

电子技术基础实验第六周实验报告

王磊 2022012972

2023 年 11 月 10 日

1 task1_3

1.1 模块设计

1.1.1 上升沿检测模块

1.1.1.1 模块代码

```
1  reg pulse1_1, pulse1_2, pulse1_3;
2  always @(posedge clk or posedge reset) begin
3      if(reset)
4          begin
5              pulse1_1 <= 1'b0;
6              pulse1_2 <= 1'b0;
7              pulse1_3 <= 1'b0;
8          end
9      else
10         begin
11             pulse1_1 <= button_io1;
12             pulse1_2 <= pulse1_1;
13             pulse1_3 <= pulse1_2;
14         end
15     end
16     wire button1_negedge = ~pulse1_2 & pulse1_3;
17     wire button1_posedge = pulse1_2 & ~pulse1_3;
```

本模块采取基本的状态机设计，用三个寄存器存储三个时刻的按键状态，通过异或门检测下降沿和上升沿。

三个按键独立设计，故有三个检测模块，实际上可以通过一个模块解决，可以优化代码臃肿度。

1.1.1.2 模块仿真

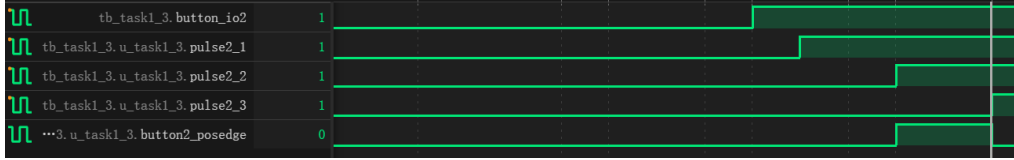


图 1: 下降沿检测模块仿真

1.1.2 按键检测模块

1.1.2.1 模块代码

```

1  reg [31:0] cnt;
2  always @ (posedge clk or posedge reset)begin
3      if(reset) begin
4          cnt <= 32'b0;
5      end
6      else if(delay_flag) begin
7          if(cnt == 'CNT_MAX-1)
8              cnt <= 32'b0;
9          else begin
10             cnt <= cnt + 32'b1;
11         end
12     end
13 end

14
15
16 reg delay_flag;
17 always @(posedge clk or posedge reset) begin
18     if(reset) begin
19         delay_flag <= 1'b0;
20     end
21     else if(button1_posedge || button2_posedge || button3_posedge) begin
22         delay_flag <= 1'b1;
23     end
24     else if(cnt == 'CNT_MAX-1) begin

```

```

25         delay_flag <= 1'b0;
26     end
27 end

```

逻辑为当检测到按键上升沿时，置延时标志位，同时开始计数，当计数到达设定值时，清除标志位，延时结束。延时的作用是消除机械按键的抖动，保证按键检测稳定。消抖后即可读取按键状态，进行相应操作。

```

1  reg button1_state, button2_state, button3_state;
2  always @(posedge clk or posedge reset) begin
3      if(reset) begin
4          button1_state <= 1'b0;
5          button2_state <= 1'b0;
6          button3_state <= 1'b0;
7      end
8      else if(cnt == 'CNT_MAX-1) begin
9          button1_state <= button_io1;
10         button2_state <= button_io2;
11         button3_state <= button_io3;
12     end
13     else begin
14         button1_state <= 1'b0;
15         button2_state <= 1'b0;
16         button3_state <= 1'b0;
17     end
18 end

```

1.1.2.2 模块仿真



图 2: 按键消抖模块仿真

可以看出检测到按键上升沿后经计数器延时到按键状态稳定时，才读取按键状态。

1.1.3 流水灯模块

1.1.3.1 模块代码

首先通过按键状态确定流水灯状态：

```
1  reg led_state1_flag, led_state2_flag, led_state3_flag;
2  always @(posedge clk or posedge reset) begin
3      if(reset) begin
4          led_state1_flag <= 1'b0;
5          led_state2_flag <= 1'b0;
6          led_state3_flag <= 1'b0;
7      end
8      else if(button1_state) begin
9          led_state1_flag <= 1'b1;
10         led_state2_flag <= 1'b0;
11         led_state3_flag <= 1'b0;
12     end
13     else if(button2_state) begin
14         led_state1_flag <= 1'b0;
15         led_state2_flag <= 1'b1;
16         led_state3_flag <= 1'b0;
17     end
18     else if(stop_flag) begin
19         led_state1_flag <= 1'b0;
20         led_state2_flag <= 1'b0;
21         led_state3_flag <= 1'b0;
22     end
23     else if(button3_state) begin
24         led_state1_flag <= 1'b0;
25         led_state2_flag <= 1'b0;
26         led_state3_flag <= 1'b1;
27     end
28 end
```

