

电子技术基础实验第八周实验报告

王磊 2022012972

2023 年 11 月 19 日

1 task1_2&task1_3

由于task1_2是task1_3的子模块，因此不单独介绍task1_2。task1_2要求的移位显示可以通过下类代码实现；

```
1 first_num <= {first_num[7:0],input_temp[3:0]};
```

1.1 模块设计

数码管显示模块、分频模块与week6一致，不再赘述。

1.1.1 矩阵按键模块

1.1.1.1 代码实现

```
1 always @(posedge clk or posedge rst) begin
2     if(rst) begin
3         state <= 2'b00;
4         hl <= 4'b1111;
5     end
6     else begin
7         case (state)
8             2'b00: begin
9                 hl <= 4'b1110;
10                state <= 2'b01;
11            end
12        end
13    end
```

```

12      2'b01: begin
13          hl <= 4'b1101;
14          state <= 2'b10;
15      end
16      2'b10: begin
17          hl <= 4'b1011;
18          state <= 2'b11;
19      end
20      2'b11: begin
21          hl <= 4'b0111;
22          state <= 2'b00;
23      end
24      default: begin
25          hl <= 4'b1111;
26          state <= 2'b00;
27      end
28  endcase
29  end
30  end
31  assign hl_v1 = {hl,v1};

```

在这个模块中，通过控制hl的输出使能矩阵键盘的不同列，实现了矩阵键盘的扫描，并将结果输出到hl_v1中。

1.1.1.2 仿真结果



图 1: 矩阵键盘仿真结果

1.1.2 按键处理模块

1.1.2.1 代码实现

```

1  //process the origin input to a simple one

```

```
2  reg [2:0] input_type;
3  parameter NUMBER = 3'b001;
4  parameter OPERATOR = 3'b010;
5  parameter EQUAL = 3'b100;
6
7  reg [4:0] input_temp;
8
9  always @ (posedge clk,posedge rst) begin
10     if (rst) begin
11         input_type <= 3'b000;
12         input_temp <= 5'b00000;
13     end
14     else begin
15         case (hl_vl)
16             8'b0111_0111: begin
17                 //for =
18                 input_type <= EQUAL;
19                 input_temp <= 5'b00000;
20             end
21             8'b1011_0111: begin
22                 //for and
23                 input_type <= OPERATOR;
24                 input_temp <= AND_OPERATOR;
25             end
26             8'b1101_0111: begin
27                 //for 0
28                 input_type <= NUMBER;
29                 input_temp <= 5'b00000;
30             end
31             8'b1110_0111: begin
32                 //for or
33                 input_type <= OPERATOR;
34                 input_temp <= OR_OPERATOR;
35             end
36             8'b0111_1011: begin
37                 //for +
38                 input_type <= OPERATOR;
39                 input_temp <= ADD_OPERATOR;
40             end
41             8'b1011_1011: begin
```

```
42         //for 3
43         input_type <= NUMBER;
44         input_temp <= 5'b00011;
45     end
46 8'b1101_1011: begin
47     //for 2
48     input_type <= NUMBER;
49     input_temp <= 5'b00010;
50 end
51 8'b1110_1011: begin
52     //for 1
53     input_type <= NUMBER;
54     input_temp <= 5'b00001;
55 end
56 8'b0111_1101: begin
57     //for -
58     input_type <= OPERATOR;
59     input_temp <= SUB_OPERATOR;
60 end
61 8'b1011_1101: begin
62     //for 6
63     input_type <= NUMBER;
64     input_temp <= 5'b00110;
65 end
66 8'b1101_1101: begin
67     //for 5
68     input_type <= NUMBER;
69     input_temp <= 5'b00101;
70 end
71 8'b1110_1101: begin
72     //for 4
73     input_type <= NUMBER;
74     input_temp <= 5'b00100;
75 end
76 8'b0111_1110: begin
77     //for compare
78     input_type <= OPERATOR;
79     input_temp <= COMPARE_OPERATOR;
80 end
81 8'b1011_1110: begin
```

```

82         //for 9
83         input_type <= NUMBER;
84         input_temp <= 5'b01001;
85     end
86     8'b1101_1110: begin
87         //for 8
88         input_type <= NUMBER;
89         input_temp <= 5'b01000;
90     end
91     8'b1110_1110: begin
92         //for 7
93         input_type <= NUMBER;
94         input_temp <= 5'b00111;
95     end
96     default: begin
97         input_type <= 3'b000;
98         input_temp <= 5'b00000;
99     end
100 endcase
101 end
102 end

