《数字电路》实验报告

实验名称： 状态机实验 指导教师：王琮，范志华

姓名：韩初晓 学号：2023K8009908002 专业：计算机科学与技术 班级：2306

实验日期：2024.10.17 实验地点：教学楼 224 是否调课/补课：否 成绩：

目录

第 1 部分　实验目的

1. 熟悉 verilog 编程、调试。
2. 熟悉状态机的工作原理，能熟练编写状态机程序。

第 2 部分　实验环境

- Vivado 2017.4 开发工具
- FPGA 开发平台（根据手册中的默认设置进行选择）

第 3 部分　实验内容

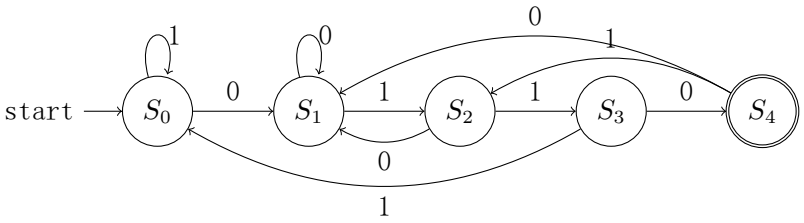
3.1 实验一：读取 0110 的有限状态自动机

3.1.1 原理说明

本实验实现了一个具有五个状态的同步状态机，接受 `0110` 为合法输入。模块根据输入信号 `in` 和当前状态 `state`，决定下一状态 `next_state` 并输出状态相关信号 `out`。有限状态机的具体状态及其行为如下：

- S0：初始状态，当输入信号 `in` 为 0 时，转移到状态 S1；否则保持在 S0。
- S1：当 `in` 为 1 时，转移到状态 S2；否则保持在 S1。
- S2：当 `in` 为 1 时，转移到状态 S3；否则返回到状态 S1。
- S3：当 `in` 为 1 时，转移回状态 S0；否则转移到状态 S4。
- S4：当 `in` 为 1 时，返回状态 S2；否则返回状态 S1。

状态转换图如下所示：



输出信号 `out` 在状态 S4 时为高电平（1），其他状态下为低电平（0）。模块的状态转移逻辑由时钟信号 `clk` 和复位信号 `rstn` 控制。

3.1.2 接口定义

- 输入信号：
  - `clk`：时钟信号，用于同步状态转移。
  - `rstn`：异步复位信号，低电平有效，复位时状态返回至初始状态 S0。
  - `in`：输入控制信号，决定状态转移路径。
- 输出信号：
  - `out`：状态机的输出信号，当状态为 S4 时为高电平（1），其他状态为低电平（0）。

3.1.3 调试过程及结果

通过编写 `fsm_1` 模块的 Testbench，对其状态转移行为进行了仿真测试。测试时，输入信号 `in` 在不同状态下进行了切换，并验证了模块的状态转移与输出行为。仿真波形如下所示：

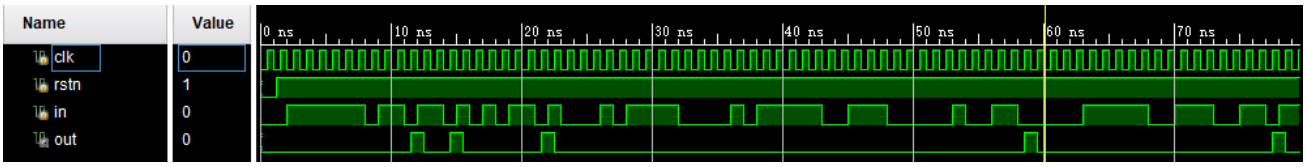


图 1: `fsm_1` 模块的仿真测试结果

通过观察仿真波形可以验证：

- 在复位信号 `rstn` 低电平时，状态机正确返回初始状态 S0。
- 状态机在每个状态下均按照设计逻辑正确转移。
- 输出信号 `out` 在状态 S4 时为高电平，表明读取到了合法信号 `0110` ，状态机的功能符合预期设计。

