Microprocessor System Design

ECE 485/585

Mark G. Faust

Microprocessor System Design

Introduction

Part of Two Quarter Sequence

- ECE 485/585 Microprocessor System Design
  - System Issues
    - Memory
    - Caches
    - Virtual Memory
    - Buses
    - I/O Devices: Disks
    - Reliability

Not a course on microprocessors (ECE 371)
Not a course on interfacing (ECE 372)
Assumes familiarity with at least one microprocessor
Comfortable with (generic) assembly language

- ECE 486/586 Computer Architecture
  - Processor Oriented
    - CPU Datapath and Control
    - Computer Arithmetic
    - Pipelining
    - Branch Prediction
    - Performance Measurement

Quick Survey

- Taken an operating systems course?
- Which microprocessor family are you familiar with?
  - Intel x86
  - ARM
  - MIPS
  - Other
- C/C++ Programming
- Verilog?   VHDL?
- Have you heard of, can you define/explain…
  - Two way set associative cache
  - Demand paged virtual memory
  - Vectored interrupt
  - TLB

Are caches desirable?

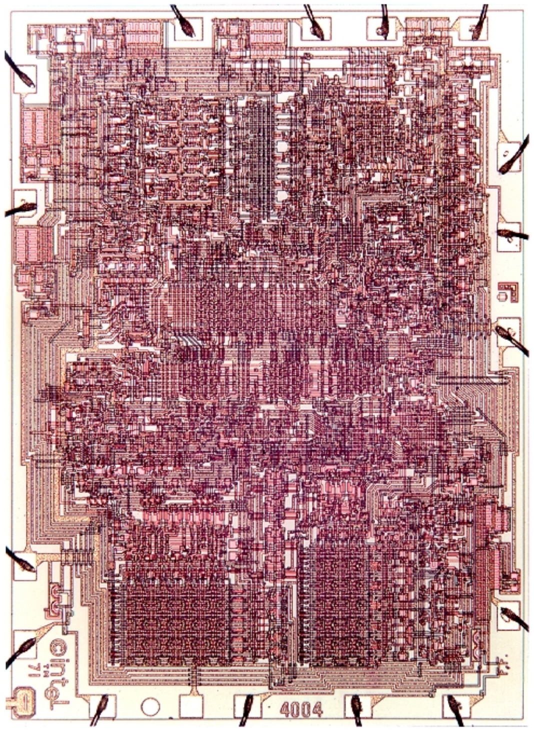
Computer architecture doesn't arise in a vacuum. It's a consequence of the constraints imposed by the real world technology and components (e.g. microprocessors, memory, disk, I/O devices) that are available (e.g. memory) or demanded by the market (e.g. wireless, graphics). For example, if you had terabytes of persistent storage with access times of 1ns you'd have little need for L1, L2 caches, swapping, etc.

Just like building architecture, we're dealing with the materials (components, subsystems) . The pyramids weren't made of timber. So, you'll find architectural decisions from past designs that wouldn't necessarily be made the same way today.

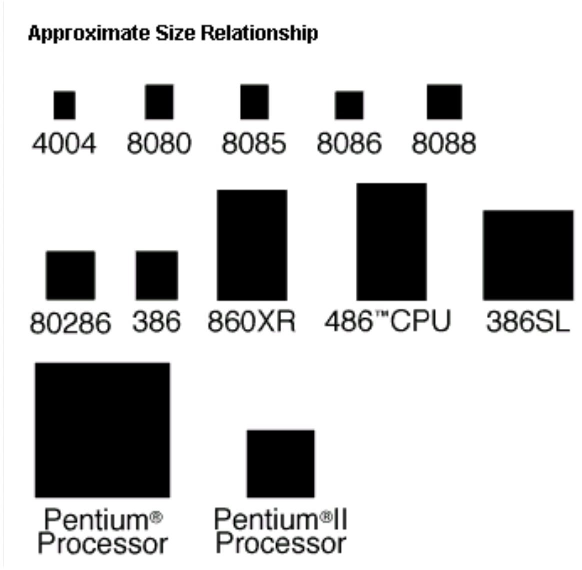
This Course

- No textbook
  - No single book or even half dozen books covers the content
  - Some content is so current no textbooks cover the material yet
    - Trade-off between timely publication and accuracy
- Will rely on
  - Lecture slides
  - Links to articles, websites, datasheets
- Lots of real world examples
  - Latest technology
  - Not always
    - Concepts sometimes more easily introduced and understood in context of older, simpler systems
    - Same techniques often used in more recent systems
    - In some cases the functionality is preserved intact
    - Studying both older and newer systems allows us to see the evolution of features over time and understand the reasons for that evolution

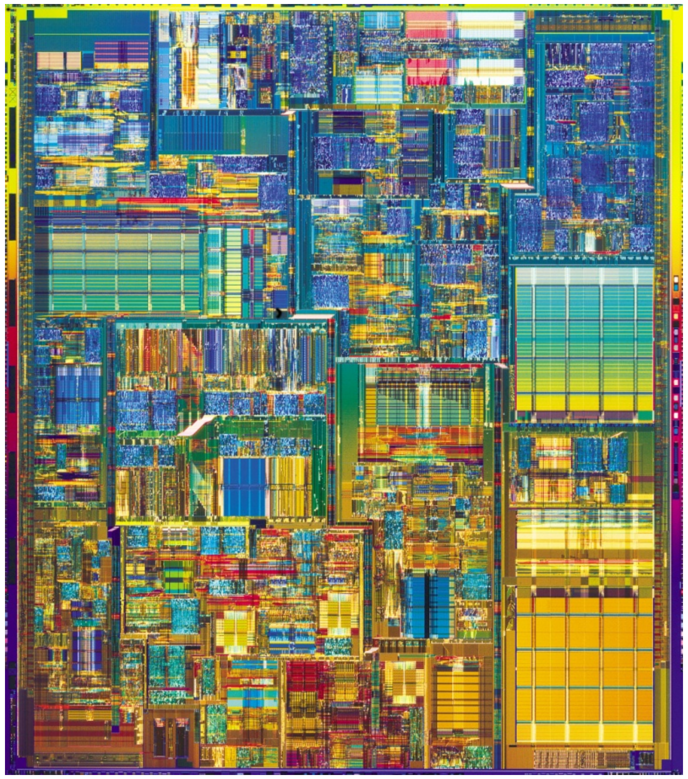
Evolution of Intel Microprocessors



Intel 4004 (November 1971)
[2,300 transistors]



8086: 29,000
Xeon: 286,000,000



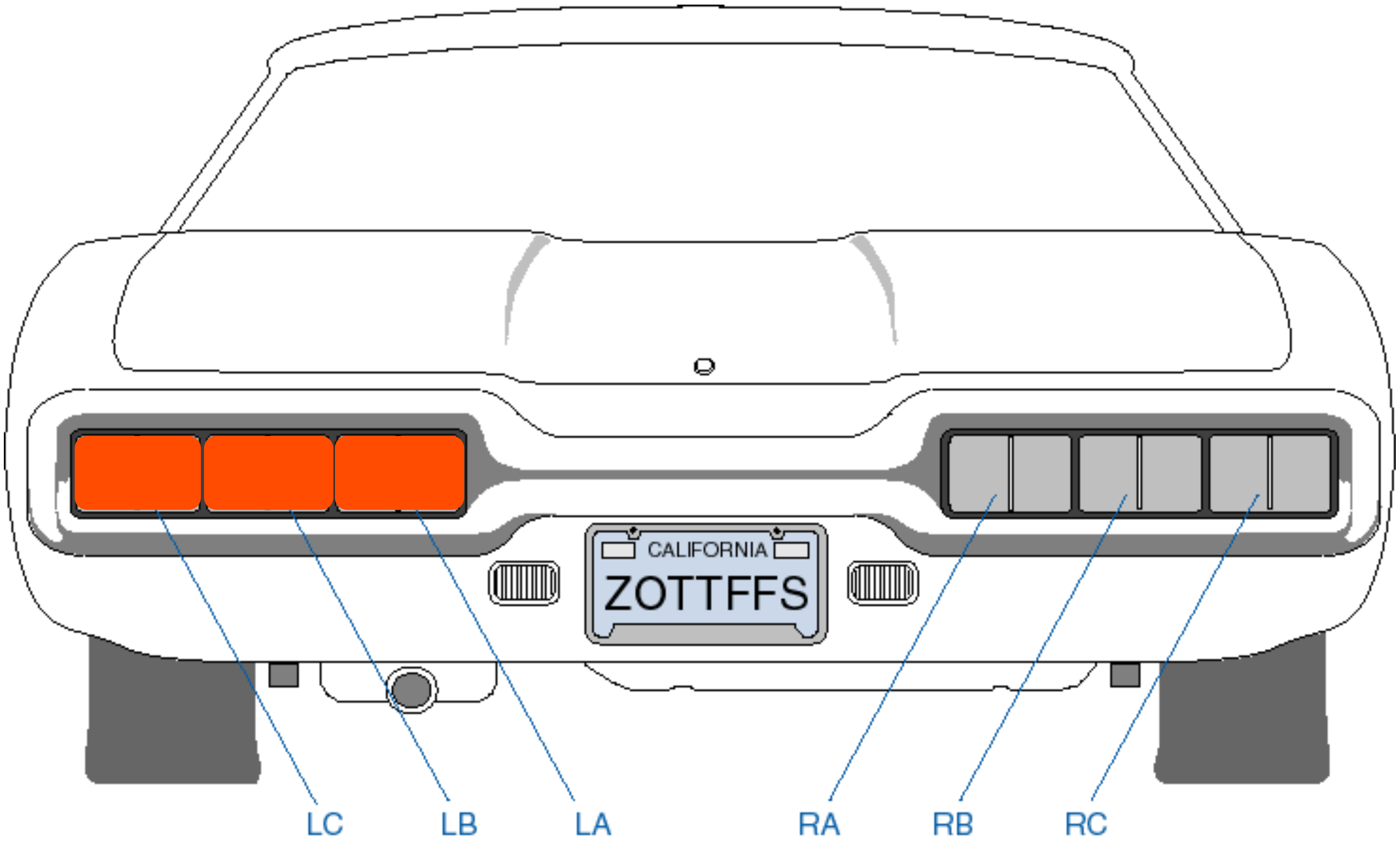
Intel Pentium P4
[55,000,000 transistors]

