
Digital Design: A Systems Approach

Lecture 8: Factoring State Machines

Readings

- L8: Chapter 17
- L9: Chapter 18

Review

- Lecture 1 – Digital abstraction
- Lecture 2 – Combinational logic design
- Lecture 3 – Combinational building blocks
- Lecture 4 – Numbers and arithmetic
- Lecture 5 – Quiz 1 Review

- Lecture 6 – Sequential Logic, FSMs
- Lecture 7 – Datapath FSMs
- *Lecture 8 - Factoring FSMs*
- Lecture 9 - Microcode

Partitioning

- Divide a big problem into several smaller problems
- Last lecture - data/control partitioning
- This lecture – factoring state machines
- Later – system partitioning

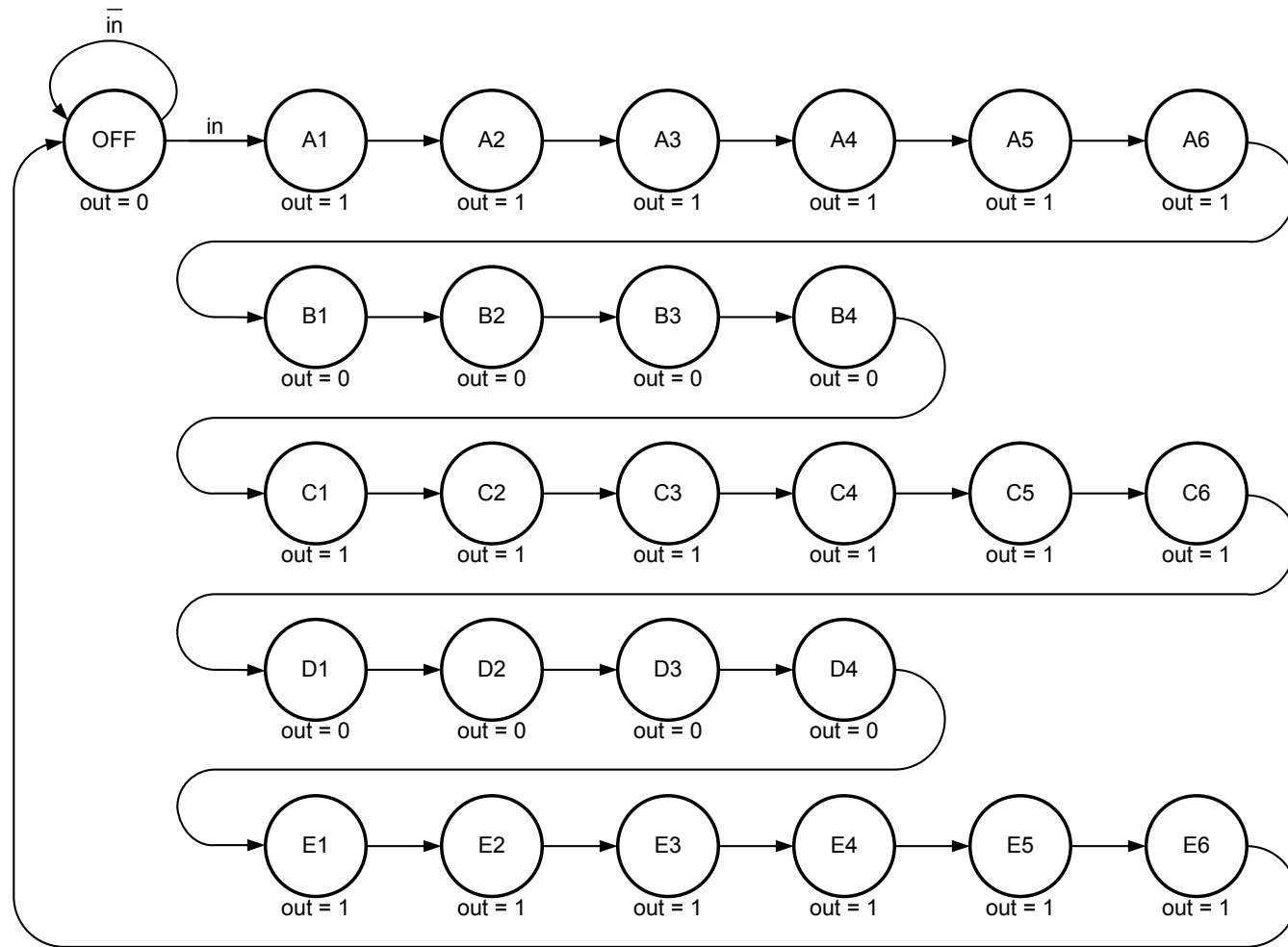
Data/Control Partitioning - Example

- Consider a 4-way digital stopwatch
- Inputs:
 - pb[3:0] - four pushbuttons - high for one clock on each press
 - Toggles start/stop for each counter
 - reset - resets and stops all timers
 - ms - high for one clock each millisecond
- Outputs - time3[15:0], time2[15:0], time1[15:0], time0[15:0]
- What is the data state of the machine?
- Assume you have a single 16-bit adder. What does the datapath look like? What are the control points?
- What control FSM is needed to sequence this?

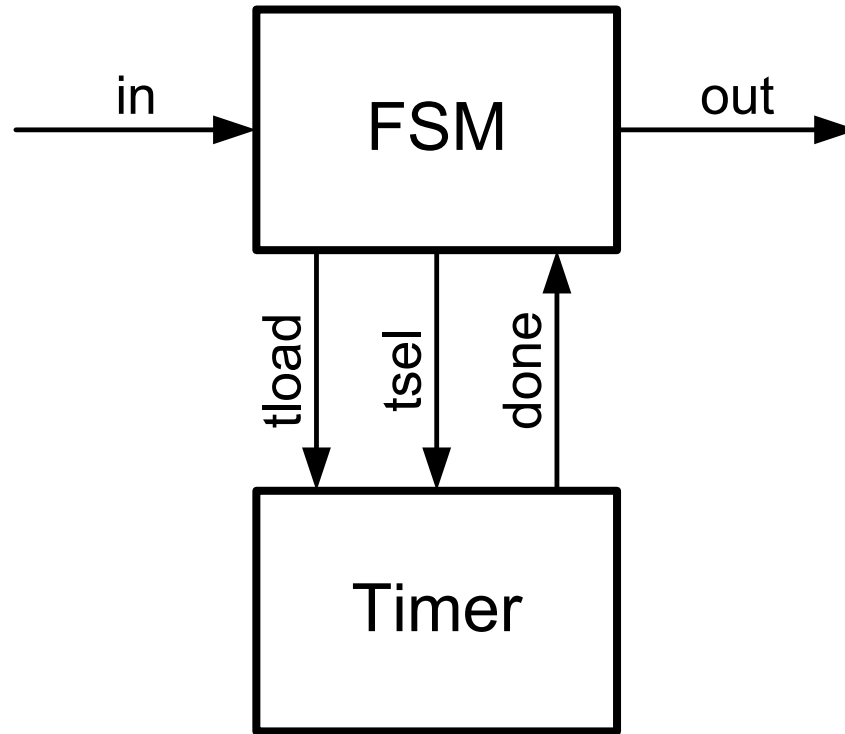
Specification Of Light Flasher

- Inputs: *in*
- Outputs: *out*
- Operation:
 - When *in* = 1, FSM goes through 5 sequences:
On-Off-On-Off-On
 - Each *On* sequence:
 - *out* = 1
 - 6 cycles long
 - Each *Off* sequence:
 - *out* = 0
 - 4 cycles long
 - After 5 sequences, FSM goes back to OFF state to wait for new input.

