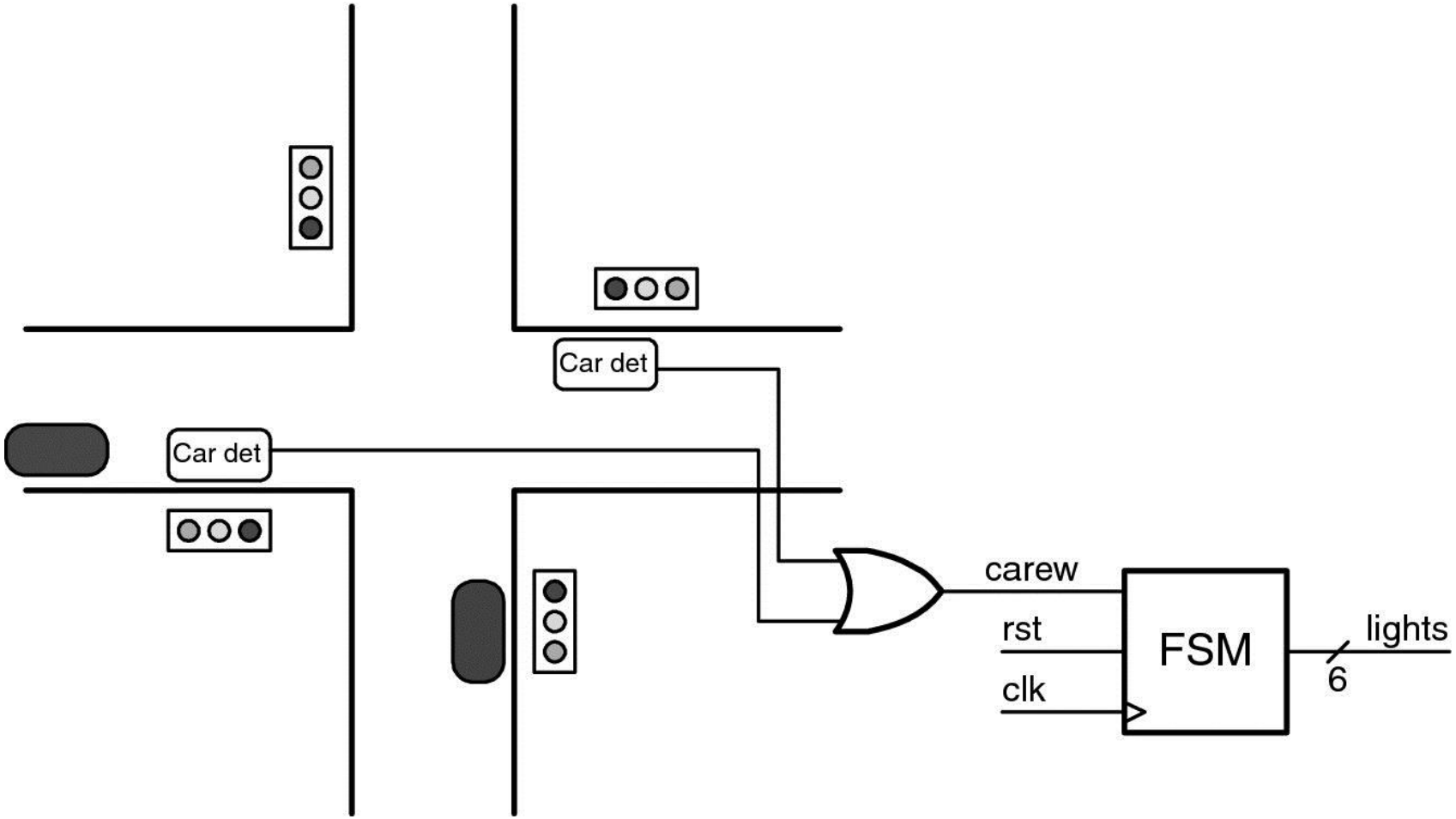
Chapter 14, Part 2

Dr. Eric Becker

CS 4341

Fall 2017

The Traffic Light Problem



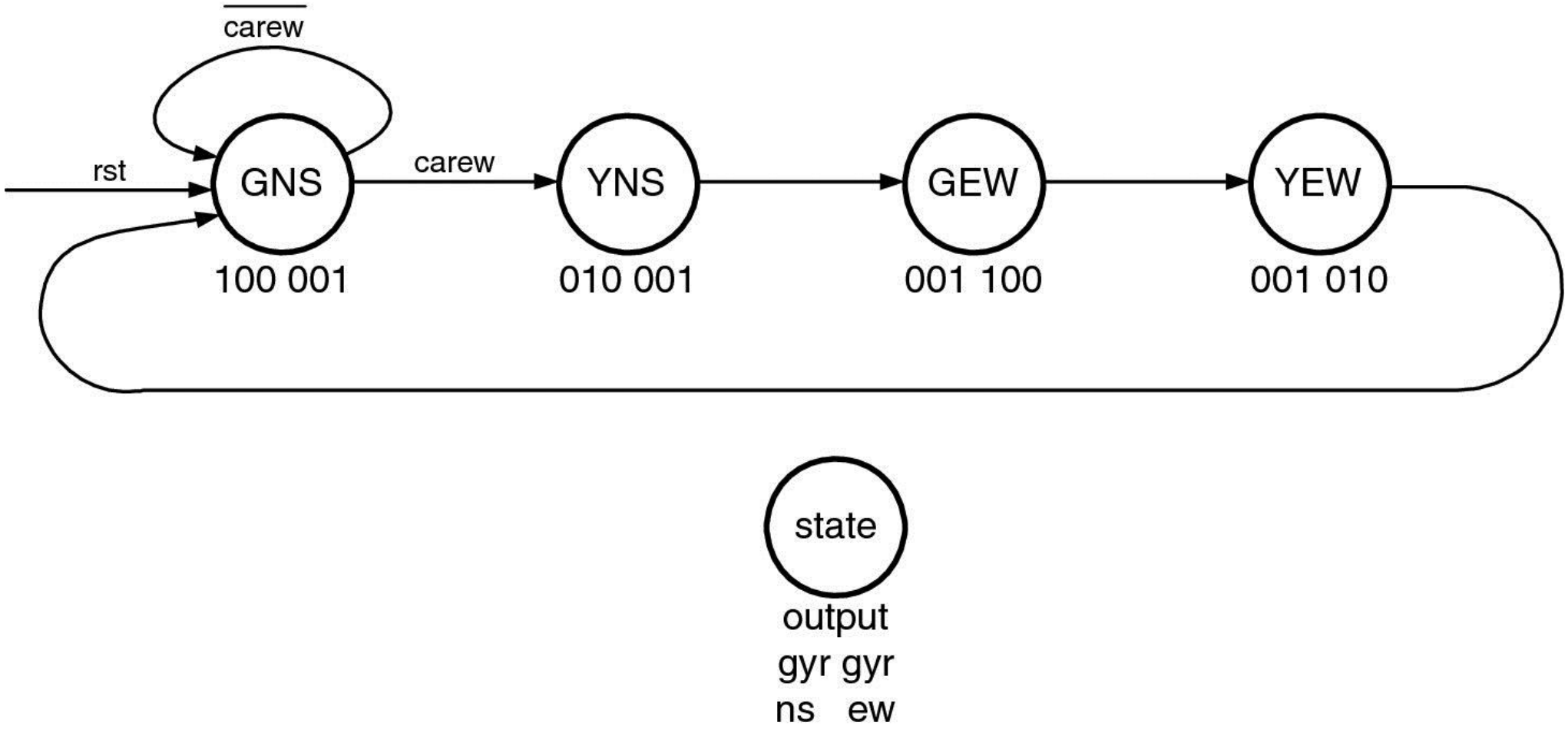
The Traffic Light Problem

- Description
  - Reset-North-south is green light, east-west is red light.
  - carew=1 means a car is detected coming from east or west. Start the sequence to change North-South to red, and east-west to green
  - A green light must change to a yellow light before changing to a red light
  - A green light can only be present if the perpendicular direction is a red light

