Lab 5 report

PB2111711 陈昕琪

实验目的与内容

> 1. 学习并掌握有限状态机的设计方法，掌握利用有限状态机实现时序逻辑电路的方法。
> 2. 了解时序逻辑电路基本元件的组成以及有限状态机的知识并自行设计时序逻辑电路。
> 3. 进一步熟悉 Verilog 语言并学会综合分析电路以及功能实现。

1. 必做内容：寄存器堆

> **要求：**
>
> 请参照实验文档中对寄存器堆的描述，设计并编写一个寄存器堆，要求满足以下功能：

1. 寄存器堆的规模为 32x32bits；
2. 0 号寄存器始终保持 0；
3. 当同时读写时，能读到正写入的最新数据。

> **逻辑实现：**
>
> 需要参照实验教程，将寄存器拓展到32位。 在写入时首先要判断要写入的地址是不是0，因为0号寄存器要保持为0，并且在每次时钟边沿要为0号寄存器赋值为0。 每次写入的时候使用的是时序逻辑电路，而读出的时候使用的是组合逻辑电路。 在检查的时候和助教交流才得知，本题同时写入读入要考虑现实情况没有仿真那么理想，可能由于延迟导致时钟上升边沿没有成功写入。要解决这个问题，可以利用读出时组合逻辑电路的特性。加入一个判断，判断本周期内要读出的值和读入的值地址位置是否一致，若一致则直接读出要写入的值。 **本题需要注意时序逻辑电路和组合逻辑电路的组合应用，注意保持0号寄存器为0。**

代码如下：

```
module Regfile (
    input                       clk,          // 时钟信号
    input           [4:0]       ra1,          // 读端口 1 地址
    input           [4:0]       ra2,          // 读端口 2 地址
    input           [4:0]       wa,           // 写端口地址
    input                       we,           // 写使能信号
    input           [31:0]      din,          // 写数据
    output  reg     [31:0]      dout1,        // 读端口 1 数据输出
    output  reg     [31:0]      dout2         // 读端口 2 数据输出
);
// Write your code here
reg [31:0] register [31:0];//32×32bits
// 写端口
always @(posedge clk) begin
```

```
    register[0] <= 32'b0;
    if (we && wa != 5'd0) begin
        register[wa] <= din;
    end else register[0] <= 32'b0;
end
always @(*)begin
    if(ra1 == wa)begin
        if (wa != 0) dout1 = din;
        else dout1 = 0;
    end else if(ra2 == wa)begin
        if (wa != 0) dout2 = din;
        else dout1 = 0;
    end else begin
        dout1 = register[ra1];
        dout2 = register[ra2];
    end
end
// End of your code
endmodule
```

