```


//********************** LEVEL 1 FSM ******************************
module lvl1FSM (
                     clk, reset, go, goJump, right, left, up, down,
                     resetAddress, drM, drML, erM, jump, jumpL, jumping, falling, drStage1,
moveRight,
                     done, jumpCounter, fall,
                     writeEn, rightColour, leftColour, ground, outofBounds, pipe, lvl1,
start, dead);

   input [4:0] jumpCounter;
   input clk, reset, left, up, down, go, goJump, right, ground, outofBounds, pipe, lvl1,
fall, start, dead;
   input [11:0] rightColour, leftColour;

   reg Facingright, Facingleft;
   initial Facingright = 0;
   initial Facingleft = 0;

   input done;
   output reg resetAddress, drM, drML, erM, jump, jumpL, jumping, falling, drStage1, writeEn
, moveRight;

   reg [10:0] current_state, next_state;

   localparam  nothing        = 10'd0,
               drawstage1     = 10'd1,
               waitmario      = 10'd2,
               drawmarioRight = 10'd3,
               drawmarioLeft  = 10'd4,
               jumpmarioRight = 10'd5,
               jumpmarioLeft  = 10'd6,
               erasemario     = 10'd7,
               jumpingerase   = 10'd8,
               fallingerase   = 10'd9,
               waitState      = 10'd10;

   always @(*)
   begin: state_table
      case(current_state)
         waitState: next_state = lvl1 ? drawstage1 : waitState;

         nothing: begin //wait state that controls direction control path should take
            if (go && ((right && !outofBounds) || (left && !outofBounds)))
               next_state = erasemario;
            else if (!ground && jumpCounter == 0 && !dead)
               next_state = fallingerase;
            else if (up && goJump)
               next_state = jumpingerase;
            else if (down && pipe)
               next_state = nothing;
            else
               next_state = nothing;
         end
         drawstage1: //draws level one - initial state
         begin
            if (done)
               next_state = waitmario;
            else
               next_state = drawstage1;
         end

         waitmario: next_state = drawmarioRight; //resets address

         drawmarioRight: //draws mario in right orientation
         begin
            if (done)
               next_state = nothing;
            else
               next_state = drawmarioRight;
         end

```

```verilog
            drawmarioLeft: begin //draws mario left orientation
               if (done)
                  next_state = nothing;
               else
                  next_state = drawmarioLeft;
            end

            jumpmarioRight: begin //draws mario jumping to the right
               if (done && !ground && goJump && fall)
                  next_state = fallingerase;
               else if (done && ground && goJump && fall)
                  next_state = drawmarioRight;
               else if (done && goJump && jumpCounter < 5'd22)
                  next_state = jumpingerase;
               else if (done && goJump && jumpCounter == 5'd22)
                  next_state = erasemario;
               else if (done && ground && goJump)
                  next_state = drawmarioRight;
               else
                  next_state = jumpmarioRight;
            end

            jumpmarioLeft: begin //draws mario jumping to the left
               if (done && !ground && goJump && fall)
                  next_state = fallingerase;
               else if (done && ground && goJump && fall)
                  next_state = drawmarioLeft;
               else if (done && goJump && jumpCounter < 5'd22)
                  next_state = jumpingerase;
               else if (done && goJump && jumpCounter == 5'd22)
                  next_state = erasemario;
               else if (done && ground && goJump)
                  next_state = drawmarioLeft;
               else
                  next_state = jumpmarioLeft;
            end

            jumpingerase: begin
               if (done && Facingright)
                  next_state = jumpmarioRight;
               else if (done && Facingleft)
                  next_state = jumpmarioLeft;
               else
                  next_state = jumpingerase;
            end

            fallingerase: begin
               if (!ground && done && Facingright)
                  next_state = jumpmarioRight;
               else if (ground && done && Facingright)
                  next_state = drawmarioRight;
               else if (!ground && done && Facingleft)
                  next_state = jumpmarioLeft;
               else if (ground && done && Facingleft)
                  next_state = drawmarioLeft;
               else
                  next_state = fallingerase;
            end

            erasemario: //erases mario by drawing background
            begin
               if(done && right)
                  next_state = drawmarioRight;
               else if (done && left)
                  next_state = drawmarioLeft;
               else if (done && Facingright)
                  next_state = drawmarioRight;
               else if (done && Facingleft)