Light Flasher State Diagram

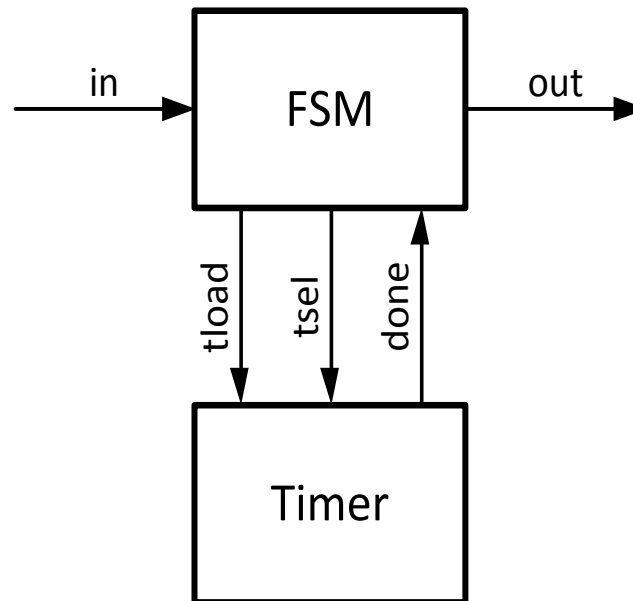
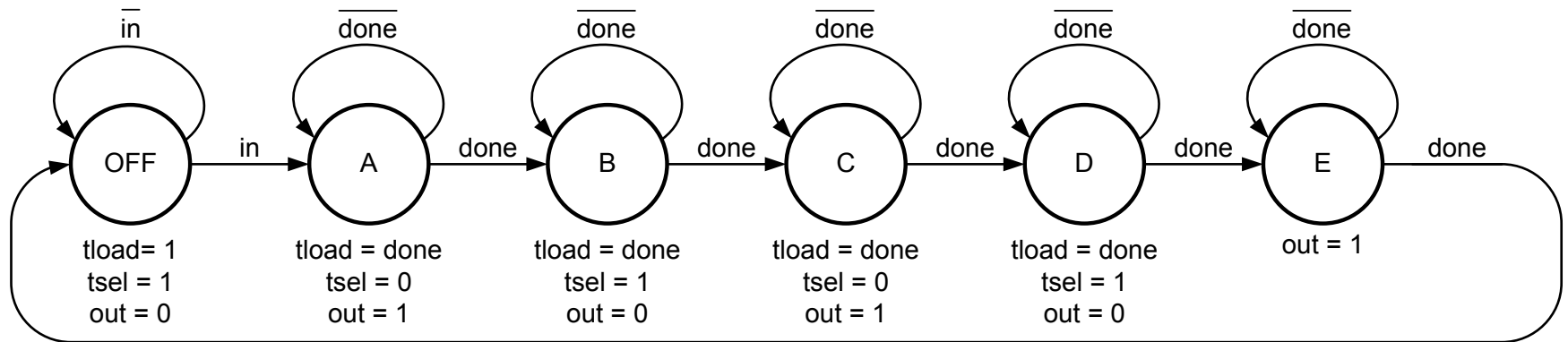


Factored light flasher



***Timer loads value 5
or 3, based on tsel**

State diagram of factored light flasher



```

module Flash(clk, rst, in, out) ;
    input clk, rst, in ; // in triggers start of flash sequence
    output out ;          // out drives LED
    reg out ;              // output
    wire [`SWIDTH-1:0] state, next ; // current state
    reg [`SWIDTH-1:0] next1 ; // next state without reset
    reg tload, tsel ;      // timer inputs
    wire done ;            // timer output

    // instantiate state register
    DFF #(`SWIDTH) state_reg(clk, next, state) ;

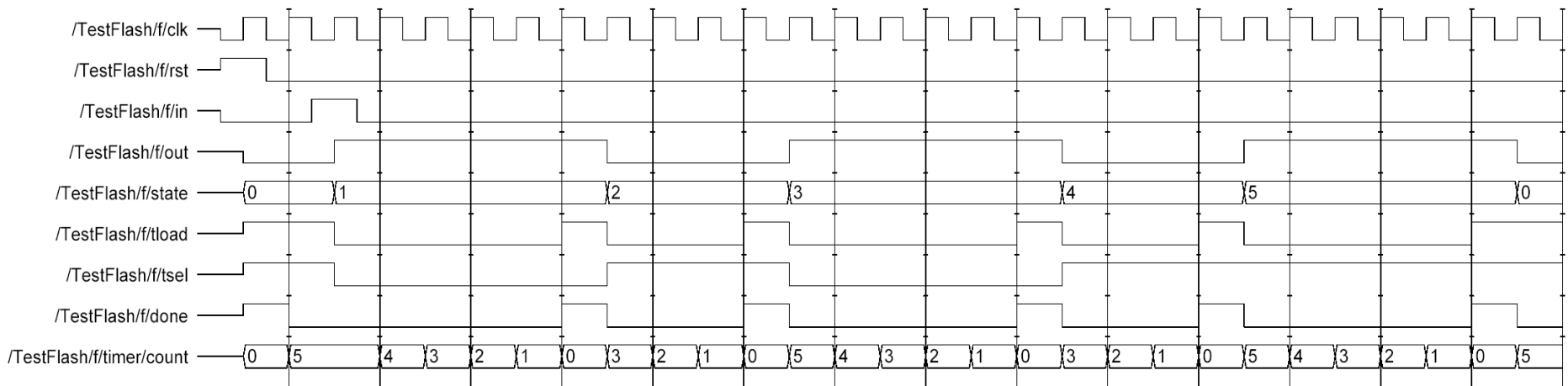
    // instantiate timer
    Timer1 timer(clk, rst, tload, tsel, done) ;

    always @* begin
        case(state)
            `S_OFF: {out, tload, tsel, next1} =
                {1'b0, 1'b1, 1'b1, in ? `S_A : `S_OFF } ;
            `S_A:   {out, tload, tsel, next1} =
                {1'b1, done, 1'b0, done ? `S_B : `S_A } ;
            `S_B:   {out, tload, tsel, next1} =
                {1'b0, done, 1'b1, done ? `S_C : `S_B } ;
            `S_C:   {out, tload, tsel, next1} =
                {1'b1, done, 1'b0, done ? `S_D : `S_C } ;
            `S_D:   {out, tload, tsel, next1} =
                {1'b0, done, 1'b1, done ? `S_E : `S_D } ;
            `S_E:   {out, tload, tsel, next1} =
                {1'b1, done, 1'b1, done ? `S_OFF : `S_E } ;
            default:{out, tload, tsel, next1} =
                {1'b1, done, 1'b1, done ? `S_OFF : `S_E } ;
        endcase
    end

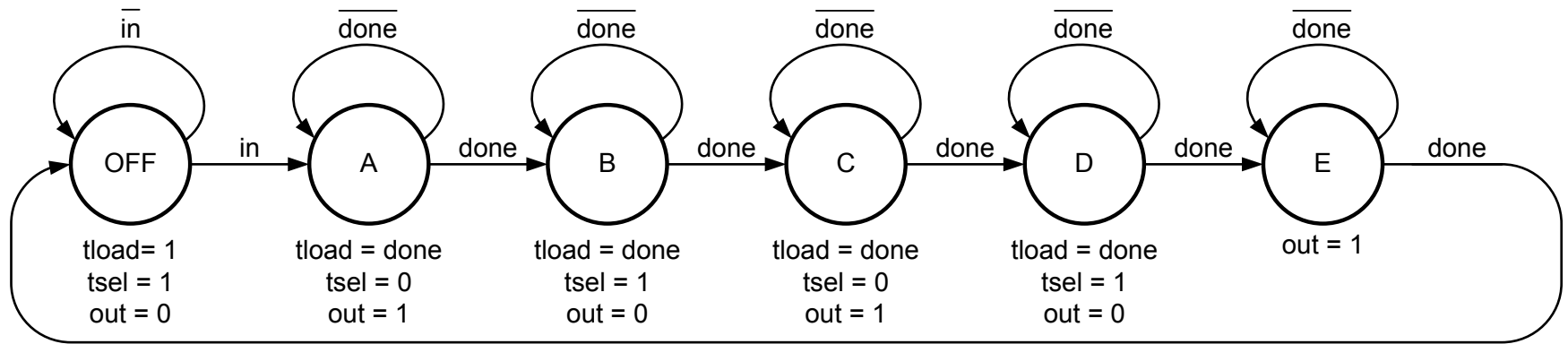
    assign next = rst ? `S_OFF : next1 ;
endmodule