仿真结果与分析

> 根据所写的代码，编写相应的仿真文件验证其正确性。 为了模拟延迟这种情况，将时钟上升沿延迟半个周期，即clk初始状态为0。

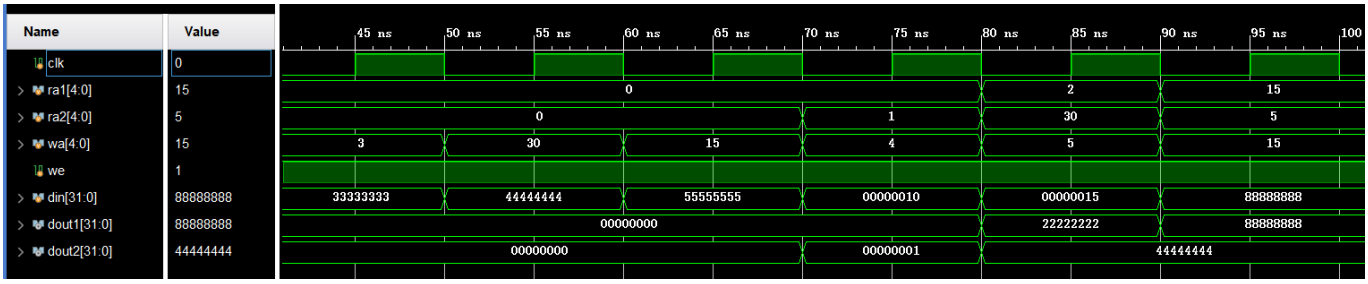
仿真文件代码如下：

```
module Regfile_tb();
reg clk;
reg [4:0] ra1, ra2, wa;
reg we;
reg [31:0] din;
wire [31:0] dout1, dout2;

Regfile regfile (
    .clk(clk),
    .ra1(ra1),
    .ra2(ra2),
    .wa(wa),
    .we(we),
    .din(din),
    .dout1(dout1),
    .dout2(dout2)
);

initial begin
    // 初始化信号
    clk = 0;
    #10;
    ra1 = 0;  // 读端口1地址
    ra2 = 0; // 读端口2地址
```

```
        wa = 1;  //
        we = 1;   // 写使能
        din = 32'h11111111; // 写数据

        #10;
        ra1 = 0;  // 读端口1地址
        ra2 = 0; // 读端口2地址
        wa = 1;//地址
        we = 1;//使能
        din = 32'h00000001;

        #10
        wa = 2;//地址
        we = 1;//使能
        din = 32'h22222222;

        #10
        wa = 3;//地址
        we = 1;//使能
        din = 32'h33333333;

        #10
        wa = 30;//地址
        we = 1;//使能
        din = 32'h44444444;

        #10
        wa = 15;//地址
        we = 1;//使能
        din = 32'h55555555;

    //前面写入了地址为0, 1, 2, 30, 15的数据
        #10;
        ra1 = 0;  // 读端口1地址
        ra2 = 1; // 读端口2地址
        wa = 4;//地址
        we = 1;//使能
        din = 32'h10;

        #10;
        ra1 = 2;  // 读端口1地址
        ra2 = 30; // 读端口2地址
        wa = 5;//地址
        we = 1;//使能
        din = 32'h15;

        //同时读写时，测试是否可以读入最新值
        #10;
        ra1 = 15;  // 读端口1地址
        ra2 = 5; // 读端口2地址
        wa = 15;
        we = 1;
        din = 32'h88888888;
```

```
end

always #5 clk = ~clk;

endmodule
```

> 仿真得出的波形图如下：



> 对比分析得知程序可以实现：
>
> 1. 寄存器堆的规模为 32x32bits；
> 2. 在写入地址为0时，0 号寄存器始终保持 0;
> 3. 在一个后期内，同时读写时，能读到正写入的最新数据。（仿真时将地址15赋初值之后再写入时可以读出新值）

2. 必做内容：数码管时钟

> **要求：**
>
> 参照实验文档中案例部分实现七段数码管制作时钟的内容，制作实现一个五位时钟，精度为 1 秒，比如七段数码管上从左到右显示 12345，则表明时间为 1 小时 23 分钟 45 秒。 除此之外，需要实现按动按钮能够将时钟复位到 12345，也就是复位值为 1 小时 23 分钟 45 秒的功能。
>
> **逻辑实现：**
>
> 首先对outh,outmin,outs都赋初值为0。然后根据一个周期为10ns，计算出1s应该是1_0000_0000个周期，即count整除1_0000_0000。同理count整除60_0000_0000表示一分钟，整除3600_0000_0000表示一小时。 同时，要实现显示部分，需要调用数码管显示器部分，显示的temp为temp = {{12'h0}, {outh}, {outmin}, {outs}}的拼接。 对于复位部分，在识别到按钮（button）之后，count要复位到1425_0000_0001。（如果复位到1425_0000_0000，可以整除1_0000_0000所以outs会直接跳到46）
> 同时，对于数码管显示部分，由于数码管是显示十六进制，所以在显示数字到达9或者19等以9结尾的数字时时，需要变为10、20等。在这里用case就可以解决。