之后根据流水灯状态控制LED灯：

```
1  reg led_cnt;
2  reg stop_flag;
3  always @(posedge clk_div or posedge reset) begin
```

```
4    if(reset) begin
5        led_io <= 8'b0;
6        led_cnt <= 8'b0;
7        stop_flag <= 1'b0;
8    end
9    else if(led_state1_flag) begin
10        if(led_io == 8'b10000000 || led_io == 8'b00000000) begin
11            led_io <= 8'b00000001;
12        end
13        else begin
14            led_io <= led_io << 1;
15        end
16    end
17
18    else if(led_state2_flag) begin
19        if(led_io == 8'b00000001 || led_io == 8'b00000000) begin
20            led_io <= 8'b10000000;
21        end
22        else begin
23            led_io <= led_io >> 1;
24        end
25    end
26    else if(led_state3_flag) begin
27        if(led_io != 8'b0) begin
28            led_io <= 8'b0;
29        end
30        else begin
31            led_io <= 8'b11111111;
32            led_cnt <= led_cnt + 1;
33        end
34    end
35
36    if(led_cnt == 8'd2 ) begin
37        stop_flag <= 1'b1;
38    end
39    else if(led_cnt == 8'd3) begin
40        led_cnt <= 0;
41    end
42    else begin
43        stop_flag <= 1'b0;
```

```
44     end
45 end
```

其中控制LED闪烁停止的模块存在一些问题，需要后续修复。

1.2 仿真设计

代码如下，结果已经给出。

```
1  //~ 'New testbench
2  `timescale 1ns / 1ps
3  `include "task1_3.v"
4  module tb_task1_3;
5
6      // task1_3 Parameters
7      parameter PERIOD = 10;
8
9      // task1_3 Inputs
10     reg clk = 0;
11     reg reset = 0;
12     reg button_io1 = 0;
13     reg button_io2 = 0;
14     reg button_io3 = 0;
15
16
17     // task1_3 Outputs
18     wire clk_div;
19     wire [7:0] led_io;
20
21     task1_3 u_task1_3 (
22         .clk          (clk),
23         .reset         (reset),
24         .button_io1    (button_io1),
25         .button_io2    (button_io2),
26         .button_io3    (button_io3),
27         .clk_div       (clk_div),
28         .led_io        (led_io[7:0])
29     );
```

```
30
31 initial begin
32     $dumpfile("./wave/tb_task1_3.vcd");
33     $dumpvars(0, tb_task1_3);
34     clk = 0;
35     reset = 1;
36     #10
37     reset = 0;
38     #10000
39     $finish;
40 end
41
42 always begin
43     #0.1 clk = ~clk;
44 end
45
46 always begin
47     button_io1 = 1;
48     #10 button_io1 = 0;
49     #1 button_io1 = 0;
50     button_io2 = 1;
51     #10 button_io2 = 0;
52     #1 button_io2 = 0;
53     button_io3 = 1;
54     #10 button_io3 = 0;
55     #1 button_io3 = 0;
56 end
57 endmodule
```

2 task2_2

2.1 模块设计

按键消抖与检测与task1_3相同，不做赘述。

2.1.1 数码管选择模块

2.1.1.1 模块代码

```
1  reg [1:0] state;
2  always @(posedge clk_div or posedge reset) begin
3      if(reset) begin
4          data_temp = 4'b0000;
5          digit = 4'b1111;
6          state = 2'b00;
7      end
8      else begin
9          case (state)
10             2'b00: begin
11                 digit = 4'b1110;
12                 data_temp = data[15:12];
13                 state = 2'b01;
14             end
15             2'b01: begin
16                 digit = 4'b1101;
17                 data_temp = data[11:8];
18                 state = 2'b10;
19             end
20             2'b10: begin
21                 digit = 4'b1011;
22                 data_temp = data[7:4];
23                 state = 2'b11;
24             end
25             2'b11: begin
26                 digit = 4'b0111;
27                 data_temp = data[3:0];
28                 state = 2'b00;
29             end
30             default: begin
31                 digit = 4'b1111;
32                 data_temp = 4'b0000;
33                 state = 2'b00;
34             end
35         endcase