```

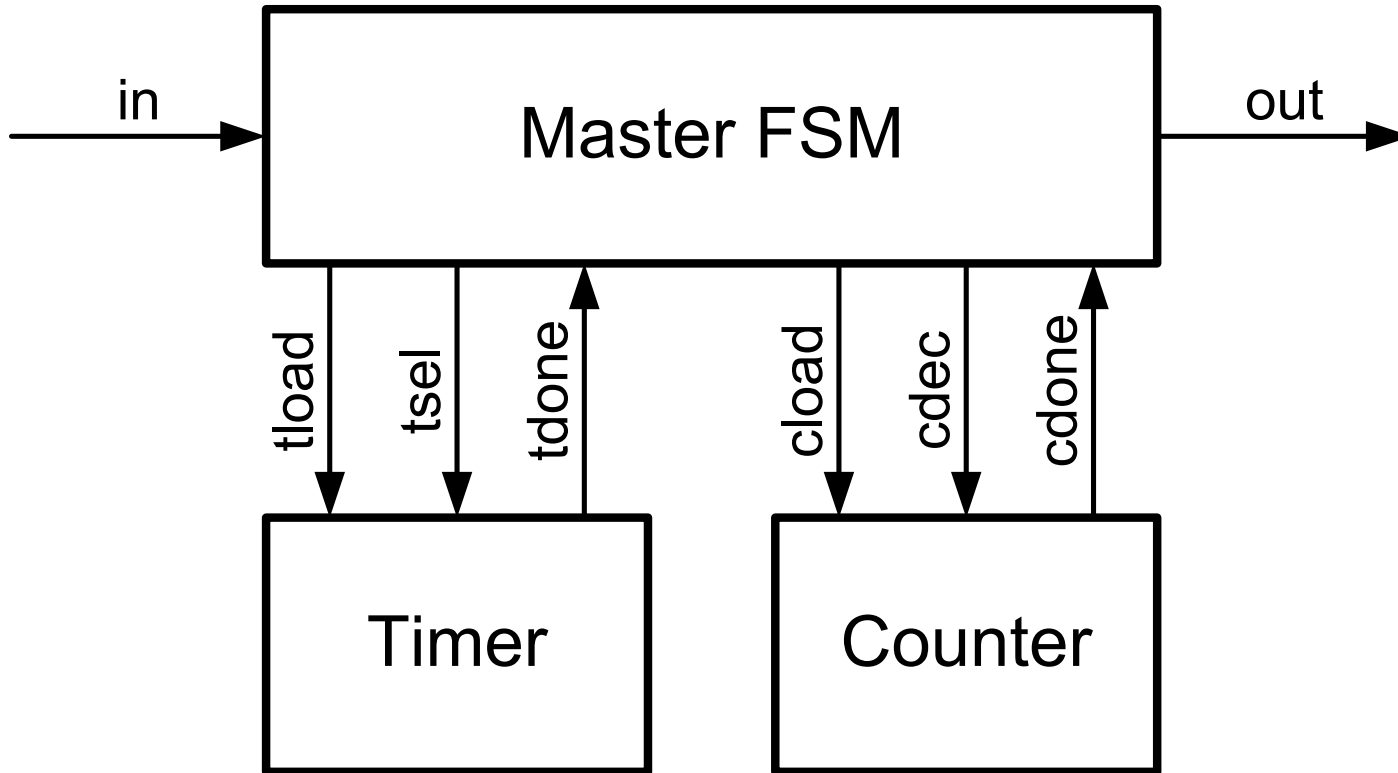
Waveforms from simulation of light-flasher FSM



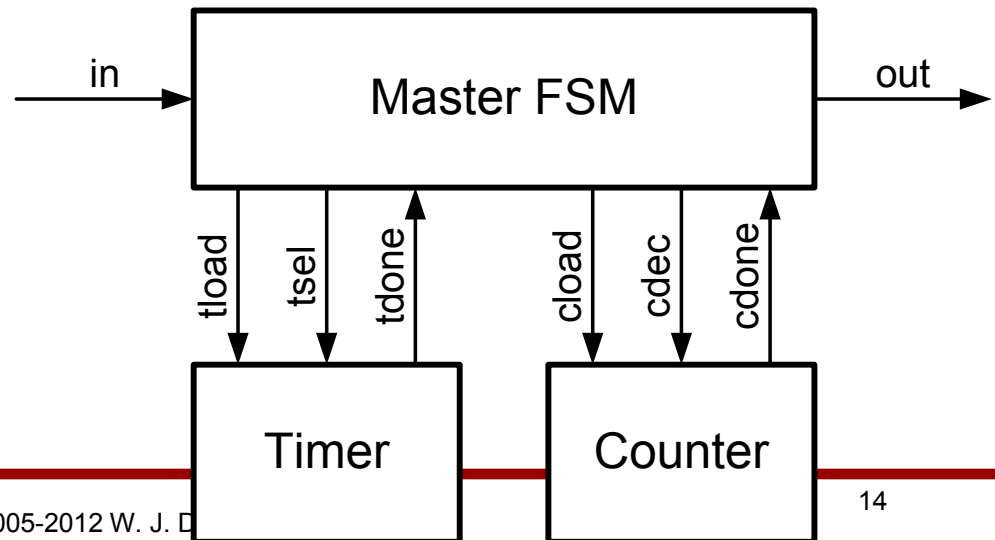
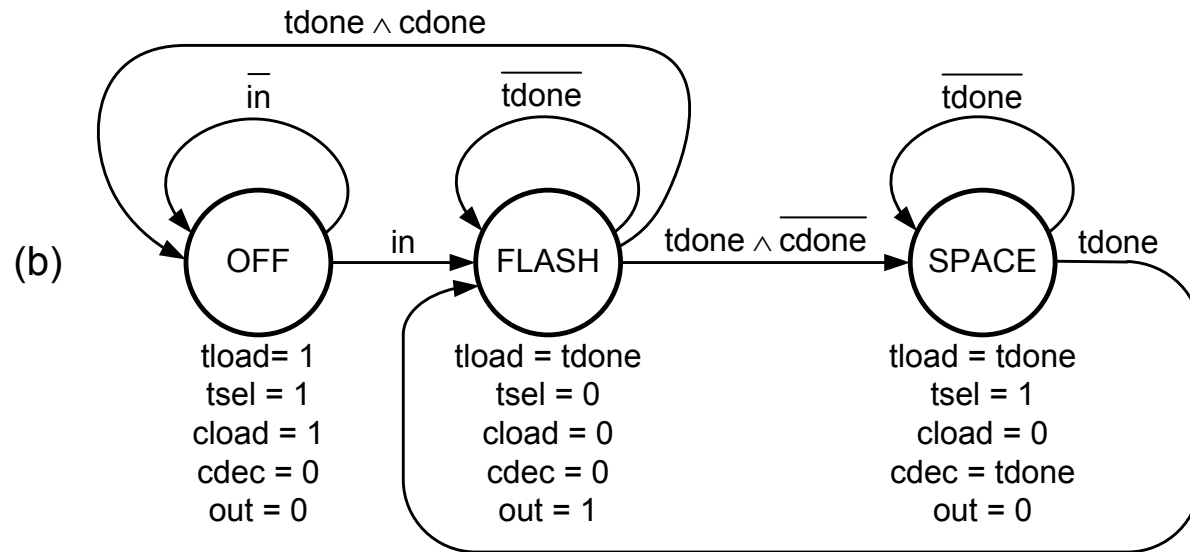
Still redundancy in state diagram



Factor out “flash number”



State diagram of twice-factored light flasher



```

module Flash2(clk, rst, in, out) ;
    input clk, rst, in ; // in triggers start of flash sequence
    output out ;          // out drives LED
    reg out ;              // output
    wire [`XWIDTH-1:0] state, next ; // current state
    reg [`XWIDTH-1:0] next1 ; // next state without reset
    reg tload, tsel, cload, cdec ; // timer and countr inputs
    wire tdone, cdone ;      // timer and counter outputs

    // instantiate state register
    DFF #(`XWIDTH) state_reg(clk, next, state) ;

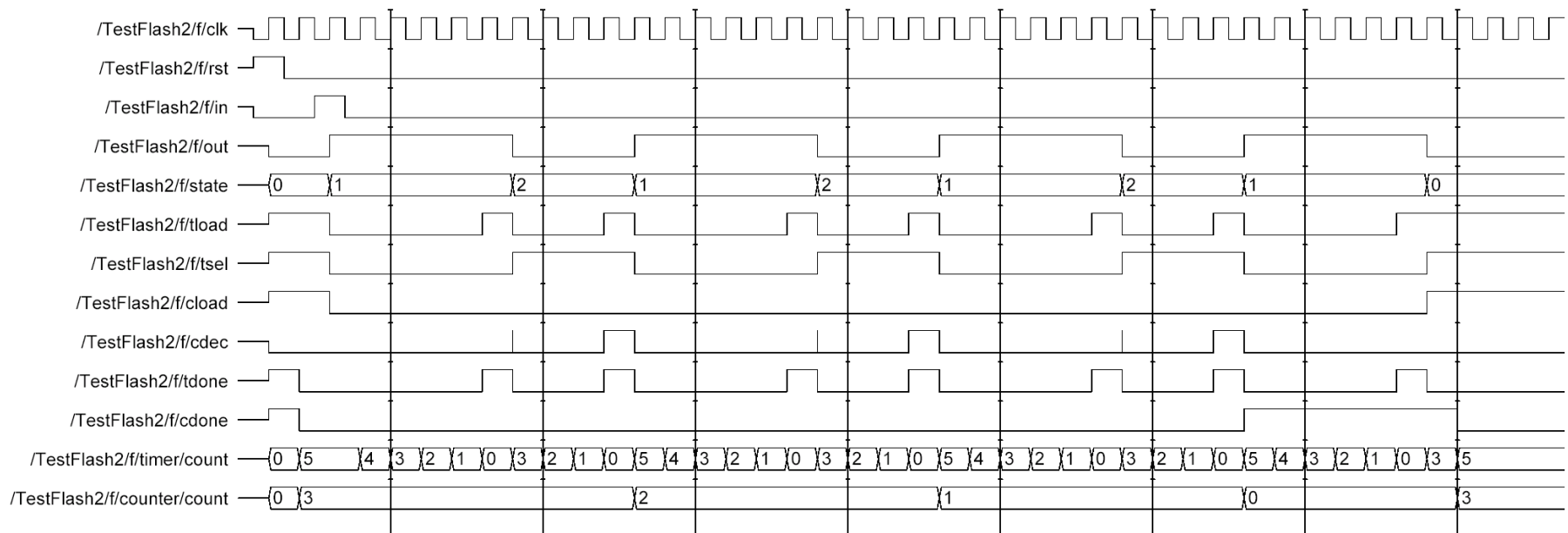
    // instantiate timer and counter
    Timer1 timer(clk, rst, tload, tsel, tdone) ;
    Counter1 counter(clk, rst, cload, cdec, cdone) ;

    always @(*) begin
        case(state)
            `X_OFF: {out, tload, tsel, cload, cdec, next1} =
                {1'b0, 1'b1, 1'b1, 1'b1, 1'b0,
                 in ? `X_FLASH : `X_OFF } ;
            `X_FLASH: {out, tload, tsel, cload, cdec, next1} =
                {1'b1, tdone, 1'b0, 1'b0, 1'b0,
                 tdone ? (cdone ? `X_OFF : `X_SPACE) : `X_FLASH } ;
            `X_SPACE: {out, tload, tsel, cload, cdec, next1} =
                {1'b0, tdone, 1'b1, 1'b0, tdone,
                 tdone ? `X_FLASH : `X_SPACE } ;
        endcase
    end

    assign next = rst ? `X_OFF : next1 ;
endmodule