                  next_state = drawmarioLeft;
               else
                  next_state = erasemario;
            end

            default: next_state = waitState; //initial state draw background
         endcase
      end

      always@(posedge clk)
      begin: logic

         drM = 0;
         drML = 0;
         writeEn = 0;
         drStage1 = 0;
         resetAddress = 0;
         moveRight = 0;
         erM = 0;
         jump = 0;
         jumpL = 0;
         jumping = 0;
         falling = 0;

         case(current_state)

         nothing: begin
            resetAddress = 1;
            writeEn = 0;
         end
         drawstage1: begin
            drStage1 = 1;
            writeEn = 1;
         end
         waitmario: begin
            writeEn = 0;
            resetAddress = 1;
         end
         drawmarioRight: begin
            drM = 1;
            writeEn = 1;
            Facingright = 1;
            Facingleft = 0;
         end
         drawmarioLeft: begin
            drML = 1;
            writeEn = 1;
            Facingleft = 1;
            Facingright = 0;
         end
         jumpmarioRight: begin
            jump = 1;
            writeEn = 1;
         end
         jumpmarioLeft: begin
            jumpL = 1;
            writeEn = 1;
         end
         jumpingerase: begin
            writeEn = 1;
            jumping = 1;
         end
         fallingerase: begin
            writeEn = 1;
            falling = 1;
         end
         erasemario: begin
            erM = 1;
            writeEn = 1;
         end

         endcase
      end

      always@(posedge clk)
       begin: state_FFs
         if (start || dead)
            current_state <= waitState;
         else
           current_state <= next_state;
       end

   endmodule



   //*********************** LEVEL 2 FSM ******************************
   module lvl2FSM (
                        clk, reset, go, goJump, right, left, up, down,
                        resetAddress, drM, drML, erM, jump, jumpL, jumping, falling, drStage2,
   moveRight,
                        done, jumpCounter, fall,
                        writeEn, rightColour, leftColour, ground, outofBounds, next, lvl2,
   start);

      input [4:0] jumpCounter;
      input clk, reset, left, up, down, go, goJump, right, ground, outofBounds, next, lvl2,
   fall, start;
      input [11:0] rightColour, leftColour;

      reg Facingright, Facingleft;
      initial Facingright = 0;
      initial Facingleft = 0;

      input done;
      output reg resetAddress, drM, drML, erM, jump, jumpL, jumping, falling, drStage2, writeEn
   , moveRight;

      reg [10:0] current_state, next_state;

      localparam  nothing        = 10'd0,
                  drawstage2     = 10'd1,
                  waitmario      = 10'd2,
                  drawmarioRight = 10'd3,
                  drawmarioLeft  = 10'd4,
                  jumpmarioRight = 10'd5,
                  jumpmarioLeft  = 10'd6,
                  erasemario     = 10'd7,
                  jumpingerase   = 10'd8,
                  fallingerase   = 10'd9,
                  waitState      = 10'd10;

      always @(*)
      begin: state_table
         case(current_state)
            waitState: next_state = lvl2 ? drawstage2 : waitState;

            nothing: begin //wait state that controls direction control path should take
               if (go && ((right && !outofBounds) || (left && !outofBounds)))
                  next_state = erasemario;
               else if (!ground && jumpCounter == 0)
                  next_state = fallingerase;
               else if (up && goJump && !outofBounds)
                  next_state = jumpingerase;
               else
                  next_state = nothing;
            end
            drawstage2: //draws level one - initial state
            begin
               if (done)
                  next_state = waitmario;
               else
                  next_state = drawstage2;
            end

            waitmario: next_state = drawmarioRight; //resets address

            drawmarioRight: //draws mario in right orientation
            begin
               if (done)
                  next_state = nothing;
```

```verilog
            else
                next_state = drawmarioRight;
        end

        drawmarioLeft: begin //draws mario left orientation
            if (done)
                next_state = nothing;
            else
                next_state = drawmarioLeft;
        end

        jumpmarioRight: begin //draws mario jumping to the right
            if (done && !ground && goJump && fall)
                next_state = fallingerase;
            else if (done && ground && goJump && fall)
                next_state = drawmarioRight;
            else if (done && goJump && jumpCounter < 5'd22)
                next_state = jumpingerase;
            else if (done && goJump && jumpCounter == 5'd22)
                next_state = erasemario;
            else if (done && ground && goJump)
                next_state = drawmarioRight;
            else
                next_state = jumpmarioRight;
        end

        jumpmarioLeft: begin //draws mario jumping to the left
