电子科技大学

University of Electronic Science and Technology of China

本科生实验报告

实验课程： 复杂数字集成电路设计（挑战性课程）

实验名称： 实验 4：序列发生和序列检测

实验地点： 无

学生姓名： 周岳恒

学　　号： 2021340105016

指导教师： 廖永波

实验时间： 2023 年 10 月 1 日

一、 实验目的

理解序列发生器、序列检测器的功能和对应电路的不同设计方法，学会通过不断优化的方式简化电路设计，提升电路性能。

通过使用移位寄存器、有限状态机、反馈移位寄存器的方式实现序列发生器，并尽可能地简化电路设计并提升性能。通过改进有限状态机的状态设计和转移逻辑、输出逻辑来实现不同序列检测的功能。

二、 实验任务

1. 用移位寄存器实现序列发生器；
2. 用有限状态机实现序列发生器；
3. 用反馈移位寄存器（Feedback Shift Register）实现序列发生器，节约资源使用；
4. 用有限状态机实现重复序列检测器；
5. 实现不重复序列检测器。

三、 实验原理

序列发生器：

使用移位寄存器实现的序列发生器，在 load 同步加载信号到来时给寄存器加载预加载值 din，且使能端无效；在 load 失效时将每个寄存器的输入设置为上一级寄存器的输出，第一个寄存器的输入为最后一个寄存器的输出。最后模块输出其中一个寄存器的输出，在时钟边沿到来时寄存器组产生移位，实现移位输出序列。

使用有限状态机实现的序列发生器不需要 load 加载信号，因为输出是通过多路选择器选择 din 输入，不需要寄存器加载外界输入。而多路选择器的选择信号来源于状态机。复位结束后，在每个时钟边沿状态机改变状态，输出选择信号逐一输出 din 的各个位，实现序列输出。

使用反馈移位寄存器实现序列发生器（6 位），在只有 3 位寄存器的情况下要使移位输出通过组合逻辑决定下一次输入，且保证 3 个状态位不重复交叠。使用卡诺图化简逻辑函数表达式，检查移位寄存器的状态即可。

序列检测器：

使用状态机实现序列检测，根据检测到的输入决定在每个时钟边沿状态机跳转到哪个状态上。决定是否检测重叠序列需要调整状态转移在遇到重复序列后条状的次态不是初始状态而是已经形成某种序列的准备状态。

四、 实验过程

序列发生器：

移位寄存器版：

verilog 代码：

```
module gen_shift(
    input clk,
    input rst_n,
    input load,
    input [5:0] din,
    output dout
);
    reg [5:0] shift;
    always @(posedge clk, negedge rst_n) begin
        if(~rst_n)
            shift <= 6'b0;
        else if(load)
            shift <= din;
        else
            shift <= {shift[4:0], shift[5]};
    end
    assign dout = shift[5];

endmodule
```

在 load 加载信号到来时寄存器装载，在 load 信号撤去后移位寄存器在时钟边沿循环左移，输出寄存器组的最高位，实现序列从左到右输出。

有限状态机版：

verilog 代码：

```
module gen_fsm(
    input clk,
    input rst_n,
    input [5:0] din,
    output dout
);
    localparam  S1 = 3'b000,
                S2 = 3'b001,
```

```
                S3 = 3'b011,
                S4 = 3'b010,
                S5 = 3'b110,
                S6 = 3'b111;
    reg [2:0] state,nxt_state;
    reg dout_reg;

    always @(posedge clk, negedge rst_n) begin
        if(~rst_n)
            state <= S1;
        else
            state <= nxt_state;
    end

    always @(*) begin
        case(state)
            S1: begin nxt_state = S2; dout_reg = din[5]; end
            S2: begin nxt_state = S3; dout_reg = din[4]; end
            S3: begin nxt_state = S4; dout_reg = din[3]; end
            S4: begin nxt_state = S5; dout_reg = din[2]; end
            S5: begin nxt_state = S6; dout_reg = din[1]; end
            default: begin nxt_state = S1; dout_reg = din[0]; end
        endcase
    end
    assign dout = dout_reg;

endmodule
```

不需要 load 加载，在时钟上升边沿就更改状态，且状态不由任何输入决定（其实可以改为计数器+多路复用器），并根据当前状态决定输出位 din 的某一位，实现序列输出。在第一个时钟边沿就准备输出下一位，所以比移位寄存器的电路要快一个节拍。

