实验名称：___Card___ 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：___Card___ 姓名：__ 学号：

专业：生物医学工程
姓名：
学号：
日期：2025.5.13
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

装 订 线

一、实验要求

玩家每次取一张牌，2~10 的牌取值与面值相同，J，Q，K 面值也是 10，A 代表 1。即简化为每次输入点数为 1—10，出现概率相同，测试文件中使用$random 产生。当总点数小于 18 点就继续叫牌，当总点数超过 21 点时算输，当总点数在[18，21]之间时算胜利。

要求完成：

1、 画出状态机的状态转换图

2、 game_fsm.v

模块要求输入输出信号如下

```
input            clk_i, 时钟输入，100MHz
input            rst_i, 高电平复位信号
input            card_rdy_i,  取牌 ready 信号，该信号为高时读入 card_value_i
input    [3:0]       card_value_i, 牌面值，取值为 1—10
output  reg    card_req_o,  叫牌请求
output  reg    lost_o,      高电平代表游戏输
output  reg [4:0]  count_o,      输出此时点数
output  reg    win_o    高电平代表游戏赢
```

3、 tb_game_fsm.v

实验名称：___Card___ 姓名：__ 学号：

使用 task 产生仿真激励，信号变化与时钟上升沿对齐，且在玩家叫牌时才会产生新的测试输入。

4、 拓展（不做要求）：两个玩家一起玩游戏（例化两个 game_fsm，添加裁判模块）。

游戏过程举例：

(1) 两个玩家准备就绪，裁判宣布开始游戏。

(2) 游戏开始时，双方点数为 0，开始叫牌，裁判把牌轮流分配给两个玩家。

(3) 玩家拿到牌后计算当前点数，根据当前点数选择继续叫牌或停止叫牌。

(4) 当一方停止叫牌，另一方仍然叫牌时，裁判只发牌给叫牌方。

(5) 当两方都停止叫牌后，游戏结束，裁判给出胜负结果。

二、实验代码与注释

装 订 线

（一）一名玩家

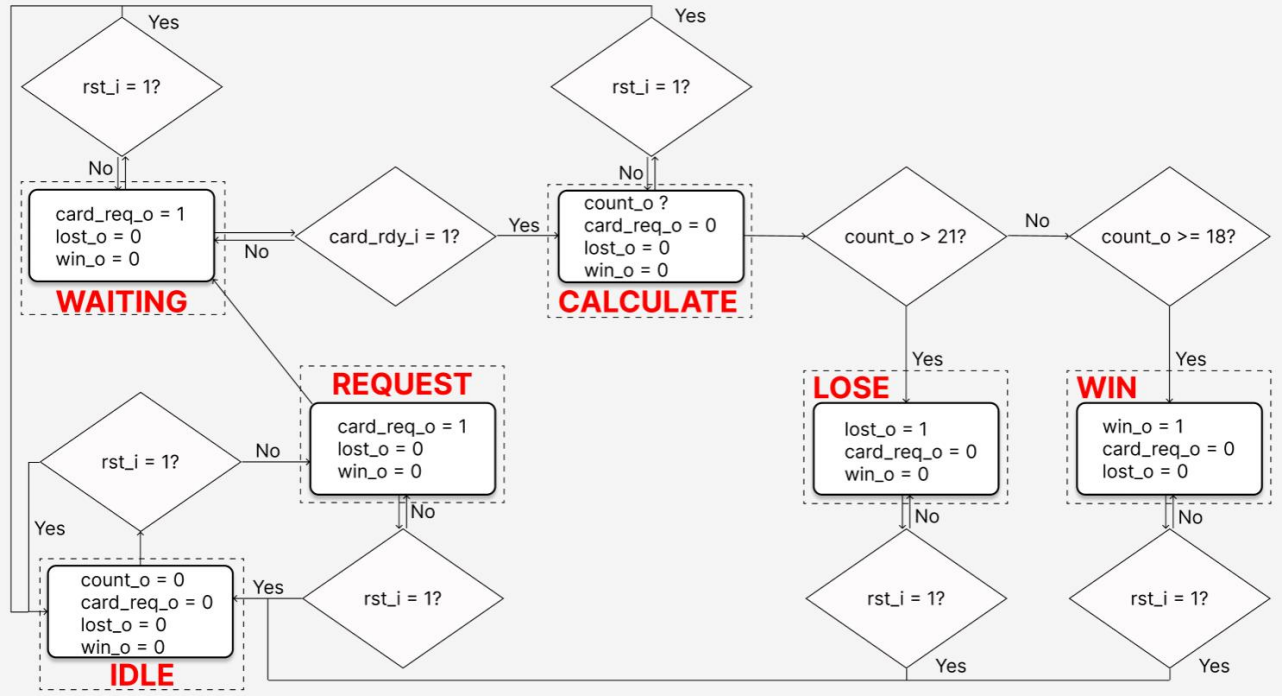


图 1 ASM 状态转移图

1、game_fsm.v

```
01 定义：
02     状态 S = {IDLE, REQUEST_CARD, WAIT_CARD, CALCULATE, WIN, LOSE}
03     输入信号 I = {clk, rst, card_rdy, card_value}
```

实验名称：___Card___ 姓名：__ 学号：

```
    输出信号 O = {card_req, lost, count, win}
    内部寄存器 R = {card_rdy_reg, card_value_reg, current_state,
next_state}

// 状态转换算法（组合逻辑部分）
ALGORITHM StateTransition(current_state)
    // 默认输出
    next_state ← current_state
    card_req ← 0
    lost ← 0
    win ← 0

    CASE current_state OF
        IDLE:
            next_state ← REQUEST_CARD

        REQUEST_CARD:
            card_req ← 1
            next_state ← WAIT_CARD

        WAIT_CARD:
            card_req ← 1  // 保持请求直到收到卡牌
            IF card_rdy_reg = 1 THEN
                next_state ← CALCULATE
            ENDIF

        CALCULATE:
            IF count > 21 THEN
                next_state ← LOSE
            ELSE IF count ≥ 18 AND count ≤ 21 THEN
                next_state ← WIN
            ELSE  // count < 18
                next_state ← REQUEST_CARD
            ENDIF

        WIN:
            win ← 1
            // 保持在 WIN 状态直到复位

        LOSE:
            lost ← 1
            // 保持在 LOSE 状态直到复位

    ENDCASE
```

装 订 线

实验名称：___Card___ 姓名：__ 学号：

```
    RETURN next_state, card_req, lost, win

// 时序逻辑算法
ALGORITHM SequentialLogic(clk, rst)
    ON RISING_EDGE(clk) OR RISING_EDGE(rst):
        IF rst = 1 THEN
            current_state ← IDLE
            count ← 0
            card_rdy_reg ← 0
            card_value_reg ← 无效值
        ELSE
            // 捕获输入信号
            card_rdy_reg ← card_rdy
            IF card_rdy = 1 THEN
                card_value_reg ← card_value
            ELSE
                card_value_reg ← 无效值
            ENDIF

            // 更新状态
            current_state ← next_state

            // 计分逻辑
            IF current_state = WAIT_CARD AND card_rdy_reg = 1 THEN
                IF card_value_reg ≥ 1 AND card_value_reg ≤ 10 THEN
                    count ← count + card_value_reg
                ENDIF
            ENDIF
        ENDIF
```

```
module game_fsm (
    input               clk_i,
    input               rst_i,
    input               card_rdy_i,
    input      [3:0]    card_value_i,
    output reg          card_req_o,
    output reg          lost_o,
    output reg [4:0]    count_o,
    output reg          win_o
);

// State definition
localparam IDLE         = 3'b000;
```