```



```
36     end
37 end
```

通过状态机控制数码管的选择，并为每个数码管分配数据。

2.1.2 数码管显示模块

2.1.2.1 模块代码

```
1  always @(posedge clk, posedge reset) begin
2      if(reset) begin
3          segment = 8'h00;
4      end
5      else begin
6          case (data_temp)
7              0: segment = 8'hc0;
8              1: segment = 8'hf9;
9              2: segment = 8'ha4;
10             3: segment = 8'hb0;
11             4: segment = 8'h99;
12             5: segment = 8'h92;
13             6: segment = 8'h82;
14             7: segment = 8'hf8;
15             8: segment = 8'h80;
16             9: segment = 8'h90;
17             10: segment = 8'h88;
18             11: segment = 8'h83;
19             12: segment = 8'hc6;
20             13: segment = 8'ha1;
21             14: segment = 8'h86;
22             15: segment = 8'h8e;
23             default: segment = 8'h00;
24         endcase
25     end
26 end
```

根据数码管选择模块分配的数据显示当前被选中的数码管的数值。

2.2 仿真设计

代码如下，结果与task1_3相同，不再给出。

```
1  //~ 'New testbench
2  'timescale 1ns/1ps
3  'include "task2_2.v"
4  module tb_task2_2;
5
6      // task2_2 Parameters
7      parameter PERIOD = 10;
8
9      // task2_2 Inputs
10     reg clk = 0;
11     reg reset = 0;
12     reg button_io = 0;
13
14     // task2_2 Outputs
15     wire [3:0] digit;
16     wire [7:0] segment;
17
18     task2_2 u_task2_2 (
19         .clk (clk),
20         .reset (reset),
21         .button_io (button_io),
22         .digit (digit[3:0]),
23         .segment (segment[7:0])
24     );
25
26     initial begin
27         $dumpfile("./wave//tb_task2_2.vcd");
28         $dumpvars(0, tb_task2_2);
29
30         // Initialize Inputs
31         clk = 0;
32         reset = 0;
33         button_io = 0;
34         reset = 1;
35         #10
36         reset = 0;
```

```

37         #10000;
38         $finish;
39     end
40     always begin
41         #0.1 clk = ~clk;
42     end
43     always begin
44         #1 button_io = 1;
45         #10 button_io = 0;
46     end
47
48 endmodule

```

3 task2_3

3.1 模块设计

数码管选择模块与显示模块与task2.2相同，蜂鸣器模块与LED控制类似，不再给出。

3.1.1 计时器模块

3.1.1.1 模块代码

```

1  reg [31:0] div_reg;
2  always @ (posedge clk or posedge reset) begin
3      if(reset)
4          begin
5              div_reg <= 32'b0;
6              clk_div <= 1'b0;
7          end
8
9      else
10         begin
11             if(div_reg < 32'd12500)
12                 div_reg <= div_reg + 32'b1;
13             else

```

```

14         begin
15             div_reg <= 32'b0;
16             clk_div <= ~clk_div;
17         end
18     end
19 end

```

3.1.2 数据处理模块

3.1.2.1 模块代码

```

1  reg buzz_flag;
2  reg buzz_cnt;
3  reg stop_flag;
4  always @(posedge clk, posedge reset) begin
5      if(reset) begin
6          data[3:0] = 4'd5;
7          data[7:4] = 4'd1;
8          data[11:8] = 4'd0;
9          data[15:12] = 4'd0;
10         real_time = 12'd15;//15s
11         buzz_flag = 1'b0;
12     end
13     else if(cnt == 'CNT_MAX-1) begin
14         if(real_time == 12'd0) begin
15             buzz_flag = 1'b1;
16             if(stop_flag == 1'b1) begin
17                 buzz_flag = 1'b0;
18             end
19         end
20         else begin
21             real_time = real_time - 12'd1;
22             data[3:0] = ((real_time)%60)%10;
23             data[7:4] = ((real_time)%60)/10;
24             data[11:8] = ((real_time)/60)%10;
25             data[15:12] = ((real_time)/60)/10;
26         end

```

```

27     end
28 end

```

其中第22-25行为核心逻辑，将以秒数表示的时间换算为分钟的十位、个位，秒钟的个位、十位四个供数码管显示的数据。

3.1.2.2 模块仿真



图 3: 数据处理模块仿真

第一个信号比后面长的原因是正在reset。

3.2 仿真设计

代码如下，结果已经给出。

```

1  //~ 'New testbench
2  `timescale 1ns/1ps
3  `include "task2_3.v"
4  module tb_task2_3;
5
6      // task2_3 Parameters
7      parameter PERIOD = 10;
8
9      // task2_3 Inputs
10     reg clk = 0;
11     reg reset = 0;
12
13     // task2_3 Outputs
14     wire clk_div;
15     wire buzz;
16     wire [3:0] digit;
17     wire [7:0] segment;
18
19     task2_3 u_task2_3 (

