                     r_ryg1          <= 3'b111;
224
225                     r_dt0_enable    <= 0;
226                     r_dt1_enable    <= 0;
227
228                     r_dt0_cnt_num   <= 6'd60;
229                     r_dt1_cnt_num   <= 6'd0;
230                 end
231             endcase
232         end
233     end
234

235 endmodule
```

(2) 时钟分频模块（ClockDivider）

（基本沿用实验 2 模块，此处略去）

(3) 数码管驱动模块

（基本沿用实验 2 模块，此处略去）

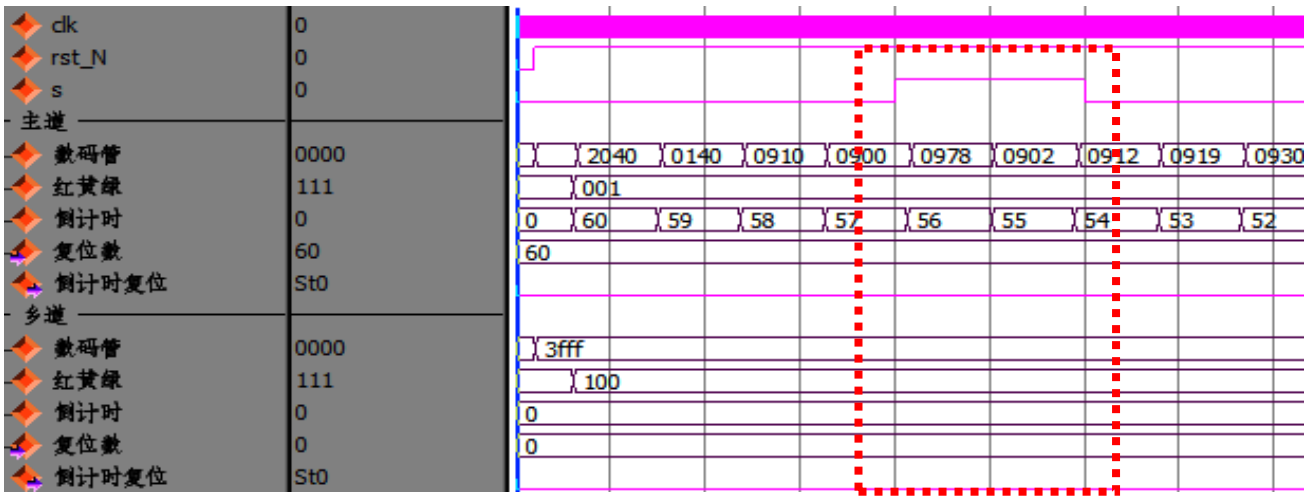
(4) 定时器模块

```
1   module Timer (
2       input                       clk                 ,
3       input                       rst_N               ,
4
5       input                       enable              ,
6       input           [5:0]       count_num           ,
7
8       output                      flag_re             ,
9       output          [7:0]       number_BCD
10  );
```

```
11
12      reg             [7:0]       r_number_BCD        ;
13      reg             [5:0]       r_count             ;
14
15
16      assign      flag_re         = (r_count == 6'd1) ? 1 : 0;
17      assign      number_BCD      = r_number_BCD      ;
18
19      always @(posedge clk or negedge rst_N)
20      begin
21          if (!rst_N) begin
22              r_count     <= 6'd0;
23          end
24          else begin
25              if (!enable) begin
26                  r_count     <= count_num;
27              end
28              else begin
29                  if ((r_count == 6'd0) || (r_count > count_num)) begin
30                      r_count     <= count_num;
31                  end else begin
32                      r_count     <= r_count - 1;
33                  end
34              end
35          end
36      end
37
38      always @(posedge clk or negedge rst_N)
39      begin
40          if (!rst_N) begin
41              r_number_BCD[3:0] <= count_num % 10;
42              r_number_BCD[7:4] <= count_num / 10;
43          end
44          else begin
45              r_number_BCD[3:0] <= r_count % 10;
46              r_number_BCD[7:4] <= r_count / 10;
47          end
48      end
49
50  endmodule
```

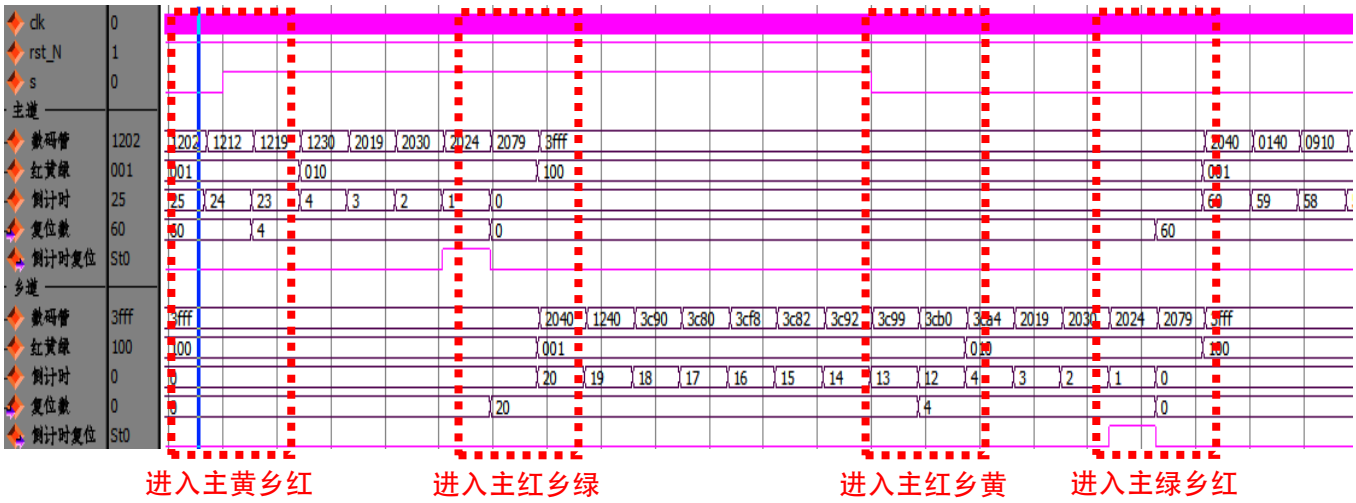
（五）仿真波形

(1) 主道第一次 60s 绿灯不被 s 信号打断

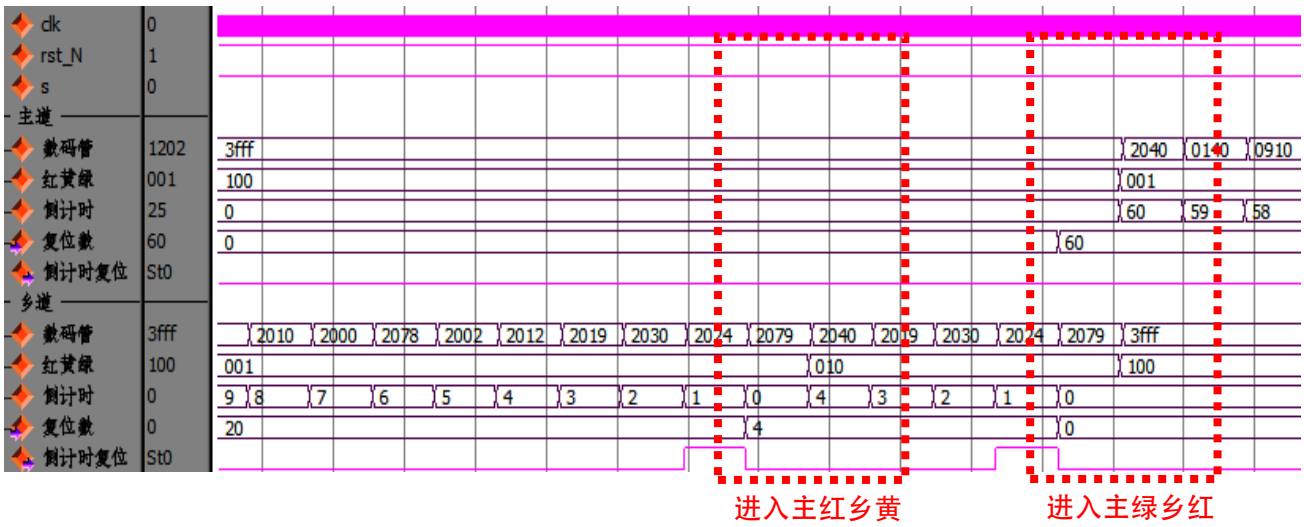


S 信号不打断第一次 60s

(2) 主道绿灯 60s 后被 s 信号打断，并依次进入“主黄乡红”→“主红乡绿”，并在 s 信号结束后依次进入“主红乡黄”→“主绿乡红”



(3) 若“主红乡绿”超过 20s，s 信号依然为高，但依然依次进入“主红乡黄”→“主绿乡红”



（六）实验效果

1. 复位时，起始状态是主绿乡红，数码管从 60 开始倒计时。

2. 当 60s 减 1 计数完成后，如果 s 为 1（代表乡干道有车要求通过）时，变为主黄乡绿状态，数码管 4s 倒计时；如果 s 为 0， 则回到起始状态，主绿乡红重新 60s 倒计时。

3. 4s 倒计时后，进入主红乡绿状态，如果此时 s 信号为 0，则立即转入主红乡黄状态；如果 s 信号一直为 1，则数码管开始 20s 倒计时，计数期间一旦出现 s 信号为 0，则立即转入主红乡黄状态，即使 s 信号一直为 1，当 20s 倒计时完成后也会入主红乡黄状态。

4. 主红乡黄，数码管开始 4s 倒计时，计数完成后进入主绿乡红状态，数码管 60s 倒计时，重复上述状态。

实验五　多功能数字钟设计

（一）实验目的

1、 学习综合且较复杂数字系统设计；

2、 学习多层次、多模块数字系统设计；

（二）设计要求

1、 数码管显示时、分、秒；

2、 具有正常计时和调时、调分等校时功能；

3、 经设置应具有整点报时功能（在 59 分 56 秒后开始报时，并用一串 LED 管显示）；

4、 经设置应具有闹钟功能（用 LED 管点亮表示，时间为一分钟）；

（三）实验程序

(1) 顶层模块（仅包含时钟分频、驱动模块和逻辑模块共三部分）