装 订 线

实验名称：___Card___ 姓名：__ 学号：

装

订

线

```
localparam REQUEST_CARD = 3'b001;
localparam WAIT_CARD    = 3'b010;
localparam CALCULATE    = 3'b011;
localparam WIN          = 3'b100;
localparam LOSE         = 3'b101;
reg [2:0] current_state;
reg [2:0] next_state;
// Internal signals for registered inputs
reg card_rdy_reg;       // Registered version of card_rdy_i
reg [3:0] card_value_reg; // Registered version of card_value_i, captured
when card_rdy_i is high
always @(*) begin
    next_state = current_state;
    card_req_o = 1'b0;
    lost_o = 1'b0;
    win_o = 1'b0;
    case (current_state)
        IDLE: begin
            next_state = REQUEST_CARD;
        end
        REQUEST_CARD: begin
            card_req_o = 1'b1;
            next_state = WAIT_CARD;
        end
        WAIT_CARD: begin
            card_req_o = 1'b1; // Keep requesting until card is ready
            if (card_rdy_reg) begin // Use registered card_rdy_reg
                next_state = CALCULATE;
            end else begin
                next_state = WAIT_CARD;
            end
        end
        CALCULATE: begin
            // card_req_o is 0 by default
            if (count_o > 5'd21) begin
                next_state = LOSE;
            end else if (count_o >= 5'd18 && count_o <= 5'd21) begin
                next_state = WIN;
            end else if (count_o < 5'd18) begin
                next_state = REQUEST_CARD;
            end else begin // Should not happen given the conditions above
                next_state = IDLE; // Safety default
            end
        end
```

实验名称：___Card___ 姓名：__ 学号：

装

订

线

```
        WIN: begin
            win_o = 1'b1;
            next_state = WIN; // Stay in WIN until reset
        end
        LOSE: begin
            lost_o = 1'b1;
            next_state = LOSE; // Stay in LOSE until reset
        end
        default: begin
            next_state = IDLE;
        end
    endcase
end
// Sequential logic for state transition and count update
always @(posedge clk_i or posedge rst_i) begin
    if (rst_i) begin
        current_state <= IDLE;
        count_o <= 5'd0;
        card_rdy_reg <= 1'b0;      // Reset registered input
        card_value_reg <= 4'bz;  // Reset registered input to an invalid state

    end else begin
        // Capture inputs first
        card_rdy_reg <= card_rdy_i;
        if (card_rdy_i) begin
            card_value_reg <= card_value_i;
        end else begin
            card_value_reg <= 4'bz; // Keep it invalid if card_rdy_i is low
        end
        current_state <= next_state; // current_state updates based on
next_state from previous cycle's inputs (via _reg)
        $display("FSM_SEQ @%t: clk=%b, rst=%b, current_state_REG=%h,
next_state_COMB=%h, card_rdy_i(input)=%b, card_value_i(input)=%d,
card_rdy_reg(old)=%b, card_value_reg(old)=%d, count_o_REG_OLD=%d",
                 $time, clk_i, rst_i, current_state, next_state, card_rdy_i,
card_value_i, card_rdy_reg, card_value_reg, count_o);
        // Accumulate count based on registered inputs from the previous cycle
        if (current_state == WAIT_CARD && card_rdy_reg) begin
            if (card_value_reg > 0 && card_value_reg <= 10) begin
                 count_o <= count_o + card_value_reg;
                 $display("FSM_SEQ @%t: ACCUMULATING! count_o <= %d + %d
(using card_value_reg). New count will be %d", $time, count_o, card_value_reg,
count_o + card_value_reg);
            end else begin
```

实验名称：___Card___ 姓名：__ 学号：

```
                $display("FSM_SEQ @%t: card_value_reg invalid (%d) or
card_rdy_reg low, not accumulating.", $time, card_value_reg);
            end
        end else if (next_state == IDLE && current_state != IDLE) begin
            // Reset count if moving to IDLE
            count_o <= 5'd0;
            $display("FSM_SEQ @%t: RESETTING count_o to 0 (transition to IDLE
from %h)", $time, current_state);
        end
        // WIN/LOSE states: count_o holds its value, handled by no assignment
here unless resetting.
    end
end
endmodule
```

2、tb_game_fsm.v

```
ALGORITHM TestBenchGameFSM()
    // 初始化
    clk ← 0
    rst ← 1
    card_rdy ← 0
    card_value ← 无效值

    // 设置随机数种子
    seed ← 设定值或当前时间

    // 生成时钟
    PARALLEL PROCESS ClockGeneration():
        FOREVER:
            WAIT(5ns)
            clk ← NOT clk

    // 提供卡牌任务
    PROCEDURE ProvideCard():
        WAIT_FOR(RISING_EDGE(clk))
        IF card_req = 1 THEN
            random_val ← RANDOM() MOD 10 + 1  // 生成 1-10 范围的随机数
            card_value ← random_val
            card_rdy ← 1

            WAIT_FOR(RISING_EDGE(clk))
            card_rdy ← 0
```

实验名称：___Card___ 姓名：__ 学号：

```
            card_value ← 无效值

            PRINT("提供卡牌值:", random_val)
        ENDIF

    // 主测试序列
    MAIN:
        // 施加复位
        WAIT(20ns)
        rst ← 0

        // 游戏循环
        FOR i = 1 TO MAX_ITERATIONS DO
            IF win = 1 OR lost = 1 THEN
                IF win = 1 THEN PRINT("游戏获胜! 最终点数:", count)
                IF lost = 1 THEN PRINT("游戏失败! 最终点数:", count)

                // 重置游戏
                rst ← 1
                WAIT(20ns)
                rst ← 0
            ENDIF

            IF card_req = 1 THEN
                ProvideCard()
            ELSE
                WAIT_FOR(RISING_EDGE(clk))
            ENDIF
        ENDFOR

        PRINT("仿真结束")
        FINISH()
```

装 订 线

```
`timescale 1ns / 1ps

module tb_game_fsm;
// Inputs
reg clk_i;
reg rst_i;
reg card_rdy_i;
reg [3:0] card_value_i;
reg [31:0] temp_rand;     // Temporary variable for $random
// Outputs
```

