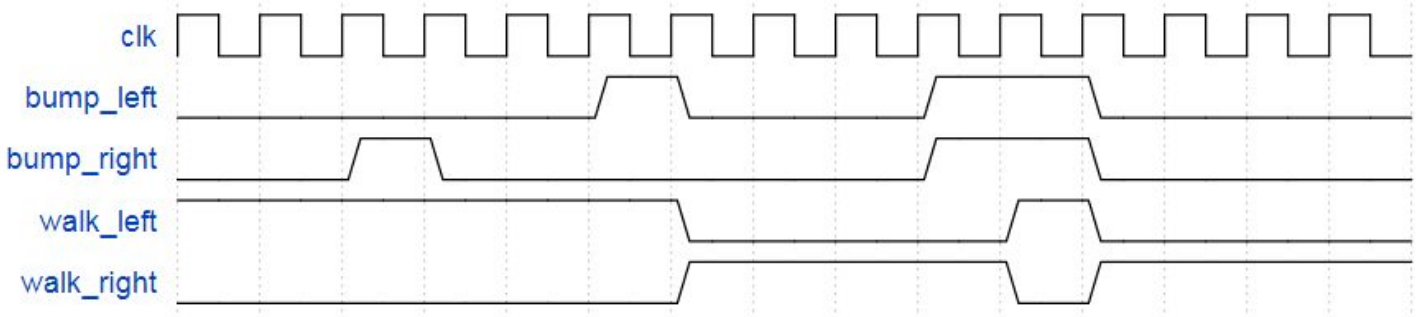
Lemmings1

The game Lemmings involves critters with fairly simple brains. So simple that we are going to model it using a finite state machine.

In the Lemmings' 2D world, Lemmings can be in one of two states: walking left or walking right. It will switch directions if it hits an obstacle. In particular, if a Lemming is bumped on the left, it will walk right. If it's bumped on the right, it will walk left. If it's bumped on both sides at the same time, it will still switch directions.

Implement a Moore state machine with two states, two inputs, and one output that models this behaviour.




```
module top_module(
    input clk,
    input areset,    // Freshly brainwashed Lemmings walk left.
    input bump_left,
    input bump_right,
    output walk_left,
    output walk_right); //

    // parameter LEFT=0, RIGHT=1, ...
    reg state, next_state;

    parameter WK_LEFT =1'b0;
    parameter WK_RIGHT=1'b1;

    always @(*) begin
        // State transition logic
        case(state)
            WK_LEFT: next_state = (bump_left) ?WK_RIGHT:WK_LEFT;
            WK_RIGHT:next_state = (bump_right)?WK_LEFT :WK_RIGHT;
        endcase
    end

    always @(posedge clk, posedge areset) begin
        // State flip-flops with asynchronous reset
        if (areset)begin
```

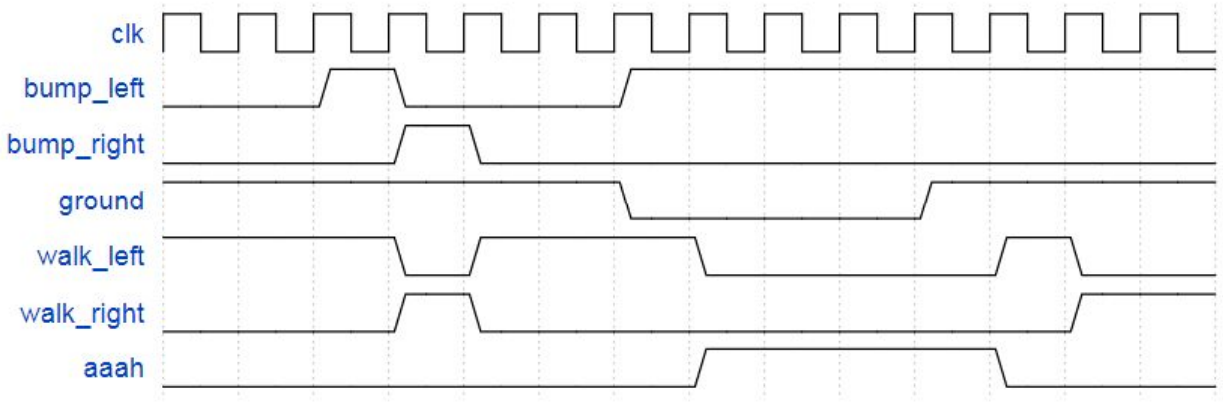
```
            state <= WK_LEFT;
        end
        else begin
            state <= next_state;
        end
    end

    // Output logic
    assign walk_left = state==WK_LEFT;
    assign walk_right= state==WK_RIGHT;


endmodule
```

Lemmings2

In addition to walking left and right, Lemmings will fall (and presumably go "aaah!") if the ground disappears underneath them.