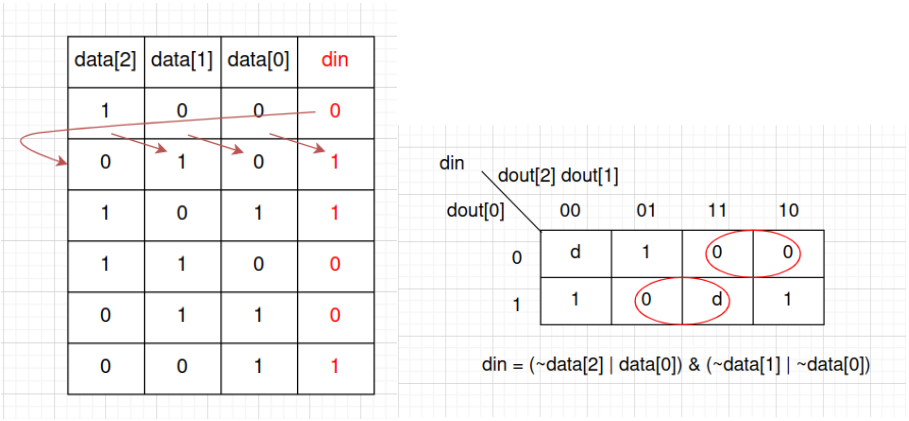
反馈移位寄存器：

verilog 代码：

```
module gen_min(
    input clk,
    input rst_n,
    output dout
);
    reg [2:0] data;
    wire df;
```

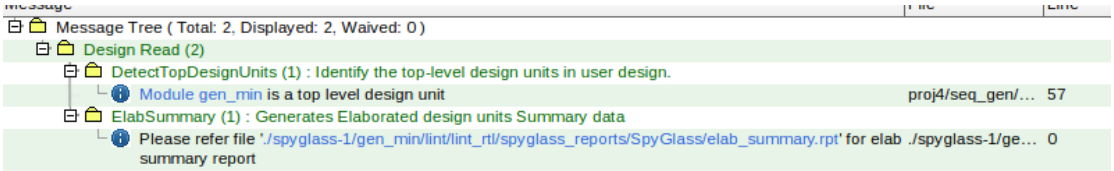
```
    assign df = (~data[2] | data[0]) & (~data[1] | ~data[0]);

    always @(posedge clk, negedge rst_n) begin
        if(~rst_n)
            data <= 3'b100;
        else
            data <= {df, data[2:1]};
    end
    assign dout = data[0];
endmodule
```

列出移位寄存器可能产生的各个状态位，以及复位状态位、无关状态位：



在不考虑可能进入无效状态（000 和 111）的情况下（可以不考虑，因为寄存器有异步复位可以保证进入有效状态），把寄存器的状态（d1，d2，d3）视为输入，下一次移位的输入（din）看做输出，列出卡诺图。由于 0 比 1 圈起来优化效果好（圈数少，圈面大），所以选择或与式表达式。复位时寄存器被置为 100，和 001011 的顺序一致，复位后就是从头开始产生序列。

spyglass 分析 rtl_lint 结果无 warning：



序列检测器：

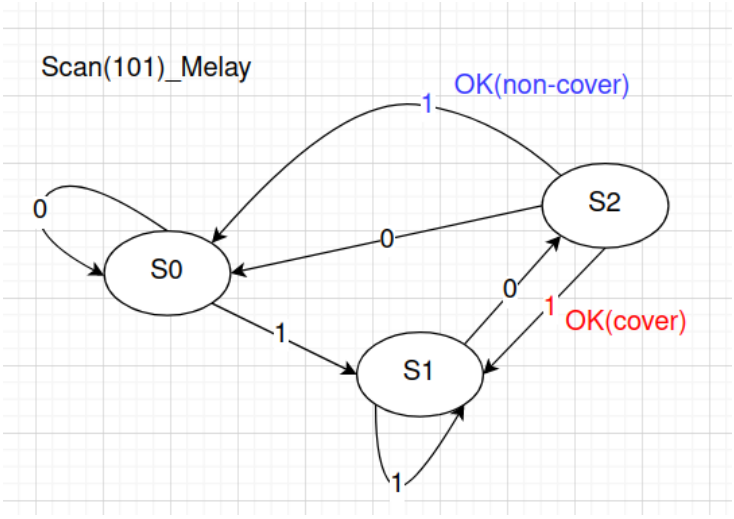
本次我选择 Mealy 机，输出取决于输入，这样状态可以减少一个。且覆盖和不覆盖的设计可以共用这些状态，改动较小。