实验名称：___Card___ 姓名：__ 学号：

装

订

线

```
wire card_req_o;
wire lost_o;
wire [4:0] count_o;
wire win_o;
game_fsm uut (
    .clk_i(clk_i),
    .rst_i(rst_i),
    .card_rdy_i(card_rdy_i),
    .card_value_i(card_value_i),
    .card_req_o(card_req_o),
    .lost_o(lost_o),
    .count_o(count_o),
    .win_o(win_o)
);
// Seed for random number generator
initial begin
    temp_rand = 12345; // Assign seed value to variable
    temp_rand = $time;
    temp_rand = $random(temp_rand); // Set seed properly
end
// Clock generation
initial begin
    clk_i = 0;
    forever #5 clk_i = ~clk_i; // 100MHz clock
end
// Task to provide a card
task provide_card;
    reg [3:0] provided_value; // To store the value for display
    begin
        // P0: card_req_o observed high by testbench.
        // FSM state could be REQUEST_CARD or WAIT_CARD (if it looped back).
        @(posedge clk_i); // Sync to P1. FSM updates its state based on P0
values.
                          // If FSM was REQUEST_CARD, it transitions to
WAIT_CARD here.
                          // If FSM was WAIT_CARD, it stays WAIT_CARD (as
card_rdy_i was low at P0 edge).
        if (card_req_o) begin // card_req_o should still be high if FSM is
in WAIT_CARD.
            temp_rand = $random;
            provided_value = (temp_rand % 10);
            if (provided_value < 0) provided_value = provided_value + 10; //
Ensure 0-9 if % is negative
            provided_value = provided_value + 1; // Shift to 1-10 range
```

实验名称：___Card___ 姓名：__ 学号：

装

订

线

```
            card_value_i = provided_value;
            card_rdy_i = 1'b1;
            // card_value_i and card_rdy_i are now set. FSM will sample these
at the P2 edge.
            @(posedge clk_i); // Sync to P2.
                              // At this P2 edge, FSM (which was in WAIT_CARD
at P1 start)
                              // samples card_value_i and card_rdy_i (which
is 1'b1).
                              // It should update count_o and transition to
CALCULATE.
            card_rdy_i = 1'b0; // De-assert card_rdy_i *after* FSM has used
it at P2 and after the #1 delay.
            card_value_i = 4'bz;
            $display("Provided card: %d at time %t. FSM state (after P2
processing): %h. New count: %d",
                     provided_value, $time, uut.current_state, count_o);
        end else begin
            $display("TB: At %t (P1 edge), FSM card_req_o went low
unexpectedly. No card provided.", $time);
        end
    end
endtask
// Test sequence
initial begin
    // Initialize Inputs
    rst_i = 1'b1;
    card_rdy_i = 1'b0;
    card_value_i = 4'bz;
    // Apply reset
    #20;
    rst_i = 1'b0;
    $display("Reset released at time %t", $time);
    // Game simulation loop
    repeat (70) begin // Max 70 cards, or until game ends
        if (win_o || lost_o) begin
            if (win_o) $display("Game WIN! Final count: %d at time %t",
count_o, $time);
            if (lost_o) $display("Game LOST! Final count: %d at time %t",
count_o, $time);

            // Reset for a new game
            rst_i = 1'b1;
```

实验名称：___Card___ 姓名：__ 学号：

```
            card_rdy_i = 1'b0;
            card_value_i = 4'bz;
            #20;
            rst_i = 1'b0;
            $display("New game started after reset at time %t", $time);
        end

        if (card_req_o) begin
            provide_card();
        end else begin
            @(posedge clk_i); // If not requesting, just wait for next cycle
        end
    end
    $display("Simulation finished after multiple attempts or timeout at
time %t", $time);
    $finish;
end
// Monitor changes
initial begin
    $monitor("Time=%t, clk=%b, rst=%b, card_req=%b, card_rdy=%b,
card_val=%d, count=%d, win=%b, lost=%b, state=%h",
             $time, clk_i, rst_i, card_req_o, card_rdy_i, card_value_i,
count_o, win_o, lost_o, uut.current_state);
    $dumpfile("tb_game_fsm.vcd");
    $dumpvars(0, tb_game_fsm);
    #20000 $finish; // Increased simulation time, e.g., to 2000ns
end
endmodule
```

（二）二名玩家

1、referee.v

```
module referee (
    input               clk_i,
    input               rst_i,

    // 玩家1接口
    input               p1_card_req_i,
    output reg          p1_card_rdy_o,
    output reg [3:0]    p1_card_value_o,
    input               p1_win_i,
    input               p1_lost_i,
    input      [4:0]    p1_count_i,
```

实验名称：___Card___ 姓名：__ 学号：

```
    // 玩家2 接口
    input               p2_card_req_i,
    output reg          p2_card_rdy_o,
    output reg [3:0]    p2_card_value_o,
    input               p2_win_i,
    input               p2_lost_i,
    input      [4:0]    p2_count_i,

    // 游戏结果输出
    output reg [1:0]    game_result_o  // 00: 未结束, 01: 玩家1 胜, 10: 玩家
2 胜, 11: 平局
);

// 状态定义
localparam INIT         = 3'b000;  // 初始状态
localparam PLAYER1_TURN = 3'b001;  // 玩家1 回合
localparam PLAYER2_TURN = 3'b010;  // 玩家2 回合
localparam DEAL_CARD    = 3'b011;  // 发牌状态
localparam JUDGE_RESULT = 3'b100;  // 判定结果
reg [2:0] current_state;
reg [2:0] next_state;
reg [3:0] card_to_deal;    // 即将发出的牌值
reg       turn_flag;       // 0: 玩家1 回合, 1: 玩家2 回合
// 随机数生成
reg [31:0] rand_temp;
always @(posedge clk_i) begin
    if (current_state == DEAL_CARD) begin
        rand_temp = $random;
        card_to_deal <= (rand_temp % 10) + 1;
        if (card_to_deal > 10 || card_to_deal < 1) begin
            card_to_deal <= (rand_temp[3:0] % 10) + 1; // 备选随机方案
        end
    end
end
// 组合逻辑部分: 状态转换和输出
always @(*) begin
    // 默认值
    next_state = current_state;
    p1_card_rdy_o = 1'b0;
    p2_card_rdy_o = 1'b0;
    p1_card_value_o = 4'b0;
    p2_card_value_o = 4'b0;
```

装 订 线

实验名称：___Card___ 姓名：__ 学号：

```
    case (current_state)
        INIT: begin
            // 初始化后转到玩家 1 回合
            next_state = PLAYER1_TURN;
        end

        PLAYER1_TURN: begin
            if (p1_card_req_i && !p1_win_i && !p1_lost_i) begin
                // 玩家 1 请求牌，转到发牌状态
                next_state = DEAL_CARD;
            end else if (!p1_card_req_i || p1_win_i || p1_lost_i) begin
                // 玩家 1 不要牌或已经完成游戏
                next_state = PLAYER2_TURN;
            end
        end

        PLAYER2_TURN: begin
            if (p2_card_req_i && !p2_win_i && !p2_lost_i) begin
                // 玩家 2 请求牌，转到发牌状态
                next_state = DEAL_CARD;
            end else if (!p2_card_req_i || p2_win_i || p2_lost_i) begin
                // 检查是否需要回到玩家 1
                if (p1_card_req_i && !p1_win_i && !p1_lost_i) begin
                    next_state = PLAYER1_TURN;
                end else if ((!p1_card_req_i || p1_win_i || p1_lost_i) &&
                             (!p2_card_req_i || p2_win_i || p2_lost_i))
begin
                    // 两位玩家都不再请求牌或已经完成游戏
                    next_state = JUDGE_RESULT;
                end else begin
                    // 保持在玩家 2 回合
                    next_state = PLAYER2_TURN;
                end
            end
        end

        DEAL_CARD: begin
            if (turn_flag == 1'b0) begin
                // 发牌给玩家 1
                p1_card_rdy_o = 1'b1;
                p1_card_value_o = card_to_deal;
                next_state = PLAYER2_TURN; // 发完牌后转到玩家 2 回合
            end else begin
                // 发牌给玩家 2
```