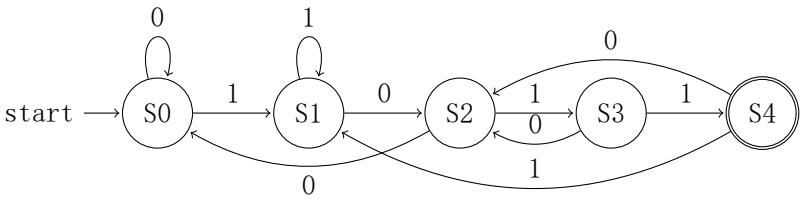
3.2 实验二：读取 1011 的有限状态自动机

3.2.1 原理说明

本实验实现了一个有限状态机（FSM），用于根据输入信号 `in` 的变化生成特定的输出信号 `out` ，接受 `1011` 为合法输入。有限状态机包含五个状态，通过输入信号和当前状态确定下一状态，并在特定状态下产生输出信号。状态的具体定义及其行为如下：

- S0：初始状态，当 `in` 为 1 时，转移到状态 S1；否则保持在 S0。
- S1：当 `in` 为 0 时，转移到状态 S2；否则保持在 S1。
- S2：当 `in` 为 1 时，转移到状态 S3；否则返回到状态 S0。
- S3：当 `in` 为 1 时，转移到状态 S4；否则转移到状态 S2。
- S4：当 `in` 为 1 时，返回状态 S1；否则转移到状态 S2。

状态转换图如下所示：



输出信号 `out` 在状态 S4 时为高电平（1），表明读取到了 `1011` 的合法输入信号，其他状态为低电平（0）。状态机由时钟信号 `clk` 驱动，并在复位信号 `rstn` 低电平时回到初始状态 S0。

3.2.2 接口定义

- 输入信号：
  - `clk`：时钟信号，用于同步状态转移。
  - `rstn`：异步复位信号，低电平有效，复位时状态返回至初始状态 S0。
  - `in`：输入信号，决定状态转移路径。
- 输出信号：
  - `out`：输出信号，在状态 S4 时为高电平（1），其余状态为低电平（0）。

3.2.3 调试过程及结果

通过编写 fsm_2 模块的 Testbench，对其状态转移和输出信号的生成逻辑进行了仿真测试。在仿真中，通过设置不同的输入信号 in，验证了状态转移路径及输出信号是否符合设计要求。仿真波形如下所示：

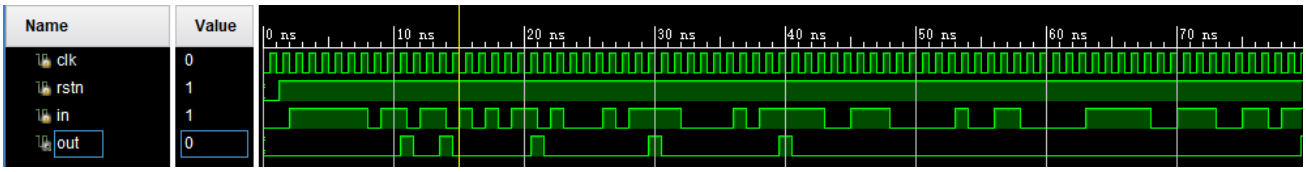


图 2: fsm_2 模块的仿真测试结果

从仿真结果中可以验证：

- 状态机在复位信号 rstn 为低电平时，正确返回初始状态 S0。
- 状态机在不同输入条件下正确完成状态转移。
- 输出信号 out 仅在状态 S4 时为高电平，表明读取到了 1011 的合法输入信号，符合设计要求。

3.3 实验三：实现信号生成器模块

3.3.1 原理说明

该信号生成器模块实现了一个具有 12 个状态的有限状态机（FSM），通过输入信号 clk 和复位信号 rstn 来控制状态转移，并根据当前状态生成输出信号 out。信号生成器的状态及其行为如下：