```
1   module E5_DigitalClock (
2       input                               clk                 ,
3       input                               rst_N               ,
4
5       input                               k                   , // alarm
6       input                               k1                  , // alarm_en
7       input                               k2                  , // stopwatch
8
9       input                               remin               ,
10      input                               rehour              ,
11
12      input                               key1_N              , // up     / pause
13      input                               key2_N              , // down   / reset
14
15      input               [ 1:0]          fast                ,
16
17      output                              LED_hourly          ,
18      output                              LED_alarm           ,
19
20      output              [41:0]          DigitalTube_out
21  );
22
23      wire                                s_clk_1kHz          ;
24      // wire                             s_clk_key           ;
25
26      wire                [ 2:0]          s_DTube_en          ;
27      wire                [ 2:0]          s_Twinkle_en        ;
28      wire                [23:0]          s_number_BCD        ;
29

30      ClockDivider ClockDivider_inst (
31          .clkin              ( clk                   ),
32          .rst_N              ( rst_N                 ),
33          .fast               ( fast                  ),
34          .clkout             ( s_clk_1kHz            ),
35          // .clkkey              ( s_clk_key             )
36      );
37
38      DigitalClock_Drive DigitalClock_Drive_inst (
39          .clk                ( clk                   ),
40          .rst_N              ( rst_N                 ),
41          .DTube_en           ( s_DTube_en            ),
42          .Twinkle_en         ( s_Twinkle_en          ),
43          .number_BCD         ( s_number_BCD          ),
44          .Dtube_out          ( DigitalTube_out       )
```

```
    );

    DigitalClock_Logic DigitalClock_Logic_inst (
        .clk                ( s_clk_1kHz            ),
        // .clkkey              ( s_clk_key             ),
        .rst_N              ( rst_N                 ),
        .alarm              ( k                     ),
        .alarm_en           ( k1                    ),
        .stopwatch          ( k2                    ),
        .remin              ( remin                 ),
        .rehour             ( rehour                ),
        .key1_N             ( key1_N                ),
        .key2_N             ( key2_N                ),
        .DTube_en           ( s_DTube_en            ),
        .Twinkle_en         ( s_Twinkle_en          ),
        .number_BCD         ( s_number_BCD          ),
        .LED_hourly         ( LED_hourly            ),
        .LED_alarm          ( LED_alarm             )
    );

endmodule
```

(2) 时钟分频模块（ClockDivider）

（基本沿用实验 2 模块，并增添加速功能便于测试，此处略去）

(3) 驱动模块（仅包含数码管驱动）

```
module DigitalClock_Drive (
    input                               clk                 ,
    input                               rst_N               ,

    input               [ 2:0]          DTube_en            ,
    input               [ 2:0]          Twinkle_en          ,
    input               [23:0]          number_BCD          ,

    output              [41:0]          Dtube_out
);


    DigitalTube_6b DigitalTube_6b_inst (
        .clk                ( clk                   ),
        .rst_N              ( rst_N                 ),
        .enable             ( { {2{DTube_en[2]}}, {2{DTube_en[1]}}, {2{DTube_en[0]}} }
  ),
        .twinkle            ( { {2{Twinkle_en[2]}}, {2{Twinkle_en[1]}}, {2{Twinkle_en[0]}} }
  ),
        .number_BCD         ( number_BCD            ),
        .pin_out            ( Dtube_out             )
    );


endmodule
```

(4) 6 位数码管驱动模块（DigitalTube_6b）

（其基本沿用实验 2 模块，此外通过使能信号增添闪烁功能，此处略去）

(5) 逻辑模块（包含按键消抖、定时器的实例化）

```
module DigitalClock_Logic (
    input                               clk                 , // 1kHz
    // input                                clkkey              ,
    input                               rst_N               ,

```

```
6       input                           alarm               ,
7       input                           stopwatch           ,
8
9       input                           alarm_en            ,
10
11      input                           remin               ,
12      input                           rehour              ,
13
14      input                           key1_N              , // up     / pause
15      input                           key2_N              , // down   / reset
16
17      output          [ 2:0]          DTube_en            ,
18      output          [ 2:0]          Twinkle_en          ,
19      output          [23:0]          number_BCD          ,
20
21      output                          LED_hourly          ,
22      output                          LED_alarm
23  );
24
25      parameter                       State_0             = 2'b00     , // clock
26                                      State_1             = 2'b01     , // alarm
27                                      State_2             = 2'b10     ; // stopwatch
28
29
30      wire            [ 1:0]          s_state             ;
31
32      wire            [ 5:0]          s_clkMode_sec       ;
33      wire            [ 5:0]          s_clkMode_min       ;
34      wire            [ 4:0]          s_clkMode_hour      ;
35
36      reg             [ 5:0]          r_alarm_min         ;
37      reg             [ 4:0]          r_alarm_hour        ;
38
39      wire            [ 5:0]          s_alarm_min         ;
40      wire            [ 4:0]          s_alarm_hour        ;
41
42      wire            [ 7:0]          s_num_1_BCD_alm     ;
43      wire            [ 7:0]          s_num_2_BCD_alm     ;
44
45      wire            [ 7:0]          s_num_0_BCD         ;
46      wire            [ 7:0]          s_num_1_BCD         ;
47      wire            [ 7:0]          s_num_2_BCD         ;
48
49      wire            [ 7:0]          s_num_0_BCD_clk     ;
50      wire            [ 7:0]          s_num_1_BCD_clk     ;
51      wire            [ 7:0]          s_num_2_BCD_clk     ;
52
53      wire            [ 7:0]          s_num_0_BCD_sw      ;
54      wire            [ 7:0]          s_num_1_BCD_sw      ;
55      wire            [ 7:0]          s_num_2_BCD_sw      ;
56
57      reg                             r_alarm_en          ;
58
59      wire                            s_flag_PressKey1    ;
60      wire                            s_flag_PressKey2    ;
61
62      reg                             r_timer_en_sw       ;
63      reg                             r_timer_rst_sw      ;
64      wire                            s_timer_en_sw       ;
65      wire                            s_timer_rst_sw      ;
66
67      wire                            s_flag_incmin       ;
68      wire                            s_flag_decmin       ;
69      wire                            s_flag_inchour      ;
70      wire                            s_flag_dechour      ;
71
```

```
72      reg                             r_flag_incmin       ;
73      reg                             r_flag_decmin       ;
74      reg                             r_flag_inchour      ;
75      reg                             r_flag_dechour      ;
76
77      assign      s_flag_incmin       = r_flag_incmin     ;
78      assign      s_flag_decmin       = r_flag_decmin     ;
79      assign      s_flag_inchour      = r_flag_inchour    ;
80      assign      s_flag_dechour      = r_flag_dechour    ;
81
82      assign      s_alarm_hour        = r_alarm_hour      ;
83      assign      s_alarm_min         = r_alarm_min       ;
84
85      assign      s_timer_en_sw       = r_timer_en_sw     ;
86      assign      s_timer_rst_sw      = r_timer_rst_sw    ;
87
88      assign      s_state             = (stopwatch ? State_2 : (alarm ? State_1 : State_0));
89
90
91      assign      LED_hourly          = (s_clkMode_min == 6'd59) ? ((s_clkMode_sec > 6'd56)
    ? 1'b1 : 1'b0) : 1'b0;
92
93      assign      LED_alarm           = (alarm_en ? (((s_clkMode_min == s_alarm_min) && (s_c
    lkMode_hour == s_alarm_hour)) ? 1'b1 : 1'b0) : 1'b0);
94
95      assign      DTube_en            = (s_state == State_1) ? 3'b110 : 3'b111;
96
97      assign      Twinkle_en[2]       = (s_state != State_2) ? (rehour ? 1'b1 : 1'b0 ) : 1'b
    0;
98      assign      Twinkle_en[1]       = (s_state != State_2) ? (remin ? 1'b1 : 1'b0 ) : 1'b0
    ;
99      assign      Twinkle_en[0]       = 1'b0;
100
101     assign      s_num_1_BCD_alm[3:0]    = s_alarm_min % 4'd10;
102     assign      s_num_1_BCD_alm[7:4]    = s_alarm_min / 4'd10;
103     assign      s_num_2_BCD_alm[3:0]    = s_alarm_hour % 4'd10;
104     assign      s_num_2_BCD_alm[7:4]    = s_alarm_hour / 4'd10;
105
106     assign      s_num_0_BCD         = (s_state == State_0) ? s_num_0_BCD_clk : ((s_state =
    = State_1) ? 8'd0 : s_num_0_BCD_sw);
107     assign      s_num_1_BCD         = (s_state == State_0) ? s_num_1_BCD_clk : ((s_state =
    = State_1) ? s_num_1_BCD_alm : s_num_1_BCD_sw);
108     assign      s_num_2_BCD         = (s_state == State_0) ? s_num_2_BCD_clk : ((s_state =
    = State_1) ? s_num_2_BCD_alm : s_num_2_BCD_sw);
109
110     assign      number_BCD          = {s_num_2_BCD, s_num_1_BCD, s_num_0_BCD};
111
112     Timer Timer_ClockMode_inst_0 (
113         .clk                ( clk                   ),
114         .rst_N              ( rst_N                 ),
115         .softrst_N          ( 1'd1                  ),
116         .enable             ( 1'd1                  ),
117         .mode               ( 1'd1                  ),
118         .flag_incmin        ( s_flag_incmin         ),
119         .flag_decmin        ( s_flag_decmin         ),
120         .flag_inchour       ( s_flag_inchour        ),
121         .flag_dechour       ( s_flag_dechour        ),
122         // .ms                  (           ),
123         .sec                ( s_clkMode_sec         ),
124         .min                ( s_clkMode_min         ),
125         .hour               ( s_clkMode_hour            ),
126         .num_0_BCD          ( s_num_0_BCD_clk           ),
127         .num_1_BCD          ( s_num_1_BCD_clk           ),
```

