实验名称：序列检测 姓名： 学号：

装

订

线

浙江大学

本科实验报告

姓名：

学院：生物医学工程与仪器科学学院

系：生物医学工程系

专业：生物医学工程

学号：

指导教师：余锋

实验名称：序列检测 姓名：__ 学号：

专业：生物医学工程
姓名：
学号：
日期：2025.4.29
地点：教 6－204

浙江大学 实验报告

课程名称：硬件描述语言 指导老师：余锋 实验名称：HDL 实验五

一、实验要求

二、实验代码与注释

三、仿真结果与分析

四、讨论与心得

装 订 线

一、实验要求

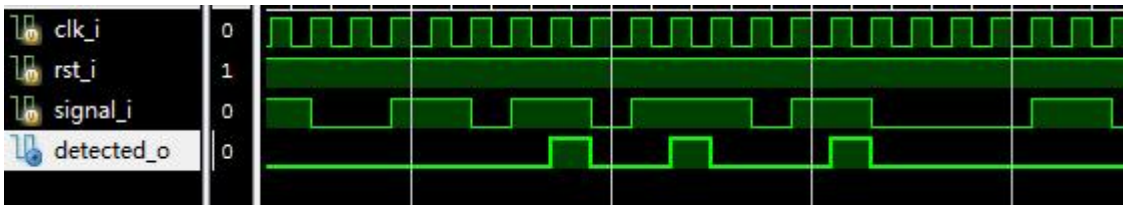
要求使用 mealy 状态机完成对一输入序列的检测，当输入序列中有 1011 的特征时，输出一个时钟周期的高电平。举例如下：

如果输入的序列为：

0001_0110_1011_0111_0010_1010_1101_0000_1011_1101_1000_0010_1101_1011_0011...

则输出的序列为：

0000_0010_0001_0010_0000_0000_0100_0000_0001_0000_1000_0000_0100_1001_0000...



要求完成：

1、 画出状态机的状态转换图

2、 detect_serial_fsm.v

```
module detect_serial_fsm(
    input  clk_i,
    input  rst_i,
```

实验名称：___序列检测___ 姓名：__ 学号：

```
    input  signal_i,
    output detected_o
);
endmodule
```

3、 tb_detect_serial_fsm.v

生成随机序列，序列变化与时钟上升沿对齐

二、实验代码与注释

装 订 线

1. Mealy 状态机 vs Moore 状态机

Mealy 状态机：输出由当前状态和当前输入共同决定。其输出在输入变化后立即响应，可能比时钟边沿早一个周期（需注意时序约束）。

Moore 状态机：输出仅由当前状态决定。输出在时钟边沿同步变化，时序更严格但可能延迟一个周期。

2. 序列检测的状态机设计核心

检测固定序列（如 "1011"）的关键是用状态表示当前已匹配的输入前缀长度：

状态 S0：未匹配任何前缀（初始状态）

状态 S1：已匹配前缀 "1"

状态 S2：已匹配前缀 "10"

状态 S3：已匹配前缀 "101"