            if (done && !ground && goJump && fall)
                next_state = fallingerase;
            else if (done && ground && goJump && fall)
                next_state = drawmarioLeft;
            else if (done && goJump && jumpCounter < 5'd22)
                next_state = jumpingerase;
            else if (done && goJump && jumpCounter == 5'd22)
                next_state = erasemario;
            else if (done && ground && goJump)
                next_state = drawmarioLeft;
            else
                next_state = jumpmarioLeft;
        end

        jumpingerase: begin
            if (done && Facingright)
                next_state = jumpmarioRight;
            else if (done && Facingleft)
                next_state = jumpmarioLeft;
            else
                next_state = jumpingerase;
        end

        fallingerase: begin
            if (!ground && done && Facingright)
                next_state = jumpmarioRight;
            else if (ground && done && Facingright)
                next_state = drawmarioRight;
            else if (!ground && done && Facingleft)
                next_state = jumpmarioLeft;
            else if (ground && done && Facingleft)
                next_state = drawmarioLeft;
            else
                next_state = fallingerase;
        end

        erasemario: //erases mario by drawing background
        begin
            if(done && right)
                next_state = drawmarioRight;
            else if (done && left)
                next_state = drawmarioLeft;
            else if (done && Facingright)
                next_state = drawmarioRight;
            else if (done && Facingleft)
                next_state = drawmarioLeft;
                else
                    next_state = erasemario;
            end

            default: next_state = waitState; //initial state draw background
        endcase
    end

    always@(posedge clk)
    begin: logic

        drM = 0;
        drML = 0;
        writeEn = 0;
        drStage2 = 0;
        resetAddress = 0;
        moveRight = 0;
        erM = 0;
        jump = 0;
        jumpL = 0;
        jumping = 0;
        falling = 0;

        case(current_state)

        nothing: begin
            resetAddress = 1;
            writeEn = 0;
        end
        drawstage2: begin
            drStage2 = 1;
            writeEn = 1;
        end
        waitmario: begin
            writeEn = 0;
            resetAddress = 1;
        end
        drawmarioRight: begin
            drM = 1;
            writeEn = 1;
            Facingright = 1;
            Facingleft = 0;
        end
        drawmarioLeft: begin
            drML = 1;
            writeEn = 1;
            Facingleft = 1;
            Facingright = 0;
        end
        jumpmarioRight: begin
            jump = 1;
            writeEn = 1;
        end
        jumpmarioLeft: begin
            jumpL = 1;
            writeEn = 1;
        end
        jumpingerase: begin
            writeEn = 1;
            jumping = 1;
        end
        fallingerase: begin
            writeEn = 1;
            falling = 1;
        end
        erasemario: begin
            erM = 1;
            writeEn = 1;
        end

        endcase
    end

```

```verilog
   always@(posedge clk)
    begin: state_FFs
      if (start)
         current_state <= waitState;
      else
        current_state <= next_state;
    end

endmodule

//*********************** LEVEL 3 FSM ******************************
module lvl3FSM (
                     clk, reset, go, goJump, right, left, up, down,
                     resetAddress, drM, drML, erM, jump, jumpL, jumping, falling, drStage3,
moveRight,
                     done, jumpCounter, fall,
                     writeEn, rightColour, leftColour, ground, outofBounds, flag, lvl3,
start);

   input [4:0] jumpCounter;
   input clk, reset, left, up, down, go, goJump, right, ground, outofBounds, flag, lvl3,
fall, start;
   input [11:0] rightColour, leftColour;

   reg Facingright, Facingleft;
   initial Facingright = 0;
   initial Facingleft = 0;

   input done;
   output reg resetAddress, drM, drML, erM, jump, jumpL, jumping, falling, drStage3, writeEn
, moveRight;

   reg [10:0] current_state, next_state;

   localparam  nothing        = 10'd0,
               drawstage3     = 10'd1,
               waitmario      = 10'd2,
               drawmarioRight = 10'd3,
               drawmarioLeft  = 10'd4,
               jumpmarioRight = 10'd5,
               jumpmarioLeft  = 10'd6,
               erasemario     = 10'd7,
               jumpingerase   = 10'd8,
               fallingerase   = 10'd9,
               waitState      = 10'd10;

   always @(*)
   begin: state_table
      case(current_state)
         waitState: next_state = lvl3 ? drawstage3 : waitState;

         nothing: begin //wait state that controls direction control path should take
            if (go && ((right && !outofBounds) || (left && !outofBounds)))
               next_state = erasemario;
            else if (!ground && jumpCounter == 0)
               next_state = fallingerase;
            else if (up && goJump)