```
128         .num_2_BCD          ( s_num_2_BCD_clk           )
129     );
130
131     Timer Timer_StopwatchMode_inst_1 (
132         .clk                ( clk                   ),
133         .rst_N              ( rst_N                 ),
134         .softrst_N          ( s_timer_rst_sw        ),
135         .enable             ( s_timer_en_sw         ),
136         .mode               ( 1'd0                  ),
137         .flag_incmin        ( 1'b0                  ),
138         .flag_decmin        ( 1'b0                  ),
139         .flag_inchour       ( 1'b0                  ),
140         .flag_dechour       ( 1'b0                  ),
141         // .ms                 (           ),
142         // .sec                (           ),
143         // .min                (           ),
144         // .hour               (           ),
145         .num_0_BCD          ( s_num_0_BCD_sw            ),
146         .num_1_BCD          ( s_num_1_BCD_sw            ),
147         .num_2_BCD          ( s_num_2_BCD_sw            )
148     );
149
150     KeyRecognition KeyRecognition_inst_0 (
151         .clk                ( clk                       ),
152         .rst_N              ( rst_N                     ),
153         .key_N              ( key1_N                    ),
154         .flag_press         ( s_flag_PressKey1          )
155     );
156
157     KeyRecognition KeyRecognition_inst_1 (
158         .clk                ( clk                       ),
159         .rst_N              ( rst_N                     ),
160         .key_N              ( key2_N                    ),
161         .flag_press         ( s_flag_PressKey2          )
162     );
163
164     always @(posedge clk or negedge rst_N) begin
165         if (!rst_N) begin
166             r_flag_incmin   <= 1'b0;
167             r_flag_inchour  <= 1'b0;
168             r_flag_decmin   <= 1'b0;
169             r_flag_dechour  <= 1'b0;
170         end else begin
171             if (s_state == State_0) begin
172                 case ({s_flag_PressKey2, s_flag_PressKey1})
173                     2'b01:      begin
174                         r_flag_inchour  <= rehour;
175                         r_flag_incmin   <= remin;
176                         r_flag_decmin   <= 1'b0;
177                         r_flag_dechour  <= 1'b0;
178                     end
179
180                     2'b10:      begin
181                         r_flag_incmin   <= 1'b0;
182                         r_flag_inchour  <= 1'b0;
183                         r_flag_dechour  <= rehour;
184                         r_flag_decmin   <= remin;
185                     end
186
187                     default:    begin
188                         r_flag_incmin   <= 1'b0;
189                         r_flag_inchour  <= 1'b0;
190                         r_flag_decmin   <= 1'b0;
191                         r_flag_dechour  <= 1'b0;
```

```
                    end
                endcase
            end
        end
    end

    always @(posedge clk or negedge rst_N) begin
        if (!rst_N) begin
            r_alarm_hour    <= 5'd0;
            r_alarm_min     <= 6'd0;
        end else begin
            if (s_state == State_1) begin
                case ({s_flag_PressKey2, s_flag_PressKey1})
                    2'b01:      begin
                        if (rehour) begin
                            if (r_alarm_hour < 5'd23) begin
                                r_alarm_hour    <= r_alarm_hour + 5'd1;
                            end else begin
                                r_alarm_hour    <= 5'd0;
                            end
                        end

                        if (remin) begin
                            if (r_alarm_min < 6'd59) begin
                                r_alarm_min     <= r_alarm_min + 6'd1;
                            end else begin
                                r_alarm_min     <= 6'd0;

                                if (!rehour) begin
                                    if (r_alarm_hour < 5'd23) begin
                                        r_alarm_hour    <= r_alarm_hour + 5'd1;
                                    end else begin
                                        r_alarm_hour    <= 5'd0;
                                    end
                                end
                            end
                        end
                    end

                    2'b10:      begin
                        if (rehour) begin
                            if (r_alarm_hour > 5'd0) begin
                                r_alarm_hour    <= r_alarm_hour - 5'd1;
                            end else begin
                                r_alarm_hour    <= 5'd23;
                            end
                        end

                        if (remin) begin
                            if (r_alarm_min > 6'd0) begin
                                r_alarm_min     <= r_alarm_min - 6'd1;
                            end else begin
                                r_alarm_min     <= 6'd59;

                                if (!rehour) begin
                                    if (r_alarm_hour > 5'd0) begin
                                        r_alarm_hour    <= r_alarm_hour - 5'd1;
                                    end else begin
                                        r_alarm_hour    <= 5'd23;
                                    end
                                end
                            end
                        end
                    end

```

```
                    default:    begin
                        r_alarm_hour    <= r_alarm_hour;
                        r_alarm_min     <= r_alarm_min;
                    end
                endcase
            end
        end
    end

    always @(posedge clk or negedge rst_N) begin
        if (!rst_N) begin
            r_timer_en_sw   <= 1'd0;
            r_timer_rst_sw  <= 1'd0;
        end else begin
            if (s_state == State_2) begin
                case ({s_flag_PressKey2, s_flag_PressKey1})
                    2'b01:      begin
                        r_timer_en_sw   <=~r_timer_en_sw;
                        r_timer_rst_sw  <= 1'd1;
                    end

                    2'b10:      begin
                        r_timer_en_sw   <= 1'd0;
                        r_timer_rst_sw  <= 1'd0;
                    end

                    2'b11:      begin
                        r_timer_en_sw   <=~r_timer_en_sw;
                        r_timer_rst_sw  <= 1'd0;
                    end

                    default:    begin
                        r_timer_en_sw   <= r_timer_en_sw;
                        r_timer_rst_sw  <= 1'd1;
                    end
                endcase
            end
        end
    end

endmodule
```

(6) 按键消抖模块

（实现原理较简单，此处略去）

(7) 定时器模块（包含软件复位、使能切换、输出模式选择、时分加减信号功能）

```
module Timer (
    input                               clk                 ,
    input                               rst_N               ,
    input                               softrst_N           ,

    input                               enable              ,
    input                               mode                ,

    // input                                flag_rstsec         ,
    input                               flag_incmin         ,
    input                               flag_decmin         ,
    input                               flag_inchour        ,
    input                               flag_dechour        ,

    output              [6:0]           ms                  ,
    output              [5:0]           sec                 ,
    output              [5:0]           min                 ,
```

```
    output              [4:0]           hour                ,

    output              [7:0]           num_0_BCD           ,
    output              [7:0]           num_1_BCD           ,
    output              [7:0]           num_2_BCD
);

    parameter                           Mode_0              = 1'd0      ,
                                        Mode_1              = 1'd1      ;

    reg                 [9:0]           r_ms                ;
    reg                 [5:0]           r_sec               ;
    reg                 [5:0]           r_min               ;
    reg                 [4:0]           r_hour              ;

    reg                 [7:0]           r_num_0_BCD         ;
    reg                 [7:0]           r_num_1_BCD         ;
    reg                 [7:0]           r_num_2_BCD         ;

    assign      ms                      = r_ms              ;
    assign      sec                     = r_sec             ;
    assign      min                     = r_min             ;
    assign      hour                    = r_hour            ;

    assign      num_0_BCD               = r_num_0_BCD       ;
    assign      num_1_BCD               = r_num_1_BCD       ;
    assign      num_2_BCD               = r_num_2_BCD       ;

    always @(posedge clk or negedge rst_N or negedge softrst_N)
    begin
        if (!rst_N || !softrst_N) begin
            r_ms    <= 10'd0;
            r_sec   <= 6'd0;
            r_min   <= 6'd0;
            r_hour  <= 5'd0;
        end
        else begin
            if (!enable) begin
                r_ms    <= r_ms;
                r_sec   <= r_sec;
                r_min   <= r_min;
                r_hour  <= r_hour;
            end
            else begin
                if (flag_incmin && !flag_decmin) begin
                    if (r_min < 6'd59) begin
                        r_min   <= r_min + 6'd1;
                    end else begin
                        r_min   <= 6'd0;

                        if (r_hour < 5'd23) begin
                            r_hour  <= r_hour + 5'd1;
                        end else begin
                            r_hour  <= 5'd0;
                        end
                    end
                end

                if (!flag_incmin && flag_decmin) begin
                    if (r_min > 6'd0) begin
                        r_min   <= r_min - 6'd1;
                    end else begin
                        r_min   <= 6'd59;

                        if (r_hour > 5'd0) begin
                            r_hour  <= r_hour - 5'd1;
```

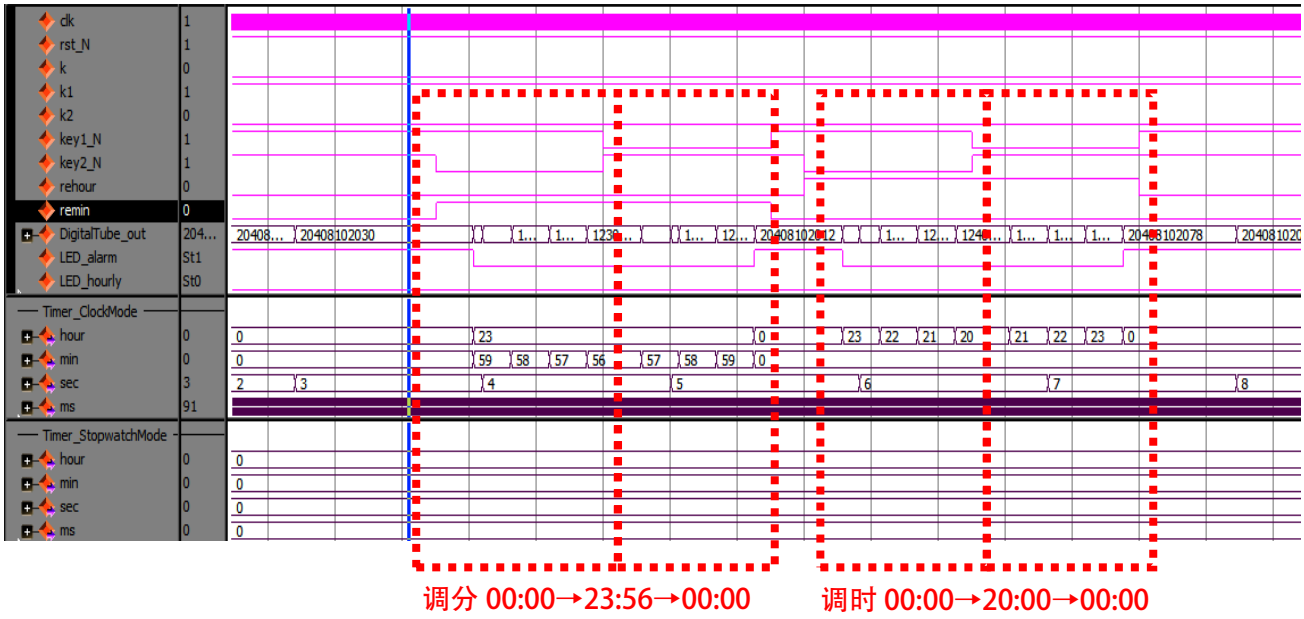
```
                    end else begin
                        r_hour  <= 5'd23;
                    end
                end
            end

            if (flag_inchour && !flag_dechour) begin
                if (r_hour < 5'd23) begin
                    r_hour  <= r_hour + 5'd1;
                end else begin
                    r_hour  <= 5'd0;
                end
            end

            if (!flag_inchour && flag_dechour) begin
                if (r_hour > 5'd0) begin
                    r_hour  <= r_hour - 5'd1;
                end else begin
                    r_hour  <= 5'd23;
                end
            end

            if (!flag_incmin && !flag_decmin && !flag_inchour && !flag_dechour) begin
                if (r_ms < 10'd999) begin
                    r_ms    <= r_ms + 10'd1;
                end else begin
                    r_ms    <= 10'd0;

                    if (r_sec < 6'd59) begin
                        r_sec   <= r_sec + 6'd1;
                    end else begin
                        r_sec   <= 6'd0;