状态 S4：已匹配完整序列 "1011"（检测成功状态）

实验名称：___序列检测___ 姓名：__ 学号：

装

订

线

3. 状态回退逻辑

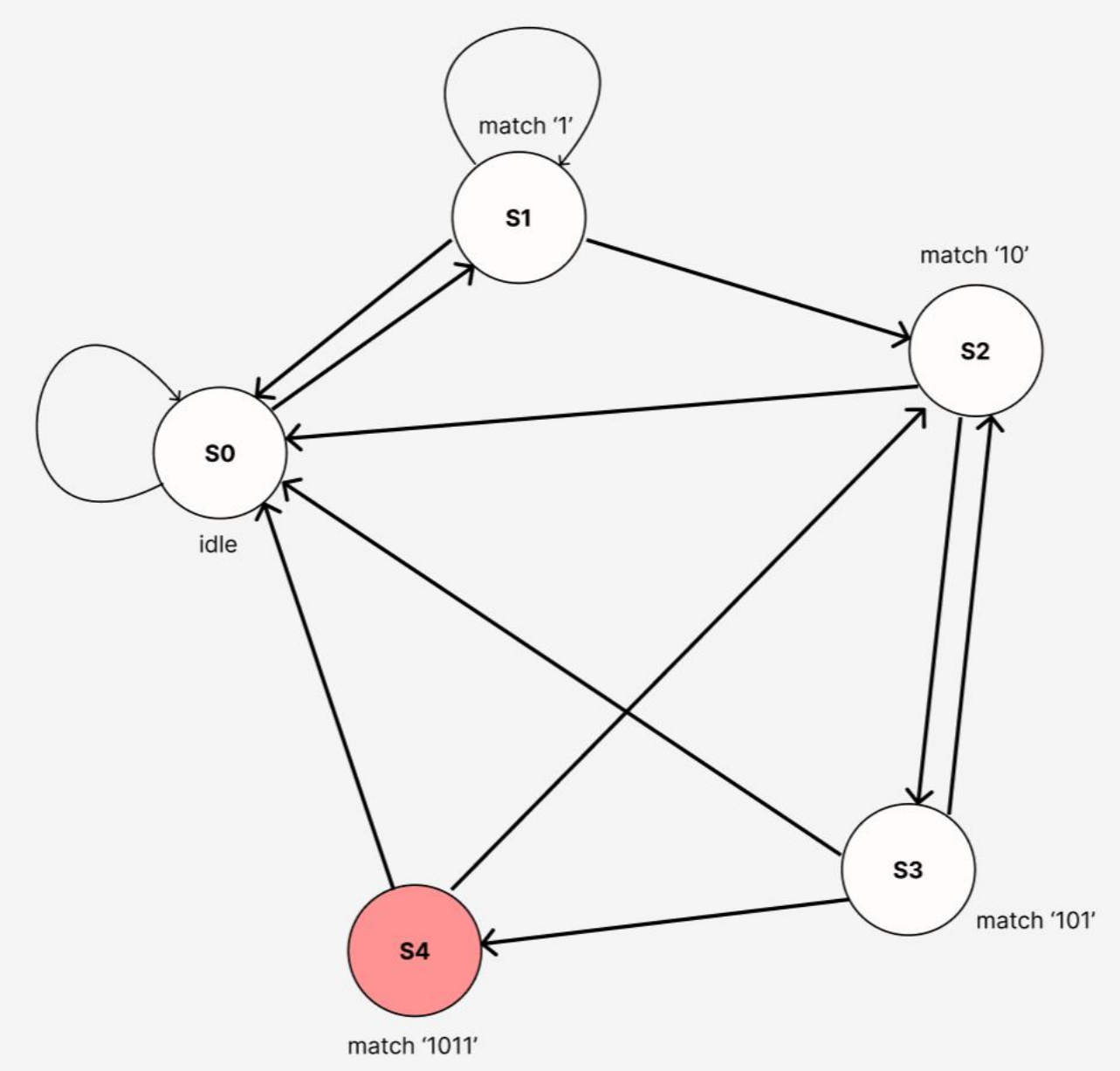
当当前输入不满足预期时，需根据历史输入判断是否能部分匹配新的前缀。例如：

在状态 S3（已匹配 "101"）时，若输入为 0（得到 "1010"），最后两位 "10" 可能成为新序列的前缀，因此回退到状态 S2（匹配 "10"）。

在状态 S2（已匹配 "10"）时，若输入为 0（得到 "100"），无法形成新前缀，回退到 S0。

当前状态	输入 signal_i	下一状态	输出 detected_o
S0	0	S0	0
S0	1	S1	0
S1	0	S2	0
S1	1	S1	0
S2	0	S0	0
S2	1	S3	0
S3	0	S2	0
S3	**1**	**S4**	1（仅当前周期）
S4	0	S2（因 "10" 可能为新前缀）	0
S4	1	S1（因 "1" 可能为新前缀）	0

实验名称：___序列检测___ 姓名：__ 学号：



4、 detect_serial_fsm.v

这里采用两段式写法。

```
module detect_serial_fsm(
    input   clk_i,
    input   rst_i,
    input   signal_i,
    output  detected_o
    );
    parameter S0 = 4'b0001;
```

装

订

线

实验名称：___序列检测___ 姓名：__ 学号：

装

订

线

```
    parameter S1 = 4'b0010;
    parameter S2 = 4'b0100;
    parameter S3 = 4'b1000;
    parameter S4 = 4'b0000;

    reg [3:0] state, next_state;
    // 状态转移组合逻辑
    always @(*) begin
        case (state)
            S0: begin // idle
                next_state = signal_i ? S1 : S0;
            end
            S1: begin // '1'
                if(signal_i) begin
                    next_state = S1;
                end else begin
                    next_state = S2;
                end
            end
            S2: begin // '0'
                if(signal_i) begin
                    next_state = S3;
                end else begin
                    next_state = S0;
                end
            end
            S3: begin // '1'
                if(signal_i) begin
                    next_state = S4; // ok
```