```

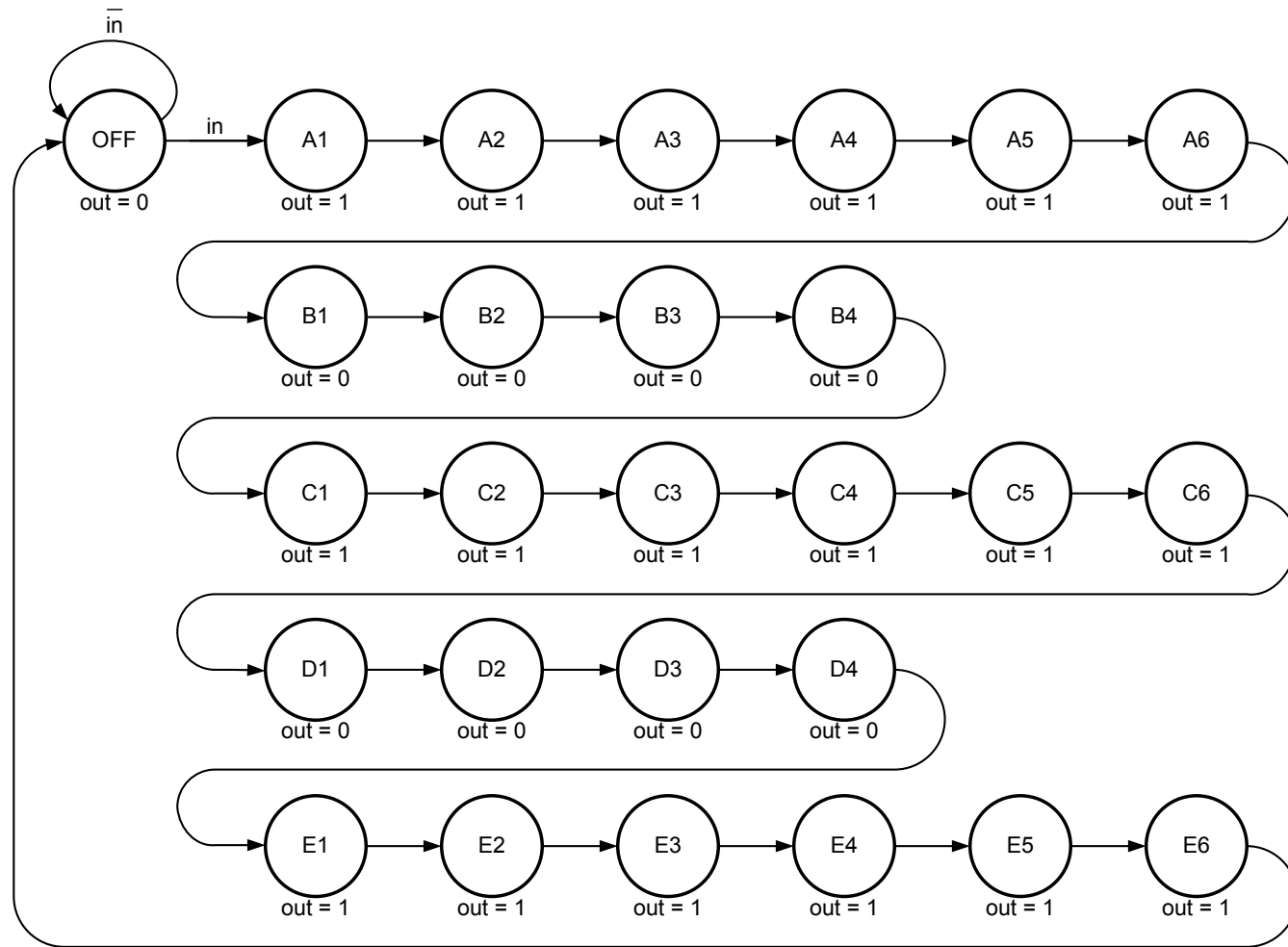
Waveforms from simulation of twice-factored light flasher



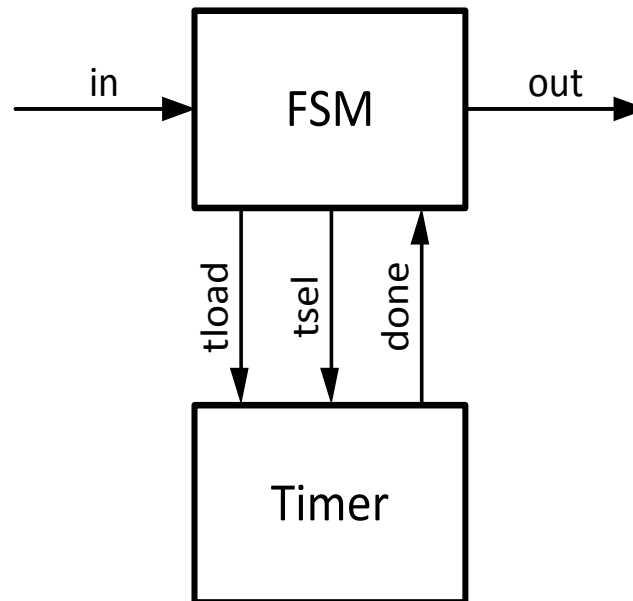
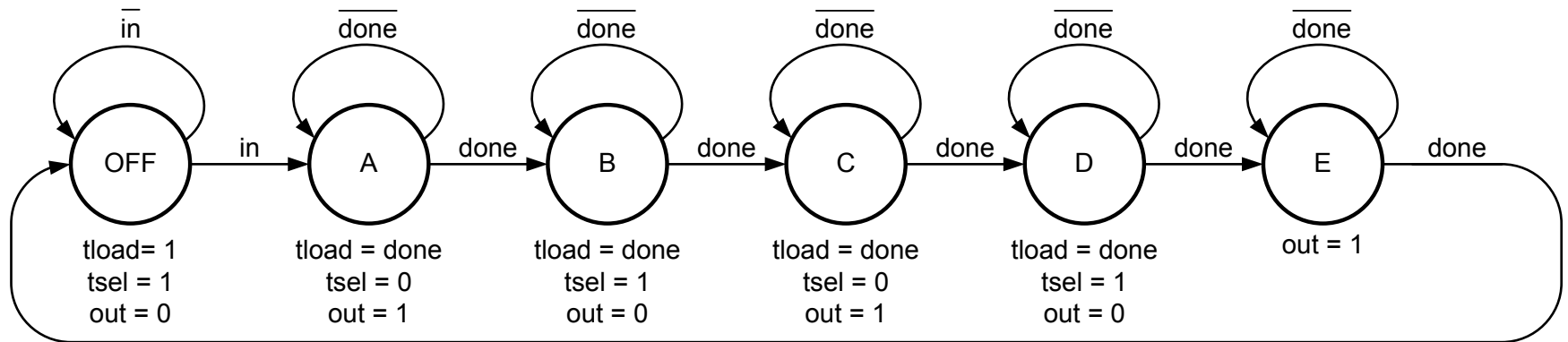
Modifying the light flasher