Class Web Site

www.ece.pdx.edu/~faustm/ece485

Finite State Machines Review

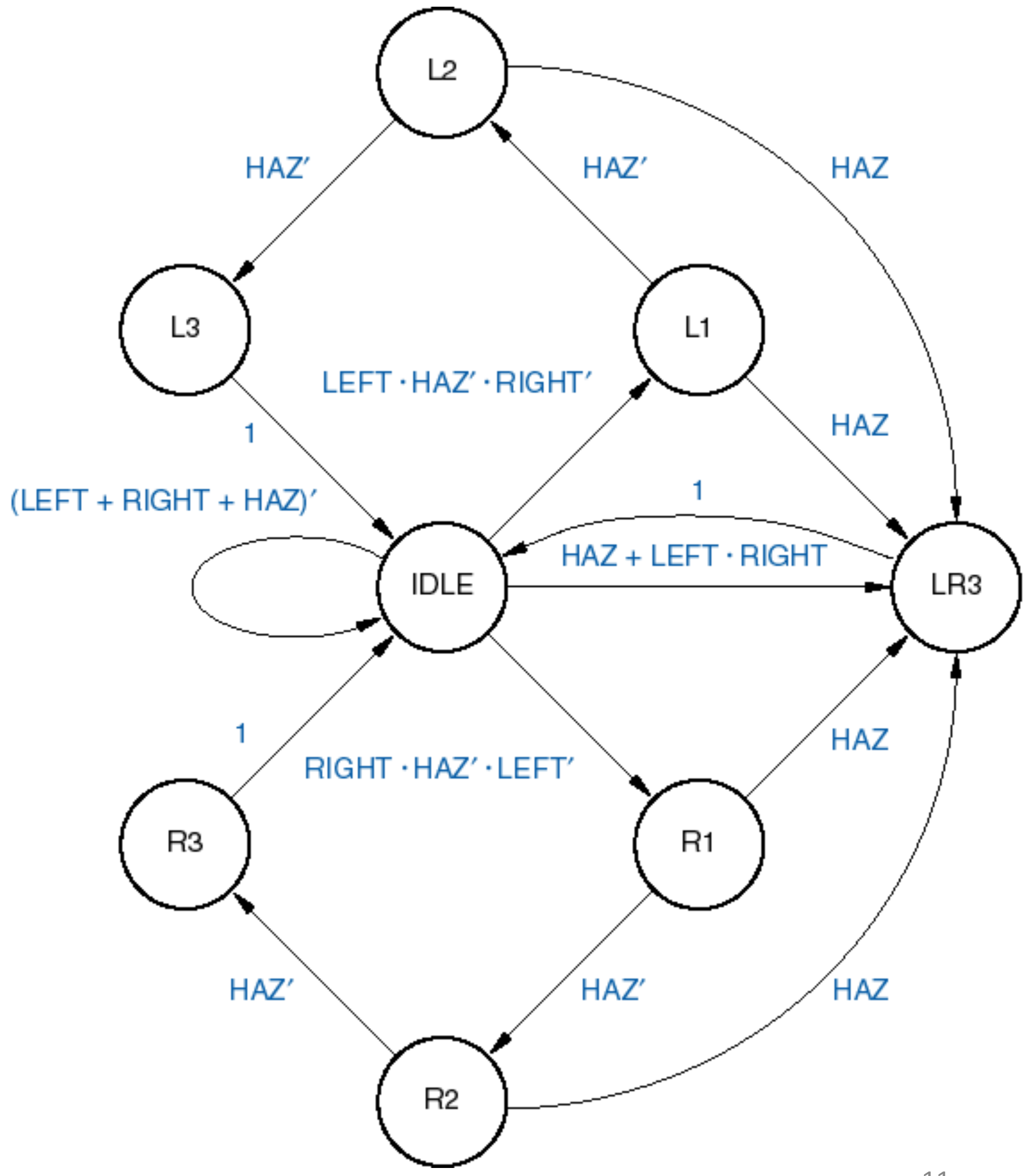
T-bird tail-lights example



State diagram

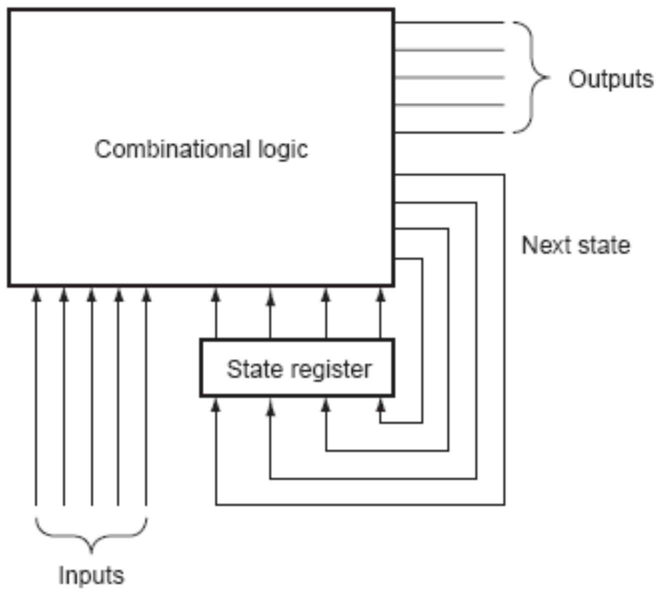
Inputs:
LEFT, RIGHT, HAZ

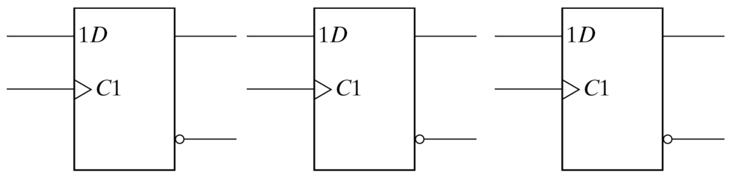
Outputs:
Six lamps
(function of state only)



