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；

```
        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

```
always @(posedge clk or posedge rst) begin
    if(rst) begin
        state <= 2'b00;
                hl <= 4'b1111;
    end
    else begin
        case (state)
            2'b00: begin
                                hl <= 4'b1110;
                state <= 2'b01;
            end
```

```
            2'b01: begin
                                hl <= 4'b1101;
                state <= 2'b10;
            end
            2'b10: begin
                                hl <= 4'b1011;
                state <= 2'b11;
            end
            2'b11: begin
                                hl <= 4'b0111;
                state <= 2'b00;
            end
            default: begin
                                hl <= 4'b1111;
                state <= 2'b00;
            end
        endcase
    end
end
assign hl_vl = {hl,vl};
```

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_vl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

```
//process the origin input to a simple one
```

```
reg [2:0] input_type;
parameter NUMBER = 3'b001;
parameter OPERATOR = 3'b010;
parameter EQUAL = 3'b100;

reg [4:0] input_temp;

always @ (posedge clk,posedge rst) begin
        if (rst) begin
                input_type <= 3'b000;
                input_temp <= 5'b00000;
        end
        else begin
                case (hl_vl)
                        8'b0111_0111: begin
                                //for =
                                input_type <= EQUAL;
                                input_temp <= 5'b00000;
                        end
                        8'b1011_0111: begin
                                //for and
                                input_type <= OPERATOR;
                                input_temp <= AND_OPERATOR;
                        end
                        8'b1101_0111: begin
                                //for 0
                                input_type <= NUMBER;
                                input_temp <= 5'b00000;
                        end
                        8'b1110_0111: begin
                                //for or
                                input_type <= OPERATOR;
                                input_temp <= OR_OPERATOR;
                        end
                        8'b0111_1011: begin
                                //for +
                                input_type <= OPERATOR;
                                input_temp <= ADD_OPERATOR;
                        end
                        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
```

```
                                //for 9
                                input_type <= NUMBER;
                                input_temp <= 5'b01001;
                        end
                        8'b1101_1110: begin
                                //for 8
                                input_type <= NUMBER;
                                input_temp <= 5'b01000;
                        end
                        8'b1110_1110: begin
                                //for 7
                                input_type <= NUMBER;
                                input_temp <= 5'b00111;
                        end
                        default: begin
                                input_type <= 3'b000;
                                input_temp <= 5'b00000;
                        end
                endcase
        end
end
```

在这个模块中，根据hl_vl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：

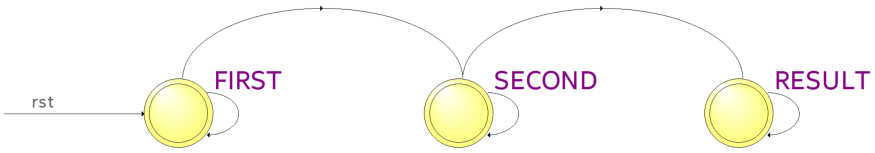

图 3: 状态机状态转移图

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

```
//try to use mealy machine to solve the problem
reg [2:0] FSM_state;
parameter FIRST = 3'b001;
parameter SECOND = 3'b010;
parameter RESULT = 3'b100;
reg [4:0] operator;
parameter OR_OPERATOR = 5'b00001;
parameter AND_OPERATOR = 5'b00010;
parameter ADD_OPERATOR = 5'b00100;
parameter SUB_OPERATOR = 5'b01000;
parameter COMPARE_OPERATOR = 5'b10000;

//the first part of the FSM, trying to get state according to input
reg [11:0] first_num;
reg [11:0] second_num;
reg [15:0] result;
always @(posedge clk, posedge rst) begin
        if(rst) begin
                FSM_state <= FIRST;
                operator <= 5'b00000;
                first_num <= 12'b0000_0000_0000;
                second_num <= 12'b0000_0000_0000;
        end
        else begin
                case(FSM_state)
```

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

```
63                                      result[7:4] <= (first_num[7:4] +
                                            second_num[7:4] + (first_num
                                            [3:0] + second_num[3:0])/10)
                                            %10;
64                                      result[11:8] <= (first_num[11:8] +
                                             second_num[11:8] + (first_num
                                            [7:4] + second_num[7:4] + (
                                            first_num[3:0] + second_num
                                            [3:0])/10)/10)%10;
65                                      result[15:12] <= (first_num[11:8]
                                            + second_num[11:8] + (first_num
                                            [7:4] + second_num[7:4] + (
                                            first_num[3:0] + second_num
                                            [3:0])/10)/10)/10;
66                                      FSM_state <= RESULT;
67                                  end
68                                  SUB_OPERATOR: begin
69                                      result[3:0] <= (first_num[3:0] -
                                            second_num[3:0])%10;
70                                      result[7:4] <= (first_num[7:4] -
                                            second_num[7:4] + (first_num
                                            [3:0] - second_num[3:0])/10)
                                            %10;
71                                      result[11:8] <= (first_num[11:8] -
                                             second_num[11:8] + (first_num
                                            [7:4] - second_num[7:4] + (
                                            first_num[3:0] - second_num
                                            [3:0])/10)/10)%10;
72                                      result[15:12] <= (first_num[11:8]
                                            - second_num[11:8] + (first_num
                                            [7:4] - second_num[7:4] + (
                                            first_num[3:0] - second_num
                                            [3:0])/10)/10)/10;
73                                      FSM_state <= RESULT;
74                                  end
75                                  COMPARE_OPERATOR: begin
76                                      if(first_num > second_num) begin
77                                              result <= 12'b1;
78                                      end
79                                      else begin
```