- Making the light's "on" state last 7 cycles
- Adding 2 more "on" states
- The 3rd "on" lasts 15 cycles
- Making the light flash 206 times, which each flash lasting an arbitrary, predetermined number of cycles

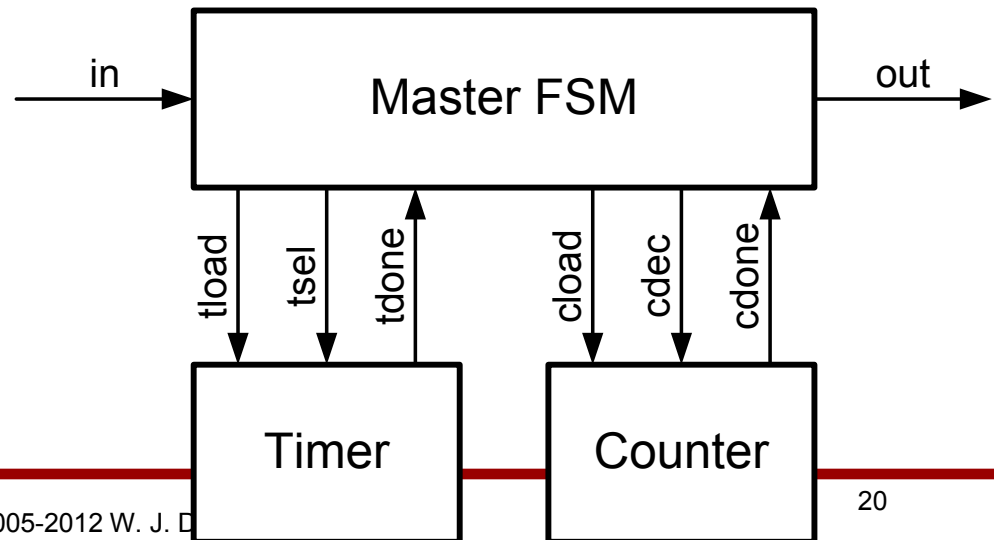
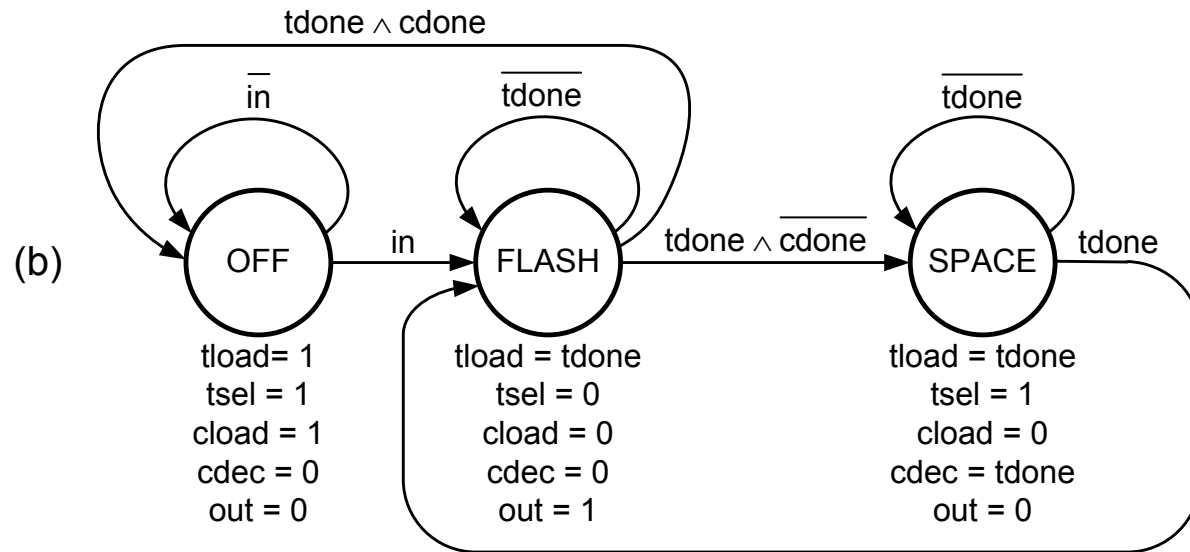
Light Flasher State Diagram



State diagram of factored light flasher



State diagram of twice-factored light flasher



Cynical view of lecture up to now

- All we've done is build a counter with three subfields
 - Each field increments when previous field “wraps”
 - Most significant field is count
 - Next is on/off
 - Least is timer
- Lets look at a less trivial problem

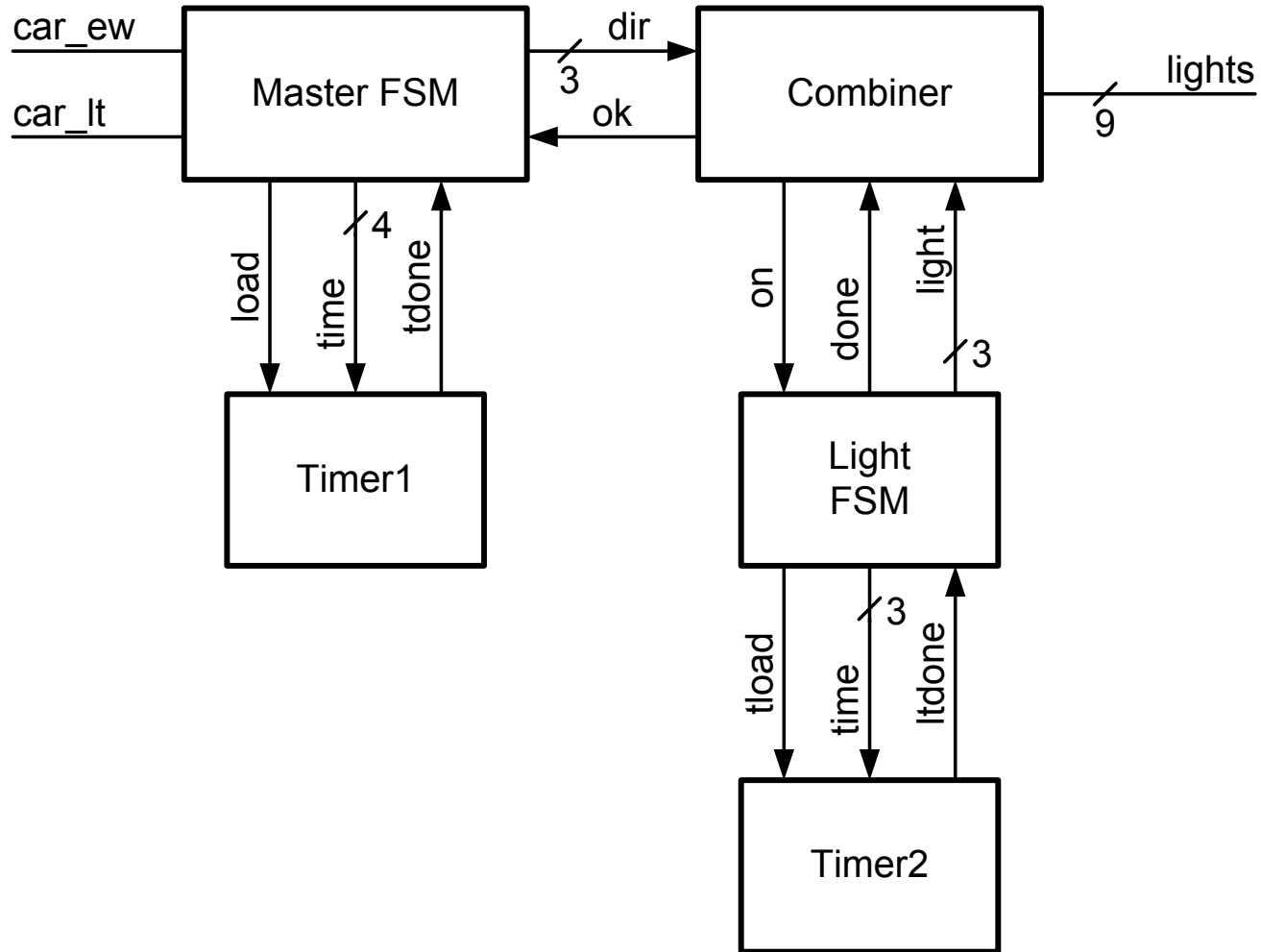
Specification Of A New Traffic-Light Controller