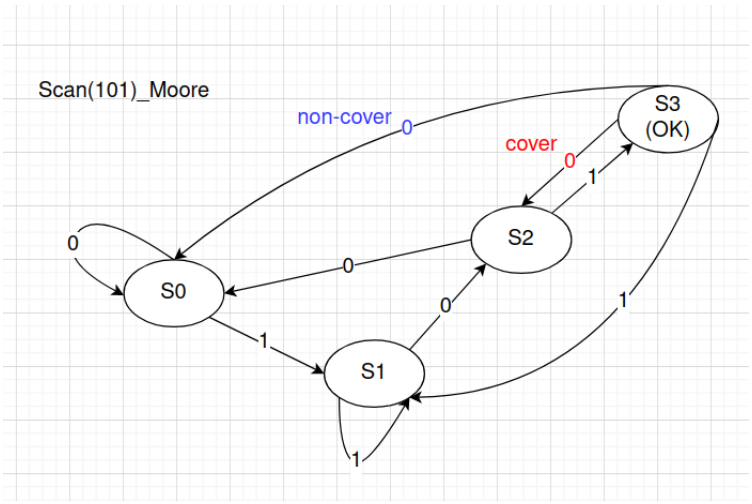
verilog 代码：

```
module seq_detection_cover(
    input clk,
    input rst_n,
    input data,
    output result
);
    localparam S0 = 2'b00,
               S1= 2'b01,
               S2= 2'b11;
    reg [1:0] state, nxt_state;

    always @(posedge clk, negedge rst_n) begin
        if(~rst_n)
            state <= S0;
        else
            state <= nxt_state;
    end

    always @(*) begin
        case(state)
            S0: nxt_state = data ? S1 : S0;
            S1: nxt_state = data ? S1 : S2;
            default: nxt_state = data ? S1 : S0;
        endcase
    end

    assign result = (state == S2 && data);

endmodule

module seq_detection_noncover(
    input clk,
    input rst_n,
    input data,
    output result
);
    localparam S0 = 2'b00,
               S1= 2'b01,
               S2= 2'b11;
    reg [1:0] state, nxt_state;

    always @(posedge clk, negedge rst_n) begin
        if(~rst_n)
```

```
            state <= S0;
        else
            state <= nxt_state;
    end

    always @(*) begin
        case(state)
            S0: nxt_state = data ? S1 : S0;
            S1: nxt_state = data ? S1 : S2;
            default: nxt_state = S0;  // change is here.
        endcase
    end

    assign result = (state == S2 && data);

endmodule
```

状态转移图：





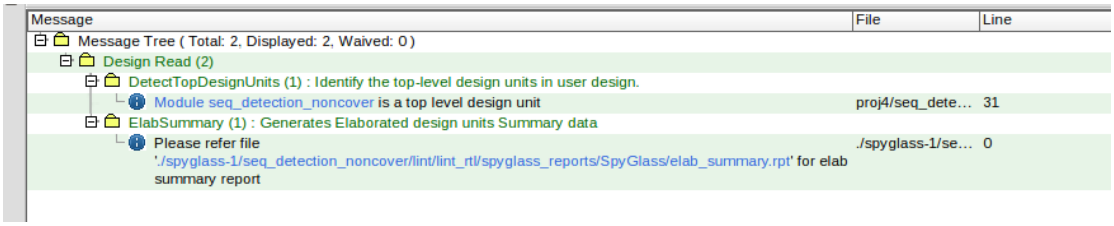
Mealy 机：以检测 101 为例：初始状态为 S0，检测到 1 进入 S1 否则保持 S0；

在 S1（检测到 1）检测到 1 保持 S1 否则进入 S2；在 S2（检测到 10）检测到 1 表示检测到 101，输出检测完成脉冲：如果是覆盖模式，则表示检测到 1 个 1，所以进入 S1，否则不覆盖应该进入 S0 表示重新开始检测；检测到 0 表示检测到 100：在 101 里没有两个 0 的情况，所以无论是否覆盖都进入 S0 等待。