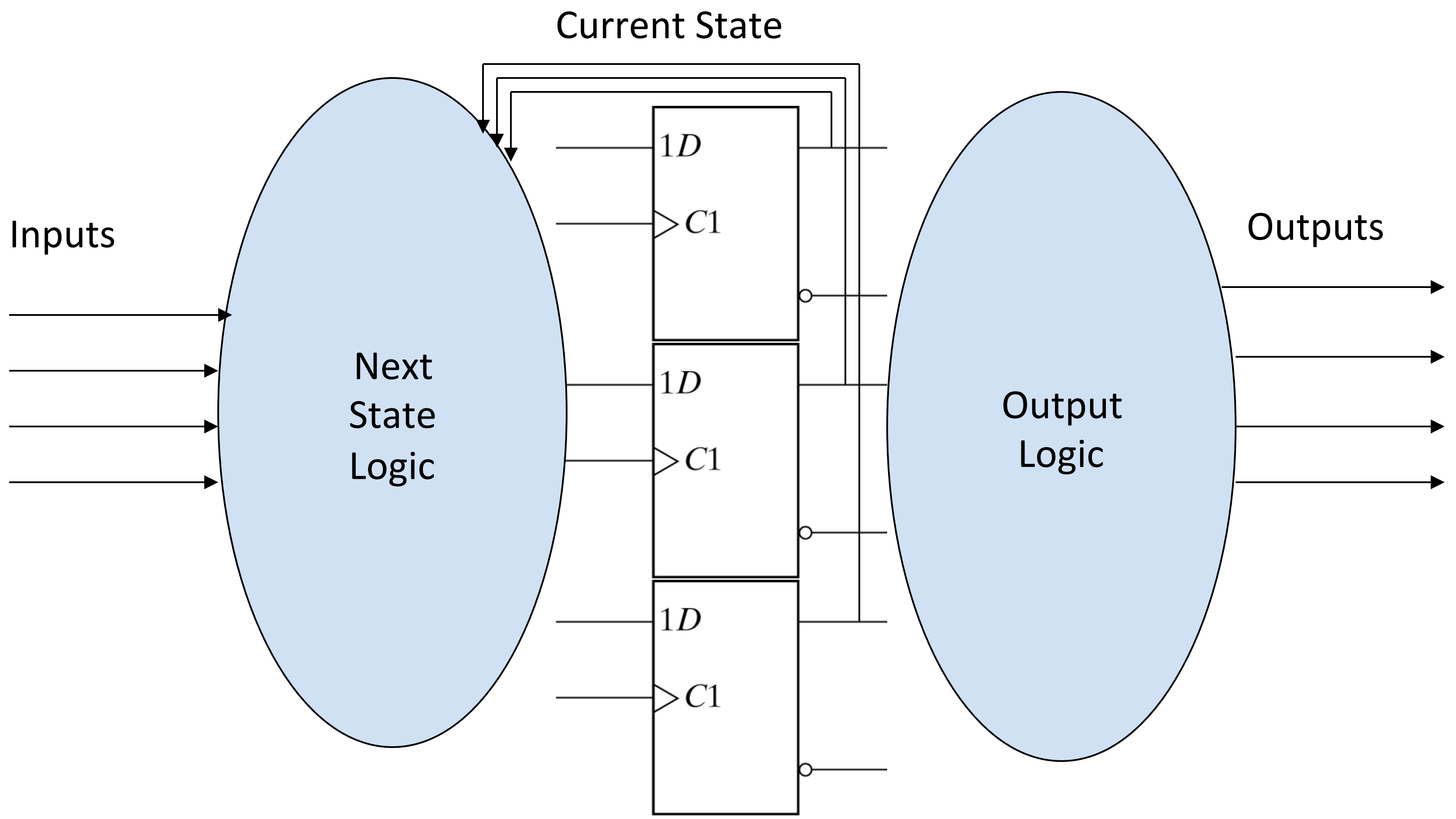
Encoded or One-Hot?

- Encoded
  - 8 states
  - $2^3 = 8$
  - Need 3 flip flops
  - Need to determine state assignment
- One-hot
  - Dedicate a flip flop per state
  - Need 8 flip flops



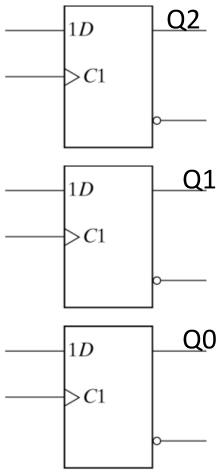


Implementation (Encoded, Moore Machine)



Output logic

State	LC	LB	LA	RA	RB	RC
IDLE	0	0	0	0	0	0
L1	0	0	1	0	0	0
L2	0	1	1	0	0	0
L3	1	1	1	0	0	0
R1	0	0	0	1	0	0
R2	0	0	0	1	1	0
R3	0	0	0	1	1	1
LR3	1	1	1	1	1	1



State	Q2	Q1	Q0
IDLE	0	0	0
L1	0	0	1
L2	0	1	1
L3	0	1	0
R1	1	0	1
R2	1	1	1
R3	1	1	0
LR3	1	0	0

LC = L3 + LR3

LB = L2 + L3 + LR3

LA = L1 + L2 + L3 + LR3

RA = R1 + R2 + R3 + LR3

RB = R2 + R3 + LR3

RC = R3 + LR3

LC = Q2’·Q1·Q0’ + Q2·Q1’·Q0’

LB = Q2’·Q1·Q0 + Q2’·Q1·Q0’ + Q2·Q1’·Q0’

LA = Q2’·Q1’·Q0 + Q2’·Q1·Q0 + Q2’·Q1·Q0’ + Q2·Q1’·Q0’

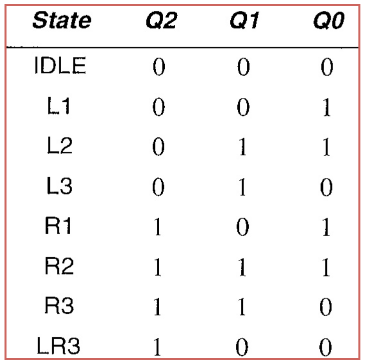
RA = Q2·Q1’·Q0 + Q2·Q1·Q0 + Q2·Q1·Q0’ + Q2·Q1’·Q0’

RB = Q2·Q1·Q0 + Q2·Q1·Q0’ + Q2·Q1’·Q0’

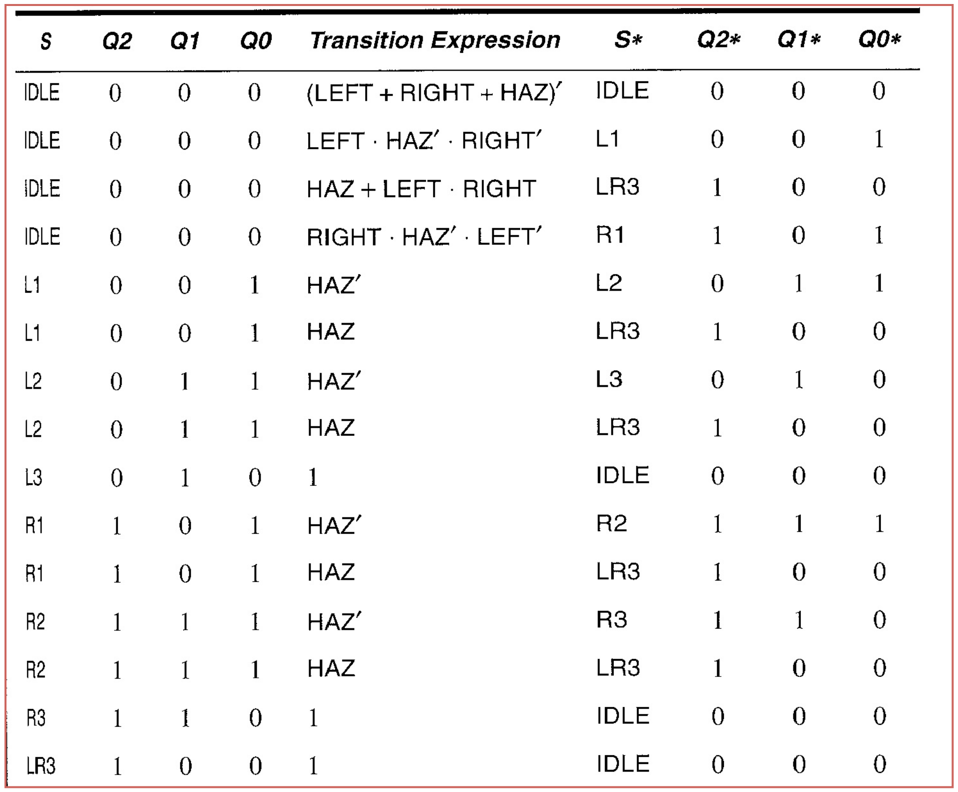
RC = Q2·Q1·Q0’ + Q2·Q1’·Q0’

Next State Logic

- State transition table for encoded states



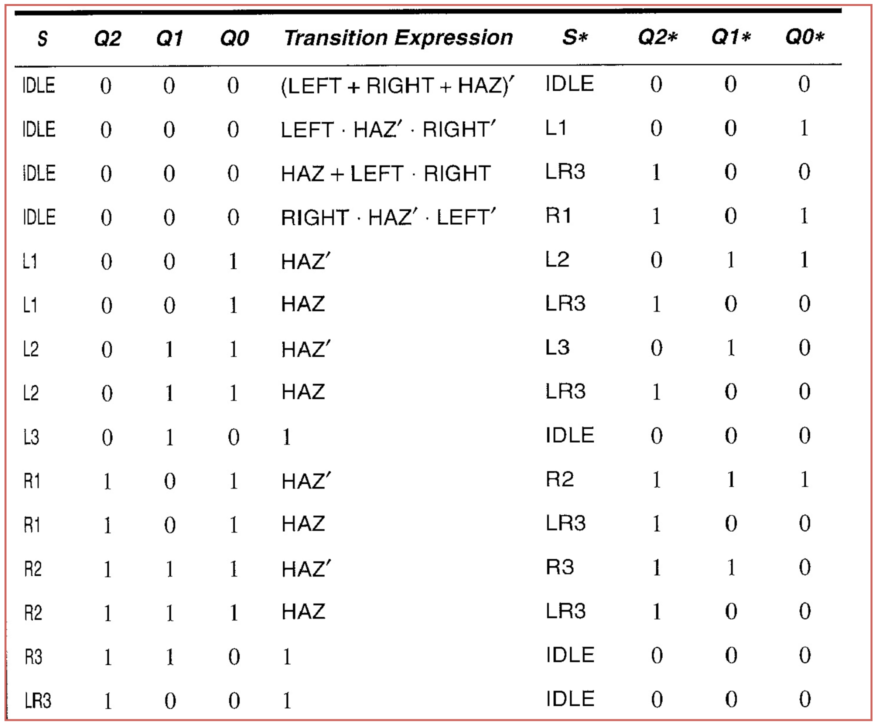
State	*Q2*	*Q1*	*Q0*
IDLE	0	0	0
L1	0	0	1
L2	0	1	1
L3	0	1	0
R1	1	0	1
R2	1	1	1
R3	1	1	0
LR3	1	0	0