- S0：初始状态，输出信号 out 为 0，转移到状态 S1。
- S1：输出信号 out 为 0，转移到状态 S2。
- S2：输出信号 out 为 1，转移到状态 S3。
- S3：输出信号 out 为 0，转移到状态 S4。
- S4：输出信号 out 为 1，转移到状态 S5。
- S5：输出信号 out 为 0，转移到状态 S6。
- S6：输出信号 out 为 0，转移到状态 S7。
- S7：输出信号 out 为 1，转移到状态 S8。
- S8：输出信号 out 为 1，转移到状态 S9。
- S9：输出信号 out 为 0，转移到状态 S10。
- S10：输出信号 out 为 1，转移到状态 S11。
- S11：输出信号 out 为 1，转移回状态 S0。

状态机通过时钟信号 clk 进行同步操作，并在复位信号 rstn 低电平时返回初始状态 S0。

3.3.2 接口定义

- 输入信号：
  - clk：时钟信号，用于同步状态转移。
  - rstn：异步复位信号，低电平有效，复位时状态返回至初始状态 S0。
- 输出信号：
  - out：输出信号，依据当前状态生成对应的输出。

3.3.3 调试过程及结果

通过编写 signal_generator 模块的 Testbench，对其状态转移和输出信号的生成进行了仿真测试。测试时，输入信号 clk 周期性变化，复位信号 rstn 用于初始化，验证了状态机在不同状态下的转移情况

及输出信号的正确性。仿真波形如下所示：

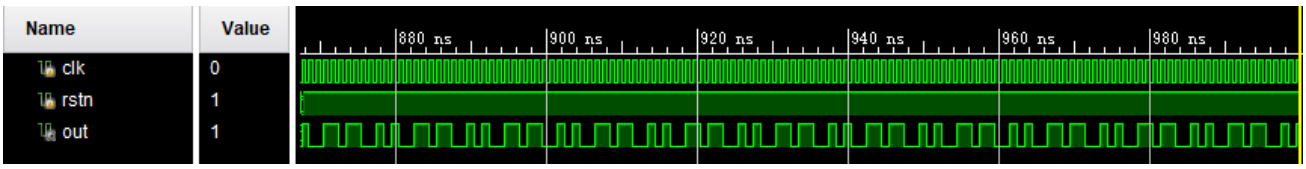


图 3: signal_generator 模块的仿真测试结果

从仿真结果中可以验证：

- 状态机在复位信号 rstn 为低电平时，正确返回初始状态 S0。
- 状态机在不同输入信号作用下，能够正确完成状态转移，周期性输出 001010011011。