装 订 线

实验名称：___Card___ 姓名：__ 学号：

```
                p2_card_rdy_o = 1'b1;
                p2_card_value_o = card_to_deal;
                next_state = PLAYER1_TURN; // 发完牌后转到玩家1回合
            end
        end

        JUDGE_RESULT: begin
            // 保持在当前状态，等待复位
            next_state = JUDGE_RESULT;
        end

        default: begin
            next_state = INIT;
        end
    endcase
end
// 时序逻辑部分：状态更新和输出寄存器
always @(posedge clk_i or posedge rst_i) begin
    if (rst_i) begin
        current_state <= INIT;
        turn_flag <= 1'b0; // 初始为玩家1回合
        game_result_o <= 2'b00;
    end else begin
        current_state <= next_state;

        // 更新回合标志
        if (current_state == DEAL_CARD) begin
            turn_flag <= ~turn_flag;
        end

        // 判定游戏结果
        if (current_state == JUDGE_RESULT) begin
            if (p1_lost_i && !p2_lost_i) begin
                // 玩家1爆牌，玩家2未爆牌，玩家2胜
                game_result_o <= 2'b10;
            end else if (!p1_lost_i && p2_lost_i) begin
                // 玩家1未爆牌，玩家2爆牌，玩家1胜
                game_result_o <= 2'b01;
            end else if (p1_lost_i && p2_lost_i) begin
                // 两人都爆牌，平局
                game_result_o <= 2'b11;
            end else begin
                // 两人都未爆牌，比点数
                if (p1_count_i > p2_count_i) begin
```

装 订 线

实验名称：___Card___ 姓名：__ 学号：

```
                    game_result_o <= 2'b01; // 玩家1胜
                end else if (p1_count_i < p2_count_i) begin
                    game_result_o <= 2'b10; // 玩家2胜
                end else begin
                    game_result_o <= 2'b11; // 平局
                end
            end
        end
    end
end
endmodule
```

2、two_player_game.v

```
module referee (
    input               clk_i,
    input               rst_i,

    // 玩家1接口
    input               p1_card_req_i,
    output reg          p1_card_rdy_o,
    output reg [3:0]    p1_card_value_o,
    input               p1_win_i,
    input               p1_lost_i,
    input      [4:0]    p1_count_i,

    // 玩家2接口
    input               p2_card_req_i,
    output reg          p2_card_rdy_o,
    output reg [3:0]    p2_card_value_o,
    input               p2_win_i,
    input               p2_lost_i,
    input      [4:0]    p2_count_i,

    // 游戏结果输出
    output reg [1:0]    game_result_o  // 00: 未结束, 01: 玩家1胜, 10: 玩家2胜, 11: 平局
);

// 状态定义
localparam INIT         = 3'b000;  // 初始状态
localparam PLAYER1_TURN = 3'b001;  // 玩家1回合
localparam PLAYER2_TURN = 3'b010;  // 玩家2回合
localparam DEAL_CARD    = 3'b011;  // 发牌状态
```

装 订 线

实验名称：___Card___ 姓名：__ 学号：

```
localparam JUDGE_RESULT = 3'b100;  // 判定结果
reg [2:0] current_state;
reg [2:0] next_state;
reg [3:0] card_to_deal;    // 即将发出的牌值
reg       turn_flag;       // 0: 玩家1 回合, 1: 玩家2 回合
// 随机数生成
reg [31:0] rand_temp;
always @(posedge clk_i) begin
    if (current_state == DEAL_CARD) begin
        rand_temp = $random;
        card_to_deal <= (rand_temp % 10) + 1;
        if (card_to_deal > 10 || card_to_deal < 1) begin
            card_to_deal <= (rand_temp[3:0] % 10) + 1; // 备选随机方案
        end
    end
end
// 组合逻辑部分：状态转换和输出
always @(*) begin
    // 默认值
    next_state = current_state;
    p1_card_rdy_o = 1'b0;
    p2_card_rdy_o = 1'b0;
    p1_card_value_o = 4'b0;
    p2_card_value_o = 4'b0;

    case (current_state)
        INIT: begin
            // 初始化后转到玩家1 回合
            next_state = PLAYER1_TURN;
        end

        PLAYER1_TURN: begin
            if (p1_card_req_i && !p1_win_i && !p1_lost_i) begin
                // 玩家1 请求牌，转到发牌状态
                next_state = DEAL_CARD;
            end else if (!p1_card_req_i || p1_win_i || p1_lost_i) begin
                // 玩家1 不要牌或已经完成游戏
                next_state = PLAYER2_TURN;
            end
        end

        PLAYER2_TURN: begin
            if (p2_card_req_i && !p2_win_i && !p2_lost_i) begin
                // 玩家2 请求牌，转到发牌状态
```

装 订 线

实验名称：___Card___ 姓名：__ 学号：

```
                next_state = DEAL_CARD;
            end else if (!p2_card_req_i || p2_win_i || p2_lost_i) begin
                // 检查是否需要回到玩家1
                if (p1_card_req_i && !p1_win_i && !p1_lost_i) begin
                    next_state = PLAYER1_TURN;
                end else if ((!p1_card_req_i || p1_win_i || p1_lost_i) &&
                             (!p2_card_req_i || p2_win_i || p2_lost_i))
begin
                    // 两位玩家都不再请求牌或已经完成游戏
                    next_state = JUDGE_RESULT;
                end else begin
                    // 保持在玩家2回合
                    next_state = PLAYER2_TURN;
                end
            end
        end

        DEAL_CARD: begin
            if (turn_flag == 1'b0) begin
                // 发牌给玩家1
                p1_card_rdy_o = 1'b1;
                p1_card_value_o = card_to_deal;
                next_state = PLAYER2_TURN; // 发完牌后转到玩家2回合
            end else begin
                // 发牌给玩家2
                p2_card_rdy_o = 1'b1;
                p2_card_value_o = card_to_deal;
                next_state = PLAYER1_TURN; // 发完牌后转到玩家1回合
            end
        end

        JUDGE_RESULT: begin
            // 保持在当前状态，等待复位
            next_state = JUDGE_RESULT;
        end

        default: begin
            next_state = INIT;
        end
    endcase
end
// 时序逻辑部分：状态更新和输出寄存器
always @(posedge clk_i or posedge rst_i) begin
    if (rst_i) begin
```

装 订 线

实验名称：___Card___ 姓名：__ 学号：

```
        current_state <= INIT;
        turn_flag <= 1'b0; // 初始为玩家1回合
        game_result_o <= 2'b00;
    end else begin
        current_state <= next_state;

        // 更新回合标志
        if (current_state == DEAL_CARD) begin
            turn_flag <= ~turn_flag;
        end

        // 判定游戏结果
        if (current_state == JUDGE_RESULT) begin
            if (p1_lost_i && !p2_lost_i) begin
                // 玩家1爆牌，玩家2未爆牌，玩家2胜
                game_result_o <= 2'b10;
            end else if (!p1_lost_i && p2_lost_i) begin
                // 玩家1未爆牌，玩家2爆牌，玩家1胜
                game_result_o <= 2'b01;
            end else if (p1_lost_i && p2_lost_i) begin
                // 两人都爆牌，平局
                game_result_o <= 2'b11;
            end else begin
                // 两人都未爆牌，比点数
                if (p1_count_i > p2_count_i) begin
                    game_result_o <= 2'b01; // 玩家1胜
                end else if (p1_count_i < p2_count_i) begin
                    game_result_o <= 2'b10; // 玩家2胜
                end else begin
                    game_result_o <= 2'b11; // 平局
                end
            end
        end
    end
end
endmodule
```