S	*Q2*	*Q1*	*Q0*	*Transition Expression*	*S**	*Q2**	*Q1**	*Q0**
IDLE	0	0	0	(LEFT + RIGHT + HAZ)′	IDLE	0	0	0
IDLE	0	0	0	LEFT · HAZ′ · RIGHT′	L1	0	0	1
IDLE	0	0	0	HAZ + LEFT · RIGHT	LR3	1	0	0
IDLE	0	0	0	RIGHT · HAZ′ · LEFT′	R1	1	0	1
L1	0	0	1	HAZ′	L2	0	1	1
L1	0	0	1	HAZ	LR3	1	0	0
L2	0	1	1	HAZ′	L3	0	1	0
L2	0	1	1	HAZ	LR3	1	0	0
L3	0	1	0	1	IDLE	0	0	0
R1	1	0	1	HAZ′	R2	1	1	1
R1	1	0	1	HAZ	LR3	1	0	0
R2	1	1	1	HAZ′	R3	1	1	0
R2	1	1	1	HAZ	LR3	1	0	0
R3	1	1	0	1	IDLE	0	0	0
LR3	1	0	0	1	IDLE	0	0	0

- Next step depends on implementation choice
  – Synthesize or Structural with choice of FFs

Transition Equations



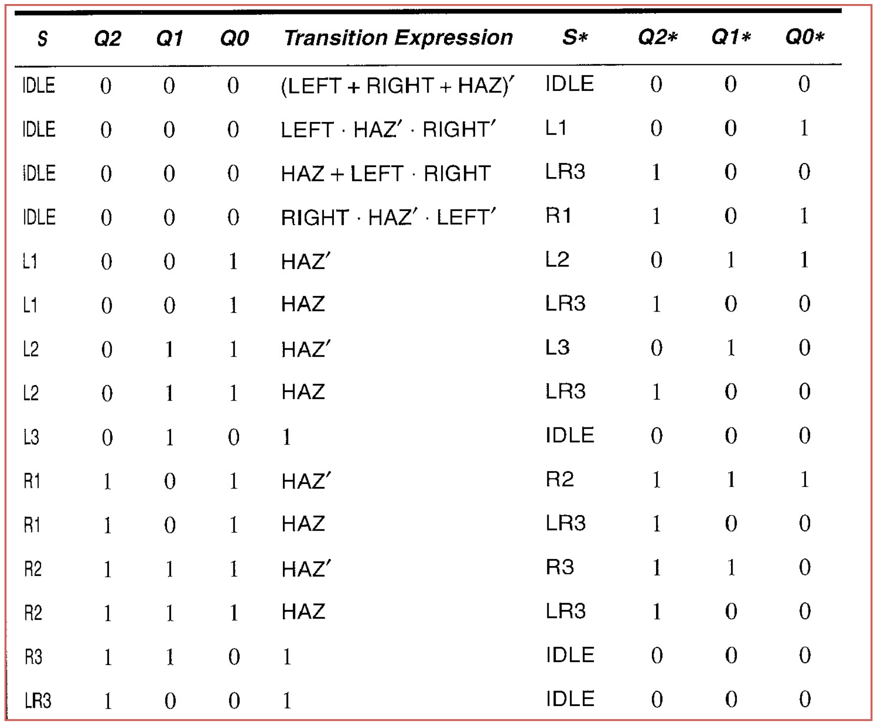
S	Q2	Q1	Q0	Transition Expression	S*	Q2*	Q1*	Q0*
IDLE	0	0	0	(LEFT + RIGHT + HAZ)′	IDLE	0	0	0
IDLE	0	0	0	LEFT · HAZ′ · RIGHT′	L1	0	0	1
IDLE	0	0	0	HAZ + LEFT · RIGHT	LR3	1	0	0
IDLE	0	0	0	RIGHT · HAZ′ · LEFT′	R1	1	0	1
L1	0	0	1	HAZ′	L2	0	1	1
L1	0	0	1	HAZ	LR3	1	0	0
L2	0	1	1	HAZ′	L3	0	1	0
L2	0	1	1	HAZ	LR3	1	0	0
L3	0	1	0	1	IDLE	0	0	0
R1	1	0	1	HAZ′	R2	1	1	1
R1	1	0	1	HAZ	LR3	1	0	0
R2	1	1	1	HAZ′	R3	1	1	0
R2	1	1	1	HAZ	LR3	1	0	0
R3	1	1	0	1	IDLE	0	0	0
LR3	1	0	0	1	IDLE	0	0	0

Q2* = Q2’· Q1’ · Q0’ · (HAZ + LEFT · RIGHT)
+ Q2’ · Q1’ · Q0’ · (RIGHT · HAZ’ · LEFT’)
+ Q2’ · Q1’ · Q0 · (HAZ)
+ Q2’ · Q1 · Q0 · (HAZ)
+ Q2 · Q1’ · Q0 · (HAZ’)
+ Q2 · Q1’ · Q0 · (HAZ)
+ Q2 · Q1 · Q0 · (HAZ’)
+ Q2 · Q1 · Q0 · (HAZ)

$\Longrightarrow$

Q2* = Q2’· Q1’ · Q0’ · (HAZ + RIGHT)
+ Q2’ · Q0 · HAZ
+ Q2 · Q0

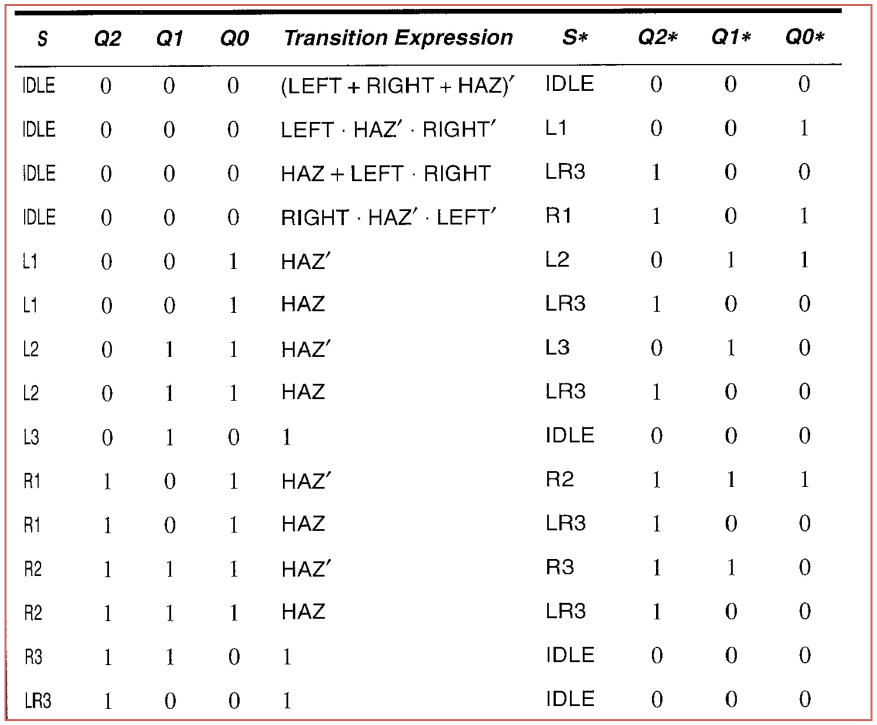
Transition Equations



S	*Q2*	*Q1*	*Q0*	*Transition Expression*	*S∗*	*Q2∗*	*Q1∗*	*Q0∗*
IDLE	0	0	0	(LEFT + RIGHT + HAZ)′	IDLE	0	0	0
IDLE	0	0	0	LEFT · HAZ′ · RIGHT′	L1	0	0	1
IDLE	0	0	0	HAZ + LEFT · RIGHT	LR3	1	0	0
IDLE	0	0	0	RIGHT · HAZ′ · LEFT′	R1	1	0	1
L1	0	0	1	HAZ′	L2	0	1	1
L1	0	0	1	HAZ	LR3	1	0	0
L2	0	1	1	HAZ′	L3	0	1	0
L2	0	1	1	HAZ	LR3	1	0	0
L3	0	1	0	1	IDLE	0	0	0
R1	1	0	1	HAZ′	R2	1	1	1
R1	1	0	1	HAZ	LR3	1	0	0
R2	1	1	1	HAZ′	R3	1	1	0
R2	1	1	1	HAZ	LR3	1	0	0
R3	1	1	0	1	IDLE	0	0	0
LR3	1	0	0	1	IDLE	0	0	0

$$\begin{aligned} Q1^* = \ & Q2' \cdot Q1' \cdot Q0 \cdot (HAZ') \\ & + Q2' \cdot Q1 \cdot Q0 \cdot (HAZ') \\ & + Q2 \cdot Q1' \cdot Q0 \cdot (HAZ') \\ & + Q2 \cdot Q1 \cdot Q0 \cdot (HAZ') \end{aligned} \quad \Longrightarrow \quad Q1^* = Q0 \cdot HAZ'$$