In addition to walking left and right and changing direction when bumped, when `ground=0`, the Lemming will fall and say "aaah!". When the ground reappears (`ground=1`), the Lemming will resume walking in the same direction as before the fall. Being bumped while falling does not affect the walking direction, and being bumped in the same cycle as ground disappears (but not yet falling), or when the ground reappears while still falling, also does not affect the walking direction.

Build a finite state machine that models this behaviour.




```
module top_module(
    input clk,
    input areset,    // Freshly brainwashed Lemmings walk left.
    input bump_left,
    input bump_right,
    input ground,
    output walk_left,
    output walk_right,
    output aaah );

    parameter LEFT   = 2'b01;
    parameter RIGHT  = 2'b00;
    parameter F_LEFT = 2'b11;
    parameter F_RIGHT= 2'b10;
    reg [2:0] curr_state, next_state;

    always @(*)begin
        case(curr_state)
            LEFT: if(~ground)begin
                next_state = F_LEFT;
            end
            else if(bump_left) begin
                next_state = RIGHT;
            end
```

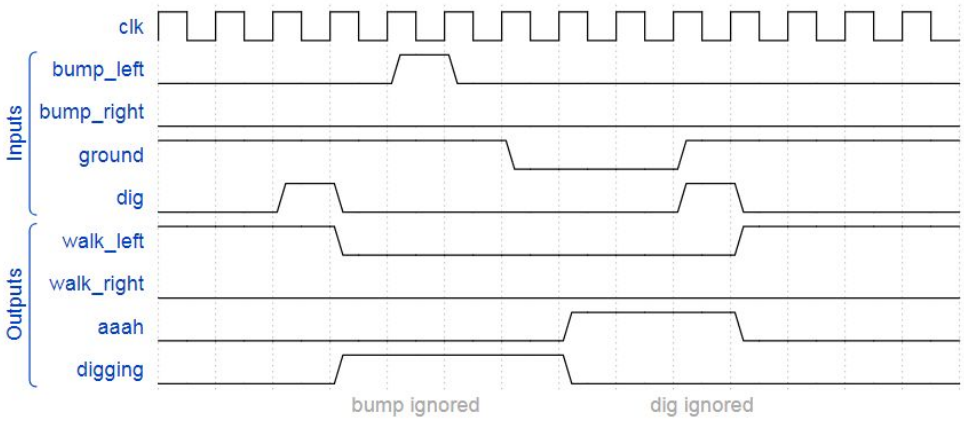
```
            else begin
                next_state = LEFT;
            end
            RIGHT: if(~ground)begin
                next_state = F_RIGHT;
            end
            else if (bump_right)begin
                next_state = LEFT;
            end
            else begin
                next_state = RIGHT;
            end
            F_LEFT: if (ground) begin
                next_state = LEFT;
            end
            else begin
                next_state = F_LEFT;
            end
            F_RIGHT: if(ground) begin
                next_state = RIGHT;
            end
            else begin
                next_state = F_RIGHT;
            end
        endcase
    end

    always @ (posedge clk or posedge areset)begin
        if(areset)begin
            curr_state <= LEFT;
        end
        else begin
            curr_state <= next_state;
        end
    end

    assign aaah = (curr_state==F_RIGHT) | (curr_state==F_LEFT);
    assign walk_left = (curr_state==LEFT);
    assign walk_right = (curr_state==RIGHT);
endmodule
```