```

```
20     .clk(clk),
21     .reset(reset),
22     .clk_div(clk_div),
23     .buzz(buzz),
24     .digit(digit[3:0]),
25     .segment(segment[7:0])
26 );
27
28 initial begin
29     $dumpfile("./wave/tb_task2_3.vcd");
30     $dumpvars(0, tb_task2_3);
31     clk = 0;
32     reset = 1;
33     #10
34     reset = 0;
35     #10000
36     $finish;
37 end
38
39 always begin
40     #0.1 clk = ~clk;
41 end
42
43 endmodule
```

4 task2_4

4.1 模块设计

task2_4相较于task2_3只多一个计时控制模块，下面给出，其余模块相同，不再给出。

4.1.1 计时控制模块

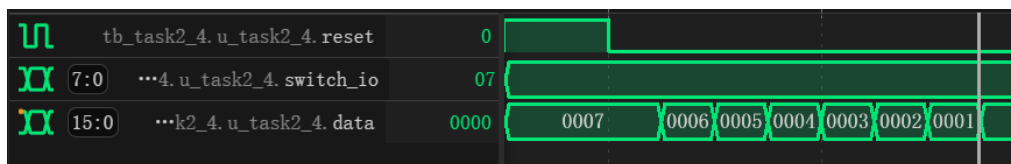
4.1.1.1 模块代码

```
1 always @(posedge clk, posedge reset) begin
```

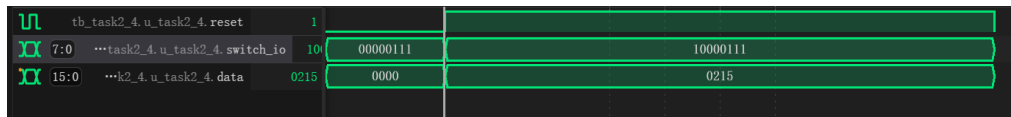
```
2      if(reset) begin
3          real_time = switch_io;
4          data[3:0] = ((real_time)%60)%10;
5          data[7:4] = ((real_time)%60)/10;
6          data[11:8] = ((real_time)/60)%10;
7          data[15:12] = ((real_time)/60)/10;
8          buzz_flag <= 1'b0;
9      end
10     else if(cnt == 'CNT_MAX-1) begin
11         if(real_time == 8'd0) begin
12             buzz_flag = 1'b1;
13             if(stop_flag == 1'b1) begin
14                 buzz_flag = 1'b0;
15             end
16         end
17     else begin
18         real_time = real_time - 8'd1;
19         data[3:0] = ((real_time)%60)%10;
20         data[7:4] = ((real_time)%60)/10;
21         data[11:8] = ((real_time)/60)%10;
22         data[15:12] = ((real_time)/60)/10;
23     end
24 end
25 end
```

第三行为该模块关键，上个模块`real_time`为固定值，此时通过拨码开关控制。

4.1.1.2 模块仿真



(a) image 1



(b) image 2

图 4: 计时控制模块仿真

4.2 仿真设计

代码如下，结果已经给出。

```

1  //~ 'New testbench
2  `timescale 1ns/1ps
3  `include "task2_4.v"
4
5  module tb_task2_4;
6
7      // task2_4 Parameters
8      parameter PERIOD = 10;
9
10     // task2_4 Inputs
11     reg clk = 0;
12     reg reset = 0;
13     reg [7:0] switch_io = 0;
14
15     // task2_4 Outputs
16     wire clk_div;
17     wire buzz;
18     wire [3:0] digit;
19     wire [7:0] segment;
20
21     task2_4 u_task2_4 (
22         .clk(clk),
23         .reset(reset),
24         .switch_io(switch_io[7:0]),

```



```
25     .clk_div(clk_div),
26     .buzz(buzz),
27     .digit(digit[3:0]),
28     .segment(segment[7:0])
29 );
30
31 initial begin
32     $dumpfile("./wave/tb_task2_4.vcd");
33     $dumpvars(0, tb_task2_4);
34     clk = 0;
35     reset = 1;
36     switch_io = 8'b00000111;
37     #10
38     reset = 0;
39     #10000
40     switch_io = 8'b10000111;
41     reset = 1;
42     #10
43     reset = 0;
44     $finish;
45 end
46
47 always begin
48     #0.1 clk = ~clk;
49 end
50
51 endmodule
```