The Traffic Light Problem: State Table

State	One-Hot	Car East-West	Next State	North-South Lights (green-yellow-red)	East-West Lights (green-yellow-red)
GNS (Green North-South)	**0001**	**0**	**GNS**	(100)	(001)
GNS	**0001**	**1**	**YNS**	(100)	(001)
YNS (Yellow North-South)	0010	0	GEW	(010)	(001)
YNS	0010	1	GEW	(010)	(001)
GEW (Green East-West)	0100	0	YEW	(001)	(100)
GEW	0100	1	YEW	(001)	(100)
YEW (Yellow East-West)	1000	0	GNS	(001)	(010)
YEW	1000	1	GNS	(001)	(010)

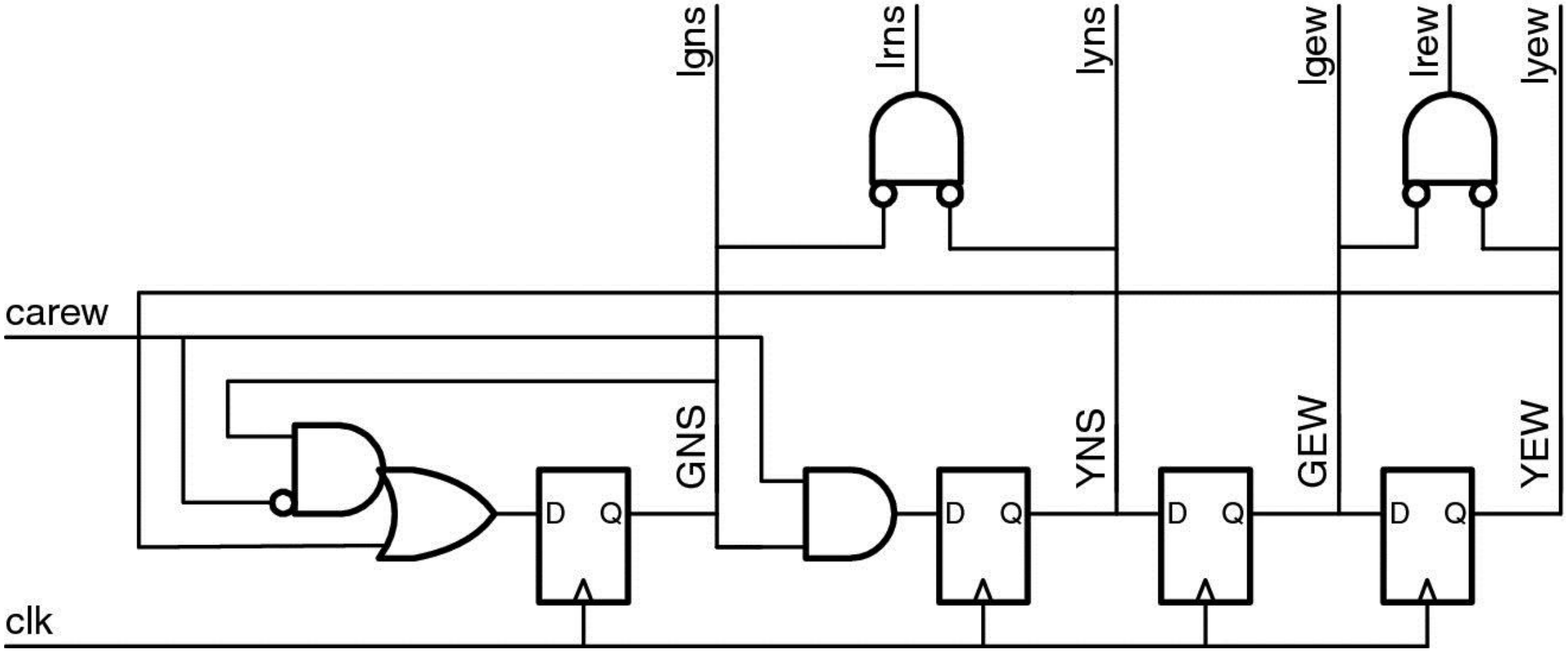
The Traffic Light Problem



The Traffic Light Problem

- Actual state is the *state variable*, as a digital encoded number
- Actual Binary Values to the FSM
  - One-hot notation for the state
  - One-bit notation for the car arriving east-west
  - Three-bit notation for the north-south lights as green-yellow-red
  - Three-bit notation for the east-west light as green-yellow-red.
- 00010100001 for
  Green North, South, Red East West, No car approaching, green light north-south, red light east-west.

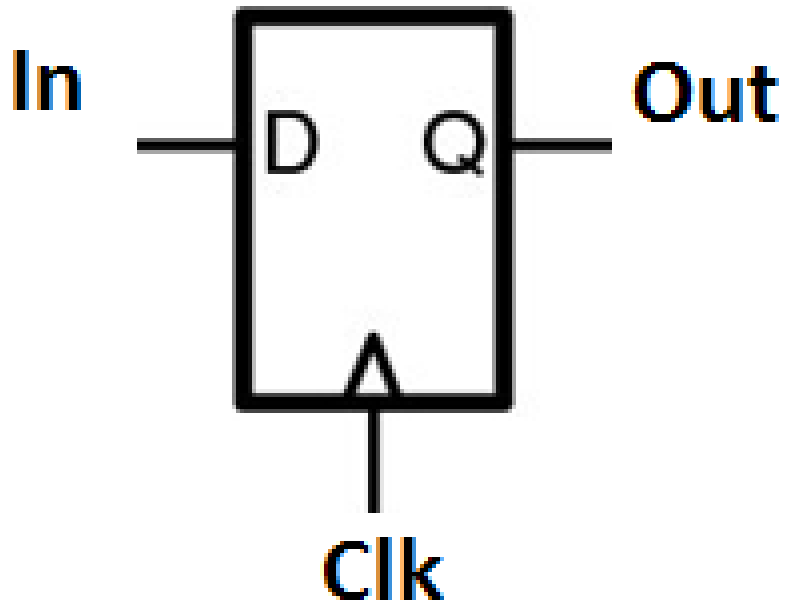