- Inputs: `car_ew`, `car_lt`
- Outputs: `nsgyr` `ewgyr` `ltgyr`
- Operation:
 - Default: green light for north-south road
 - If `car_lt`: green light for left-turn
 - If $\sim\text{car_lt}$ & `car_ew`: green light for east-west road
 - Green light for left-turn and east-west roads stay on until either:
 - No more cars, or
 - Green light timer expires
 - Each green, yellow, and red light stay on for a time interval
 - Lighting sequence: green -> yellow -> red

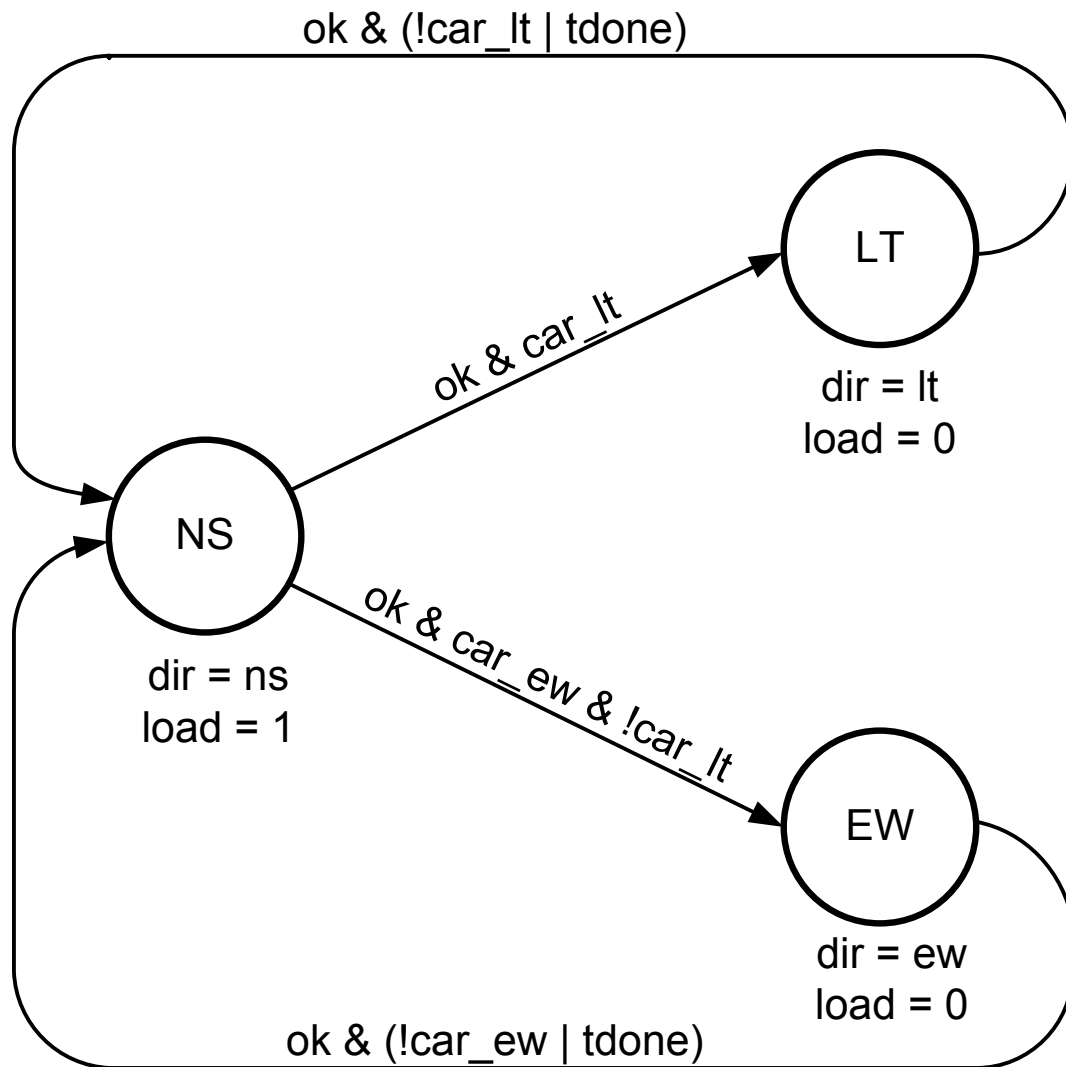
Number of states required by 'flat' machine

- Sequences for NS, EW, LT (3)
- States in each sequence Yellow, Red, Green (3)
- Time in each state 9, 5, or 4 cycles (6)
- Time remaining before return to NS (13)
- Total states $3 \times 3 \times 6 \times 13 = 702$