3.4 实验四：自动报纸贩卖机

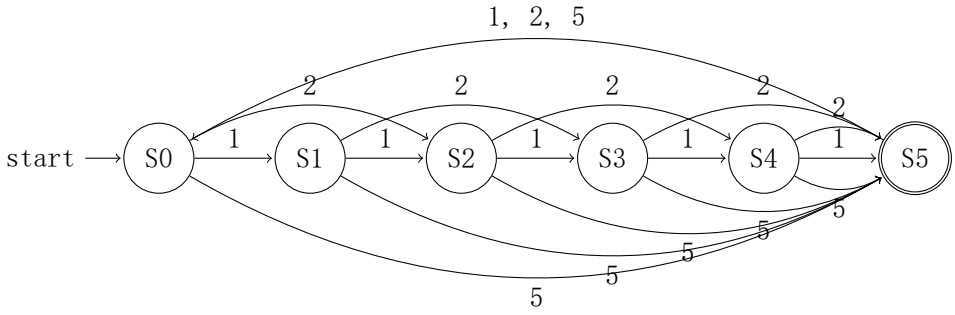
3.4.1 原理说明

实验中实现了自动报纸贩卖机模块，它接受 1 分、2 分和 5 分硬币，并根据投入的硬币金额判断是否发放报纸。每当用户投入总额达到 5 分时，系统会发放报纸。

该有限状态机共有 6 个状态，每个状态表示售货机的不同金额累计情况，描述如下：

- S0：初始状态，未收到任何硬币。
- S1：收到 1 分硬币，总金额为 1 分。
- S2：收到 2 分硬币，总金额为 2 分。
- S3：收到 3 分硬币，总金额为 3 分。
- S4：收到 4 分硬币，总金额为 4 分。
- S5：收到 5 分硬币，总金额为 5 分，发放报纸。

状态机的状态转换图如下所示：



每次投入硬币时，模块会根据硬币的面值调整状态，直到累计金额达到 5 分为止。如果投入的金额已满足 5 分，输出信号 dispense 为 1，表示发放报纸。

3.4.2 接口定义

- 输入信号：
  - clk：时钟信号，用于同步状态转移。
  - rstn：异步复位信号，低电平有效，复位时状态返回至初始状态 S0。
  - coin[2:0]：输入的硬币面值，支持三种硬币：
    - 1 分硬币对应 3’b001
    - 2 分硬币对应 3’b010
    - 5 分硬币对应 3’b100

- 输出信号：
  - dispense：输出信号，当总金额达到 5 分时为高电平（1），表示发放报纸；否则为低电平（0）。

3.4.3 调试过程及结果

通过编写 newspaper_machine 模块的 Testbench，对其状态转移和发放报纸逻辑进行了仿真测试。在测试过程中，输入不同面额的硬币并观察状态变化和输出信号 dispense 的变化。仿真结果如下所示：

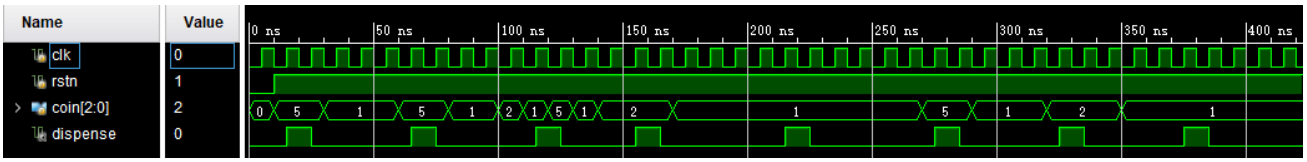


图 4: newspaper_machine 模块的仿真测试结果

从仿真结果中可以验证：

- 状态机能够正确识别投入的硬币（1 分、2 分或 5 分），并进行正确的状态转移。
- 当总金额达到 5 分时，输出信号 dispense 正确地变为高电平，表示报纸已发放。
- 如果硬币不足以达到 5 分，状态机会根据硬币继续增加总金额，而不会发放报纸。

第 4 部分 实验总结

在本实验中，我实现了四个有限状态自动机模块，深入体会了在 verilog 中设计时序电路的方法。在过度实验中，我了解了一段式、两段式和三段式自动机的设计，发现三段式状态机分为状态转移快，状态获取块和结果输出块三部分的代码结构更加清晰，也更加易于设计。本实验涉及到的有限状态自动机均为 Moore 型状态机，即状态机的输出仅与当前状态有关，而不依赖于输入信号。这代表着结果输出块可以通过将 state 作为敏感信号来实现。同时，在状态转移块中非阻塞赋值的应用也是本实验中的重要内容，在这一部分使用非阻塞赋值时需要规定信号的上升沿和下降沿。但是，在状态获取块中则需要使用阻塞赋值，这是特别需要注意的一点。在编写 Testbench 的时候，我注意到，对于检测输入序列是否合法的有限状态自动机，输入 in 信号的变化频率应当与模块中定义的时钟翻转频率相同，否则会造成错误。总的来说，本次实验让我对有限状态自动机的设计有了更深入的了解，也提高了我对 verilog 编程的熟练程度。