A 4-bit Traffic Light Problem: A Cirucit



Implementation of Finite State Machines

- Use a multiple flip-flops.
- A set of multiple Flip Flops (D Flip Flops) is called a *register.*
- And a *state register* is using the flip-flops to record the current state.
  - Hey, look! We've found a memory variable!
- One flip-flop per state.
  - For example, 4 states, 4 flip-flops, and its one-hot…only one flip-flop at a time will be equal to one.

The D Flip Flop?




```
module DFF(clk,in,out);
  parameter n=1;//width
  input clk;
  input [n-1:0] in;
  output [n-1:0] out;
  reg [n-1:0] out;

  always @(posedge clk)
     out = in;
endmodule
```

A 2-Bit Traffic Light Example

State	carew	Next	Comment	Light Pattern
00	0	00	GNS, carew=0	NS=(100) EW=(001)
00	1	01	GNS, carew=1	NS=(100) EW=(001)
01	0	11	YNS, carew=0	NS=(010) EW=(001)
01	1	11	YNS, carew=1	NS=(010) EW=(001)
11	0	10	GEW, carew=0	NS=(001) EW=(100)
11	1	10	GEW, carew=1	NS=(001) EW=(100)
10	0	00	YEW, carew=0	NS=(001) EW=(010)
10	1	00	YEW, carew=1	NS=(001) EW=(010)

Why 2 bit? Easier K-Mapping!