                        if (r_min < 6'd59) begin
                            r_min   <= r_min + 6'd1;
                        end else begin
                            r_min   <= 6'd0;

                            if (r_hour < 5'd23) begin
                                r_hour  <= r_hour + 5'd1;
                            end else begin
                                r_hour  <= 5'd0;
                            end
                        end
                    end
                end
            end
        end
    end
end


always @(posedge clk or negedge rst_N or negedge softrst_N)
begin
    if (!rst_N || !softrst_N) begin
        r_num_0_BCD[3:0]    <= 4'd0;
        r_num_0_BCD[7:4]    <= 4'd0;
    end
    else begin
        if ((mode == Mode_0) && (r_hour == 7'd0)) begin
            r_num_0_BCD[3:0]    <= r_ms / 10 % 10;
            r_num_0_BCD[7:4]    <= r_ms / 100;
        end else begin
            r_num_0_BCD[3:0]    <= r_sec % 10;
            r_num_0_BCD[7:4]    <= r_sec / 10;
        end
```

```
        end
    end

    always @(posedge clk or negedge rst_N or negedge softrst_N)
    begin
        if (!rst_N || !softrst_N) begin
            r_num_1_BCD[3:0]    <= 4'd0;
            r_num_1_BCD[7:4]    <= 4'd0;
        end
        else begin
            if ((mode == Mode_0) && (r_hour == 7'd0)) begin
                r_num_1_BCD[3:0]    <= r_sec % 10;
                r_num_1_BCD[7:4]    <= r_sec / 10;
            end else begin
                r_num_1_BCD[3:0]    <= r_min % 10;
                r_num_1_BCD[7:4]    <= r_min / 10;
            end
        end
    end

    always @(posedge clk or negedge rst_N or negedge softrst_N)
    begin
        if (!rst_N || !softrst_N) begin
            r_num_2_BCD[3:0]    <= 4'd0;
            r_num_2_BCD[7:4]    <= 4'd0;
        end
        else begin
            if ((mode == Mode_0) && (r_hour == 7'd0)) begin
                r_num_2_BCD[3:0]    <= r_min % 10;
                r_num_2_BCD[7:4]    <= r_min / 10;
            end else begin
                r_num_2_BCD[3:0]    <= r_hour % 10;
                r_num_2_BCD[7:4]    <= r_hour / 10;
            end
        end
    end