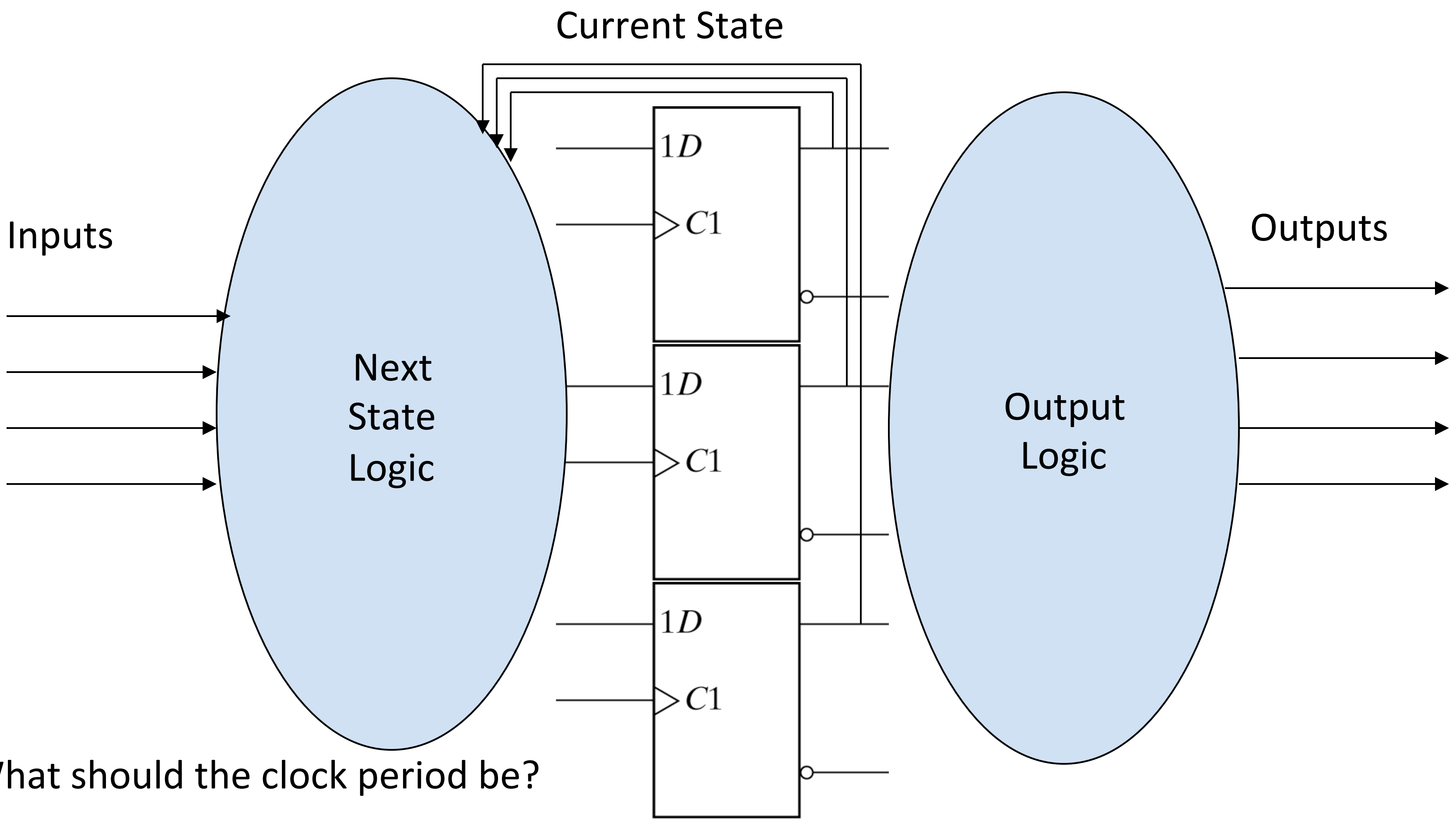
Transition Equations



S	*Q2*	*Q1*	*Q0*	*Transition Expression*	*S**	*Q2**	*Q1**	*Q0**
IDLE	0	0	0	(LEFT + RIGHT + HAZ)′	IDLE	0	0	0
IDLE	0	0	0	LEFT · HAZ′ · RIGHT′	L1	0	0	1
IDLE	0	0	0	HAZ + LEFT · RIGHT	LR3	1	0	0
IDLE	0	0	0	RIGHT · HAZ′ · LEFT′	R1	1	0	1
L1	0	0	1	HAZ′	L2	0	1	1
L1	0	0	1	HAZ	LR3	1	0	0
L2	0	1	1	HAZ′	L3	0	1	0
L2	0	1	1	HAZ	LR3	1	0	0
L3	0	1	0	1	IDLE	0	0	0
R1	1	0	1	HAZ′	R2	1	1	1
R1	1	0	1	HAZ	LR3	1	0	0
R2	1	1	1	HAZ′	R3	1	1	0
R2	1	1	1	HAZ	LR3	1	0	0
R3	1	1	0	1	IDLE	0	0	0
LR3	1	0	0	1	IDLE	0	0	0

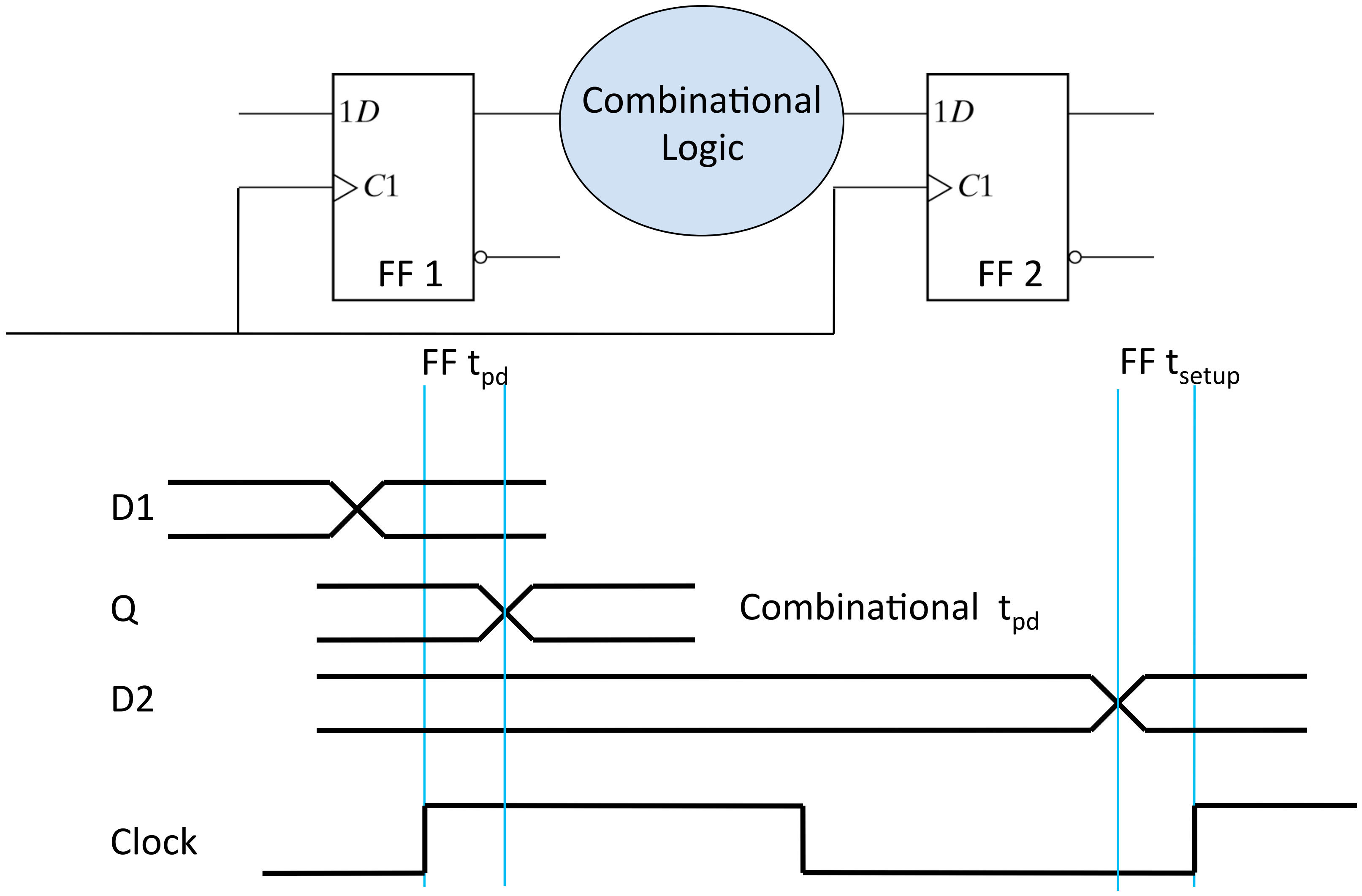
$$\begin{aligned} Q0^* = \; & Q2' \cdot Q1' \cdot Q0' \cdot (\mathrm{LEFT} \cdot \mathrm{HAZ}' \cdot \mathrm{RIGHT}') \\ & + Q2' \cdot Q1' \cdot Q0' \cdot (\mathrm{RIGHT} \cdot \mathrm{HAZ}' \cdot \mathrm{LEFT}') \\ & + Q2' \cdot Q1' \cdot Q0 \cdot (\mathrm{HAZ}') \\ & + Q2 \cdot Q1' \cdot Q0 \cdot (\mathrm{HAZ}') \end{aligned} \quad \Rightarrow \quad \begin{aligned} Q0^* = \; & Q2' \cdot Q1' \cdot Q0' \cdot \mathrm{HAZ}' \cdot (\mathrm{LEFT} \oplus \mathrm{RIGHT}) \\ & + Q1' \cdot Q0 \cdot \mathrm{HAZ}' \end{aligned}$$