Next State, MSB

carew\S1S0	00	01	11	10
0	0	**1**	**1**	0
1	0	**1**	**1**	0

MSB=XX1

MSB=s0

Next State, LSB

carew\S1S0	00	01	11	10
0	0	1	0	0
1	1	1	0	0

LSB=X01 V 10X

LSB= ~s1s0 + carew~s1

LSB=(~s1)(~s0+carew)

LSB=$(\overline{s1}) \wedge (s0 \vee carew)$

What about the Lights?

State	NS Red	NS Yellow	NS Green
00	0	0	1
01	0	1	0
10	1	0	0
11	1	0	0

Normal form will be simple.

NS Red = $(s1 \wedge \overline{s0}) \vee (s1 \wedge s0) = s1$

NS Yellow = $(\overline{s1}) \wedge (s0)$

NS Green = $(\overline{s1}) \wedge (\overline{s0})$

State	EW Red	EW Yellow	EW Green
00	1	0	0
01	1	0	0
10	0	1	0
11	0	0	1

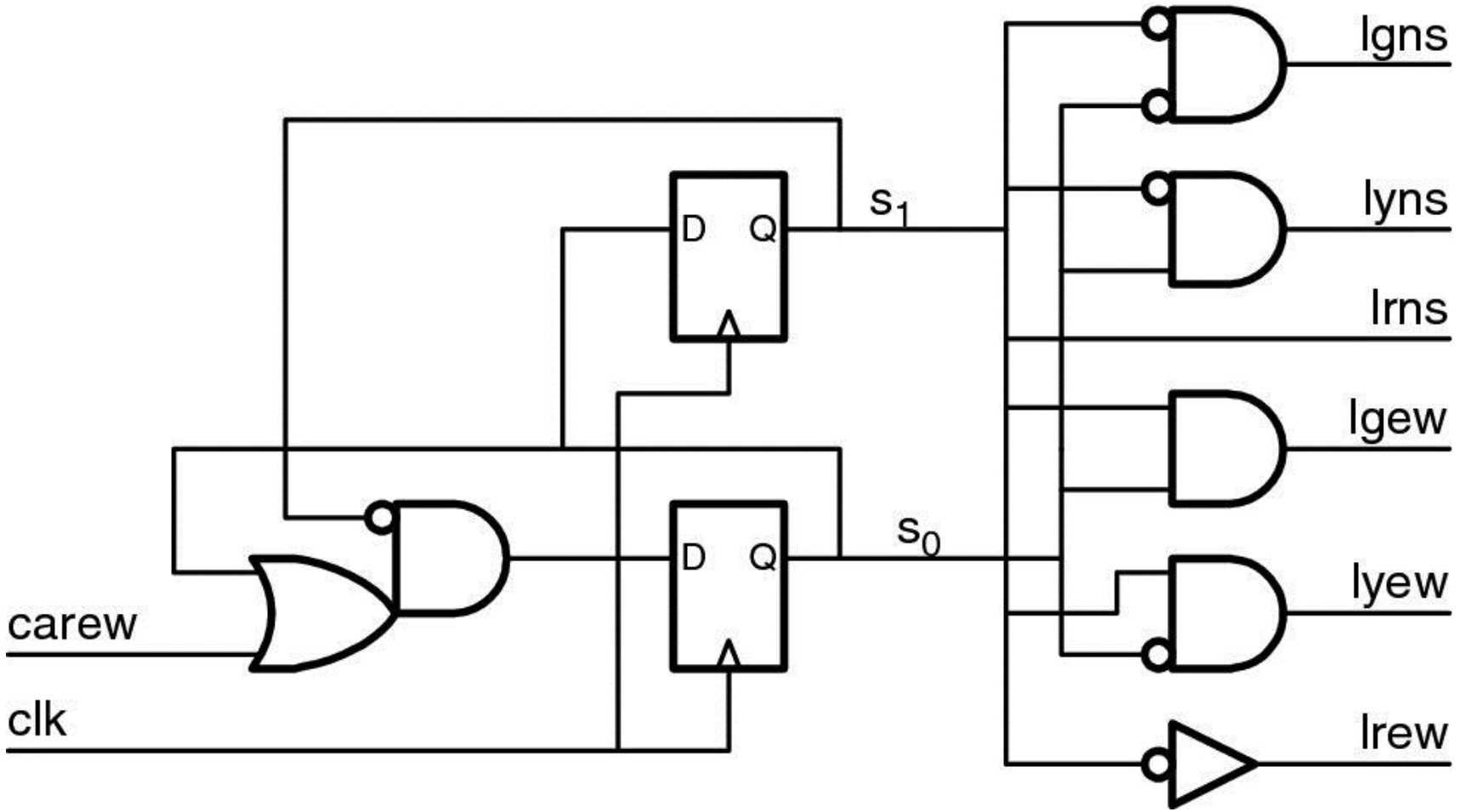
Normal form will be simple.