> 本题代码包括三个部分：Timer、数码管显示部分、Top模块

代码如下：

1. Timer部分

```
module TIMER (
    input                           clk,rst,
    output      wire     [3:0]       seg_data,
    output      wire     [2:0]       seg_an
);

    reg [3:0]outh ;
    reg [7:0] outmin;
    reg [7:0] outs;
    reg [39:0] count;//到10^8表示1s，最多为60×60s,即3600_0000_0000

    reg [32:0] temp;
    always @(*) begin
        temp = {{12'h0}, {outh}, {outmin}, {outs}};
    end

    initial begin
        outh    = 4'h0;
        outmin   = 8'h0;
        outs   = 8'h0;
        count   = 40'h1;
    end

    //计数器部分
    always@ (posedge clk) begin
        if (rst) begin
            count <= 40'd1425_0000_0001;//仿真时参数是14250001
        end
        else begin
            if (count > 40'd3600_0000_0000)//conut需要从1到3600_0000_0001计数
                count <= 40'h1;
            else
                count <= count + 40'h1;
        end
    end

    /*使用 Lab3 实验练习中编写的数码管显示模块*/
    Segment segment(
        .clk                (clk),
        .rst                (rst),
        .output_data        (temp),
        .output_valid       (8'HFF),      // 如果你没有实现，可以不需要这个端口
        .seg_data           (seg_data),
        .seg_an             (seg_an)
    );

    always@ (posedge clk) begin
        if(rst) begin//复位为12345
            outh <= 4'h1;
            outmin <= 8'h23;
            outs <= 8'h45;
        end
        else begin
```

```
            if(count == 40'd3600_0000_0000) begin//仿真时参数是36000000
                if(outh == 4'h9) begin
                    outh <= 4'h0;
                end else outh <= outh + 1;
            end else outh <= outh;

            if(count % 40'd60_0000_0000 ==0) begin//仿真时参数是600000
                case(outmin)
                    8'h09:   outmin <= 8'h10;
                    8'h19:   outmin <= 8'h20;
                    8'h29:   outmin <= 8'h30;
                    8'h39:   outmin <= 8'h40;
                    8'h49:   outmin <= 8'h50;
                    8'h59:   outmin <= 8'h00;
                    default: outmin <= outmin + 1;
                endcase
                end else outmin <= outmin;

            if(count % 40'd1_0000_0000 == 0) begin//仿真时参数是10000
                case(outs)
                    8'h09:   outs <= 8'h10;
                    8'h19:   outs <= 8'h20;
                    8'h29:   outs <= 8'h30;
                    8'h39:   outs <= 8'h40;
                    8'h49:   outs <= 8'h50;
                    8'h59:   outs <= 8'h00;
                    default: outs <= outs + 1;
                endcase
            end else outs <= outs;
        end
    end
endmodule
```

2. 数码管显示部分

```
module Segment(
    input                       clk,
    input                       rst,
    input       [31:0]          output_data,
    input       [ 7:0]          output_valid,

    output reg  [ 3:0]          seg_data,
    output reg  [ 2:0]          seg_an
);
// 计时器部分
reg [31:0] counter;
//计数器的上限为2.5*10^5
parameter TIME_CNT = 25_0000;//仿真时参数是25
always @(posedge clk) begin
    if (rst) begin
        counter <= 0;
```

```
    end else begin
        if (counter >= TIME_CNT) begin
            counter <= 0;
        end else begin
            counter <= counter + 1;
        end
    end
end

// 数码管编号部分
reg [2:0] seg_id;
initial begin
    seg_id = 0;
    counter = 0;
end
always @(posedge clk) begin
    if (rst) begin
        seg_id <= 0;
    end else begin
        if(counter == 1) begin
            if (seg_id < 3'b111) begin
                seg_id <= seg_id + 1;
            end else seg_id <= 0;
        end else seg_id <= seg_id;
    end
end

// 数码管显示内容部分
always @(*) begin
    seg_data = 0;
    if(rst) seg_an = 3'b000;
    else begin
        seg_an = seg_id;   // <- 对于所有情况都相同
        case (seg_an)
            0: seg_data = output_data[3:0];
            1: if(output_valid[1] == 1) begin
                seg_data = output_data[7:4];
            end else begin
                seg_an = 0;
                seg_data = output_data[3:0];
            end
            2:if(output_valid[2] == 1) begin
                seg_data = output_data[11:8];
            end else begin
                seg_an = 0;
                seg_data = output_data[3:0];
            end
            3: if(output_valid[3] == 1) begin
                seg_data = output_data[15:12];
            end else begin
                seg_an = 0;
                seg_data = output_data[3:0];
            end
            4: if(output_valid[4] == 1) begin
```

```
                seg_data = output_data[19:16];
            end else begin
                seg_an = 0;
                seg_data = output_data[3:0];
            end
            5: if(output_valid[5] == 1) begin
                seg_data = output_data[23:20];
            end else begin
                seg_an = 0;
                seg_data = output_data[3:0];
            end
            6: if(output_valid[6] == 1) begin
                seg_data = output_data[27:24];
            end else begin
                seg_an = 0;
                seg_data = output_data[3:0];
            end
            7: if(output_valid[7] == 1) begin
                seg_data = output_data[31:28];
            end else begin
                seg_an = 0;
                seg_data = output_data[3:0];
            end
            default: ;
        endcase
    end
end
endmodule
```