               next_state = jumpingerase;
            else
               next_state = nothing;
         end
         drawstage3: //draws level one - initial state
         begin
            if (done)
               next_state = waitmario;
            else
               next_state = drawstage3;
         end

         waitmario: next_state = drawmarioRight; //resets address

         drawmarioRight: //draws mario in right orientation
         begin
```

```
                if (done)
                    next_state = nothing;
                else
                    next_state = drawmarioRight;
            end

            drawmarioLeft: begin //draws mario left orientation
                if (done)
                    next_state = nothing;
                else
                    next_state = drawmarioLeft;
            end

            jumpmarioRight: begin //draws mario jumping to the right
                if (done && !ground && goJump && fall)
                    next_state = fallingerase;
                else if (done && ground && goJump && fall)
                    next_state = drawmarioRight;
                else if (done && goJump && jumpCounter < 5'd22)
                    next_state = jumpingerase;
                else if (done && goJump && jumpCounter == 5'd22)
                    next_state = erasemario;
                else if (done && ground && goJump)
                    next_state = drawmarioRight;
                else
                    next_state = jumpmarioRight;
            end

            jumpmarioLeft: begin //draws mario jumping to the left
                if (done && !ground && goJump && fall)
                    next_state = fallingerase;
                else if (done && ground && goJump && fall)
                    next_state = drawmarioLeft;
                else if (done && goJump && jumpCounter < 5'd22)
                    next_state = jumpingerase;
                else if (done && goJump && jumpCounter == 5'd22)
                    next_state = erasemario;
                else if (done && ground && goJump)
                    next_state = drawmarioLeft;
                else
                    next_state = jumpmarioLeft;
            end

            jumpingerase: begin
                if (done && Facingright)
                    next_state = jumpmarioRight;
                else if (done && Facingleft)
                    next_state = jumpmarioLeft;
                else
                    next_state = jumpingerase;
            end

            fallingerase: begin
                if (!ground && done && Facingright)
                    next_state = jumpmarioRight;
                else if (ground && done && Facingright)
                    next_state = drawmarioRight;
                else if (!ground && done && Facingleft)
                    next_state = jumpmarioLeft;
                else if (ground && done && Facingleft)
                    next_state = drawmarioLeft;
                else
                    next_state = fallingerase;
            end

            erasemario: //erases mario by drawing background
            begin
                if(done && right)
                    next_state = drawmarioRight;
                else if (done && left)
                    next_state = drawmarioLeft;
                else if (done && Facingright)
                    next_state = drawmarioRight;
                else if (done && Facingleft)
                    next_state = drawmarioLeft;
                else
                    next_state = erasemario;
            end

            default: next_state = waitState; //initial state draw background
        endcase
    end

    always@(posedge clk)
    begin: logic

        drM = 0;
        drML = 0;
        writeEn = 0;
        drStage3 = 0;
        resetAddress = 0;
        moveRight = 0;
        erM = 0;
        jump = 0;
        jumpL = 0;
        jumping = 0;
        falling = 0;

        case(current_state)

        nothing: begin
            resetAddress = 1;
            writeEn = 0;
        end
        drawstage3: begin
            drStage3 = 1;
            writeEn = 1;
        end
        waitmario: begin
            writeEn = 0;
            resetAddress = 1;
        end
        drawmarioRight: begin
            drM = 1;
            writeEn = 1;
            Facingright = 1;
            Facingleft = 0;
        end
        drawmarioLeft: begin
            drML = 1;
            writeEn = 1;
            Facingleft = 1;
            Facingright = 0;
        end
        jumpmarioRight: begin
            jump = 1;
            writeEn = 1;
        end
        jumpmarioLeft: begin
            jumpL = 1;
            writeEn = 1;
        end
        jumpingerase: begin
            writeEn = 1;
            jumping = 1;
        end
        fallingerase: begin
            writeEn = 1;
            falling = 1;
        end
        erasemario: begin
            erM = 1;
            writeEn = 1;
        end

        endcase
    end

    always@(posedge clk)
     begin: state_FFs
       if (start)
          current_state <= waitState;
       else
         current_state <= next_state;
     end

endmodule

```