endmodule
```

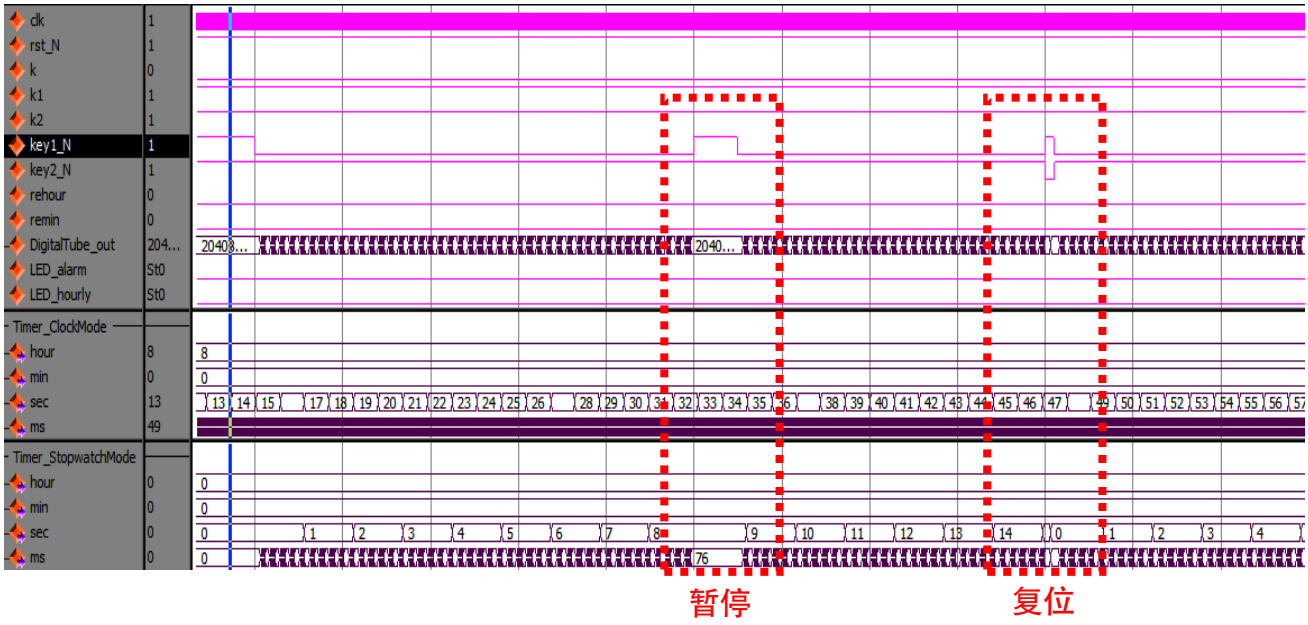
（四）实验波形

由于利用 ModelSim 可直接编写 Testbench 代码，因此以下图片均作反色处理以便于印刷。

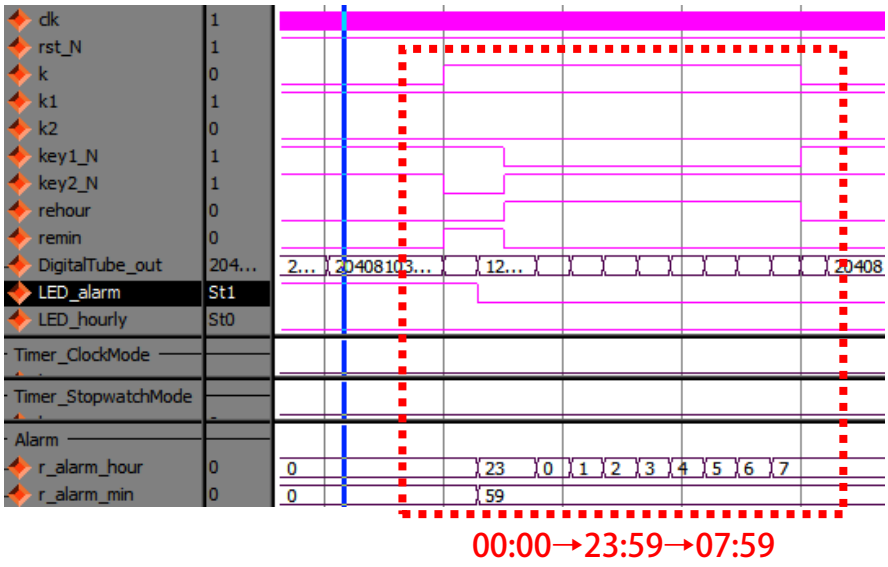
(1) 调时间——此处可看到时钟的定时器根据 remin、key1_N、key2_N 信号进行变化



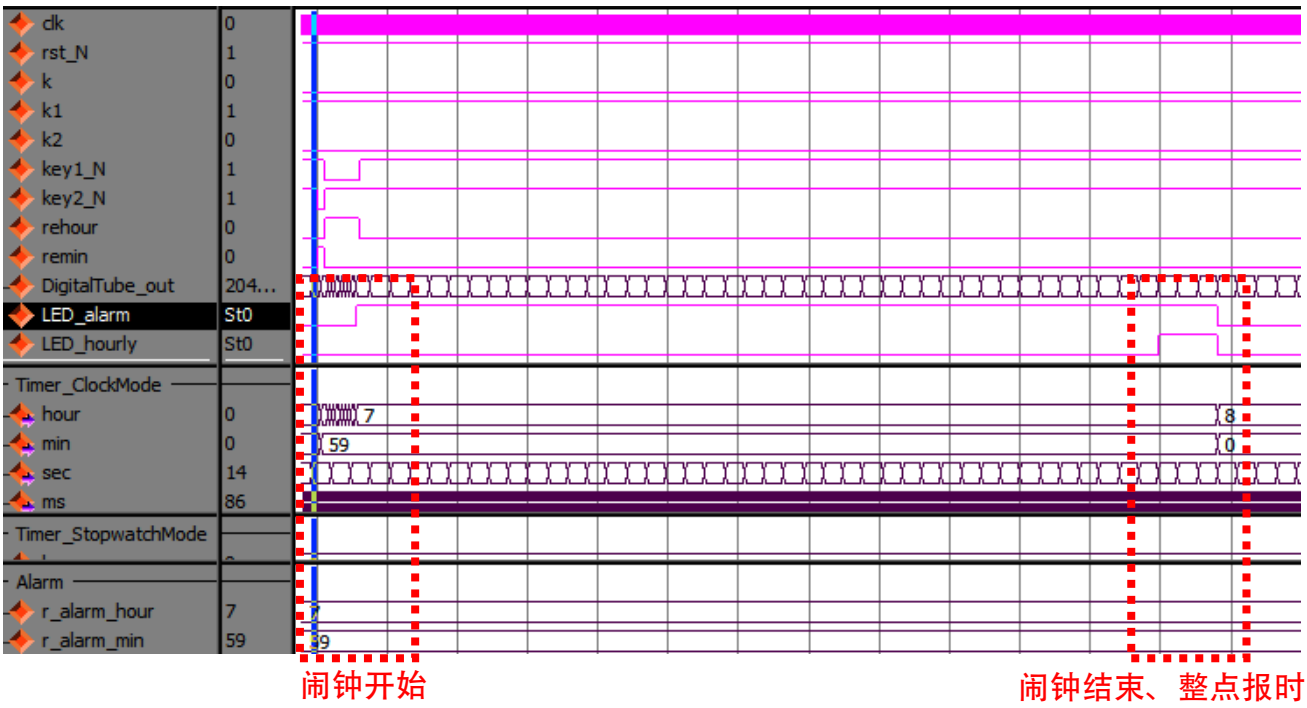
(2) 秒表——此处可看到 key1_N 信号执行了暂停功能、key2_N 信号执行了复位功能，同时不影响时钟的进行



(3) 闹钟调节——此处可看到闹钟时间发生变化



(4) 闹钟与整点报时——此处可看到闹钟 LED_alarm 在修改后的时间持续 1 分钟，整点报时 LED_hourly 在整点前持续 3 秒



（五）实验效果

当 RST 为 0 时，计数值清零，当 RST 为高电平时，开始计数，当计时到 59 分 56 秒后开始三秒的整点报时，用一串 LED 管显示，当 K 为高时进行正常计数，K 为低时进行正常的闹钟设置切换，当 REMIN 为低时，进行调分，当 REHOUR 为低时，进行调时，当到达闹钟所设定的时间时，进行闹铃功能，显示为 LED 管点亮一分钟。

此外，还在上述基础上增加调节时数码管对应位闪烁、长按连续调节、秒表（精度百分之一毫秒）等功能。

实验六　综合实验设计——VGA 数字钟

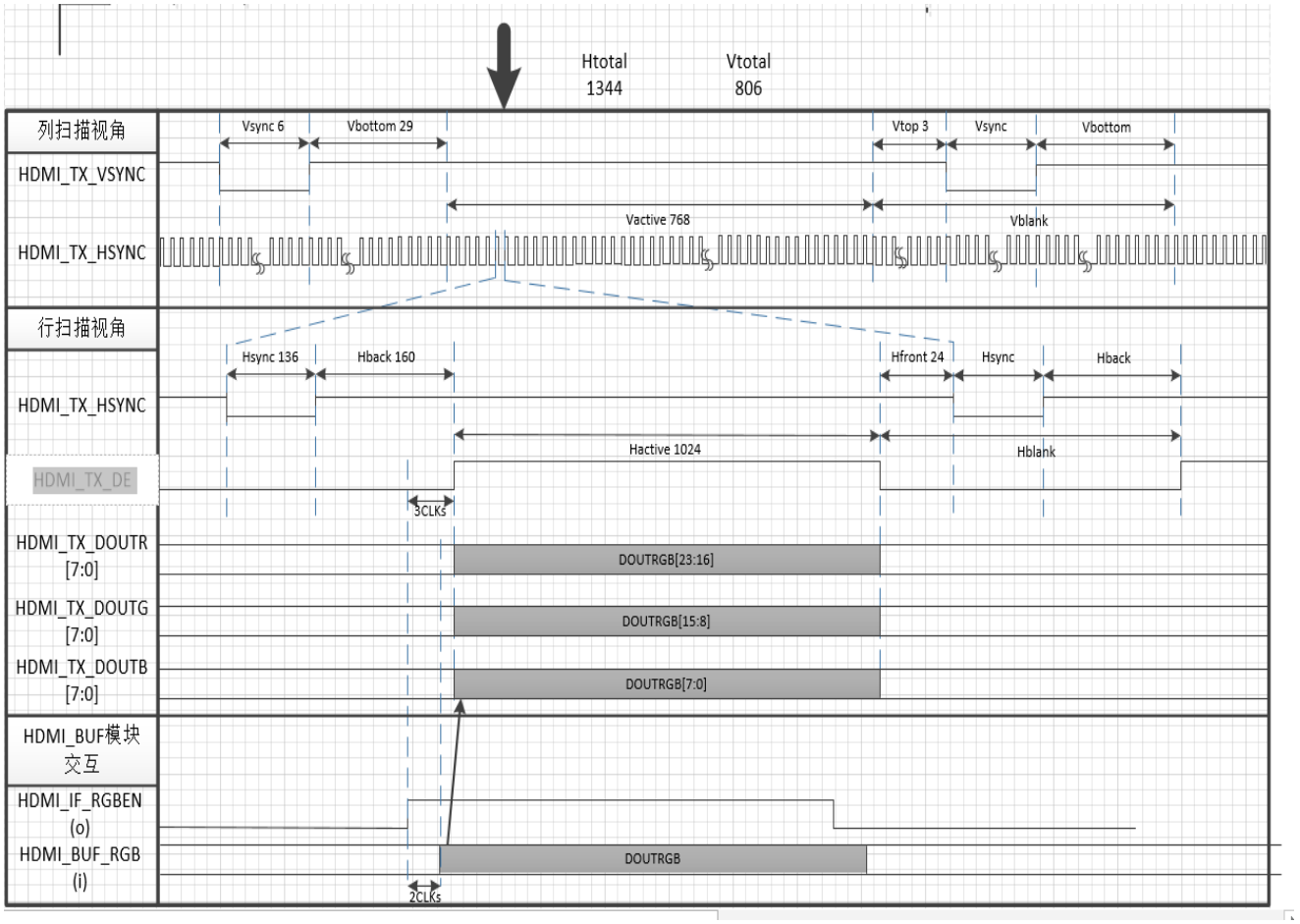
（一）实验目的：综合运用所学知识独立设计并完成一个综合实验。

（二）实验要求：

1. 完成 VGA 时序的编写，在分辨率 1024*768、刷新率 60Hz 的显示器上进行显示；
2. 使用 IP 核完成上述 VGA 时序要求的 65MHz 时钟生成；
3. 显示器背景实现循环彩虹渐变（即红橙黄绿蓝靛紫在 8 位色深下共 1536 色渐变）；
4. 按实验五同样要求在 VGA 显示器上显示数字钟；
5. 时钟、闹钟、整点报时等显示与背景实现透明叠加。

（三）实验原理：

1. VGA 时序图（在时序模块中将严格按照此时序进行编写）



2. 使用 IP 核实现 65MHz 时钟的生成
   由于 PLL 资源的分频倍频限制，因此共需要使用 2 个 PLL。
   第一个 PLL 输入 50MHz 产生 200MHz 时钟，第二个 PLL 输入 200MHz 产生 65MHz 时钟。
3. 循环彩虹渐变背景实现
   在 8 位色深显示下，RGB 三色按照(255,0,0)→(255,255,0)→(0,255,0)→(0,255,255)→(0,0,255)→(255,0,255)的规律循环改变，共计 256*6=1536 种颜色。
   使用一个寄存器存放当前 x 坐标为 0 的颜色，通过定时器将其不断改变。
   此外，页面中其他颜色均根据上述初始颜色进行依次上述计算并输出。
4. 透明叠加
   通过输出覆盖信号，然后使用或运算即可实现。

（四）实验程序

(1) 顶层模块（仅包含时钟分频、IP 核、驱动模块和逻辑模块共三部分）

```
module E6_VGA (
    input                       RST_N                   , // (i)
    input                       CLK                     , // (i)

    output                      VGA_CLK                 , // (o)
    output                      VGA_HSYNC               , // (o) VGA Horizontal SYNC
    output                      VGA_VSYNC               , // (o) VGA Vertical SYNC
    output          [ 7:0]      VGA_R                   , // (o) VGA Red
    output          [ 7:0]      VGA_G                   , // (o) VGA Green
    output          [ 7:0]      VGA_B                   , // (o) VGA Blue
    output                      VGA_SYNC_N              , // (o) 0
    output                      VGA_BLANK_N             , // (o) 1

    input                       k                       , // alarm
    input                       k1                      , // alarm_en
    input                       k2                      , // stopwatch

    input                       remin                   ,
    input                       rehour                  ,

    input                       key1_N                  , // up     / pause
    input                       key2_N                  , // down   / reset

    input           [ 1:0]      fast                    ,

    output                      LED_hourly              ,
    output                      LED_alarm               ,

    output          [41:0]      DigitalTube_out
);


    wire                        w_CLK_100M              ;
    wire                        w_CLK_200M              ;
    wire                        w_CLK_65M               ;

    wire                        w_CLK_1K                ;

    wire            [ 2:0]      w_DTube_en              ;
    wire            [ 2:0]      w_Twinkle_en            ;
    wire            [23:0]      w_number_BCD            ;

    wire                        w_HOURLY                ;
    wire                        w_ALARM                 ;


    PLL_0 PLL_0_inst (
        .refclk                 ( CLK                   ), // refclk.clk
        .rst                    (~RST_N                 ), // reset.reset
        .outclk_0               ( w_CLK_100M            ), // outclk0.clk
        .outclk_1               ( w_CLK_200M            ), // outclk1.clk
        .locked                 (                       )
    );

    PLL_1 PLL_1_inst (
        .refclk                 ( w_CLK_200M            ), // refclk.clk
        .rst                    (~RST_N                 ), // reset.reset
        .outclk_0               ( w_CLK_65M             ), // outclk0.clk
        .locked                 (                       )
    );

```