Block diagram of factored machine



State diagram of master FSM



```

//-----
//Master FSM
// car_ew - car waiting on east-west road
// car_lt - car waiting in left-turn lane
// ok      - signal that it is ok to request a new direction
// dir     - output signaling new requested direction
//-----
module TLC_Master(clk, rst, car_ew, car_lt, ok, dir) ;
    input clk, rst, car_ew, car_lt, ok ;
    output [1:0] dir ;

    wire [`MWIDTH-1:0] state, next ; // current state and next state
    reg  [`MWIDTH-1:0] next1 ;       // next state without reset
    reg  tload ;                     // timer load
    reg  [1:0] dir ;                  // direction output
    wire tdone ;                     // timer completion

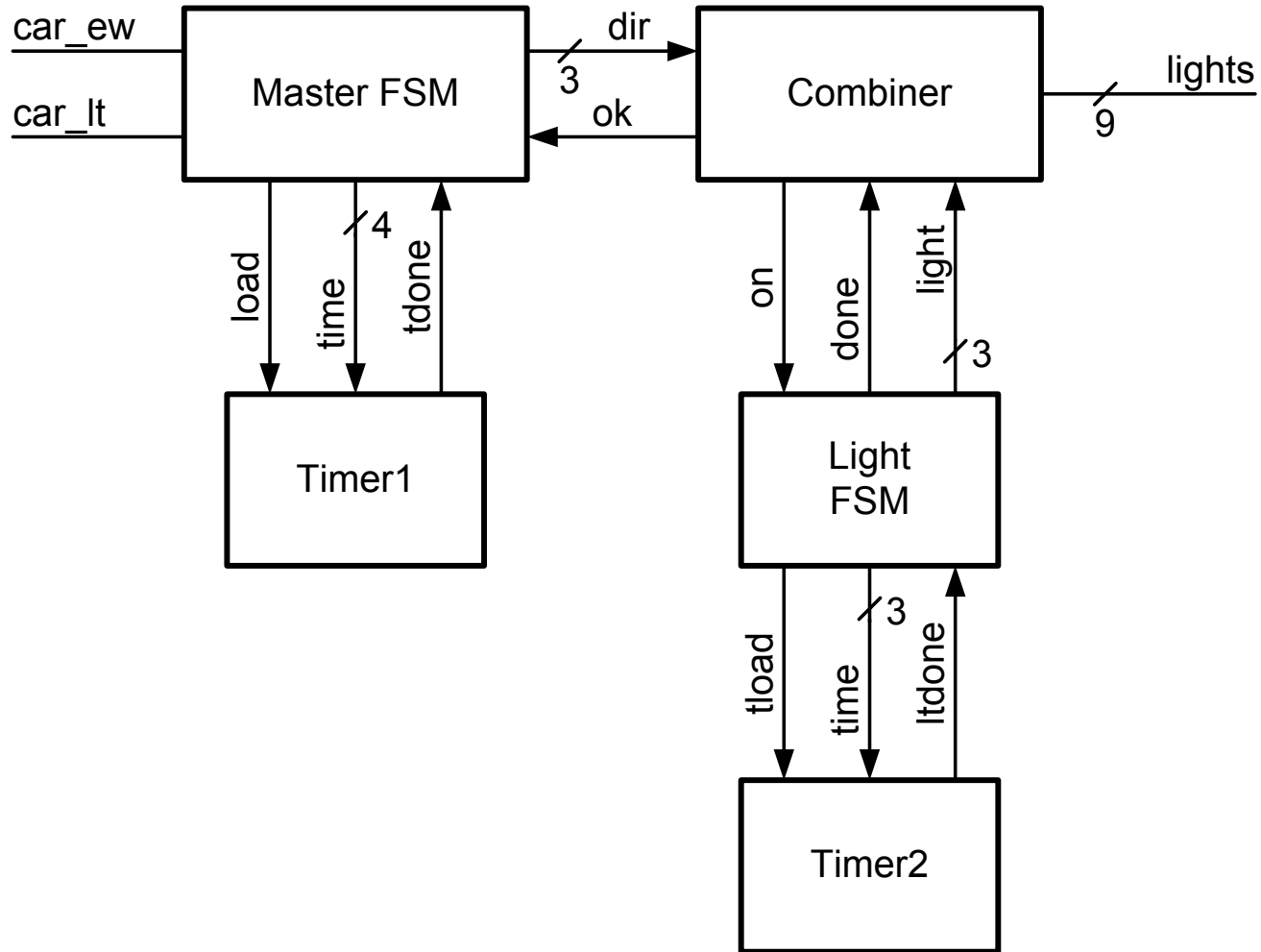
    // instantiate state register
    DFF #(`MWIDTH) state_reg(clk, next, state) ;

    // instantiate timer
    Timer #(`TWIDTH) timer(clk, rst, tload, `T_EXP, tdone) ;

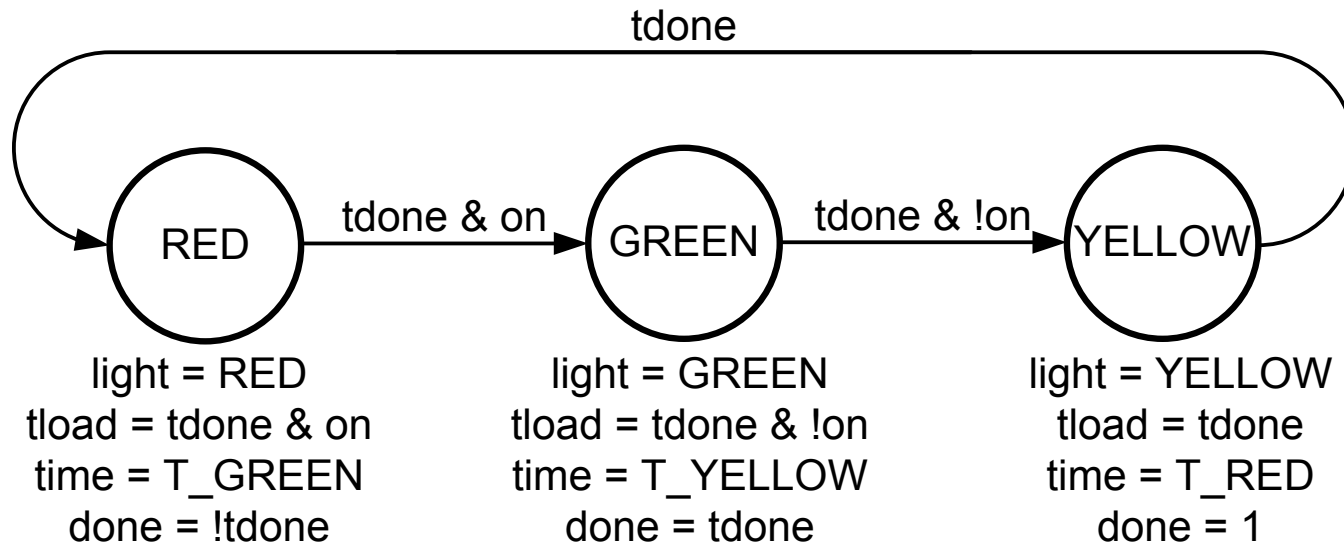
    always @(state or rst or car_ew or car_lt or ok or tdone) begin
        case(state)
            `M_NS: {dir, tload, next1} =
                {`M_NS, 1'b1, ok ? (car_lt ? `M_LT
                                     : (car_ew ? `M_EW : `M_NS))
                 : `M_NS} ;
            `M_EW: {dir, tload, next1} =
                {`M_EW, 1'b0, (ok & (!car_ew | tdone)) ? `M_NS : `M_EW} ;
            `M_LT: {dir, tload, next1} =
                {`M_LT, 1'b0, (ok & (!car_ew | tdone)) ? `M_NS : `M_LT} ;
            default: {dir, tload, next1} =
                {`M_LT, 1'b0, (ok & (!car_ew | tdone)) ? `M_NS : `M_LT} ;
        endcase
    end
    assign next = rst ? `M_NS : next1 ;
endmodule

```

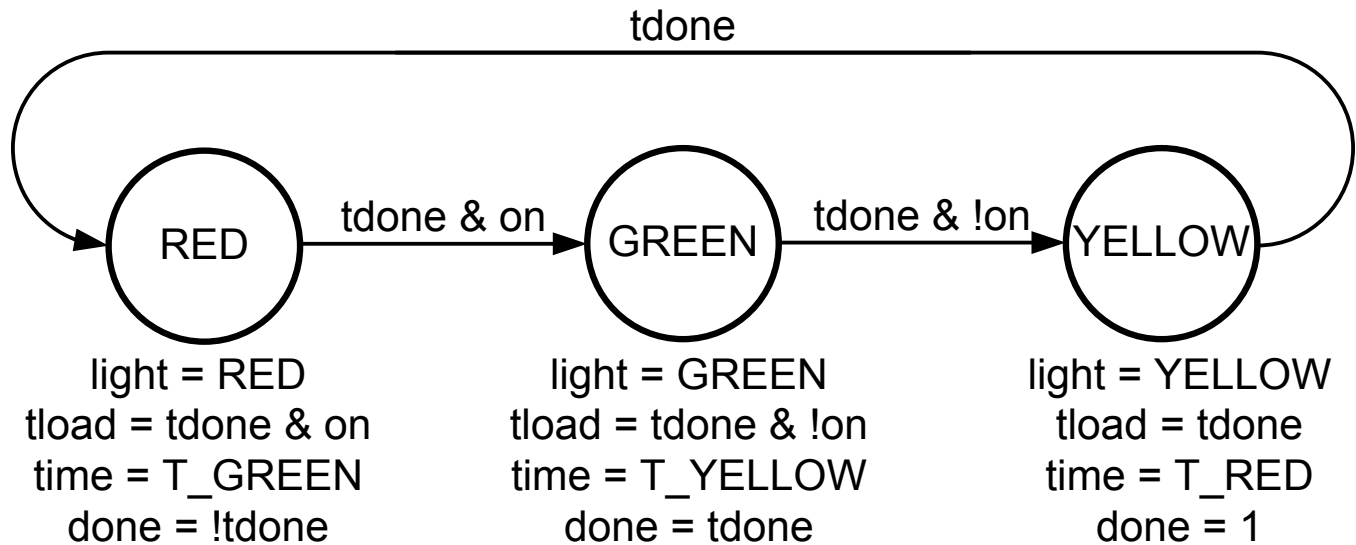
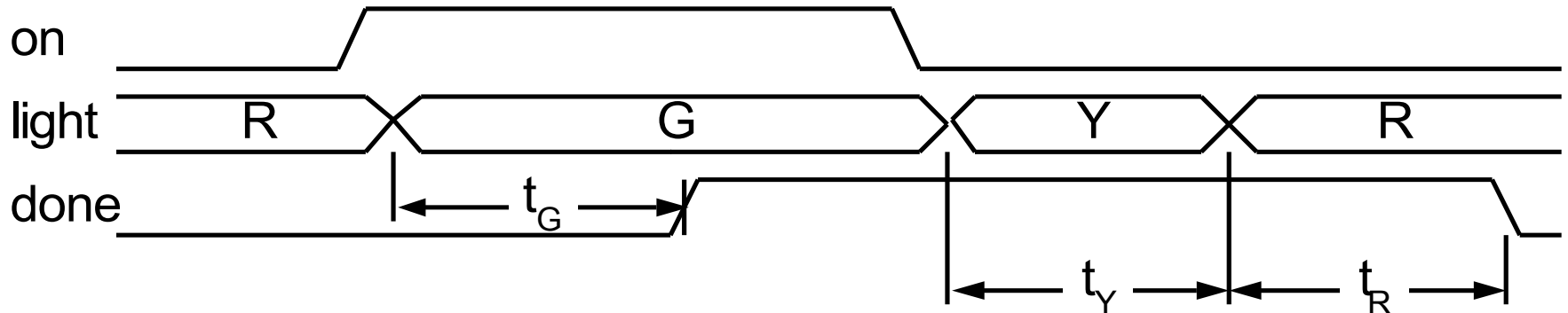
Block diagram of factored machine