3、tb_two_player_game.v

```
`timescale 1ns / 1ps

module tb_two_player_game;
// 输入信号
reg clk_i;
```

装 订 线

实验名称：___Card___ 姓名：__ 学号：

装

订

线

```
reg rst_i;
// 输出信号
wire [1:0] game_result;
wire [4:0] p1_count;
wire [4:0] p2_count;
wire p1_win;
wire p1_lost;
wire p2_win;
wire p2_lost;
// 统计多局游戏的结果
integer total_games;
integer p1_wins;
integer p2_wins;
integer draws;
integer game_timer;
integer is_timeout;
// 例化待测模块
two_player_game uut (
    .clk_i(clk_i),
    .rst_i(rst_i),
    .game_result_o(game_result),
    .p1_count_o(p1_count),
    .p2_count_o(p2_count),
    .p1_win_o(p1_win),
    .p1_lost_o(p1_lost),
    .p2_win_o(p2_win),
    .p2_lost_o(p2_lost)
);
// 生成时钟信号
initial begin
    clk_i = 0;
    forever #5 clk_i = ~clk_i; // 100MHz 时钟
end
// 测试序列
initial begin
    // 初始化统计数据
    total_games = 0;
    p1_wins = 0;
    p2_wins = 0;
    draws = 0;
    game_timer = 0;

    // 运行多局游戏
    repeat (5) begin  // 设置运行 5 局游戏
```

实验名称：___Card___ 姓名：__ 学号：

```
        // 初始化
        rst_i = 1;
        is_timeout = 0;

        // 复位释放
        #20 rst_i = 0;
        $display("第%0d 局游戏开始，复位释放于时间 %t", total_games+1, $time);

        // 简单的超时检测
        game_timer = 0;
        while (game_result == 2'b00 && game_timer < 200) begin
            #50 game_timer = game_timer + 1;
        end

        // 检查是否超时
        if (game_result == 2'b00) begin
            $display("第%0d 局游戏超时，强制结束", total_games+1);
            is_timeout = 1;
        end else begin
            // 记录游戏结果
            total_games = total_games + 1;

            if (game_result == 2'b01) p1_wins = p1_wins + 1;
            else if (game_result == 2'b10) p2_wins = p2_wins + 1;
            else if (game_result == 2'b11) draws = draws + 1;

            // 显示本局结果
            $display("第%0d 局游戏结束于时间 %t", total_games, $time);

            if (game_result == 2'b01) $display("本局结果: 玩家 1 获胜");
            else if (game_result == 2'b10) $display("本局结果: 玩家 2 获胜");
            else if (game_result == 2'b11) $display("本局结果: 平局");

            $display("玩家 1: 点数 = %d, 赢 = %b, 输 = %b", p1_count, p1_win,
p1_lost);
            $display("玩家 2: 点数 = %d, 赢 = %b, 输 = %b", p2_count, p2_win,
p2_lost);
            $display("-----------------------------------------------");
        end

        // 在进入下一局之前等待一段时间
        #100;
    end
```

装 订 线

实验名称：___Card___ 姓名：__ 学号：

```
    // 显示总体结果
    $display("\n========== 游戏统计 ==========");
    $display("总局数: %0d", total_games);
    $display("玩家 1 胜场: %0d", p1_wins);
    $display("玩家 2 胜场: %0d", p2_wins);
    $display("平局: %0d", draws);
    $display("============================\n");

    #100 $finish;
end
// 监视信号变化
initial begin
    $monitor("时间=%t, 复位=%b, 结果=%b, 玩家 1(点数=%d,赢=%b,输=%b), 玩家
2(点数=%d,赢=%b,输=%b)",
        $time, rst_i, game_result, p1_count, p1_win, p1_lost, p2_count,
p2_win, p2_lost);

    $dumpfile("tb_two_player_game.vcd");
    $dumpvars(0, tb_two_player_game);
end
endmodule
```