Moore 机：由于输出不直接取决于输入，需要单独的状态位 S3（OK）标志输入情况并存储，在这里用 S3 表征检测到了 101。而且覆盖和非覆盖版本的区别在于在 S3 状态检测到 0 之后跳转的下一状态：覆盖到 S2，不覆盖到 S0 从头开始，这一部分和 Mealy 机的过程等效。

最终结果上 Mealy 由于少一个状态位，所以反馈输出会比 Moore 机快，但是由于组成是组合逻辑，如果输入不稳定，可能输出会随之也不稳定产生噪声问题。

spyglass 分析 rtl_lint 结果无 warning：



五、 仿真结果

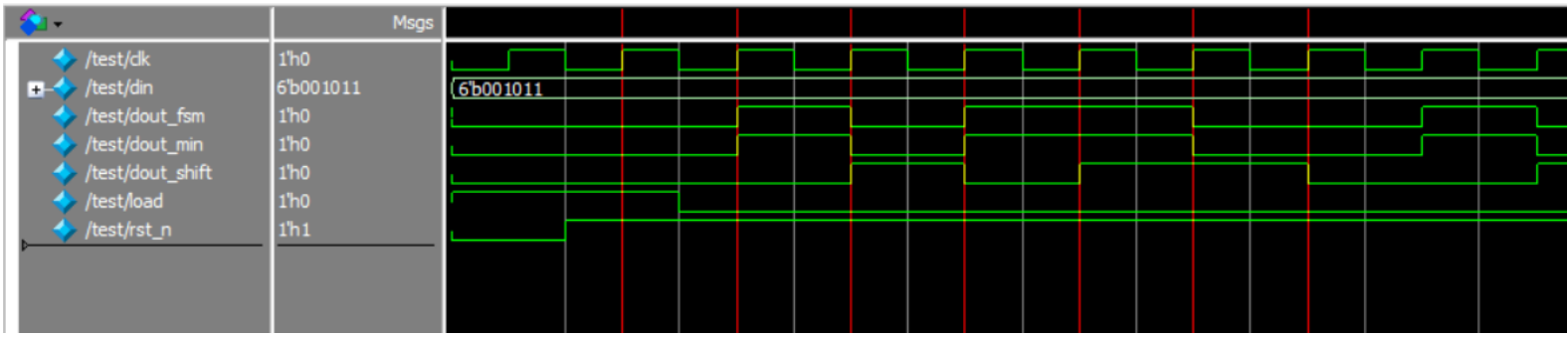
序列发生器：

testbench：

```
`timescale 1ps/1ps
module test();
    reg clk,rst_n,load;
    reg [5:0] din;
    wire dout_shift,dout_fsm,dout_min;

    gen_shift u1(
        .clk(clk),
        .rst_n(rst_n),
        .load(load),
        .din(din),
        .dout(dout_shift)
    );
    gen_fsm u2(
        .clk(clk),
```

```
        .rst_n(rst_n),
        .din(din),
        .dout(dout_fsm)
    );
    gen_min u3(
        .clk(clk),
        .rst_n(rst_n),
        .dout(dout_min)
    );

    initial begin
        clk = 1'b0;
        forever #5 clk = ~clk;
    end

    initial begin
        rst_n = 1'b0;
        load = 1'b1;
        din = 6'b001011;
        #10 rst_n = 1'b1;
        #10 load = 1'b0;
        // #60 din = 6'b101010;
        // load = 1'b1;
        // #10 load = 1'b0;
    end
endmodule
```