State diagram of light FSM



Light “handshake”



```

module TLC_Light(clk, rst, on, done, light) ;
    input clk, rst, on ;
    output done ;
    output [2:0] light ;
    reg [2:0] light ;
    reg done ;
    wire [`LWIDTH-1:0] state, next ; // current state, next state
    reg [`LWIDTH-1:0] next1 ;        // next state w/o reset
    reg tload ;
    reg [`TWIDTH-1:0] tin ;
    wire tdone ;

    // instantiate state register
    DFF #(`LWIDTH) state_reg(clk, next, state) ;

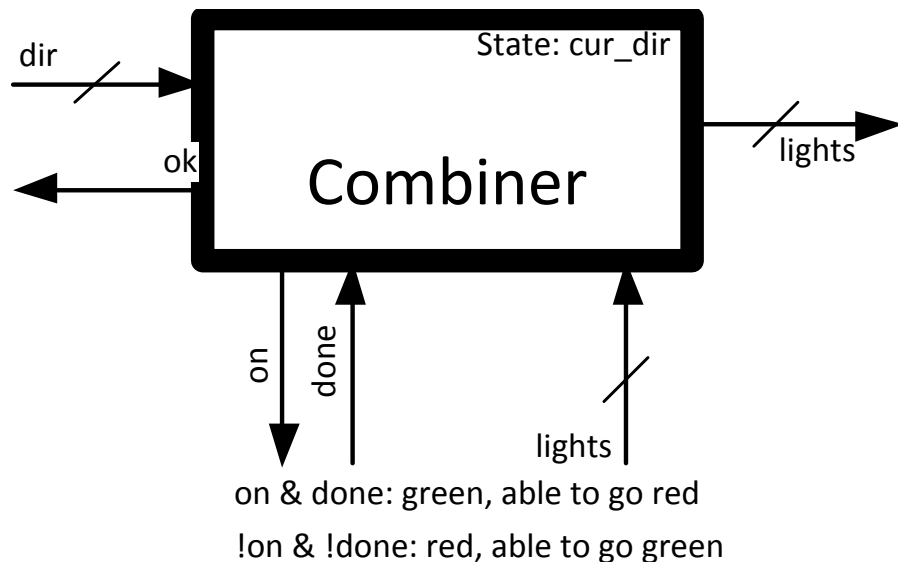
    // instantiate timer
    Timer timer(clk, rst, tload, tin, tdone) ;

    always @(state or rst or on or tdone) begin
        case(state)
            `L_RED: {tload, tin, light, done, next1} =
                {tdone & on, `T_GREEN, `RED, !tdone,
                 (tdone & on) ? `L_GREEN : `L_RED} ;
            `L_GREEN: {tload, tin, light, done, next1} =
                {tdone & !on, `T_YELLOW, `GREEN, tdone,
                 (tdone & !on) ? `L_YELLOW : `L_GREEN} ;
            `L_YELLOW: {tload, tin, light, done, next1} =
                {tdone, `T_RED, `YELLOW, 1'b1, tdone ? `L_RED : `L_YELLOW} ;
            default: {tload, tin, light, done, next1} =
                {tdone, `T_RED, `YELLOW, 1'b1, tdone ? `L_RED : `L_YELLOW} ;
        endcase
    end

    assign next = rst ? `L_RED : next1;
endmodule

```

The Combiner



- Notify master when legal to change (green, able to go red)
 - `ok = on & done`
- Trigger a red light when master changes direction
 - `on = (dir == cur_dir)`
- Update the current direction to input **dir** when all lights are red and able to go green
 - `next_dir = (!on & !done) ?
dir : cur_dir`
- Trigger a green light when `cur_dir` has been updated
 - `on = (dir == cur_dir)`
- Output red lights for the two directions that are not `cur_dir`

```

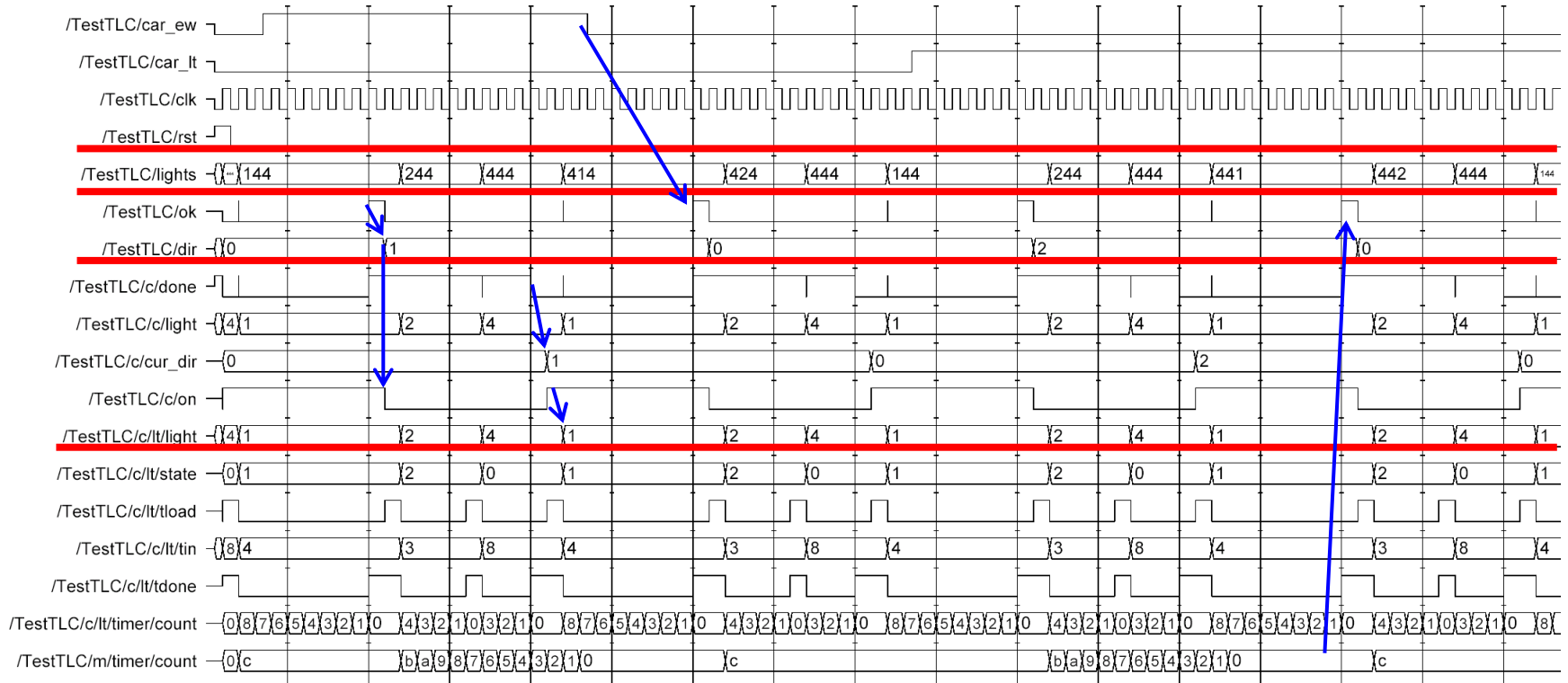
//-----
// Combiner -
//   dir - direction request from master FSM
//   ok  - acknowledge to master FSM
//   lights - 9-bits to control traffic lights {NS,EW,LT}
//-----
module TLC_Combiner(clk, rst, dir, ok , lights) ;
    input clk, rst ;
    input [1:0] dir ;
    output ok ;
    output [8:0] lights ;
    wire done ;
    wire [2:0] light ;
    reg [8:0] lights ;
    wire [1:0] cur_dir ;

    // current direction register
    DFF #(2) dir_reg(clk, next_dir, cur_dir) ;
    // light FSM
    TLC_Light lt(clk, rst, on, done, light) ;
    // request green from light FSM until direction changes
    wire on = (cur_dir == dir) ;
    // update direction when light FSM has made lights red
    wire [1:0] next_dir = rst ? 2'b0 : ((!on & !done) ? dir : cur_dir) ;
    // ok to take another change when light FSM is done
    wire ok = on & done ;

    // combine cur_dir and light to get lights
    always @(cur_dir or light) begin
        case(cur_dir)
            `M_NS: lights = {light, `RED, `RED} ;
            `M_EW: lights = `{RED, light, `RED} ;
            `M_LT: lights = `{RED, `RED, light} ;
            default: lights = `{RED, `RED, `RED} ;
        endcase
    end
endmodule

```


Waveforms from simulation of factored machine



Summary

- Factoring state machines
 - Separate state into multiple ‘orthogonal’ state variables
 - Each is simpler to handle (fewer states)
 - Total number of states is product of numbers of sub-states
 - “Factors out” repetitive sequences
 - Hierarchical structure