装 订 线

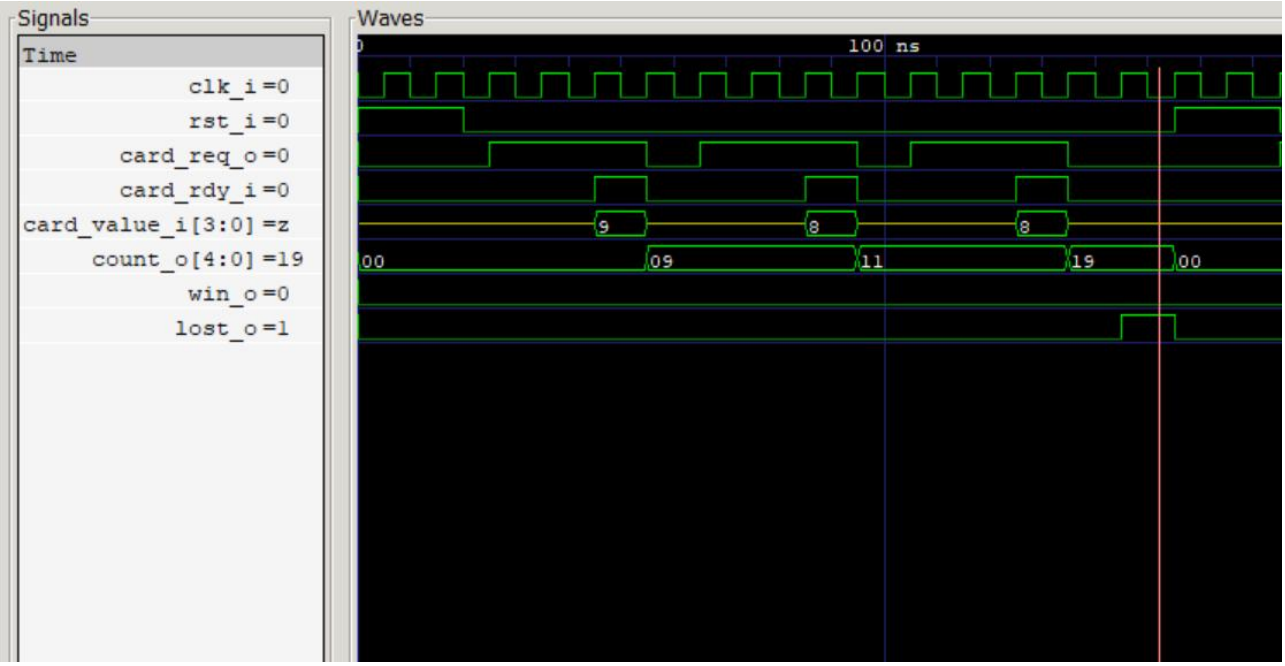
三、仿真结果与分析

Gtkwave 的 count_o 为十六进制，【11】（hex）实际上是【17】（dec）。

（一）单名玩家

实验名称：___Card___ 姓名：__ 学号：

1、第一轮

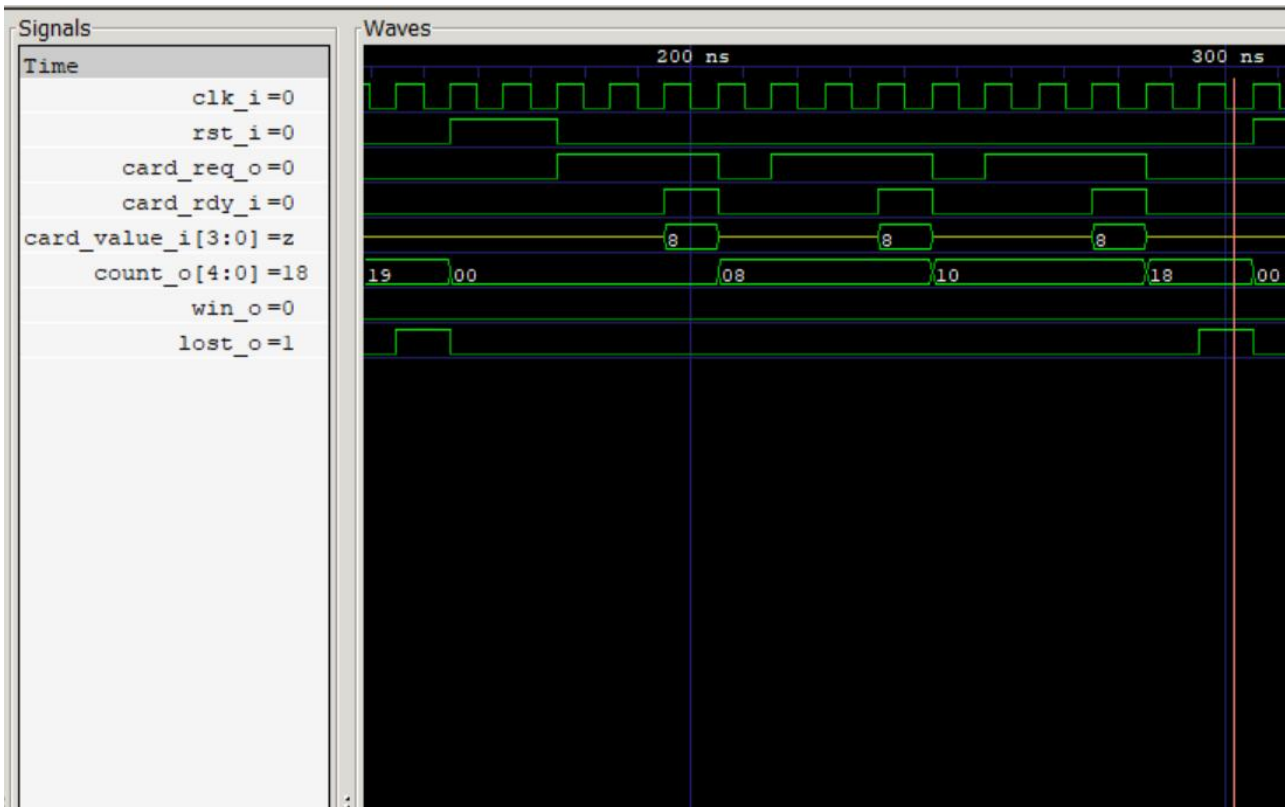


可见 card_value_i 随机发牌 3 次，分别为 9、8、8，累积得到的 count_o 为 25>21，因此在 count_o 判断结束后的下一个上升沿将其置 1。经过一个时钟周期后 rst 置位，开始下一轮游戏。