Implementation (Encoded, Moore Machine)

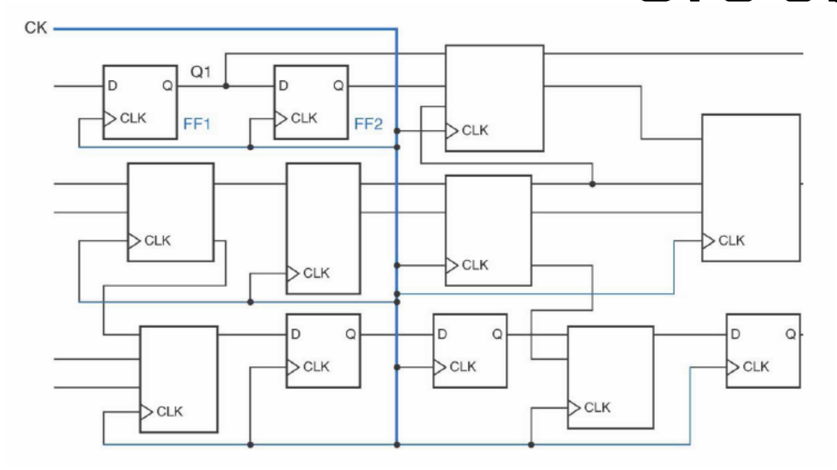


What should the clock period be?

How Fast Can the Clock Be?

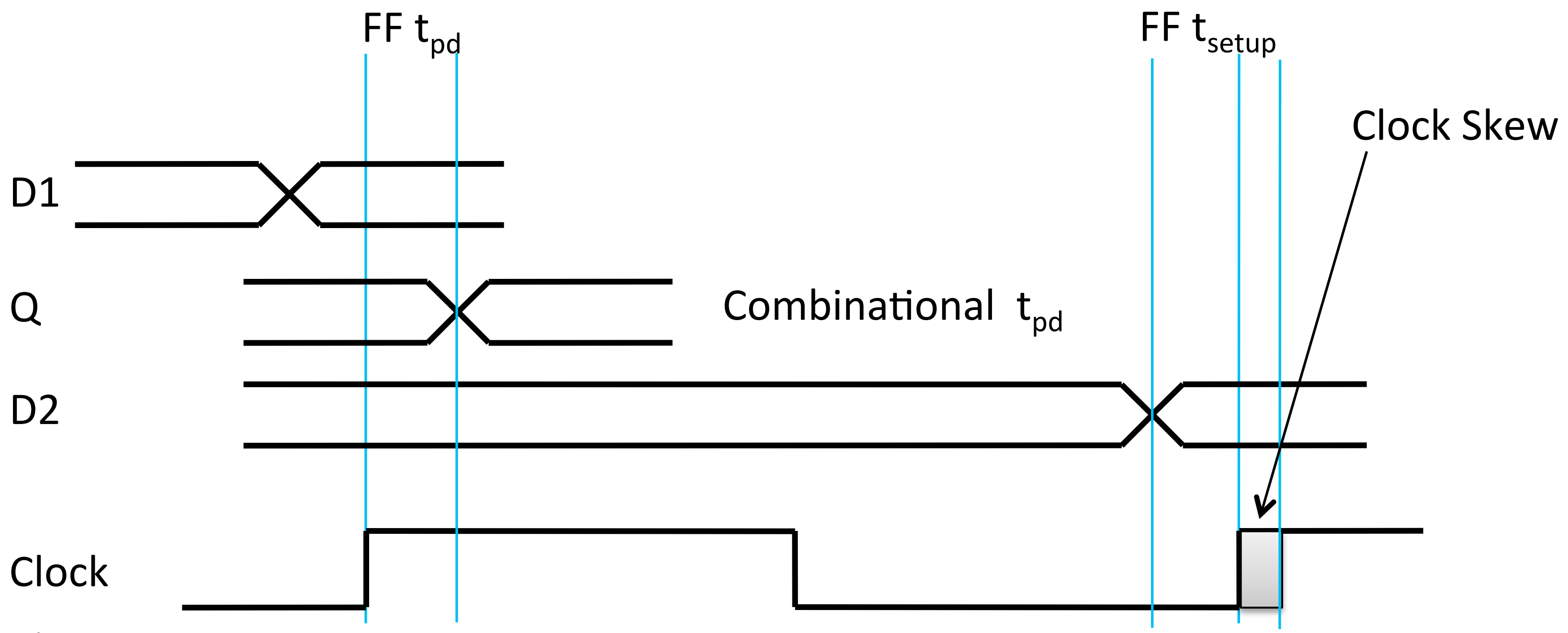


Clock Skew



Even with careful routing, clock will not arrive at all FFs at the same time. This skew in clock arrival time affects max clock rate.

$$\text{Clock Period}_{min} = \text{FF } t_{pd} + \text{FF } t_{setup} + \text{C } t_{pd} + t_{skew}$$



One-Hot

IDLE* = IDLE · (HAZ + LEFT + RIGHT)’ + L3 + R3 + LR3

L1* = IDLE · LEFT · HAZ’ · RIGHT’

R1* = IDLE · RIGHT · HAZ’ · LEFT’

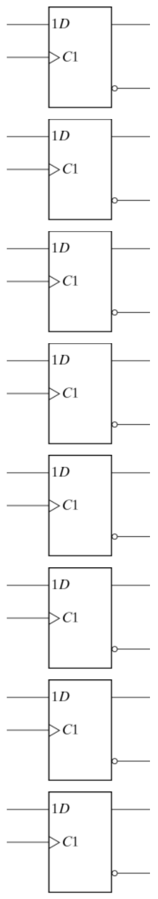
L2* = L1 · HAZ’

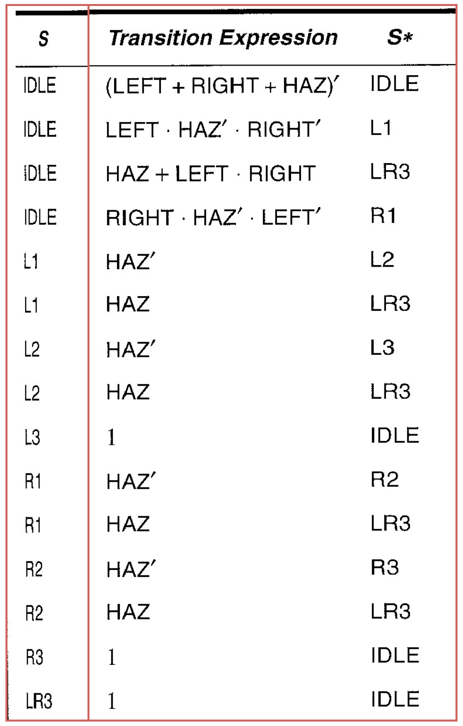
R2* = R1 · HAZ’