```
60      ClockDivider ClockDivider_inst (
61          .clkin                  ( CLK                   ),
62          .rst_N                  ( RST_N                 ),
63          .fast                   ( fast                  ),
64          .clkout                 ( w_CLK_1K              )
65      );
66
67      DigitalClock_Drive DigitalClock_Drive_inst (
68          .clk                    ( CLK                   ),
69          .rst_N                  ( RST_N                 ),
70          .DTube_en               ( w_DTube_en            ),
71          .Twinkle_en             ( w_Twinkle_en          ),
72          .number_BCD             ( w_number_BCD          ),
73          .HOURLY                 ( w_HOURLY              ),
74          .ALARM                  ( w_ALARM               ),
75          .Dtube_out              ( DigitalTube_out       ),
76          .LED_hourly             ( LED_hourly            ),
77          .LED_alarm              ( LED_alarm             ),
78          .VGA_CLK_IN             ( w_CLK_65M             ),
79          .VGA_CLK_OUT            ( VGA_CLK               ),
80          .VGA_HSYNC              ( VGA_HSYNC             ),
81          .VGA_VSYNC              ( VGA_VSYNC             ),
82          .VGA_R                  ( VGA_R                 ),
83          .VGA_G                  ( VGA_G                 ),
84          .VGA_B                  ( VGA_B                 ),
85          .VGA_SYNC_N             ( VGA_SYNC_N            ),
86          .VGA_BLANK_N            ( VGA_BLANK_N           ),
87      );
88
89      DigitalClock_Logic DigitalClock_Logic_inst (
90          .clk                    ( w_CLK_1K              ),
91          .rst_N                  ( RST_N                 ),
92          .alarm                  ( k                     ),
93          .alarm_en               ( k1                    ),
94          .stopwatch              ( k2                    ),
95          .remin                  ( remin                 ),
96          .rehour                 ( rehour                ),
97          .key1_N                 ( key1_N                ),
98          .key2_N                 ( key2_N                ),
99          .DTube_en               ( w_DTube_en            ),
100         .Twinkle_en             ( w_Twinkle_en          ),
101         .number_BCD             ( w_number_BCD          ),
102         .LED_hourly             ( w_HOURLY              ),
103         .LED_alarm              ( w_ALARM               )
104     );
105
106 endmodule
```

(2) IP 核 PLL 实例

（根据 IP 核新建向导点击即可自动生成模块，此处略去）

(3) 时钟分频模块（ClockDivider）

（完全沿用实验 5 模块，此处略去）

(4) 驱动模块（包含数码管驱动、VGA 时序、VGA 显示、数字闪烁逻辑）

```
1   module DigitalClock_Drive (
2       input                       clk                     ,
3       input                       rst_N                   ,
4
5       input           [ 2:0]      DTube_en                ,
6       input           [ 2:0]      Twinkle_en              ,
7       input           [23:0]      number_BCD              ,
```

```
8       input                       HOURLY                  ,
9       input                       ALARM                   ,
10
11      output          [41:0]      Dtube_out               ,
12      output                      LED_hourly              ,
13      output                      LED_alarm               ,
14
15      input                       VGA_CLK_IN              , // (i)
16      output                      VGA_CLK_OUT             , // (o)
17      output                      VGA_HSYNC               , // (o) VGA Horizontal SYNC
18      output                      VGA_VSYNC               , // (o) VGA Vertical SYNC
19      output          [ 7:0]      VGA_R                   , // (o) VGA Red
20      output          [ 7:0]      VGA_G                   , // (o) VGA Green
21      output          [ 7:0]      VGA_B                   , // (o) VGA Blue
22      output                      VGA_SYNC_N              , // (o) 0
23      output                      VGA_BLANK_N               // (o) 1
24  );
25
26      parameter                   Twinkle_Cnt             = 23'd5_000_000;
27
28
29      reg             [22:0]      r_cnt                   ;
30      reg                         r_twinkle               ;
31
32      reg             [ 5:0]      r_enable                ;
33      wire            [ 5:0]      w_enable                ;
34
35
36      wire                        w_VGA_IF_RGBEN          ;
37      wire            [23:0]      w_VGA_BUF_RGB           ;
38
39      assign                      LED_hourly              = HOURLY        ;
40      assign                      LED_alarm               = ALARM         ;
41
42      assign                      w_enable                = r_enable      ;
43
44      assign                      VGA_CLK_OUT             = VGA_CLK_IN    ;
45
46      DigitalTube_6b DigitalTube_6b_inst (
47          .clk                    ( clk                   ),
48          .rst_N                  ( rst_N                 ),
49          .enable                 ( w_enable              ),
50          .number_BCD             ( number_BCD            ),
51          .pin_out                ( Dtube_out             )
52      );
53
54      VGA_TIMING_8b VGA_TIMING_8b_inst(
55          .VGA_CLK                ( VGA_CLK_IN            ), // (i) 65M clock
56          .VGA_RST_N              ( rst_N                 ), // (i) reset, High Active
57
58          .VGA_HSYNC              ( VGA_HSYNC             ), // (o) HSYNC signal
59          .VGA_VSYNC              ( VGA_VSYNC             ), // (o) VSYNC signal
60          .VGA_R                  ( VGA_R                 ), // (o) Red signal
61          .VGA_G                  ( VGA_G                 ), // (o) Green signal
62          .VGA_B                  ( VGA_B                 ), // (o) Blue signal
63          .VGA_SYNC_N             ( VGA_SYNC_N            ), // (o) 0
64          .VGA_BLANK_N            ( VGA_BLANK_N           ), // (o) 1
65
66          .VGA_DE                 (                       ), // (o) DE signal
67          .VGA_IF_RGBEN           ( w_VGA_IF_RGBEN        ), // (o) if RGB enable
68          .VGA_BUF_RGB            ( w_VGA_BUF_RGB         )  // (i) buffer RGB
69      );
```

```
70

71      VGA_DISPLAY VGA_DISPLAY_inst(
72          .VGA_CLK                ( VGA_CLK_IN            ), // (i) vga clk in
73          .RST_N                  ( rst_N                 ), // (i) reset, High Active
74          .VGA_IF_RGBEN           ( w_VGA_IF_RGBEN        ), // (i)
75          .NUMBER_BCD             ( number_BCD            ), // (i)
76          .NUMBER_ENABLE          ( w_enable              ), // (i)
77          .HOURLY                 ( HOURLY                ), // (i)
78          .ALARM                  ( ALARM                 ), // (i)
79          .VGA_BUF_RGB            ( w_VGA_BUF_RGB         )  // (o) out
80      );
```

……

此处略去数字闪烁代码（与实验 5 相同）

……

```
81  endmodule
```

(5) 6 位数码管驱动模块（DigitalTube_6b）

（基本沿用实验 5 模块，但是将闪烁功能上移一层，此处略去）

(6) VGA 时序模块（8 位色深，严格按照 VGA 时序图实现）

```
1   module VGA_TIMING_8b(
2       input                       VGA_CLK                 , // (i) 65M clock
3       input                       VGA_RST_N               , // (i) reset, High Active
4
5       output                      VGA_HSYNC               , // (o) HSYNC signal
6       output                      VGA_VSYNC               , // (o) VSYNC signal
7       output          [ 7:0]      VGA_R                   , // (o) Red signal
8       output          [ 7:0]      VGA_G                   , // (o) Green signal
9       output          [ 7:0]      VGA_B                   , // (o) Blue signal
10      output                      VGA_SYNC_N              , // (o) 0
11      output                      VGA_BLANK_N             , // (o) 1
12
13      output                      VGA_DE                  , // (o) DE signal
14      output                      VGA_IF_RGBEN            , // (o) if RGB enable
15      input           [23:0]      VGA_BUF_RGB               // (i) buffer RGB
16  );
17
18  // ============================================================================
19  // Defination of parameter
20  // ============================================================================
21      parameter                   VGA_H_SyncPulse         = 11'd136           ; // Horizontal Sy
    nc Pulse      //  = 11'd4;    //
22      parameter                   VGA_H_BackPorch         = 11'd160           ; // Horizontal ba
    ck porch      //  = 11'd4;    //
23      parameter                   VGA_H_ActiveVideo       = 11'd1024          ; // Horizontal ac
    tive video    //  = 11'd5;    //
24      parameter                   VGA_H_FrontPorch        = 11'd24            ; // Horizontal fr
    ont porch     //  = 11'd4;    //
25
26      parameter                   VGA_V_SyncPulse         = 10'd6             ; // Vertical   Sy
    nc Pulse      //  = 11'd4;    //
27      parameter                   VGA_V_BackPorch         = 10'd29            ; // Vertical   ba
    ck porch      //  = 11'd4;    //
28      parameter                   VGA_V_ActiveVideo       = 10'd768           ; // Vertical   ac
    tive video    //  = 11'd4;    //
29      parameter                   VGA_V_FrontPorch        = 10'd3             ; // Vertical   fr
    ont porch     //  = 11'd4;    //
30
31      parameter                   VGA_H_time1             = VGA_H_SyncPulse                   ;
    // Horizontal Sync Pulse
```

```
32      parameter                   VGA_H_time2             = VGA_H_time1 + VGA_H_BackPorch     ;
    // Horizontal back porch
33      parameter                   VGA_H_time3             = VGA_H_time2 + VGA_H_ActiveVideo   ;
    // Horizontal active video
34      parameter                   VGA_H_time4             = VGA_H_time3 + VGA_H_FrontPorch    ;
    // Horizontal front porch
35
36      parameter                   VGA_V_time1             = VGA_V_SyncPulse                   ;
    // Vertical   Sync Pulse
37      parameter                   VGA_V_time2             = VGA_V_time1 + VGA_V_BackPorch     ;
    // Vertical   back porch
38      parameter                   VGA_V_time3             = VGA_V_time2 + VGA_V_ActiveVideo   ;
    // Vertical   active video
39      parameter                   VGA_V_time4             = VGA_V_time3 + VGA_V_FrontPorch    ;
    // Vertical   front porch
40
41  // ============================================================================
42  // Defination of Internal Signals
43  // ============================================================================
44      reg                         r_VGA_HSYNC             ;
45      reg                         r_VGA_VSYNC             ;
46      reg                         r_VGA_DE                ;
47      reg                         r_VGA_IF_RGBEN          ;
48      reg             [ 7:0]      r_VGA_R                 ;
49      reg             [ 7:0]      r_VGA_G                 ;
50      reg             [ 7:0]      r_VGA_B                 ;
51
52      reg             [10:0]      r_Hcount                ; // H sync pulse count
53      reg             [09:0]      r_Vcount                ; // H sync pulse count
54
55      wire                        w_if_acitve             ; // Active
56
57  // ============================================================================
58  // RTL Body
59  // ============================================================================
60      assign      VGA_HSYNC       = r_VGA_HSYNC           ;
61      assign      VGA_VSYNC       = r_VGA_VSYNC           ;
62      assign      VGA_DE          = r_VGA_DE              ;
63      assign      VGA_IF_RGBEN    = r_VGA_IF_RGBEN        ;
64      assign      VGA_R           = r_VGA_R               ;
65      assign      VGA_G           = r_VGA_G               ;
66      assign      VGA_B           = r_VGA_B               ;
67
68      assign      VGA_SYNC_N      = 1'b0                  ;
69      assign      VGA_BLANK_N     = VGA_HSYNC & r_VGA_VSYNC   ; //1'b1                   ;   //
70
71      assign      w_if_acitve     = ((VGA_V_time2 < r_Vcount) && (r_Vcount <= VGA_V_time3)) ? 1'
    b1 : 1'b0;
72