```

在这个模块中，根据hl_v1的输入，将其转换为对应的操作数或操作符，并将其输出到input_type和input_temp中。

1.1.2.2 仿真结果

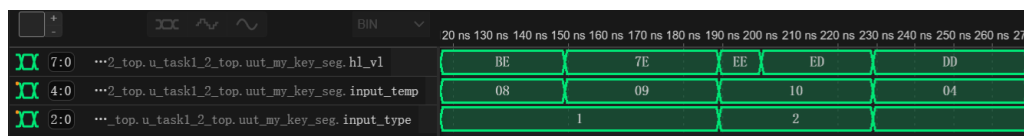


图 2: 按键处理模块仿真结果

1.1.3 状态机模块

为了实现计算器的功能，我实现了一个简单的mearly状态机，其状态转移图如下：

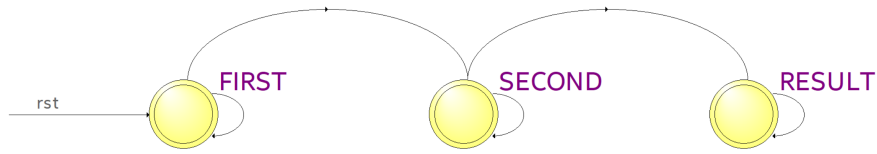


图 3: 状态机状态转移图

其中，FIRST为显示第一个操作数，SECOND为显示第二个操作数，RESULT为显示计算结果。

1.1.3.1 代码实现

```

1  //try to use mealy machine to solve the problem
2  reg [2:0] FSM_state;
3  parameter FIRST = 3'b001;
4  parameter SECOND = 3'b010;
5  parameter RESULT = 3'b100;
6  reg [4:0] operator;
7  parameter OR_OPERATOR = 5'b00001;
8  parameter AND_OPERATOR = 5'b00010;
9  parameter ADD_OPERATOR = 5'b00100;
10 parameter SUB_OPERATOR = 5'b01000;
11 parameter COMPARE_OPERATOR = 5'b10000;
12
13 //the first part of the FSM, trying to get state according to input
14 reg [11:0] first_num;
15 reg [11:0] second_num;
16 reg [15:0] result;
17 always @(posedge clk, posedge rst) begin
18     if(rst) begin
19         FSM_state <= FIRST;
20         operator <= 5'b00000;
21         first_num <= 12'b0000_0000_0000;
22         second_num <= 12'b0000_0000_0000;
23     end
24     else begin
25         case(FSM_state)

```

```
26 FIRST: begin
27     if(input_type == NUMBER) begin
28         first_num <= {first_num[7:0],input_temp
29             [3:0]};
30         FSM_state <= FIRST;
31     end
32     else if(input_type == OPERATOR) begin
33         operator <= input_temp;
34         FSM_state <= SECOND;
35     end
36     else begin
37         FSM_state <= FIRST;
38     end
39 end
40 SECOND: begin
41     if(input_type == NUMBER) begin
42         second_num <= {second_num[7:0],input_temp
43             [3:0]};
44         FSM_state <= SECOND;
45     end
46     else if(input_type == EQUAL) begin
47         FSM_state <= RESULT;
48     end
49     else begin
50         FSM_state <= SECOND;
51     end
52 end
53 RESULT: begin
54     case(operator)
55     OR_OPERATOR: begin
56         result <= first_num | second_num;
57         FSM_state <= RESULT;
58     end
59     AND_OPERATOR: begin
60         result <= first_num & second_num;
61         FSM_state <= RESULT;
62     end
63     ADD_OPERATOR: begin
64         result[3:0] <= (first_num[3:0] +
65             second_num[3:0])%10;
```

```

63         result[7:4] <= (first_num[7:4] +
64             second_num[7:4] + (first_num
65                 [3:0] + second_num[3:0])/10)
66                 %10;
67         result[11:8] <= (first_num[11:8] +
68             second_num[11:8] + (first_num
69                 [7:4] + second_num[7:4] + (
70                 first_num[3:0] + second_num
71                 [3:0])/10)/10)%10;
72         result[15:12] <= (first_num[11:8]
73             + second_num[11:8] + (first_num
74                 [7:4] + second_num[7:4] + (
75                 first_num[3:0] + second_num
76                 [3:0])/10)/10)/10;
77         FSM_state <= RESULT;
78     end
79     SUB_OPERATOR: begin
80         result[3:0] <= (first_num[3:0] -
81             second_num[3:0])%10;
82         result[7:4] <= (first_num[7:4] -
83             second_num[7:4] + (first_num
84                 [3:0] - second_num[3:0])/10)
85                 %10;
86         result[11:8] <= (first_num[11:8] -
87             second_num[11:8] + (first_num
88                 [7:4] - second_num[7:4] + (
89                 first_num[3:0] - second_num
90                 [3:0])/10)/10)%10;
91         result[15:12] <= (first_num[11:8]
92             - second_num[11:8] + (first_num
93                 [7:4] - second_num[7:4] + (
94                 first_num[3:0] - second_num
95                 [3:0])/10)/10)/10;
96         FSM_state <= RESULT;
97     end
98     COMPARE_OPERATOR: begin
99         if(first_num > second_num) begin
100             result <= 12'b1;
101         end
102         else begin