第 5 部分 源代码

5.1 实验一：读取 0110 的有限状态自动机

5.1.1 fsm_1 模块源代码

```
1
2 module fsm_1(
3     input clk,
4     input rstn,
5     input in,
6     output reg out
7     );
```

```
8
9     localparam S0 = 3'b000;
10    localparam S1 = 3'b001;
11    localparam S2 = 3'b010;
12    localparam S3 = 3'b011;
13    localparam S4 = 3'b100;
14
15    reg [2:0] state, next_state;
16
17    always @(posedge clk or negedge rstn) begin
18      if (~rstn) begin
19          state <= S0;
20      end
21      else begin
22          state <= next_state;
23      end
24    end
25
26    always @(state or in) begin
27      case(state)
28        S0: begin
29          if(in) begin
30            next_state = S0;
31          end
32          else begin
33            next_state = S1;
34          end
35        end
36        S1: begin
37          if(in) begin
38            next_state = S2;
39          end
40          else begin
41            next_state = S1;
42          end
43        end
44        S2: begin
45          if(in) begin
46            next_state = S3;
47          end
48          else begin
49            next_state = S1;
50          end
51        end
52        S3: begin
53          if(in) begin
54            next_state = S0;
55          end
56          else begin
57            next_state = S4;
58          end
59        end
```

```
        S4: begin
          if(in) begin
            next_state = S2;
          end
          else begin
            next_state = S1;
          end
        end
        default: next_state = S0;
      endcase
    end

    always @(state) begin
      case(state)
        S0:out = 1'b0;
        S1:out = 1'b0;
        S2:out = 1'b0;
        S3:out = 1'b0;
        S4:out = 1'b1;
        default: out = 1'b0;
      endcase
    end
endmodule
```

5.1.2 fsm_1 模块 Testbench

```
module test_fsm_1(
    );
    reg clk;
    reg rstn;
    reg in;
    wire out;

    fsm_1 instance_fsm_1(
      .clk(clk),
      .rstn(rstn),
      .in(in),
      .out(out)
      );

    initial begin
      clk = 0;
      rstn = 1;
      #0.1 rstn = 0;
      #1.1 rstn = 1;
    end

    initial begin
      in = 0;
      #1 in = 0;
      #1 in = 1;
```

```
      #1 in = 1;
      #1 in = 0;
      #1 in = 1;
      #1 in = 0;
      #1 in = 0;
    end

    always begin
      #1 in = $random() % 2;
    end

    always begin
      #0.5 clk = ~clk;
    end
endmodule
```

5.2 实验二：读取 1011 的有限状态自动机

5.2.1 fsm_2 模块源代码

```

module fsm_2(
    input clk,
    input rstn,
    input in,
    output reg out
    );

    localparam S0 = 3'b000;
    localparam S1 = 3'b001;
    localparam S2 = 3'b010;
    localparam S3 = 3'b011;
    localparam S4 = 3'b100;

    reg [2:0] state, next_state;

    always @(posedge clk or negedge rstn) begin
      if (~rstn) begin
          state <= S0;
      end
      else begin
          state <= next_state;
      end
    end

    always @(state or in) begin
      case(state)
        S0: begin
          if(in) begin
            next_state = S1;
          end
```

```
          else begin
            next_state = S0;
          end
        end
        S1: begin
          if(in) begin
            next_state = S1;
          end
          else begin
            next_state = S2;
          end
        end
        S2: begin
          if(in) begin
            next_state = S3;
          end
          else begin
            next_state = S0;
          end
        end
        S3: begin
          if(in) begin
            next_state = S4;
          end
          else begin
            next_state = S2;
          end
        end
        S4: begin
          if(in) begin
            next_state = S1;
          end
          else begin
            next_state = S2;
          end
        end
        default: next_state = S0;
      endcase
    end

    always @(state) begin
      case(state)
        S0:out = 1'b0;
        S1:out = 1'b0;
        S2:out = 1'b0;
        S3:out = 1'b0;
        S4:out = 1'b1;
        default: out = 1'b0;
      endcase
    end
endmodule
```