L3* = L2 · HAZ’

R3* = R2 · HAZ’

LR3* = IDLE · (HAZ + LEFT · RIGHT) + (L1 + L2 + R1 + R2) · HAZ

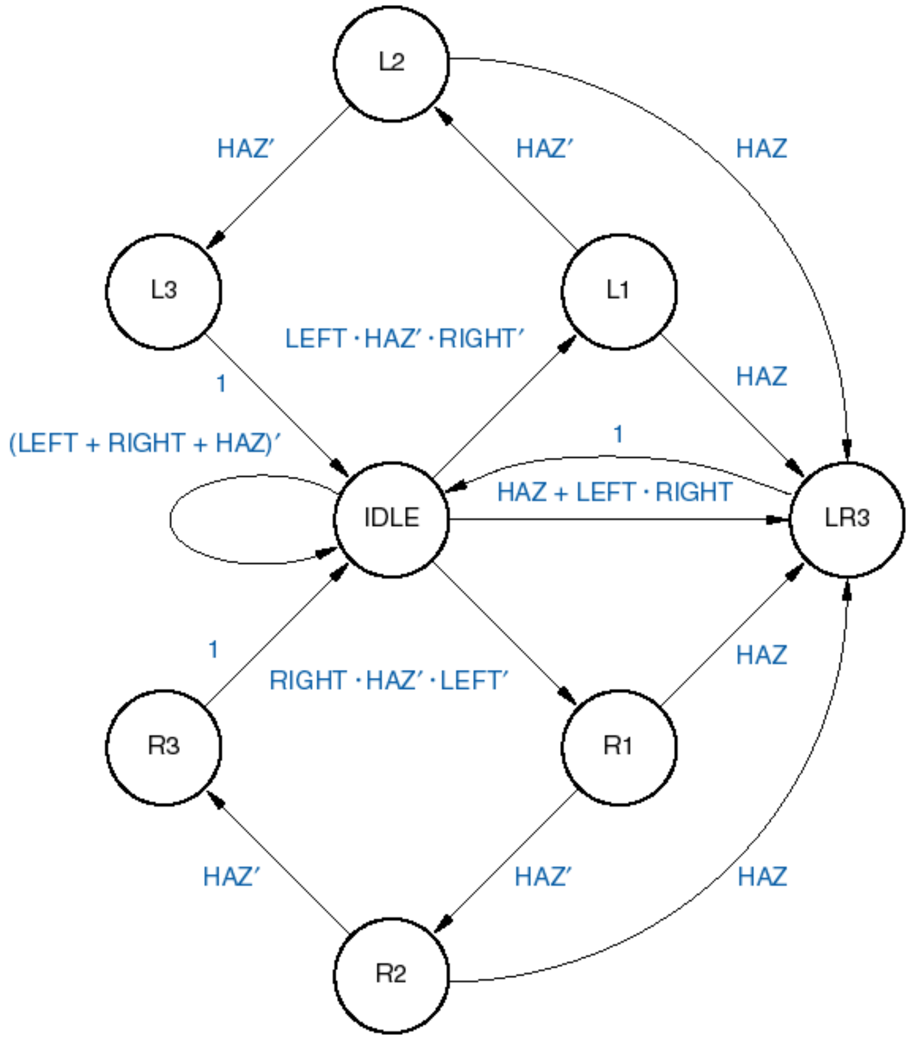




S	*Transition Expression*	*S**
IDLE	(LEFT + RIGHT + HAZ)′	IDLE
IDLE	LEFT · HAZ′ · RIGHT′	L1
IDLE	HAZ + LEFT · RIGHT	LR3
IDLE	RIGHT · HAZ′ · LEFT′	R1
L1	HAZ′	L2
L1	HAZ	LR3
L2	HAZ′	L3
L2	HAZ	LR3
L3	1	IDLE
R1	HAZ′	R2
R1	HAZ	LR3
R2	HAZ′	R3
R2	HAZ	LR3
R3	1	IDLE
LR3	1	IDLE

Better Still – Behavioral Verilog

```
parameter
                IDLE  = 8'b00000001,
                L1    = 8'b00000010,
                L2    = 8'b00000100,
                L3    = 8'b00001000,
                R1    = 8'b00010000,
                R2    = 8'b00100000,
                R3    = 8'b01000000,
                LR3   = 8'b10000000;

reg [7:0] State, NextState;

case (State)
  IDLE:
    begin
    if (Hazard || Left && Right)
      NextState = LR3;
    else if (Left)
      NextState = L1;
    else if (Right)
      NextState = R1;
    else
      NextState = IDLE;
    end

  L1: begin
    if (Hazard)
      NextState = LR3;
    else
      NextState = L2;
    end

  L2: begin
    if (Hazard)
      NextState = LR3;
    else
      NextState = L3;
    end

  L3: begin
    NextState = IDLE;
    end

  R1: begin
    if (Hazard)
      NextState = LR3;
    else
      NextState = R2;
    end

  R2: begin
    if (Hazard)
      NextState = LR3;
    else
      NextState = R3;
    end

  R3: begin
    NextState = IDLE;
    end

  LR3:begin
    NextState = IDLE;
    end
endcase
```

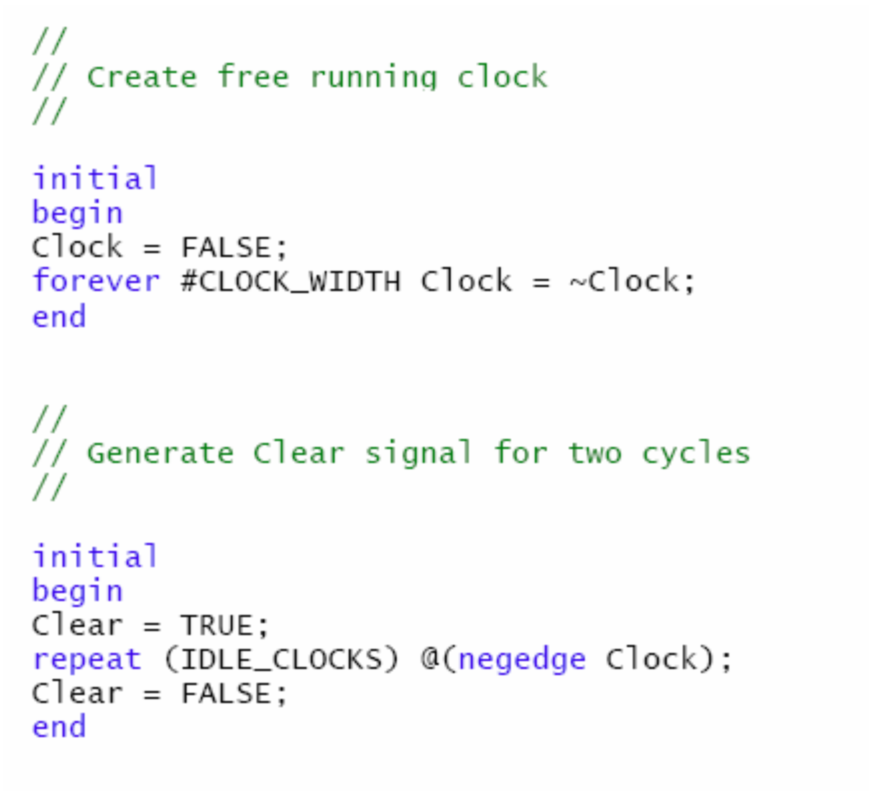


```
//
// behavioral model of the T Bird Tail Light FSM
//


module TBirdTailLights(Clock, Clear, Left, Right, Hazard, LA, LB, LC, RA, RB, RC);
input Clock, Clear, Left, Right, Hazard;
output LA,LB,LC,RA,RB,RC;
reg LA,LB,LC,RA,RB,RC;


parameter ON  = 1'b1;
parameter OFF = 1'b0;

// define states using same names and state assignments as state diagram and table
// Using one-hot method, we have one bit per state

parameter
    IDLE  = 8'b00000001,
    L1    = 8'b00000010,
    L2    = 8'b00000100,
    L3    = 8'b00001000,
    R1    = 8'b00010000,
    R2    = 8'b00100000,
    R3    = 8'b01000000,
    LR3   = 8'b10000000;


reg [7:0] State, NextState;
//
// Update state or reset on every + clock edge.  Use non-blocking assignments
//

always @(posedge Clock)
begin
if (Clear)
    State <= IDLE;
else
    State <= NextState;
end
```

```
//
// Outputs depend only upon state (Moore machine)
//

always @(State)
begin
case (State)
    IDLE:   begin
        LA = OFF;
        LB = OFF;
        LC = OFF;
        RA = OFF;
        RB = OFF;
        RC = OFF;
        end
    L1: begin
        LA = ON;
        LB = OFF;
        LC = OFF;
        RA = OFF;
        RB = OFF;
        RC = OFF;
        end

    L2: begin
        LA = ON;
        LB = ON;
        LC = OFF;
        RA = OFF;
        RB = OFF;
        RC = OFF;
        end
    L3: begin
        LA = ON;
        LB = ON;
        LC = ON;
        RA = OFF;
        RB = OFF;
        RC = OFF;
        end

    R1: begin
        LA = OFF;
        LB = OFF;
        LC = OFF;
        RA = ON;
        RB = OFF;
        RC = OFF;
        end

    R2: begin
        LA = OFF;
        LB = OFF;
        LC = OFF;
        RA = ON;
        RB = ON;
        RC = OFF;
        end
    R3: begin
        LA = OFF;
        LB = OFF;
        LC = OFF;
        RA = ON;
        RB = ON;
        RC = ON;
        end

    LR3:    begin
        LA = ON;
        LB = ON;
        LC = ON;
        RA = ON;
        RB = ON;
        RC = ON;
        end

endcase
end
```