3. Top模块

```
module Top(
    input                   clk,
    input                   btn,
    output [2:0]            seg_an,
    output [3:0]            seg_data
);
    TIMER timer(
        .clk(clk),
        .rst(btn),
        .seg_data(seg_data),
        .seg_an(seg_an)
    );
endmodule
```

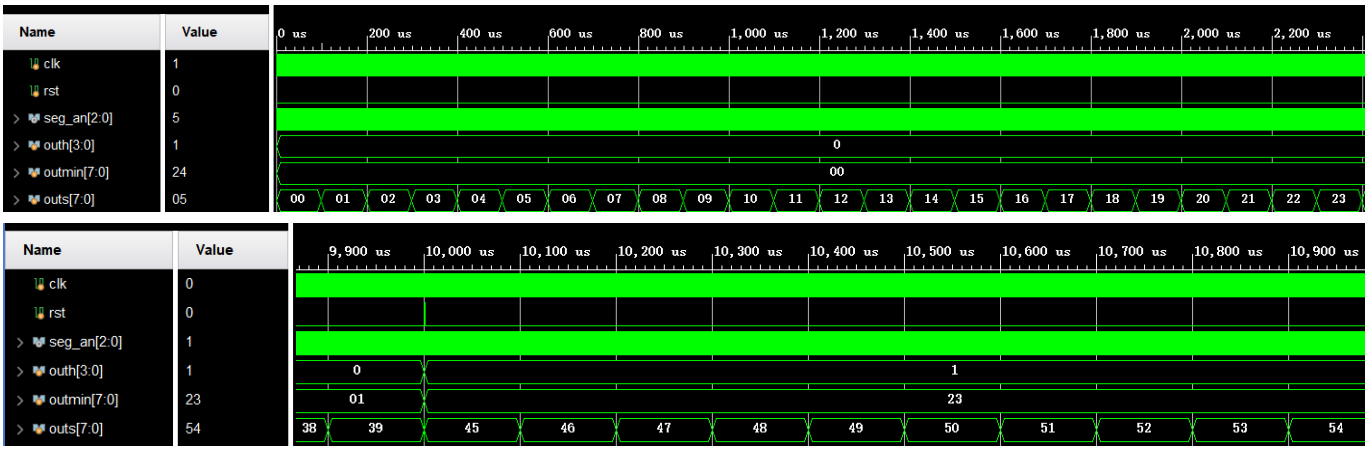
仿真结果与分析

> 根据所写的代码，编写相应的仿真文件验证其正确性。 **为了仿真的快速性，在仿真时更改了参数，降低了10^4倍，以实现快速观察结果。**

仿真文件代码如下：

```
module Timer_tb();
reg clk,rst;
wire [3:0] seg_data;
wire [2:0] seg_an;
initial begin
    clk = 1;
    rst = 0;
    #10000000;
    rst = 1;
    #10;
    rst = 0;
end
TIMER timer(
    .clk(clk),
    .rst(rst),
    .seg_data(seg_data),
    .seg_an(seg_an)
);
always #5 clk = ~clk;
endmodule
```