装 订 线

实验名称：___Card___ 姓名：__ 学号：

2、第二轮



第二轮三次发牌累积得 24>21，依然判负。

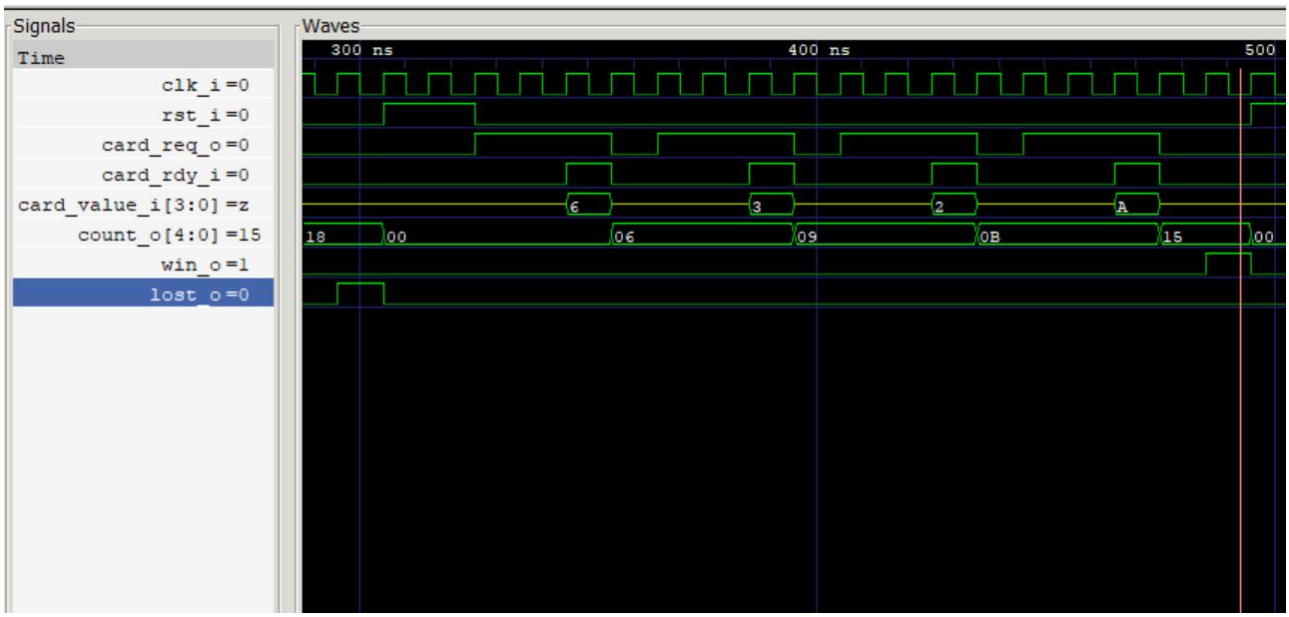
装

订

线

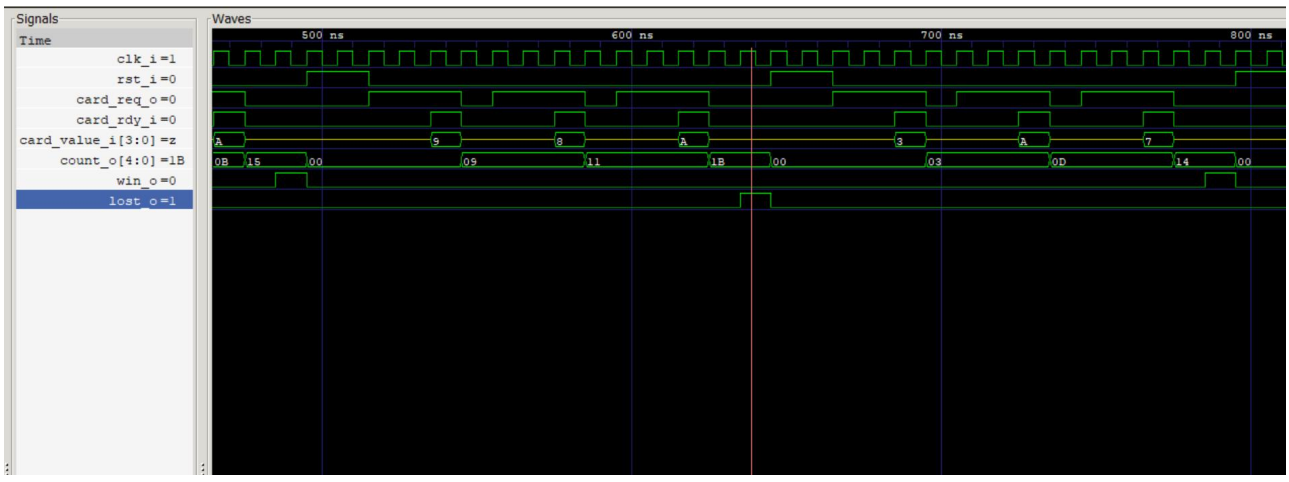
实验名称：___Card___ 姓名：__ 学号：

3、第三轮



这一轮随机发了 4 次牌才得到最终结果 21∈[18,21]，因此举起 win_o。

4、后续二轮



这两轮一次是 27，一次是 20。最后状态转换达到主循环设定 repeat 的上限 70，仿真结束。

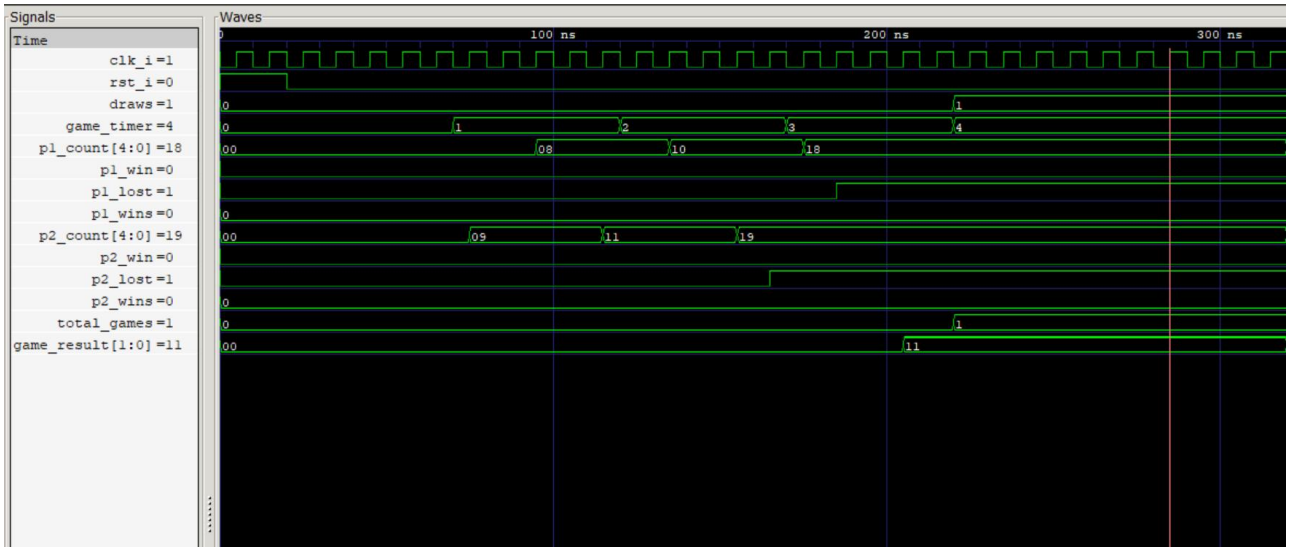
（二）二名玩家

此处使用 iverilog -g2001 -o two_player_game_tb.vvp edge/tb_two_player_game.v edge/two_player_game.v edge/referee.v game_fsm.v 来进行编译，也就是将单玩家状态机

