デジタル【問】レベル2

VerilogHDLでステート・マシンを設計してシミュレーションで確認すると仕様どおりに動作していない（ステート2からステート0へ状態が遷移しない現象）。RTL、テストベンチ、シミュレーション結果、状態遷移図を参考にして、原因を選びなさい。

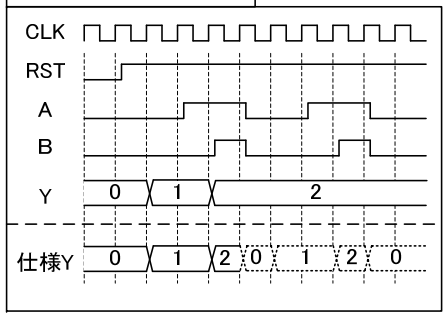
RTL

```
module state(RST, CLK, A, B, Y) ;
input             RST, CLK, A, B ;
output [1:0]      Y ;

parameter STATE0 = 2'b00 ;
parameter STATE1 = 2'b01 ;
parameter STATE2 = 2'b10 ;
reg [1:0] now_state;
reg [1:0] next_state;

    always @(posedge CLK or negedge RST)
    if(!RST)
        now_state <= STATE0 ;
    else
        now_state <= next_state ;

    always @(now_state, A)
        case (now_state)
            STATE0 : if (!A) next_state <= STATE1 ;
                     else    next_state <= STATE0 ;
            STATE1 : if (A)  next_state <= STATE2 ;
                     else    next_state <= STATE1 ;
            STATE2 : if (B)  next_state <= STATE0 ;
                     else    next_state <= STATE2 ;
            default :        next_state <= now_state ;
        endcase

    assign Y = now_state ;

endmodule
```

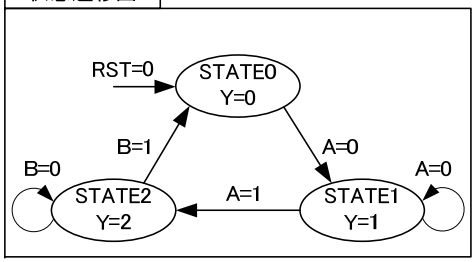
テストベンチ

```
`timescale 1 ns / 1 ns

module test_test ;
    parameter CYCLE = 100 ;
    parameter DLY   = 1 ;
    reg             rst_sig, clk_sig, a_sig, b_sig ;
    wire [1:0]      y_sig ;
    integer i ;

    state Ustate(.RST(rst_sig), .CLK(clk_sig),
                 .A(a_sig), .B(b_sig), .Y(y_sig));

    always begin
                 clk_sig = 1'b0 ;
        #(CYCLE) clk_sig = 1'b1 ;
        #(CYCLE) ;
    end

    initial begin
        rst_sig <= 1'b0 ;
        a_sig   <= 1'b0 ;
        b_sig   <= 1'b0 ;
        #DLY ;
        @(posedge clk_sig) #DLY rst_sig <= 1'b1 ;
        @(posedge clk_sig) #DLY a_sig <= 1'b0 ;
        @(posedge clk_sig) #DLY a_sig <= 1'b1 ;
        @(posedge clk_sig) #DLY b_sig <= 1'b1 ;
        @(posedge clk_sig) #DLY a_sig <= 1'b0 ; b_sig <= 1'b0 ;
        @(posedge clk_sig) ;
        @(posedge clk_sig) ;
        @(posedge clk_sig) #DLY a_sig <= 1'b1 ;
        @(posedge clk_sig) #DLY b_sig <= 1'b1 ;
        @(posedge clk_sig) #DLY a_sig <= 1'b0 ; b_sig <= 1'b0 ;
        @(posedge clk_sig) ;
        @(posedge clk_sig) ;
        #DLY ;
        $stop ;
    end
endmodule
```