实验名称：___序列检测___ 姓名：__ 学号：

```
                end else begin
                    next_state = S2;
                end
            end
            S4: begin // '1'
                if(signal_i) begin
                    next_state = S1;
                end else begin
                    next_state = S2;
                end
            end
            default: next_state = S0;
        endcase
    end
    // 状态寄存器
    always @(posedge clk_i or posedge rst_i) begin
        if (rst_i) begin
            state <= S0;
        end else begin
            state <= next_state;
        end
    end
    // 输出逻辑(Mealy)
    assign #2 detected_o = (state == S3) && (signal_i);
endmodule
```

5、tb_detect_serial_fsm.v

```
`timescale 1ns/1ps
```

装 订 线

实验名称：___序列检测___ 姓名：__ 学号：

装

订

线

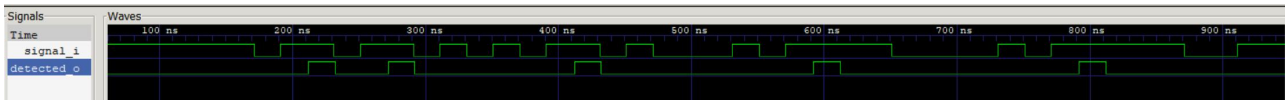
```
module tb_detect_serial_fsm();
    reg clk_i;
    reg rst_i;
    reg signal_i;
    wire detected_o;
    // Instantiate the DUT (Device Under Test)
    detect_serial_fsm dut (
        .clk_i(clk_i),
        .rst_i(rst_i),
        .signal_i(signal_i),
        .detected_o(detected_o)
    );
    // Clock generation
    initial begin
        clk_i = 0;
        forever #10 clk_i = ~clk_i;
    end
    // Test sequence
    initial begin
        // Initialize inputs
        rst_i = 1;
        signal_i = 0;
        // Release reset after 20 ns
        #20 rst_i = 0;
        repeat(100) begin  // 生成 100 位测试序列
            @(posedge clk_i);  // 等待时钟上升沿
            signal_i <= $random % 2;  // 随机生成 0 或 1
        end
        // Finish simulation after some time
```

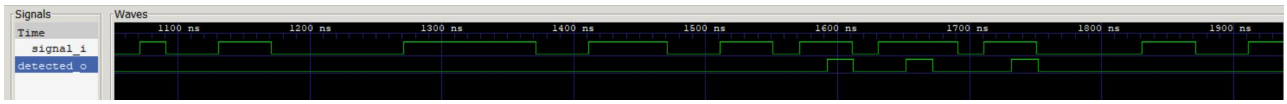
实验名称：___序列检测___ 姓名：__ 学号：

```
        #50 $finish;
    end
    initial begin
        // Monitor the signals
        $monitor("Time: %0t | clk_i: %b | rst_i: %b | signal_i: %b |
detected_o: %b", $time, clk_i, rst_i, signal_i, detected_o);
        $dumpfile("tb_detect_serial_fsm.vcd");
        $dumpvars(0, tb_detect_serial_fsm);
    end
endmodule
```

装 订 线

三、仿真结果与分析





这里使用 assign detected_o = (state == S3) && signal_i 作为标准的 Mealy 机输出逻辑，但是正如 PPT 所说，这里的组合逻辑和实验四一样有**毛刺**，本质上是数字电路中竞争-冒险的固有现象。

可以在加一个 reg 引入寄存器节拍，但经确认会延迟一个周期，在 assign 的时候手动添加了延时固然能解决，但并不是一个合理的方法。

后面锁定原因在 signal_i 上，因为我在 tb 用了 signal_i = $random % 2 这个阻塞赋值，把这里改掉之后就能在当前时间步结束时统一更新变量值，避免了组合逻辑因输入信号过早变化而产生的毛刺。

最终结果如图，无毛刺且实现了功能。通过$monitor 打印信息和波形图，验证了输出序

实验名称：___序列检测___ 姓名：__ 学号：

列与输入序列的对应关系，满足实验要求的检测功能。

四、讨论与心得

本次实验深入理解了 Mealy 状态机的工作原理，其输出依赖当前状态和输入，能实时响应输入变化。在代码编写中，状态转移逻辑是关键，需确保每种输入情况都正确转移。仿真中观察到的毛刺是组合逻辑竞争 - 冒险的体现，虽然理论上存在，但不影响功能实现。通过本次实验，掌握了序列检测的状态机设计方法，以及 Verilog 中状态机的建模与仿真技巧。