```
// (C) Tomonori Izumi <izumi@ieee.org>, Oct. 2007

// リハビリ問題　順序回路（ステートマシン）編　信号機

// 信号機を設計しなさい。
// 入力 xreset       ... 1bit リセット信号（負論理）
// 入力 clock        ... 1bit クロック（立ち上がりトリガ）
// 入力 walker       ... 1bit 歩行者からの横断要求（正論理）
// 入力 next         ... 1bit 次状態への移行指示（正論理）
// 出力 light_blue   ... 1bit 青信号点灯（正論理）
// 出力 light_yellow ... 1bit 黄信号点灯（正論理）
// 出力 light_red    ... 1bit 赤信号点灯（正論理）

// 非同期リセット（負論理）付き、クロック同期回路とする。
// 初期状態は青にする。
// 青状態のときに next が入力されるか歩行者からの横断要求があったら黄にする。
// 黄状態のときに next が入力されると赤にする。
// 赤状態のときに next が入力されると青にする。
// それ以外のときは状態はかわらない。

// 順序回路記述のアドバイス
// レジスタやフリップフロップは reg として宣言する。
// 各 reg （のグループ）ひとつにつきひとつの always ブロックを用意する。
// その（グループの）reg の代入は対応する always ブロック内でのみ行なう。
// 代入は <= で行なう。
// always ブロックの概形は雛形に沿ったものにする。

module trafficsignal_module(xreset, clock, walker, next, light_blue, light_yellow, light_red);

   _____ xreset;
   _____ clock;
   _____ walker;
   _____ next;
   ______ light_blue, light_yellow, light_red;

   ___ [___]   state;

   p________   STATE_BLUE      = 2'b00;
   p________   STATE_YELLOW    = 2'b01;
   p________   STATE_RED       = 2'b11;

   always @(posedge clock or negedge xreset)
     begin
        if (!xreset) begin
           __________________;
        end else begin
           case (state)
             STATE_BLUE: begin
                if (_______________) begin
                   ____________________;
```

```
                end
            end
            STATE_YELLOW: begin
                if (_______________) begin
                    ____________________;
                end
            end
            STATE_RED: begin
                if (_______________) begin
                    ____________________;
                end
            end
          endcase
        end
    end

  assign light_blue   = (____________________) ? 1'b1 : 1'b0;
  assign light_yellow = (____________________) ? 1'b1 : 1'b0;
  assign light_red    = (____________________) ? 1'b1 : 1'b0;

endmodule
```