73      //------------------------------------------------------------------------
74      // V Control
75      //------------------------------------------------------------------------
76      always @(negedge VGA_HSYNC or negedge VGA_RST_N) begin
77          if (VGA_RST_N == 1'b0) begin
78              r_Vcount        <= 10'd0;
79              r_VGA_VSYNC     <= 1'b1;
80          end else begin
81
82              if (r_Vcount < VGA_V_time1) begin
83                  r_Vcount        <= r_Vcount + 10'd1;
84                  r_VGA_VSYNC     <= 1'b0;
85
86              end else if (r_Vcount < VGA_V_time2) begin
87                  r_Vcount        <= r_Vcount + 10'd1;
88                  r_VGA_VSYNC     <= 1'b1;
```

```
89
90              end else if (r_Vcount < VGA_V_time3) begin
91                  r_Vcount        <= r_Vcount + 10'd1;
92                  r_VGA_VSYNC     <= 1'b1;
93
94              end else if (r_Vcount < VGA_V_time4) begin
95                  if (r_Vcount == VGA_V_time4 - 10'd1) begin
96                      r_Vcount        <= 10'd0;
97                  end else begin
98                      r_Vcount        <= r_Vcount + 10'd1;
99                  end
100
101                 r_VGA_VSYNC     <= 1'b1;
102
103             end else begin
104                 r_Vcount        <= 10'd0;
105                 r_VGA_VSYNC     <= 1'b1;
106             end
107         end
108     end
109

110     //---------------------------------------------------------------------
111     // H Control
112     //---------------------------------------------------------------------
113     always @(posedge VGA_CLK or negedge VGA_RST_N) begin
114         if (VGA_RST_N == 1'b0) begin
115             r_Hcount        <= 0;
116
117             r_VGA_HSYNC     <= 1'b1;
118             r_VGA_DE        <= 1'b0;
119             r_VGA_IF_RGBEN  <= 1'b0;
120
121             r_VGA_R         <= 8'h0;
122             r_VGA_G         <= 8'h0;
123             r_VGA_B         <= 8'h0;
124         end else begin
125
126             if (r_Hcount < VGA_H_time1) begin
127                 r_Hcount        <= r_Hcount + 11'd1;
128
129                 r_VGA_HSYNC     <= 1'b0;
130                 r_VGA_DE        <= 1'b0;
131                 r_VGA_IF_RGBEN  <= 1'b0;
132
133                 r_VGA_R         <= 8'h0;
134                 r_VGA_G         <= 8'h0;
135                 r_VGA_B         <= 8'h0;
136
137             end else if (r_Hcount < VGA_H_time2) begin
138                 r_Hcount        <= r_Hcount + 11'd1;
139
140                 r_VGA_HSYNC     <= 1'b1;
141                 r_VGA_DE        <= 1'b0;
142
143                 if (w_if_acitve) begin
144                     if (r_Hcount < VGA_H_time2 - 11'd3) begin
145                         r_VGA_IF_RGBEN  <= 1'b0;
146                     end else begin
147                         r_VGA_IF_RGBEN  <= 1'b1;
148                     end
149                 end else begin
150                     r_VGA_IF_RGBEN  <= 1'b0;
151                 end
152
153                 r_VGA_R         <= 8'h0;
```

```
154                 r_VGA_G         <= 8'h0;
155                 r_VGA_B         <= 8'h0;
156
157             end else if (r_Hcount < VGA_H_time3) begin
158                 r_Hcount        <= r_Hcount + 11'd1;
159
160                 r_VGA_HSYNC     <= 1'b1;
161
162                 if (w_if_acitve) begin
163                     r_VGA_DE        <= 1'b1;
164                     if (r_Hcount < VGA_H_time3 - 11'd3) begin
165                         r_VGA_IF_RGBEN  <= 1'b1;
166                     end else begin
167                         r_VGA_IF_RGBEN  <= 1'b0;
168                     end
169
170                     r_VGA_R         <= VGA_BUF_RGB[23:16];
171                     r_VGA_G         <= VGA_BUF_RGB[15: 8];
172                     r_VGA_B         <= VGA_BUF_RGB[ 7: 0];
173                 end else begin
174                     r_VGA_DE        <= 1'b0;
175                     r_VGA_IF_RGBEN  <= 1'b0;
176
177                     r_VGA_R         <= 8'h0;
178                     r_VGA_G         <= 8'h0;
179                     r_VGA_B         <= 8'h0;
180                 end
181
182
183             end else if (r_Hcount < VGA_H_time4) begin
184                 if (r_Hcount == VGA_H_time4 - 11'd1) begin
185                     r_Hcount        <= 11'd0;
186                 end else begin
187                     r_Hcount        <= r_Hcount + 11'd1;
188                 end
189
190                 r_VGA_HSYNC     <= 1'b1;
191                 r_VGA_DE        <= 1'b0;
192                 r_VGA_IF_RGBEN  <= 1'b0;
193
194                 r_VGA_R         <= 8'h0;
195                 r_VGA_G         <= 8'h0;
196                 r_VGA_B         <= 8'h0;
197
198
199             end else begin
200                 r_Hcount        <= 0;
201
202                 r_VGA_HSYNC     <= 1'b1;
203                 r_VGA_DE        <= 1'b0;
204                 r_VGA_IF_RGBEN  <= 1'b0;
205
206                 r_VGA_R         <= 8'h0;
207                 r_VGA_G         <= 8'h0;
208                 r_VGA_B         <= 8'h0;
209             end
210         end
211     end
212
213 endmodule
```

(7) VGA 显示模块（给 VGA 时序提供显示数据，由于篇幅过长且主要为模块实例化代码，因此仅给出计算当前显示 XY 坐标的代码）

```
    always @(posedge VGA_CLK or negedge RST_N) begin
        if (!RST_N) begin
            r_VGA_IF_RGBEN_1    <= 1'b0;
        end else begin
            if (VGA_IF_RGBEN) begin
                r_VGA_IF_RGBEN_1    <= 1'b1;
            end else begin
                r_VGA_IF_RGBEN_1    <= 1'b0;
            end
        end
    end


    always @(posedge VGA_CLK or negedge RST_N) begin
        if (!RST_N) begin
            r_X     <= 11'd0;
            r_Y     <= 11'd0;
        end else begin
            if (w_VGA_IF_RGBEN_1) begin
                if (r_X < (p_DISPLAY_X - 11'd1)) begin
                    r_X     <= r_X + 11'd1;
                end else begin
                    r_X     <= 11'd0;
                    if (r_Y < (p_DISPLAY_Y - 11'd1)) begin
                        r_Y     <= r_Y + 11'd1;
                    end else begin
                        r_Y     <= 11'd0;
                    end
                end
            end else begin
                r_X     <= r_X;
                r_Y     <= r_Y;
            end
        end
    end
```

(8) 背景显示模块（实现循环彩虹渐变）

```
module VGA_DISPLAY_BACK (
    input                       VGA_CLK                 , // (i) vga clk in
    input                       RST_N                   , // (i) reset, High Active
    input                       VGA_IF_RGBEN_1          , // (i)
    input           [10:0]      DISPLAY_X               , // (i)
    input           [10:0]      DISPLAY_Y               , // (i)
    input           [10:0]      CURRENT_X               , // (i)
    input           [10:0]      CURRENT_Y               , // (i)
    output          [23:0]      VGA_BUF_RGB               // (o)
);

// ============================================================================
// Defination of parameter
// ============================================================================
    parameter                   Color_Cnt_Num           = 24'd2_000_000 ;   // = 24'd6  ;   //

    // parameter                    RGB_Cnt_Num1            = 4'd0          ;
    parameter                   RGB_Cnt_Num2            = 4'd0          ;

// ============================================================================
// Defination of Internal Signals
// ============================================================================
    reg             [23:0]      r_VGA_BUF_RGB           ;

    reg             [23:0]      r_color_cnt             ;
    reg             [23:0]      r_rgb                   ;
```