シミュレーション結果



状態遷移図



ア　RTLのステート・マシン記述で、ステート2からステート0への移行条件が記述されていないため、ステート・マシンが動作しない。

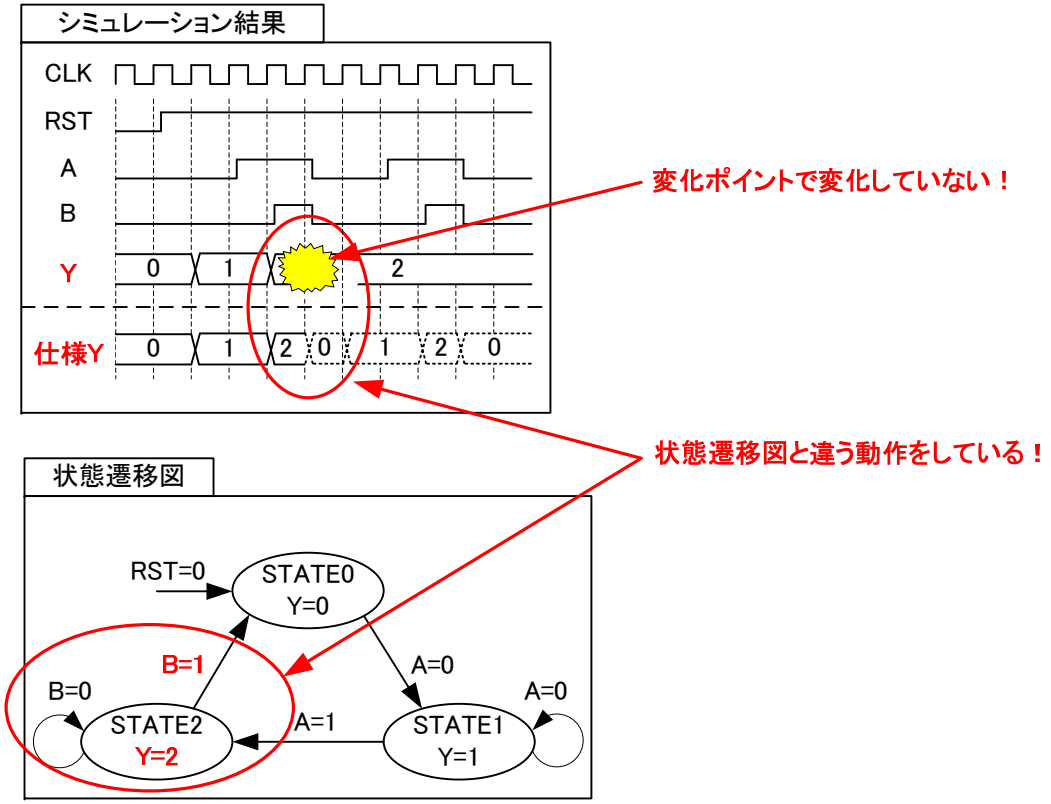
イ　RTLの組み合わせ回路always記述で、センシティビティ・リストが不完全なため、ステート・マシンが動作しない。

ウ　テストベンチの記述（initial文）にあるB=1の入力時にA=1も成立しているため、ステート・マシンが動作しない。

エ　テストベンチの記述にある設計対象のインスタンス化で、接続される信号が間違っているため、ステート・マシンが動作しない。

【答】イ

【解説】
問題で**「シミュレーションで確認すると仕様どおりに動作していない」**と問われていることから、**「シミュレーション結果」**の見るべきポイントは Y と **仕様Y** の部分と、**「状態遷移図」**の STATE2 の部分です。**「シミュレーション結果」**が示す A や B の入力信号の状態を見ても、入力の仕様的に問題無いことから**「テストベンチ」**に問題が無いことがわかります。この時点で選択肢**ア～エ**で問われている**ウ**と**エ**に関しては**選択対象外**になります。よって**「RTL」**のステート・マシン記述に原因があることになります。



ステート・マシンの状態遷移条件が揃っているのにもかかわらず、変化するべきポイントで変化しない原因は、always文で記述されたステート・マシンの信号定義です。always文はセンシティビティ・リストに定義する信号の変化を常に見ることを行っています。よってセンシティビティ・リストが不完全であると、STATE2の状態遷移条件である入力信号Bの信号をalways文の中に記述してあっても、その信号の変化を見つけることができないので、シミュレーション結果のような動作になってしまいます。