> 仿真得出的波形图如下：

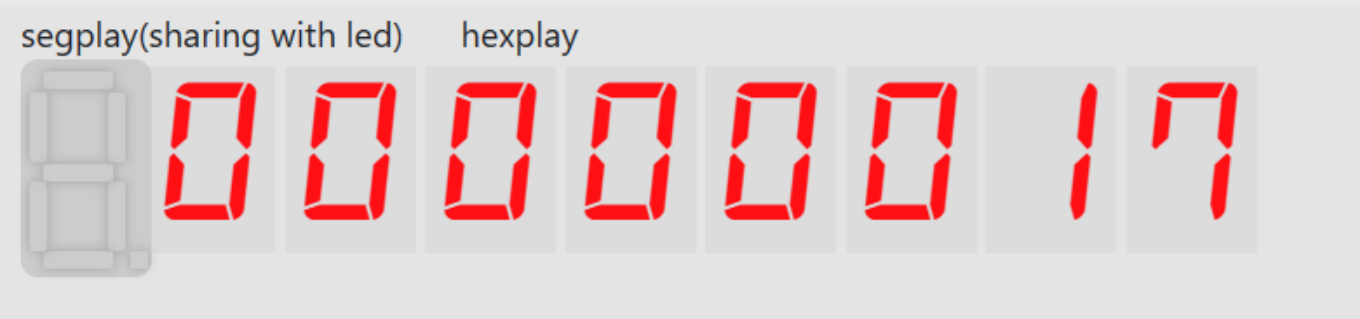


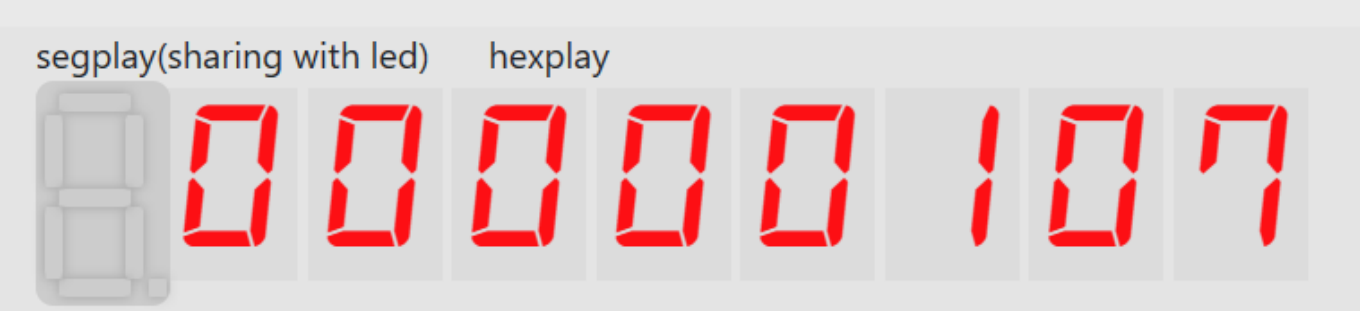
> 对比分析得知程序正确。

测试结果与分析

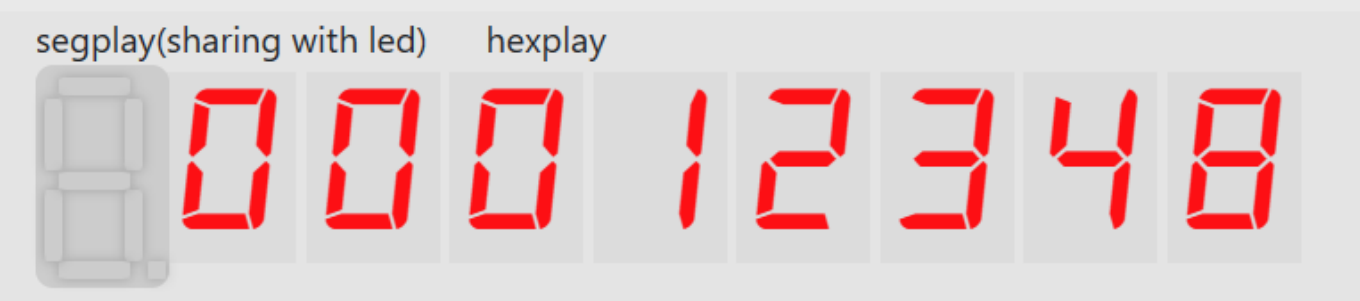
> 对约束文件，开放相应的sw端口，并烧写成bit流文件，上板烧写结果如下。

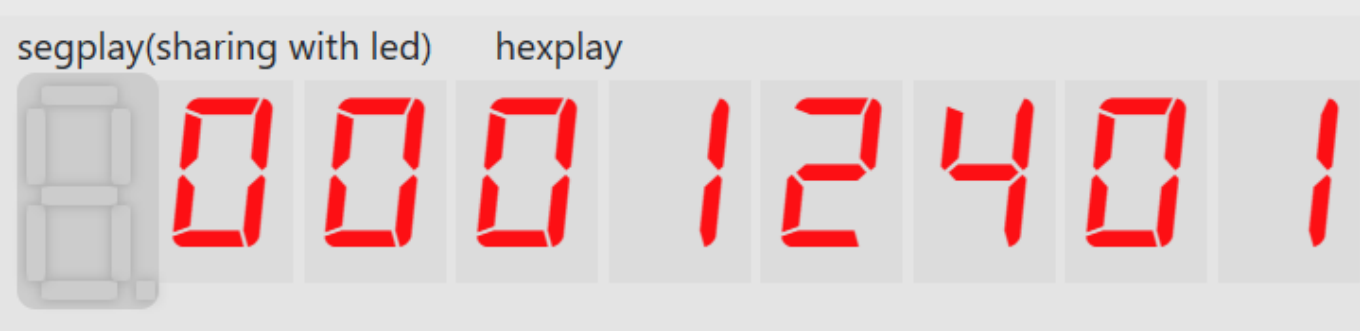
1. 初始时计数图如下：





2. 复位为12345后计数图如下：





> 上板烧写后，分析对判断程序正确。

3. 选择性必做内容：倍数检测器——再临

> **要求：**
>
> 利用 FSM 来制作一个 32bits 7 的倍数检测器,并通过仿真测试。 具体检测流程为：每次复位后，将输出有效位设置为 0，ready 位置为 1 ，等待输入有效位为 1 时接收输入并计算，计算过程中 ready 位保持 0 显示正在检测，此时输入会被忽略，计算出结果后输出并将输出有效位设置为 1， ready 位设为 1 并不再变化，直到下一次输入有效值变为 1 后进行下一次检测。

> **逻辑实现：**

> 首先需要定义8个状态，分别表示

- IDLE:初始状态
- S1:模七余1
- S2:模七余2
- S3:模七余3
- S4:模七余4
- S5:模七余5
- S6:模七余6
- S7:模七余0（即为7的倍数）。

> 从头开始遍历输入的每一位，每次遍历一位，相当于前面检测过的数字左移一位加现在正在检测的数字，那么就可以求出现在已经遍历过的数字模七余几。同时每个情况需要加入输入有效的判断。 当检测到最后一位的时候，判断此时是否是模七余0的状态，若是则输出为1，否则输出为0。 有关循环的判断:这里用了一个变量i,初始化为32，并在输入有效时开始减1操作，每次检测temp[i]，直到i=0,判断到最后一位，可以结束操作，将输出有效设置为1，并将输出值输出。 为了实现题目中的"计算过程中 ready 位保持 0 显示正在检测，此时输入会被忽略"，增加一个temp变量，在开始检测（即满足"i == 31 && ready == 0"）时将输入的数字赋值给temp，但是在检测过程中，由于增加了"if(i == 31 && ready == 0)"的判断，所以在检测过程中temp的值不会变化。