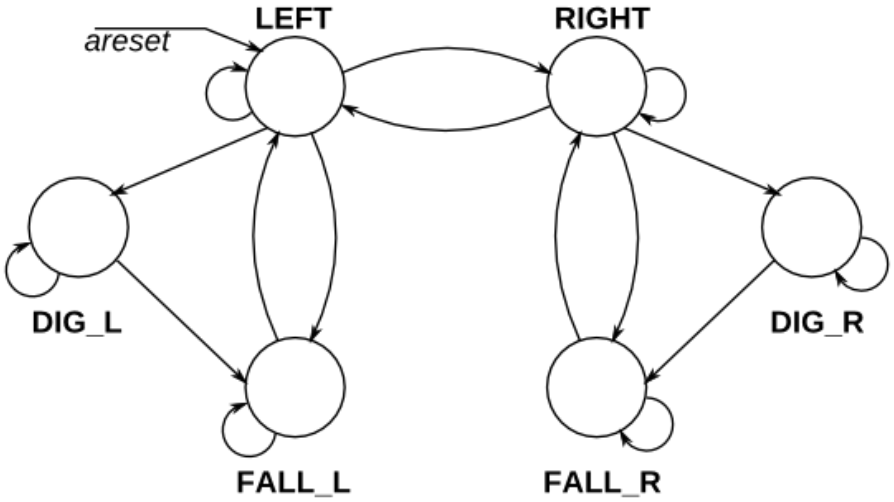
Lemmings3

In addition to walking and falling, Lemmings can sometimes be told to do useful things, like dig (it starts digging when `dig=1`). A Lemming can dig if it is currently walking on ground (`ground=1` and not falling), and will continue digging until it reaches the other side (`ground=0`). At that point, since there is no ground, it will fall (aaah!), then continue walking in its original direction once it hits ground again. As with falling, being bumped while digging has no effect, and being told to dig when falling or when there is no ground is ignored.

(In other words, a walking Lemming can fall, dig, or switch directions. If more than one of these conditions are satisfied, fall has higher precedence than dig, which has higher precedence than switching directions.)

Extend your finite state machine to model this behaviour.





```
module top_module(
    input clk,
    input areset,    // Freshly brainwashed Lemmings walk left.
    input bump_left,
    input bump_right,
    input ground,
    input dig,
    output walk_left,
    output walk_right,
    output aaah,
    output digging );

    parameter LEFT  =3'b000;
    parameter L_DIG =3'b001;
    parameter L_FALL=3'b010;

    parameter RIGHT =3'b100;
    parameter R_DIG =3'b101;
    parameter R_FALL=3'b110;

    reg [2:0] curr_state, next_state;

    always @ (*)begin
        case(curr_state)
            LEFT: if (~ground) begin
                        next_state = L_FALL;
                end
                        else if (dig) begin
                    next_state = L_DIG;
                        end
                        else if (bump_left)begin
                    next_state = RIGHT;
                        end
                else begin
                    next_state = LEFT;
                end
            RIGHT:if (~ground) begin
                        next_state = R_FALL;
                end
                        else if (dig) begin
                        next_state = R_DIG;
                        end
                        else if (bump_right)begin
                    next_state = LEFT;
                        end
                        else begin
                    next_state = RIGHT;
                end
            L_DIG:if (~ground) begin
                        next_state = L_FALL;
                end
                else begin
                    next_state = L_DIG;
                end
            R_DIG:if (~ground) begin
                        next_state = R_FALL;
                        end
                        else begin
```

```
                    next_state = R_DIG;
                    end
        L_FALL: next_state = (ground)?LEFT :L_FALL;
        R_FALL: next_state = (ground)?RIGHT:R_FALL;
        default:next_state = LEFT;
      endcase
   end

   always @(posedge clk or posedge areset)begin
      if (areset)begin
         curr_state <= LEFT;
      end
      else begin
         curr_state <= next_state;
      end
   end

   assign aaah    = (curr_state == L_FALL) | (curr_state==R_FALL);

   assign digging = (curr_state == L_DIG ) | (curr_state==R_DIG );

   assign walk_left  = (curr_state == LEFT );
   assign walk_right = (curr_state == RIGHT );
endmodule
```

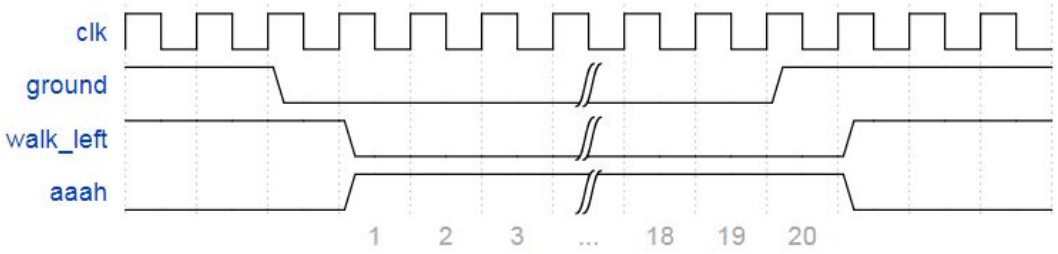
Lemmings4

See also: Lemmings1, Lemmings2, and Lemmings3.