5.2.2 fsm_2 模块 Testbench

```
module test_fsm_2(
    );
    reg clk;
    reg rstn;
    reg in;
    wire out;

    fsm_2 instance_fsm_2(
      .clk(clk),
      .rstn(rstn),
      .in(in),
      .out(out)
      );

    initial begin
      clk = 0;
      rstn = 1;
      #0.1 rstn = 0;
      #1.1 rstn = 1;
    end

    initial begin
      in = 0;
      #1 in = 1;
      #1 in = 0;
      #1 in = 1;
      #1 in = 1;
      #1 in = 0;
      #1 in = 1;
      #1 in = 1;
      #1 in = 0;
      #1 in = 1;
      #1 in = 0;
      #1 in = 0;
      #1 in = 1;
      #1 in = 0;
    end

    always begin
      #1 in = $random() % 2;
    end

    always begin
      #0.5 clk = ~clk;
    end
endmodule
```

5.3 实验三：实现信号生成器模块

5.3.1 signal_generator 模块源代码

```
1
2  module signal_generator(
3      input clk,
4      input rstn,
5      output reg out
6      );
7
8      localparam S0 = 4'b0000;
9      localparam S1 = 4'b0001;
10     localparam S2 = 4'b0010;
11     localparam S3 = 4'b0011;
12     localparam S4 = 4'b0100;
13     localparam S5 = 4'b0101;
14     localparam S6 = 4'b0110;
15     localparam S7 = 4'b0111;
16     localparam S8 = 4'b1000;
17     localparam S9 = 4'b1001;
18     localparam S10 = 4'b1010;
19     localparam S11 = 4'b1011;
20
21     reg [3:0] state, next_state;
22
23     always @(posedge clk or negedge rstn) begin
24       if (~rstn) begin
25         state <= S0;
26       end
27       else begin
28         state <= next_state;
29       end
30     end
31
32     always @(state) begin
33       case(state)
34         S0: begin
35           next_state = S1;
36           out = 1'b0;
37         end
38         S1: begin
39           next_state = S2;
40           out = 1'b0;
41         end
42         S2: begin
43           next_state = S3;
44           out = 1'b1;
45         end
46         S3: begin
47           next_state = S4;
48           out = 1'b0;
```

```
        end
        S4: begin
          next_state = S5;
          out = 1'b1;
        end
        S5: begin
          next_state = S6;
          out = 1'b0;
        end
        S6: begin
          next_state = S7;
          out = 1'b0;
        end
        S7: begin
          next_state = S8;
          out = 1'b1;
        end
        S8: begin
          next_state = S9;
          out = 1'b1;
        end
        S9: begin
          next_state = S10;
          out = 1'b0;
        end
        S10: begin
          next_state = S11;
          out = 1'b1;
        end
        S11: begin
          next_state = S0;
          out = 1'b1;
        end
        endcase
    end
endmodule
```

5.3.2 signal_generator 模块 Testbench

```
module test_signal_generator(

    );
    reg clk;
    reg rstn;
    wire out;

    signal_generator instance_signal_generator(
      .clk(clk),
      .rstn(rstn),
      .out(out)
      );
```

```

    initial begin
      clk = 0;
      rstn = 1;
      #0.1 rstn = 0;
      #0.1 rstn = 1;
    end

    always begin
      #0.5 clk = ~clk;
    end
endmodule
```