```



```

80                                     result <= 12'b0;
81                                     end
82                                     FSM_state <= RESULT;
83                                 end
84                                 default: begin
85                                     result <= 12'b0;
86                                     FSM_state <= RESULT;
87                                 end
88                             endcase
89                         end
90                         default: begin
91                             FSM_state <= FIRST;
92                         end
93                     endcase
94                 end
95             end

```

以上为mearly状态机的第一部分，作用为根据输入切换状态，同时计算结果。

```

1  //the second part of the FSM, trying to show the result according to the state
2  always @(posedge clk, posedge rst) begin
3      if(rst) begin
4          data <= 16'b0000_0000_0000_0000;
5      end
6      else begin
7          case(FSM_state)
8              FIRST: begin
9                  data <= first_num;
10             end
11             SECOND: begin
12                 data <= second_num;
13             end
14             RESULT: begin
15                 data <= result;
16             end
17             default: begin
18                 data <= 16'b0000_0000_0000_0000;
19             end

```

```

20         endcase
21     end
22 end

```

mealy状态机的第二部分，作用为根据状态显示对应的结果。

1.1.3.2 仿真结果

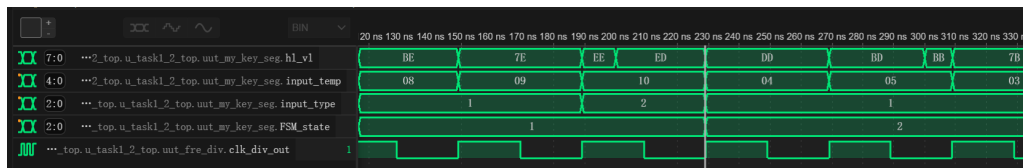


图 4: 状态机模块仿真结果1

由上图可以看出，当input_type为1时，FSM_state会维持在FIRST状态。而当input_type为2时，FSM_state会转移到SECOND状态。证明状态机切换正常。

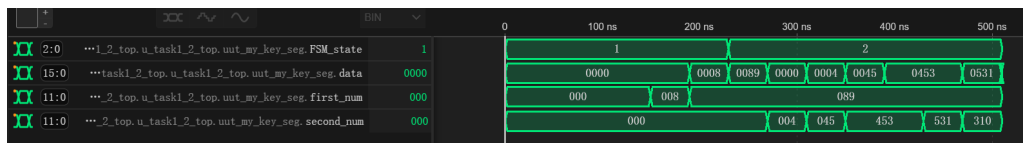


图 5: 状态机模块仿真结果2

由上图可以看出，当FSM_state为FIRST状态时，data与first_num保持一致，而当FSM_state为SECOND状态时，data与second_num保持一致。证明状态机显示正常。

1.2 仿真设计

仿真结果已在上文中给出，代码实现如下。

1.2.1 代码实现

```

1  //~ 'New testbench
2  `timescale 1ns/1ps
3  `include "task1_2_top.v"
4
5  module tb_task1_2_top;

```

```
6
7 // task1_2_top Parameters
8 parameter PERIOD = 10;
9
10 // task1_2_top Inputs
11 reg [3:0] vl = 0;
12 reg clk = 0;
13 reg rst = 0;
14
15 // task1_2_top Outputs
16 wire [3:0] hl;
17 wire [3:0] digit;
18 wire [7:0] seg;
19
20 task1_2_top u_task1_2_top (
21     .clk(clk),
22     .rst(rst),
23     .vl(vl[3:0]),
24     .hl(hl[3:0]),
25     .digit(digit[3:0]),
26     .seg(seg[7:0])
27 );
28
29 initial begin
30     $dumpfile(".\\wave\\tb_task1_2_top.vcd");
31     $dumpvars(0, tb_task1_2_top);
32     clk = 0;
33     rst = 0;
34     vl = 4'b1111;
35     rst = 1;
36     #10
37     rst = 0;
38     #500
39     $finish;
40 end
41
42 always begin
43     #1 clk = ~clk;
44 end
45
```

```
46     always begin
47         #100
48         v1 = 4'b1110;
49         #100
50         v1 = 4'b1101;
51         #100
52         v1 = 4'b1011;
53         #100
54         v1 = 4'b0111;
55     end
56
57 endmodule
```

2 task2_2

2.1 模块设计

分频模块、数码管显示模块与week6一致，不再赘述。

2.1.1 地址生成模块

2.1.1.1 代码实现

```
1  module addrgen(
2      input clk,
3      input rst,
4      output reg [7:0] addr
5  );
6
7  always @(posedge clk or posedge rst) begin
8      if(rst) begin
9          addr <= 8'b00000000;
10     end
11     else begin
12         addr <= addr + 1;
13     end
14 end
```

```
14 end  
15 endmodule
```

作用为每1s将addr加1。

2.1.1.2 仿真结果

图 6: 地址生成模块仿真结果