EW Red = $(\overline{s1} \wedge \overline{s0}) \vee (\overline{s1} \wedge s0) = \overline{s1}$

EW Yellow = $(s1) \wedge (\overline{s0})$

EW Green = $(s1) \wedge (s0)$

And the circuit would be…



Verilog

```
//----------------------
// define states
//----------------------
`define GNS 2'b00
`define YNS 2'b01
`define GEW 2'b11
`define YEW 2'b10
`define SWIDTH 2 //2Bit
```

```
//----------------------
// define output codes
//----------------------
`define GNSL 6'b100001
`define YNSL 6'b010001
`define GEWL 6'b001100
`define YEWL 6'b001010
```

Verilog

```
// Traffic_Light
// Inputs:
//   clk - system clock
//   rst - reset - high true
//   carew - car east/west - true when car is waiting in east-
west direction
// Outputs:
//   lights - (6 bits) {gns, yns, rns, gew, yew, rew}
// Waits in state GNS until carew is true,
// then sequences YNS, GEW, YEW and back to GNS.
//-----------------------------------------------------
module Traffic_Light(clk, rst, carew, lights) ;
  input clk ;
  input rst ;            // reset
  input carew ;          // car present on east-west road
  output [5:0] lights ;  // {gns, yns, rns, gew, yew, rew}
  wire [`SWIDTH-1:0] state, next ; // current and next state
  reg [`SWIDTH-1:0] next1 ;        // next state w/o reset
  reg [5:0] lights ;               // output - six lights 1=on

  // instantiate state register
  DFF #(`SWIDTH) state_reg(clk, next, state) ;
// next state and output equations - this is combinational logic
always @(*) begin
    case(state)
      `GNS: {next1, lights} = {(carew ? `YNS : `GNS), `GNSL} ;
      `YNS: {next1, lights} = {`GEW, `YNSL} ;
      `GEW: {next1, lights} = {`YEW, `GEWL} ;
      `YEW: {next1, lights} = {`GNS, `YEWL} ;
      default: {next1, lights} = {`SWIDTH+5{1'bx}};
    endcase
  end
   // add reset
   assign next = rst ? `GNS : next1 ;
endmodule // Traffic_Light
```

Verilog

```
module Test_Fsm1 ;
  reg clk, rst, carew ;
  wire [5:0] lights ;

  Traffic_Light tl(clk, rst, carew, lights) ;

  // clock with period of 10 units
  initial begin
    clk = 1 ; #5 clk = 0 ;
    forever
      begin
        $display("%b %b %b %b", rst, carew,
tl.state, lights ) ;
        #5 clk = 1 ; #5 clk = 0 ;
      end
    end

  // input stimuli
  initial begin
    rst = 0 ; carew=0 ;
    #15 rst = 1 ; carew = 0 ;  // reset
    #10 rst = 0 ;   // remove reset
    #20 carew = 1 ; //car arrives
    #30 carew = 0 ; // car leaves
    #20 carew = 1   // car comes and stays
    #60             // 6 more cycles
    $stop ;
  end
endmodule
```