状态转换表

当前状态	检测到1	检测到0
IDLE	S1	S7
S1	S3	S2
S2	S5	S4
S3	S7	S6
S4	S2	S1
S5	S4	S3
S6	S6	S5
S7	S1	S0

代码如下：

```
module MUL7(
    input                           clk,            // 时钟信号
    input                           rst,            // 复位信号，使状态机回到初
始态
    input               [31 : 0]    src,            // 输入数据
    input                           src_valid,      // 表明输入结果是否有效
    output      reg                 ready,          // 表明是否正在检测
    output      reg                 res,            // 输出结果
    output      reg                 res_valid       // 表明输出结果是否有效
```

```
);

// Write your code here
    // 状态机定义
    localparam IDLE = 3'b000;
    localparam S1 = 3'b001;
    localparam S2 = 3'b010;
    localparam S3 = 3'b011;
    localparam S4 = 3'b100;
    localparam S5 = 3'b101;
    localparam S6 = 3'b110;
    localparam S7 = 3'b111;
    reg [5:0] i;
    reg [2:0] current_state,next_state;
    reg [31:0] temp;

initial begin
    i = 32;
end

    always @(posedge clk) begin
        if (rst) begin//表示复位
            current_state <= IDLE;
            ready <= 1'b1;//未在检测
            res <= 32'h0;
            res_valid <= 1'b0;//输出无效
            i <= 32;
        end
        else if (src_valid && i > 0) begin//开始输入
            current_state <= next_state;
            i = i - 1;
            ready <= 1'b0;
        end else if(i == 0) begin//表示未在检测
            ready <= 1'b1;
            res_valid  <= 1'b0;
            res <= 1'b0;
        end
    end

    always@(*) begin
        if(i == 31 && ready == 0) begin
            temp <= src;//正在检测且输入数据
        end
    end

    always@(*) begin
        next_state = current_state;
        if(i <= 31) begin
            case (current_state)
            IDLE: begin
                if (src_valid) begin
                    if(temp[i] == 1)begin
                        next_state = S1;
                    end else next_state = S7;
```

```
                    ready <= 1'b0;
                end else begin
                        ready <= 1'b1;
                        res_valid  <= 1'b1;
                    end
                end
            S1: begin
                if (src_valid) begin
                    if(temp[i] == 1)begin
                        next_state = S3;
                    end else next_state = S2;
                    ready <= 1'b0;
                end else begin
                        ready <= 1'b1;
                        res_valid  <= 1'b1;
                    end
                end
            S2: begin
                if (src_valid) begin
                    if(temp[i] == 1)begin
                        next_state = S5;
                    end else next_state = S4;
                    ready <= 1'b0;
                end else begin
                        ready <= 1'b1;
                        res_valid  <= 1'b1;
                    end
                end
            S3: begin
                if (src_valid) begin
                    if(temp[i] == 1)begin
                        next_state = S7;
                    end else next_state = S6;
                    ready <= 1'b0;
                end else begin
                        ready <= 1'b1;
                        res_valid  <= 1'b1;
                    end
                end
            S4: begin
                if (src_valid) begin
                    if(temp[i] == 1)begin
                        next_state = S2;
                    end else next_state = S1;
                    ready <= 1'b0;
                end else begin
                        ready <= 1'b1;
                        res_valid  <= 1'b1;
                    end
                end
            S5: begin
                    if (src_valid) begin
                        if(temp[i] == 1)begin
                            next_state = S4;
```

```
                        end else next_state = S3;
                        ready <= 1'b0;
                    end else begin
                            ready <= 1'b1;
                            res_valid  <= 1'b1;
                        end
                    end
            S6: begin
                if (src_valid) begin
                    if(temp[i] == 1)begin
                        next_state = S6;
                    end else next_state = S5;
                    ready <= 1'b0;
                end else begin
                        ready <= 1'b1;
                        res_valid  <= 1'b1;
                    end
                end
            S7: begin
                    if (src_valid) begin
                        if(temp[i] == 1)begin
                            next_state = S1;
                        end else next_state = S7;
                        ready <= 1'b0;
                    end else begin
                            ready <= 1'b1;
                            res_valid  <= 1'b1;
                        end
                    end
            endcase
        end
        if(i == 0) begin
            res = (next_state == S7);
            ready = 0;
            res_valid = 1;
        end
    end
// End of your code
endmodule
```