```
28      reg             [23:0]      r_rgb1                  ;
29      // reg              [03:0]      r_rgbcnt1               ;
30
31      reg             [23:0]      r_rgb2                  ;
32      reg             [03:0]      r_rgbcnt2               ;
33
34  // ============================================================================
35  // RTL Body
36  // ============================================================================
37      assign                      VGA_BUF_RGB             = r_VGA_BUF_RGB;
38
39      always @(posedge VGA_CLK or negedge RST_N) begin
40          if (RST_N == 1'b0) begin
41              r_VGA_BUF_RGB   <= 24'hff_ff_ff;
42              r_rgb1          <= 24'hff_00_00;
43              r_rgb2          <= 24'hff_00_00;
44              // r_rgbcnt1        <= 4'd0;
45              r_rgbcnt2       <= 4'd0;
46          end else begin
47              if (VGA_IF_RGBEN_1) begin
48                  r_VGA_BUF_RGB   <= r_rgb2;
49
50                  if (CURRENT_X == (DISPLAY_X - 11'd1)) begin
51                      if (CURRENT_Y == (DISPLAY_Y - 11'd1)) begin
52                          r_rgb1      <= r_rgb;
53                          r_rgb2      <= r_rgb;
54                      end else begin
55                          r_rgb1      <= r_rgb1;
56                          r_rgb2      <= r_rgb1;
57                      end
58
59                  end else begin
60                      r_rgb1      <= r_rgb1;
61
62                      r_rgbcnt2   <= r_rgbcnt2 + 4'd1;
63                      if (r_rgbcnt2 == RGB_Cnt_Num2) begin
64                          r_rgbcnt2   <= 4'd0;
65
66                          if ((r_rgb2[23:16] == 8'hff) && (r_rgb2[15:8] < 8'hff) && (r_rgb2[7:0]
 == 8'h00)) begin
67                              r_rgb2[15:8]    <= r_rgb2[15:8] + 8'h1;
68
69                          end else if ((r_rgb2[23:16] > 8'h00) && (r_rgb2[15:8] == 8'hff) && (r_
rgb2[7:0] == 8'h00)) begin
70                              r_rgb2[23:16]   <= r_rgb2[23:16] - 8'h1;
71
72                          end else if ((r_rgb2[23:16] == 8'h00) && (r_rgb2[15:8] == 8'hff) && (r
_rgb2[7:0] < 8'hff)) begin
73                              r_rgb2[7:0]     <= r_rgb2[7:0] + 8'h1;
74
75                          end else if ((r_rgb2[23:16] == 8'h00) && (r_rgb2[15:8] > 8'h00) && (r_
rgb2[7:0] == 8'hff)) begin
76                              r_rgb2[15:8]    <= r_rgb2[15:8] - 8'h1;
77
78                          end else if ((r_rgb2[23:16] < 8'hff) && (r_rgb2[15:8] == 8'h00) && (r_
rgb2[7:0] == 8'hff)) begin
79                              r_rgb2[23:16]   <= r_rgb2[23:16] + 8'h1;
80
81                          end else if ((r_rgb2[23:16] == 8'hff) && (r_rgb2[15:8] == 8'h00) && (r
_rgb2[7:0] > 8'h00)) begin
82                              r_rgb2[7:0]     <= r_rgb2[7:0] - 8'h1;
83                          end
84                      end
85                  end
```

```
86
87              end else begin
88                  r_VGA_BUF_RGB    <= 24'h00_00_00;
89              end
90          end
91      end
92
93      always @(posedge VGA_CLK or negedge RST_N) begin
94          if (RST_N == 1'b0) begin
95              r_color_cnt <= 24'd0;
96              r_rgb       <= 24'hff_00_00;
97          end else begin
98              if (VGA_IF_RGBEN_1) begin
99                  if (r_color_cnt < Color_Cnt_Num) begin
100                     r_color_cnt <= r_color_cnt + 24'd1;
101                 end else begin
102                     r_color_cnt <= 24'd0;
103
104                     if ((r_rgb[23:16] == 8'hff) && (r_rgb[15:8] < 8'hff) && (r_rgb[7:0] == 8'h
    00)) begin
105                         r_rgb[15:8]     <= r_rgb[15:8] + 8'h1;
106
107                     end else if ((r_rgb[23:16] > 8'h00) && (r_rgb[15:8] == 8'hff) && (r_rgb[7:
    0] == 8'h00)) begin
108                         r_rgb[23:16]    <= r_rgb[23:16] - 8'h1;
109
110                     end else if ((r_rgb[23:16] == 8'h00) && (r_rgb[15:8] == 8'hff) && (r_rgb[7
    :0] < 8'hff)) begin
111                         r_rgb[7:0]      <= r_rgb[7:0] + 8'h1;
112
113                     end else if ((r_rgb[23:16] == 8'h00) && (r_rgb[15:8] > 8'h00) && (r_rgb[7:
    0] == 8'hff)) begin
114                         r_rgb[15:8]     <= r_rgb[15:8] - 8'h1;
115
116                     end else if ((r_rgb[23:16] < 8'hff) && (r_rgb[15:8] == 8'h00) && (r_rgb[7:
    0] == 8'hff)) begin
117                         r_rgb[23:16]    <= r_rgb[23:16] + 8'h1;
118
119                     end else if ((r_rgb[23:16] == 8'hff) && (r_rgb[15:8] == 8'h00) && (r_rgb[7
    :0] > 8'h00)) begin
120                         r_rgb[7:0]      <= r_rgb[7:0] - 8'h1;
121
122                     end
123                 end
124             end
125         end
126     end
127
128 endmodule
```

(9) 1 位数字显示模块（实现指定位置指定数字的显示）

```
1   module VGA_DISPLAY_NUM (
2       input                       VGA_CLK             , // (i) vga clk in
3       input                       RST_N               , // (i) reset, High Active
4       input                       ENABLE              , // (i)
5       input                       VGA_IF_RGBEN_1      , // (i)
6       input           [10:0]      POSITION_X          , // (i)
7       input           [10:0]      POSITION_Y          , // (i)
8       input           [10:0]      CURRENT_X           , // (i)
9       input           [10:0]      CURRENT_Y           , // (i)
10      input           [ 3:0]      NUMBER              , // (i)
11      output                      COVER               , // (o)
12      output          [23:0]      VGA_BUF_RGB           // (o)
```

```
13  );
14
15  // ============================================================================
16  // Defination of parameter
17  // ============================================================================
18      parameter       [10:0]      p_NUM_X                 = 11'd16        ;
19      parameter       [10:0]      p_NUM_Y                 = 11'd32        ;
20
```

……

由于篇幅所限，此处略去 0~9 数字的编码常数

……

```
21  // ============================================================================
22  // Defination of Internal Signals
23  // ============================================================================
24      reg                         r_COVER                 = 1'b0              ;
25      reg             [23:0]      r_VGA_BUF_RGB           = 24'h00_00_00      ;
26
27      wire                        w_IN_POSTION            = ( (POSITION_Y <= CURRENT_Y)
28                                                          && (CURRENT_Y < POSITION_Y + p_NUM_Y)
29                                                          && (POSITION_X <= CURRENT_X)
30                                                          && (CURRENT_X < POSITION_X + p_NUM_X)
    ) ? 1'b1 : 1'b0;
31
32  // ============================================================================
33  // RTL Body
34  // ============================================================================
35      assign                      COVER                   = r_COVER           ;
36      assign                      VGA_BUF_RGB             = r_VGA_BUF_RGB     ;
37
38
39      always @(posedge VGA_CLK or negedge RST_N) begin
40          if (RST_N == 1'b0) begin
41              r_COVER         <= 1'b0;
42              r_VGA_BUF_RGB   <= 24'h00_00_00;
43          end else begin
44              if ((ENABLE) && (w_IN_POSTION == 1'b1)) begin
45                  case (CURRENT_Y)
46                      (POSITION_Y + 11'd00):  r_COVER <= p_NUM_00[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
47                      (POSITION_Y + 11'd01):  r_COVER <= p_NUM_01[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
48                      (POSITION_Y + 11'd02):  r_COVER <= p_NUM_02[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
49                      (POSITION_Y + 11'd03):  r_COVER <= p_NUM_03[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
50                      (POSITION_Y + 11'd04):  r_COVER <= p_NUM_04[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
51                      (POSITION_Y + 11'd05):  r_COVER <= p_NUM_05[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
52                      (POSITION_Y + 11'd06):  r_COVER <= p_NUM_06[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
53                      (POSITION_Y + 11'd07):  r_COVER <= p_NUM_07[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
54                      (POSITION_Y + 11'd08):  r_COVER <= p_NUM_08[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
55                      (POSITION_Y + 11'd09):  r_COVER <= p_NUM_09[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
56                      (POSITION_Y + 11'd10):  r_COVER <= p_NUM_10[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
57                      (POSITION_Y + 11'd11):  r_COVER <= p_NUM_11[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
58                      (POSITION_Y + 11'd12):  r_COVER <= p_NUM_12[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
```

```
59                      (POSITION_Y + 11'd13):  r_COVER <= p_NUM_13[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
60                      (POSITION_Y + 11'd14):  r_COVER <= p_NUM_14[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
61                      (POSITION_Y + 11'd15):  r_COVER <= p_NUM_15[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
62                      (POSITION_Y + 11'd16):  r_COVER <= p_NUM_16[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
63                      (POSITION_Y + 11'd17):  r_COVER <= p_NUM_17[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
64                      (POSITION_Y + 11'd18):  r_COVER <= p_NUM_18[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
65                      (POSITION_Y + 11'd19):  r_COVER <= p_NUM_19[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
66                      (POSITION_Y + 11'd20):  r_COVER <= p_NUM_20[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
67                      (POSITION_Y + 11'd21):  r_COVER <= p_NUM_21[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
68                      (POSITION_Y + 11'd22):  r_COVER <= p_NUM_22[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
69                      (POSITION_Y + 11'd23):  r_COVER <= p_NUM_23[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
70                      (POSITION_Y + 11'd24):  r_COVER <= p_NUM_24[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
71                      (POSITION_Y + 11'd25):  r_COVER <= p_NUM_25[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
72                      (POSITION_Y + 11'd26):  r_COVER <= p_NUM_26[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
73                      (POSITION_Y + 11'd27):  r_COVER <= p_NUM_27[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
74                      (POSITION_Y + 11'd28):  r_COVER <= p_NUM_28[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
75                      (POSITION_Y + 11'd29):  r_COVER <= p_NUM_29[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
76                      (POSITION_Y + 11'd30):  r_COVER <= p_NUM_30[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
77                      (POSITION_Y + 11'd31):  r_COVER <= p_NUM_31[CURRENT_X - POSITION_X + (NUMB
    ER << 4)];
78                      default:                r_COVER <= 1'b0;
79                  endcase
80                  r_VGA_BUF_RGB   <= 24'hff_ff_ff;
81              end else begin
82                  r_COVER         <= 1'b0;
83                  r_VGA_BUF_RGB   <= 24'h00_00_00;
84              end
85          end
86      end
87
88  endmodule
```

(10)　其他显示模块（包含符号、文字等）

（由于原理与数字显示模块均相同，此处略去）

(11)　逻辑模块（DigitalClock_Logic）

（沿用实验 5 模块，此处略去）

（五）实验效果

最终完整实现功能。

(1) 时钟


时钟 闹钟
13:13:22
SONGREN
MENU DOWN UP EXIT

(2) 秒表


时钟 闹钟
00:21:83
SONGREN
MENU DOWN UP EXIT