仿真波形：



可以看到除了移位寄存器因为 load 需要延长一个节拍外，所有输出都能产生序列 001011。

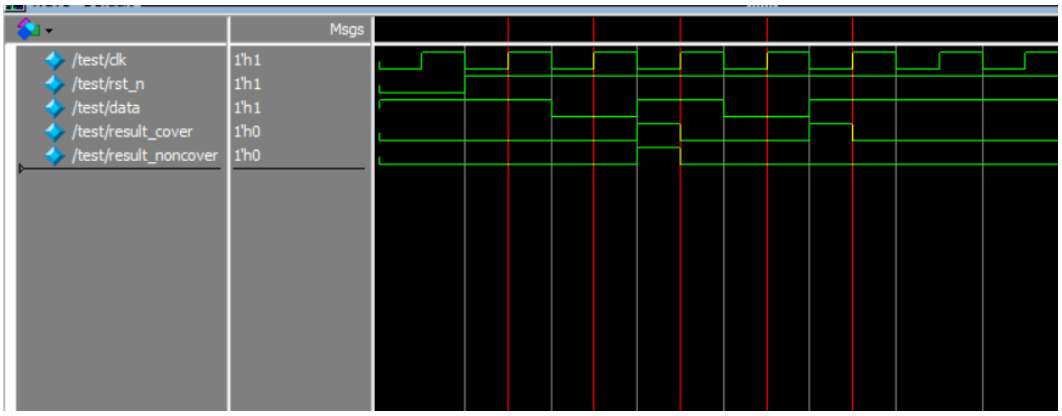
序列检测器：

testbench：

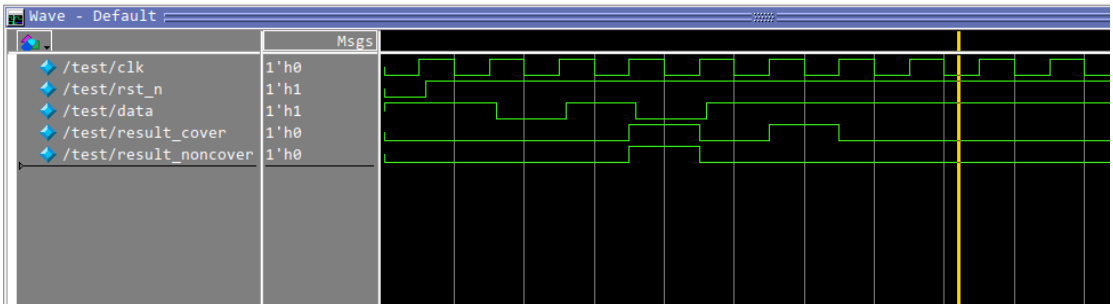
```
`timescale 1ps/1ps
module test();
```

```
    reg clk,rst_n,data;
    wire result_cover,result_noncover;

    seq_detection_cover u1(
        .clk(clk),
        .rst_n(rst_n),
        .data(data),
        .result(result_cover)
    );
    seq_detection_noncover u2(
        .clk(clk),
        .rst_n(rst_n),
        .data(data),
        .result(result_noncover)
    );

    initial begin
        clk = 1'b0;
        forever #5 clk = ~clk;
    end

    initial begin
        rst_n = 1'b0;
        data = 1'b1;
        #10 rst_n = 1'b1;
        #10 data = 1'b0;
        #10 data = 1'b1;
        #10 data = 1'b0;
        #10 data = 1'b1;
        #10 data = 1'b1;
        #10 data = 1'b1;
    end
endmodule
```

仿真波形：



可以看到 data 输入为 10101，覆盖版本在第三位和第五位都会输出脉冲，不覆盖版本只在第三位输出脉冲，第五位不输出脉冲，符合期望。



使用 Moore 机实现序列检测功能：由于 Moore 机需要单独的状态位记录输入情况，所以输出结果会根据时钟信号定时一拍，且输出结果只在时钟边沿变化，不会像组合逻辑一样随时随输入变化。在输入 101 后下一拍两个结果都保持 1，输入 10101 后 coverge 版本输出 0，non-coverge 版本输出 1，验证了功能的正确性。

六、 实验总结

本次实验实现了序列发生器和序列检测器的设计，使用移位寄存器、有限状态机、反馈移位寄存器的不同设计导致不同优化结果。

移位寄存器需要 load 信号加载导致慢一拍；有限状态机可以直接优化成计数器和多路选择器的组合，因为此状态机不会因为输入改变状态；反馈移位寄存器设计注意卡诺图化简和无效状态位的检查。

在实现序列检测器的有限状态机的设计中对状态机的转移逻辑进行修改，达到了实现重叠序列检测和不重叠序列检测的效果，对状态机的设计更加谨慎。同时也要注意使用 Moore 机和 Mealy 机设计的不同。

此次实验使用 spyglass 工具检查 rtl 代码可能出现的问题，幸运的是我的设计暂时还没遇到过有问题的情况。今后我在设计中会特别注意组合逻辑在 always 块综合意外锁存器之类的情况，减少设计失误。