仿真结果与分析

> 根据所写的代码，编写相应的仿真文件验证其正确性。 分别测试7、14、15（增加了对ready的判断）、896、0、1、165，并观察结果。

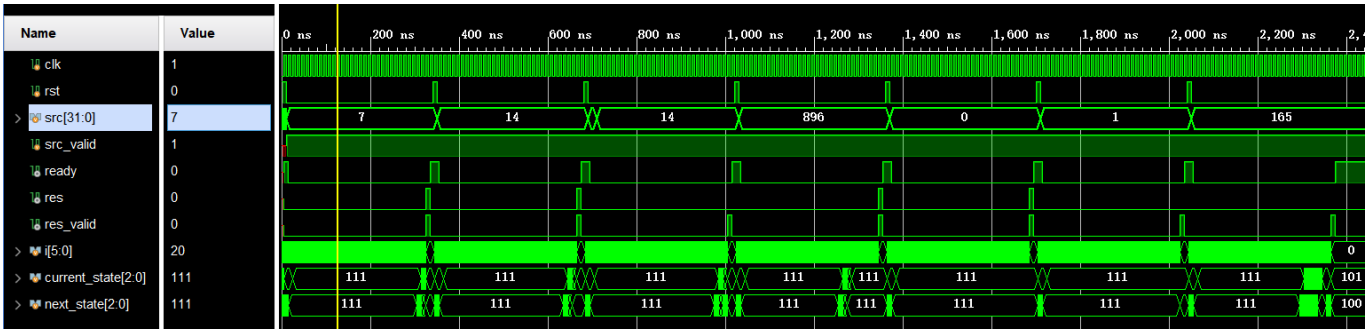
仿真文件代码如下：

```
module MUL7_tb();
reg clk,rst;
reg [31:0] src;
reg src_valid;
```

```
    wire ready,res,res_valid;

    initial begin
        clk = 0;
        rst = 1;

        #10;
        rst = 0;
        src = 32'b111;//7
        src_valid = 1;

        #330
        rst = 1;
        #10;
        rst = 0;
        src = 32'b1110;//14
        src_valid = 1;

        #330
        rst = 1;
        #10;
        rst = 0;
        src = 32'b1111;//15
        src_valid = 1;
        #20
        src = 32'b1110;//14
        src_valid = 1;

        #310
        rst = 1;
        #10;
        rst = 0;
        src = 32'b1110000000;//896
        src_valid = 1;

        #330
        rst = 1;
        #10;
        rst = 0;
        src = 32'b0;//0
        src_valid = 1;

        #330
        rst = 1;
        #10;
        rst = 0;
        src = 32'b1;//1
        src_valid = 1;

        #330
        rst = 1;
        #10;
        rst = 0;
        src = 32'b1010_0101;//165
```

```
    src_valid = 1;



end
always #5 clk = ~ clk;

MUL7 mul(
    .clk(clk),
    .rst(rst),
    .src(src),
    .src_valid(src_valid),
    .ready(ready),
    .res(res),
    .res_valid(res_valid)
);
endmodule
```

> 仿真得出的波形图如下：



> 观察比对得知7、14、896、0为7的倍数，且程序对ready为0时再次输入无效判断正确，由此可见程序正确。

总结

> 1. 本次实验，对于 Verilog 语言有了更深入的了解，同时学会利用状态机和时序逻辑元件来实现电路。
> 2. 对七段数码管能够熟练掌握运用，并且可以学会调用自己之前写过的模块解决更困难的问题。
> 3. 对于较难的问题，可以通过分析题干，查找资料获取灵感，能够独立编写程序并完成仿真，并可以基本保证程序正确性。
> 4. 要有整体化和模块化思想，理清楚程序的过程，程序编写就更加轻松容易。