5.4 实验四：实现自动报纸贩卖机

5.4.1 newspaper_machine 模块源代码

```

module newspaper_machine(
    input clk,
    input rstn,
    input [2:0] coin,
    output reg dispense
    );

    reg [2:0] state, next_state;
    localparam S0 = 3'b000;
    localparam S1 = 3'b001;
    localparam S2 = 3'b010;
    localparam S3 = 3'b011;
    localparam S4 = 3'b100;
    localparam S5 = 3'b101;

    always @(posedge clk or negedge rstn) begin
      if (~rstn) begin
        state <= S0;
      end else begin
        state <= next_state;
      end
    end

    always @(coin or state) begin
      case (state)
      S0: begin
        if (coin == 3'b001) begin
          next_state = S1;
        end
        else if (coin == 3'b010) begin
          next_state = S2;
        end
        else if (coin == 3'b101) begin
```

```
          next_state = S5;
        end
        else begin
          next_state = S0;
        end
      end
      S1: begin
        if (coin == 3'b001) begin
          next_state = S2;
        end
        else if (coin == 3'b010) begin
          next_state = S3;
        end
        else if (coin == 3'b101) begin
          next_state = S5;
        end
        else begin
          next_state = S1;
        end
      end
      S2: begin
        if (coin == 3'b001) begin
          next_state = S3;
        end
        else if (coin == 3'b010) begin
          next_state = S4;
        end
        else if (coin == 3'b101) begin
          next_state = S5;
        end
        else begin
          next_state = S2;
        end
      end
      S3: begin
        if (coin == 3'b001) begin
          next_state = S4;
        end
        else if (coin == 3'b010) begin
          next_state = S5;
        end
        else if (coin == 3'b101) begin
          next_state = S5;
        end
        else begin
          next_state = S3;
        end
      end
      S4: begin
        if (coin == 3'b001) begin
          next_state = S5;
        end
```

```
        else if (coin == 3'b010) begin
          next_state = S5;
        end
        else if (coin == 3'b101) begin
          next_state = S5;
        end
        else begin
          next_state = S4;
        end
      end
      S5: begin
        next_state = S0;
      end
      endcase
    end
    always @(state) begin
      case(state)
        S0: dispense = 1'b0;
        S1: dispense = 1'b0;
        S2: dispense = 1'b0;
        S3: dispense = 1'b0;
        S4: dispense = 1'b0;
        S5: dispense = 1'b1;
      endcase
    end
endmodule
```

5.4.2 newspaper_machine 模块 Testbench

```
module test_newspaper_machine(

    );
    reg clk;
    reg rstn;
    reg [2:0] coin;
    reg [2:0] random_coin;
    wire dispense;

    newspaper_machine instance_newspaper_machine(
      .clk(clk),
      .rstn(rstn),
      .coin(coin),
      .dispense(dispense)
      );

    initial begin
      clk = 0;
      rstn = 0;
      coin = 3'b000;

      #10;
```

```
      rstn = 1;
//以下是一个特定的测试序列。我在使用随机数测试的时候将这个序列注释掉了。
//       coin = 3'b001;#10;
//       coin = 3'b001;#10;
//       coin = 3'b001;#10;
//       coin = 3'b001;#10;
//       coin = 3'b001;#10;
//       coin = 3'b000;
//
//       rstn = 0;#10;
//       rstn = 1;
//       coin = 3'b001;#10;
//       coin = 3'b010;#10;
//       coin = 3'b010;#10;
//       coin = 3'b000;
//
//       rstn = 0;#10;
//       rstn = 1;
//       coin = 3'b101;#10;
//       coin = 3'b000;
//
//       rstn = 0;#10;
//       rstn = 1;
//       coin = 3'b001;#10;
//       coin = 3'b010;#10;
//       coin = 3'b101;#10;
//       coin = 3'b000;
//
//       rstn = 0;#10;
//       rstn = 1;#10;
    end

    always begin
      #10 random_coin = $random() % 3;
      case (random_coin)
        0: coin = 3'b001;
        1: coin = 3'b010;
        2: coin = 3'b101;
      endcase
    end

    always begin
      #5 clk = ~clk;
    end
endmodule
```