RTL（間違った記述）

```
module state(RST, CLK, A, B, Y) ;
input              RST, CLK, A, B ;
output [1:0]       Y ;

parameter STATE0 = 2'b00 ;
parameter STATE1 = 2'b01 ;
parameter STATE2 = 2'b10 ;
reg [1:0] now_state;
reg [1:0] next_state;

    always @(posedge CLK or negedge RST)
    if(!RST)
        now_state <= STATE0 ;
    else
        now_state <= next_state ;

    always @(now_state, A)
        case (now_state)
            STATE0 : if (!A) next_state <= STATE1 ;
                     else    next_state <= STATE0 ;
            STATE1 : if (A)  next_state <= STATE2 ;
                     else    next_state <= STATE1 ;
            STATE2 : if (B)  next_state <= STATE0 ;
                     else    next_state <= STATE2 ;
            default :        next_state <= now_state ;
        endcase

    assign Y = now_state ;

endmodule
```

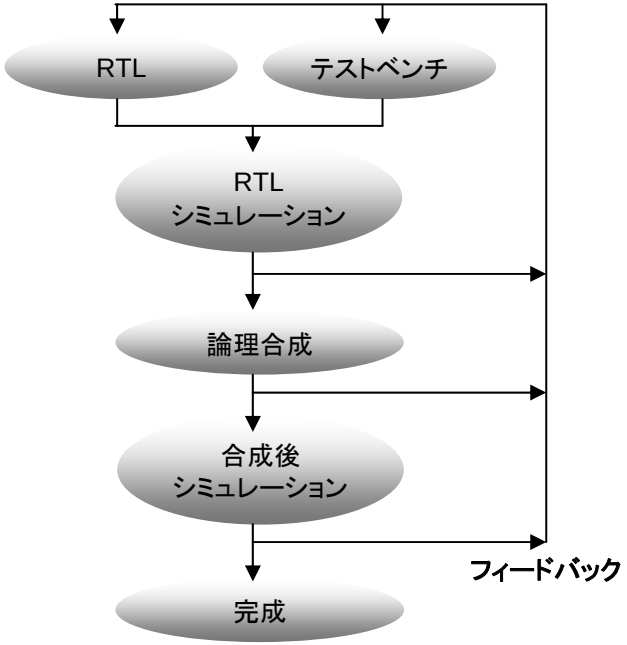
always文中にBは使用されているのに、センシティビティ・リストに入力信号が定義されていない！

RTL（正しい記述）

```
module state(RST, CLK, A, B, Y) ;
input              RST, CLK, A, B ;
output [1:0]       Y ;

parameter STATE0 = 2'b00 ;
parameter STATE1 = 2'b01 ;
parameter STATE2 = 2'b10 ;
reg [1:0] now_state;
reg [1:0] next_state;

    always @(posedge CLK or negedge RST)
    if(!RST)
        now_state <= STATE0 ;
    else
        now_state <= next_state ;

    always @(now_state, A, B)
        case (now_state)
            STATE0 : if (!A) next_state <= STATE1 ;
                     else    next_state <= STATE0 ;
            STATE1 : if (A)  next_state <= STATE2 ;
                     else    next_state <= STATE1 ;
            STATE2 : if (B)  next_state <= STATE0 ;
                     else    next_state <= STATE2 ;
            default :        next_state <= now_state ;
        endcase

    assign Y = now_state ;

endmodule
```

【問題の意図】
この問題に記載されている情報は、**「RTL」**、**「テストベンチ」**、**「シミュレーション結果」**、**「状態遷移図」**と、回答者にとって参考にするものが多く感じてもらうのがポイント（情報が有り過ぎてどれを見ていいかわからない状況になる）です。HDL言語で設計する上では、「RTL」、「テストベンチ」、「シミュレーション」、「仕様」のすべてを理解できることが必須であることから、参考にすべき情報の多い問題にしています。

求められるスキル
・状態遷移図やタイムチャートが理解できる
・VerilogHDL言語による論理記述を理解している



この問題はHDLによる回路設計でよくある間違いから出題しています。
センシティビティ・リストの問題は単純ですが誤りが起きやすい問題です。

HDL設計では、RTL→テストベンチ→シミュレーション→RTL・・・と検証をフィードバックしながら設計するスタイルが基本です。RTLを記述した際には、コンパイラによるチェックを行いますが、出題されている問題のセンシティビティ・リストの定義漏れは、多くのコンパイラでエラーが発生しません（論理合成ではWarningとして発生します）。