实验名称：___Card___ 姓名：__ 学号：

复用两次并添加裁判模块协调。

1、第一轮



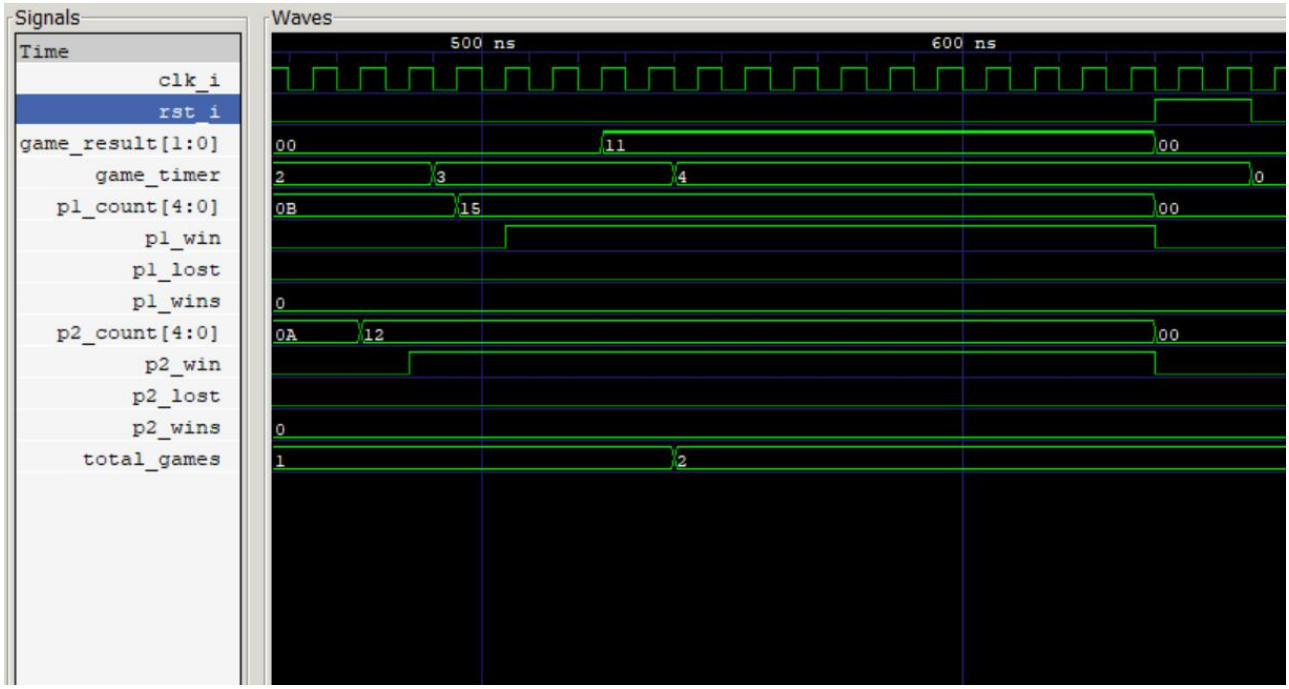
game_timer 用于计时计数，最开始是打算作为超时和发牌数的计时器使用。

玩家 2（p2）**先**获得牌：9 → 17 → 25（爆牌）

玩家 1（p1）**后**获得牌：8 → 16 → 24（爆牌）

两位玩家都爆牌了，分别置位各自的 lost，游戏最终结果 game_result 向量设为 11（二进制），表示平局。

2、第二轮



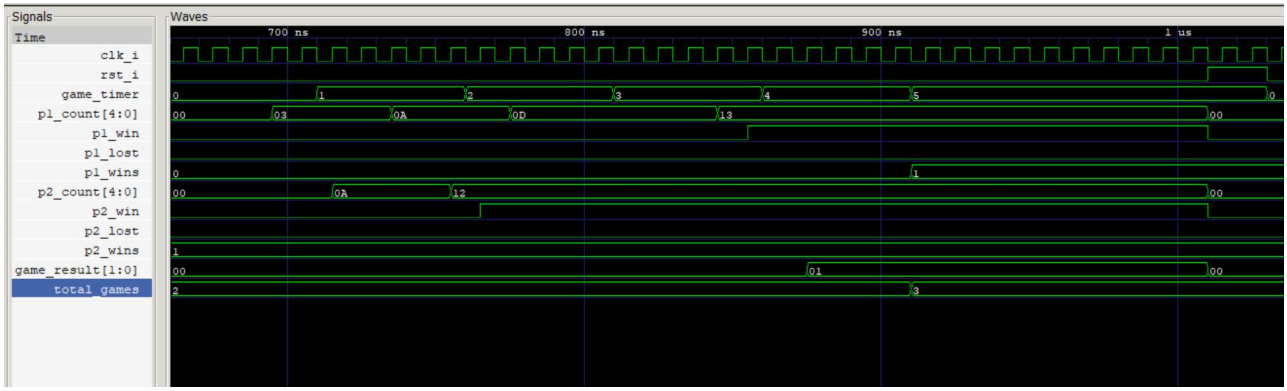
实验名称：___Card___ 姓名：__ 学号：

玩家 2：10 → 18

玩家 1：2 → 11 → 21

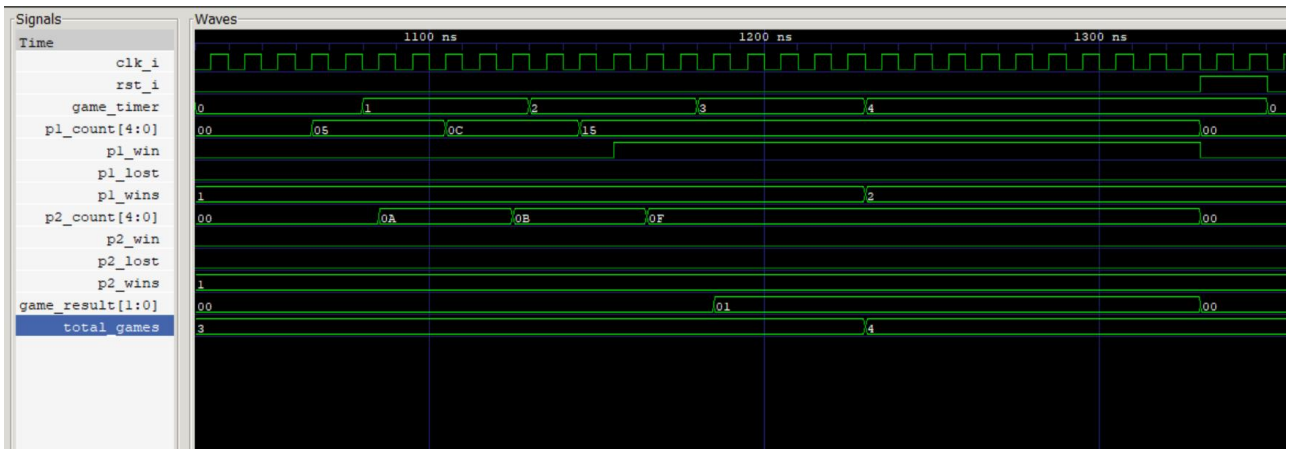
是个 win-win 局面，所以 referee 还是判平局。同时 total_games++自增。

3、第三轮



本次是玩家 1 爆牌，因此玩家 2 胜利，game_result<=01。

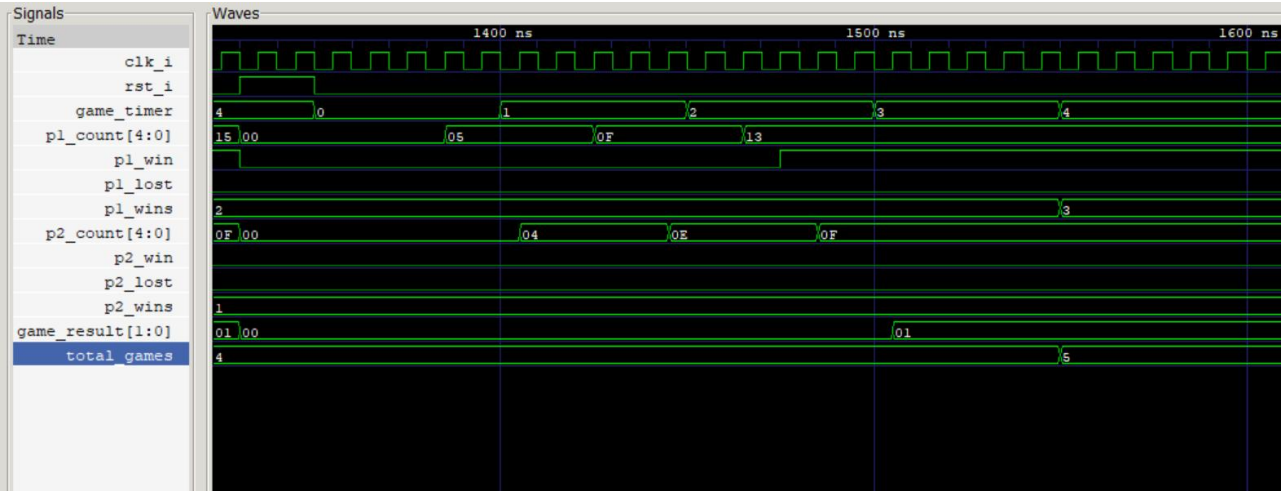
4、第四轮



5、第五轮

装 订 线

实验名称：___Card___ 姓名：__ 学号：



四、讨论与心得

我在 5.13 下午的课才了解到余锋老师更倾向于 ASM（Algorithmic State Machine）的状态转换图，所以吸取上一次实验使用 figma 的经验，修改了此次的 FSM 图。

总体而言，这次实验不仅巩固了我对有限状态机设计的理解，还提高了模块化设计能力和综合应用能力。通过扩展实现双玩家系统，我对复杂数字系统的分析和构建有了更全面的认识。通过对比单玩家和双玩家系统的不同，我也理解了系统复杂度增加时如何合理抽象和分解问题，裁判模块的设计让我学会了如何协调多个独立模块的工作。

装 订 线