```
                                                result <= 12'b0;
                                        end
                                        FSM_state <= RESULT;
                                end
                                default: begin
                                        result <= 12'b0;
                                        FSM_state <= RESULT;
                                end
                        endcase
                end
                default: begin
                        FSM_state <= FIRST;
                end
        endcase
        end
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。

```
//the second part of the FSM, trying to show the result according to the state
always @(posedge clk, posedge rst) begin
        if(rst) begin
                data <= 16'b0000_0000_0000_0000;
        end
        else begin
                case(FSM_state)
                        FIRST: begin
                                data <= first_num;
                        end
                        SECOND: begin
                                data <= second_num;
                        end
                        RESULT: begin
                                data <= result;
                        end
                        default: begin
                                data <= 16'b0000_0000_0000_0000;
                        end
```

```
                endcase
        end
end
```

mearly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

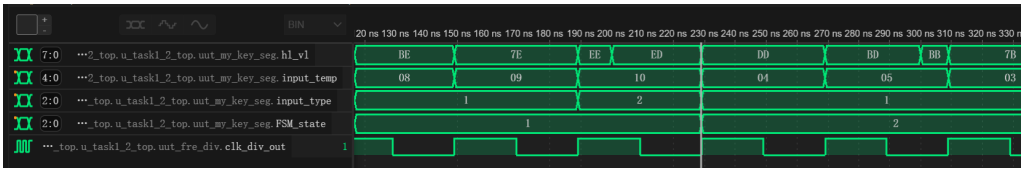

图 4: 状态机模块仿真结果1

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

```
//~ `New testbench
`timescale 1ns/1ps
`include "task1_2_top.v"

module tb_task1_2_top;
```

```

    // task1_2_top Parameters
    parameter PERIOD = 10;

    // task1_2_top Inputs
    reg [3:0] vl = 0;
    reg clk = 0;
    reg rst = 0;

    // task1_2_top Outputs
    wire [3:0] hl;
    wire [3:0] digit;
    wire [7:0] seg;

    task1_2_top u_task1_2_top (
        .clk(clk),
        .rst(rst),
        .vl(vl[3:0]),
        .hl(hl[3:0]),
        .digit(digit[3:0]),
        .seg(seg[7:0])
    );

    initial begin
        $dumpfile(".\\wave\\tb_task1_2_top.vcd");
        $dumpvars(0, tb_task1_2_top);
        clk = 0;
        rst = 0;
        vl = 4'b1111;
        rst = 1;
        #10
        rst = 0;
        #500
        $finish;
    end

    always begin
        #1 clk = ~clk;
    end

```

```
    always begin
        #100
        vl = 4'b1110;
        #100
        vl = 4'b1101;
        #100
        vl = 4'b1011;
        #100
        vl = 4'b0111;
    end

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

```
module addrgen(
    input clk,
    input rst,
    output reg [7:0] addr
);

always @(posedge clk or posedge rst) begin
    if(rst) begin
        addr <= 8'b00000000;
    end
    else begin
        addr <= addr + 1;
    end
```

```
14 end
15 endmodule
```

作用为每1s将addr加1。

2.2 仿真设计

2.2.1 代码实现

```
1                 //~ `New testbench
2  `timescale 1ns / 1ps
3  `include "task2_2_top.v"
4  module tb_task2_2;
5
6    // task2_2_top Parameters
7    parameter PERIOD = 10;
8
9
10   // task2_2_top Inputs
11   reg         clk = 0;
12   reg         rst = 0;
13
14   // task2_2_top Outputs
15   wire        clk_div_out;
16   wire        clk_div_out1;
17   wire [3:0] digit;
18   wire [7:0] seg;
19
20   task2_2_top u_task2_2_top (
21       .clk(clk),
22       .rst(rst),
23       .clk_div_out (clk_div_out),
24       .clk_div_out1(clk_div_out1),
25       .digit       (digit[3:0]),
26       .seg         (seg[7:0])
27   );
28
29   initial begin
```

```
30      //$dumpfile(".\\wave\\tb_task2_2_top.vcd");
31      //$dumpvars(0, tb_task2_2);
32      clk = 0;
33      rst = 0;
34      #100 rst = 1;
35      #100 rst = 0;
36      #500
37      $finish;
38    end
39
40    always begin
41      #1 clk = ~clk;
42    end
43
44  endmodule
```