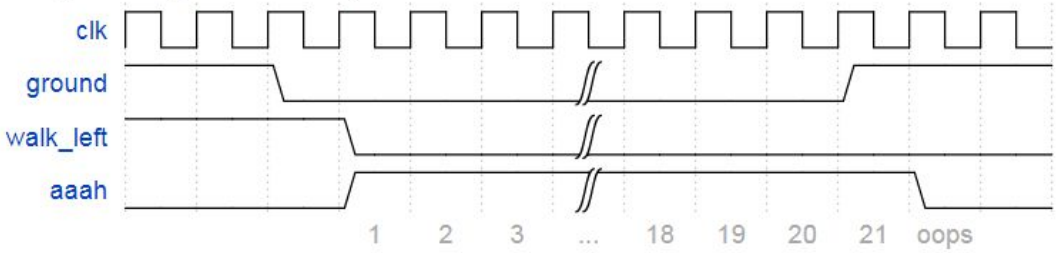
Although Lemmings can walk, fall, and dig, Lemmings aren't invulnerable. If a Lemming falls for too long then hits the ground, it can splatter. In particular, if a Lemming falls for more than 20 clock cycles then hits the ground, it will splatter and cease walking, falling, or digging (all 4 outputs become 0), forever (Or until the FSM gets reset). There is no upper limit on how far a Lemming can fall before hitting the ground. Lemmings only splatter when hitting the ground; they do not splatter in mid-air.

Extend your finite state machine to model this behaviour.

Falling for 20 cycles is survivable:



Falling for 21 cycles causes splatter:




```
module top_module(
    input clk,
    input areset,    // Freshly brainwashed Lemmings walk left.
    input bump_left,
    input bump_right,
    input ground,
    input dig,
    output walk_left,
    output walk_right,
    output aaah,
    output digging );

    parameter LEFT  =3'b000;
    parameter L_DIG =3'b001;
    parameter L_FALL=3'b010;

    parameter RIGHT =3'b100;
    parameter R_DIG =3'b101;
```

```
parameter R_FALL=3'b110;

parameter HEAVEN=3'b111;

reg [2:0] curr_state, next_state;
wire      yet_heaven, heaven_call;

always @ (*)begin
   case(curr_state)
      LEFT: if (~ground) begin
                  next_state = L_FALL;
            end
        else if (dig) begin
              next_state = L_DIG;
        end
        else if (bump_left)begin
              next_state = RIGHT;
        end
          else begin
             next_state = LEFT;
            end
      RIGHT:if (~ground) begin
                  next_state = R_FALL;
            end
        else if (dig) begin
                  next_state = R_DIG;
        end
        else if (bump_right)begin
              next_state = LEFT;
        end
        else begin
              next_state = RIGHT;
            end
      L_DIG:if (~ground) begin
              next_state = L_FALL;
            end
            else begin
              next_state = L_DIG;
            end
      R_DIG:if (~ground) begin
                  next_state = R_FALL;
        end
        else begin
                  next_state = R_DIG;
        end
      L_FALL:if (ground & yet_heaven) begin
              next_state = LEFT;
             end
             else if(heaven_call) begin
               next_state = HEAVEN;
             end
             else begin
               next_state = L_FALL;
             end

      R_FALL:if (ground & yet_heaven) begin
              next_state = RIGHT;
             end
             else if(heaven_call) begin
               next_state = HEAVEN;
             end
             else begin
               next_state = R_FALL;
             end
      HEAVEN: next_state = HEAVEN;
```

```
        default:next_state = LEFT;
      endcase
   end

   reg [4:0] cnt;
   wire      fall = (curr_state==L_FALL) | (curr_state==R_FALL);
   wire      cnt_rst = fall & ((next_state==RIGHT) | (next_state==LEFT));
   assign    heaven_call = (cnt==5'd19) & fall & ~ground;
   assign    yet_heaven  = ~heaven_call;


   always @ (posedge clk or posedge areset)begin
      if(areset)begin
         cnt <= 5'd0;
      end
      else if (cnt_rst)begin
         cnt<=5'd0;
      end
      else if (fall)begin
         cnt <= cnt+5'd1;
      end
   end

   always @(posedge clk or posedge areset)begin
      if (areset)begin
         curr_state <= LEFT;
      end
      else begin
         curr_state <= next_state;
      end
   end

   wire aaah_trig = (curr_state==LEFT & next_state==L_FALL) |(curr_state==RIGHT & next_state==R_FALL)|
                    (curr_state==L_DIG & next_state==L_FALL)|(curr_state==R_DIG & next_state==R_FALL);

   wire aaah_end  = (curr_state==L_FALL & next_state==LEFT)|(curr_state==R_FALL & next_state==RIGHT)|
                      (curr_state==HEAVEN & ground);

   always @ (posedge clk or posedge areset) begin
      if(areset)begin
         aaah<=1'b0;
      end
      else if (aaah_end)begin
         aaah<=1'b0;
      end
      else if (aaah_trig)begin
         aaah<=1'b1;
      end
   end

   assign digging = (curr_state == L_DIG ) | (curr_state==R_DIG );

   assign walk_left  = (curr_state == LEFT );
   assign walk_right = (curr_state == RIGHT );
endmodule
```