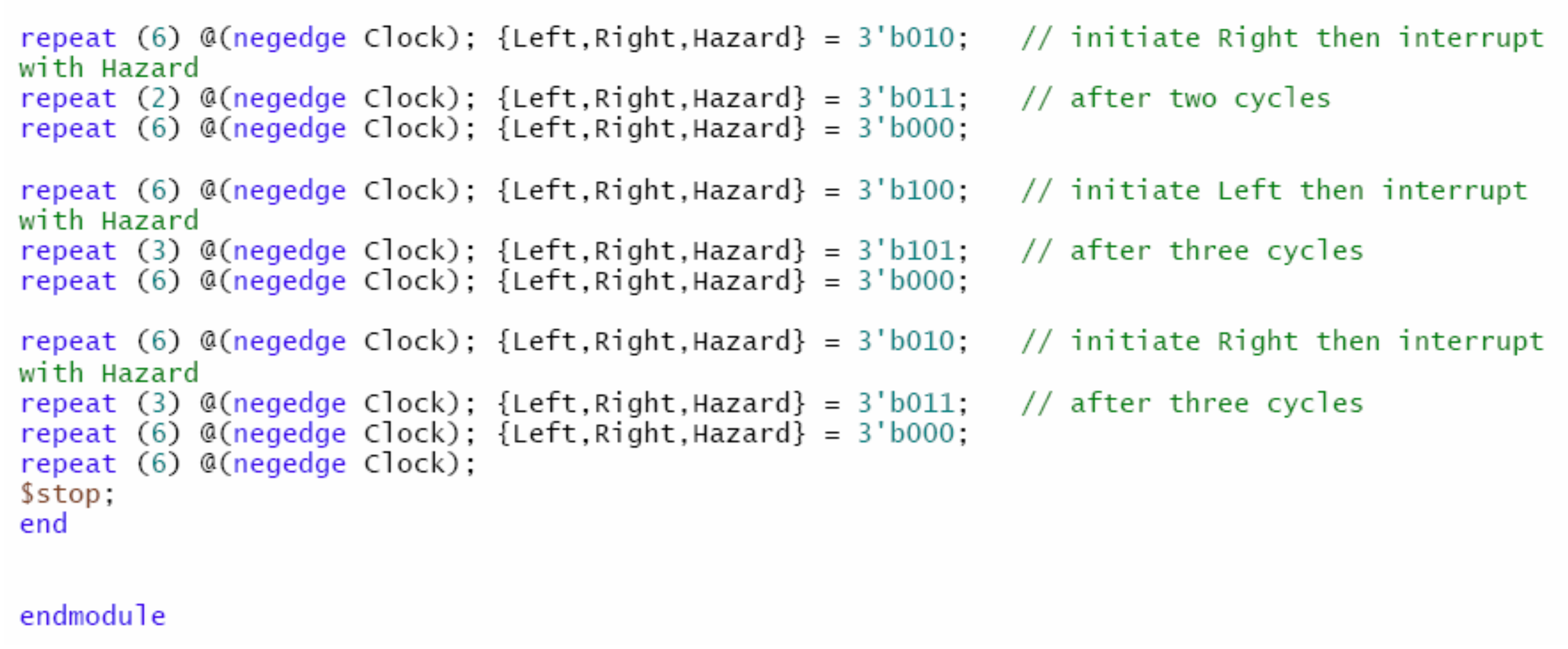
```
//
// Next state generation logic
//

always @(State or Left or Right or Hazard)
begin
case (State)
    IDLE:   begin
        if (Hazard || Left && Right)
            NextState = LR3;
        else if (Left)
            NextState = L1;
        else if (Right)
            NextState = R1;
        else
            NextState = IDLE;
        end

    L1: begin
        if (Hazard)
            NextState = LR3;
        else
            NextState = L2;
        end

    L2: begin
        if (Hazard)
            NextState = LR3;
        else
            NextState = L3;
        end

    L3: begin
        NextState = IDLE;
        end

    R1: begin
        if (Hazard)
            NextState = LR3;
        else
            NextState = R2;
        end

    R2: begin
        if (Hazard)
            NextState = LR3;
        else
            NextState = R3;
        end

    R3: begin
        NextState = IDLE;
        end

    LR3:begin
        NextState = IDLE;
        end
    default: NextState = IDLE;
endcase
end


endmodule
```

```
//
// Testbench for TBirdTailLights
//



module TestBench;
reg Left, Right, Hazard, Clear, Clock;
wire LA,LB,LC,RA,RB,RC;


parameter TRUE    = 1'b1;
parameter FALSE   = 1'b0;
parameter CLOCK_CYCLE  = 20;
parameter CLOCK_WIDTH  = CLOCK_CYCLE/2;
parameter IDLE_CLOCKS  = 2;



TBirdTailLights TFSM(Clock, Clear, Left, Right, Hazard, LA, LB, LC, RA, RB, RC);

//
// set up monitor
//

initial
begin
$display("                Time Clear Left Right Hazard  LA LB LC RA RB RC\n");
$monitor($time, "   %b    %b    %b     %b        %b   %b   %b   %b   %b   %b",Clear,Left,Right,
Hazard,LA,LB,LC,RA,RB,RC);
end
```



```
//
// Create free running clock
//

initial
begin
Clock = FALSE;
forever #CLOCK_WIDTH Clock = ~Clock;
end


//
// Generate Clear signal for two cycles
//

initial
begin
Clear = TRUE;
repeat (IDLE_CLOCKS) @(negedge Clock);
Clear = FALSE;
end
```

```
//
// Generate stimulus after waiting for reset
//

initial
begin

repeat (6) @(negedge Clock); {Left,Right,Hazard} = 3'b100;   // simple Left, Right, Hazard
tests
repeat (6) @(negedge Clock); {Left,Right,Hazard} = 3'b000;
repeat (6) @(negedge Clock); {Left,Right,Hazard} = 3'b010;
repeat (6) @(negedge Clock); {Left,Right,Hazard} = 3'b000;
repeat (6) @(negedge Clock); {Left,Right,Hazard} = 3'b001;
repeat (6) @(negedge Clock); {Left,Right,Hazard} = 3'b000;

repeat (6) @(negedge Clock); {Left,Right,Hazard} = 3'b100;   // initiate Left then interrupt
with Hazard
repeat (1) @(negedge Clock); {Left,Right,Hazard} = 3'b101;   // after one cycle
repeat (6) @(negedge Clock); {Left,Right,Hazard} = 3'b000;

repeat (6) @(negedge Clock); {Left,Right,Hazard} = 3'b010;   // initiate Right then interrupt
with Hazard
repeat (1) @(negedge Clock); {Left,Right,Hazard} = 3'b011;   // after one cycle
repeat (6) @(negedge Clock); {Left,Right,Hazard} = 3'b000;

repeat (6) @(negedge Clock); {Left,Right,Hazard} = 3'b100;   // initiate Left then interrupt
with Hazard
repeat (2) @(negedge Clock); {Left,Right,Hazard} = 3'b101;   // after two cycles
repeat (6) @(negedge Clock); {Left,Right,Hazard} = 3'b000;
```



```
repeat (6) @(negedge Clock); {Left,Right,Hazard} = 3'b010;   // initiate Right then interrupt
with Hazard
repeat (2) @(negedge Clock); {Left,Right,Hazard} = 3'b011;   // after two cycles
repeat (6) @(negedge Clock); {Left,Right,Hazard} = 3'b000;

repeat (6) @(negedge Clock); {Left,Right,Hazard} = 3'b100;   // initiate Left then interrupt
with Hazard
repeat (3) @(negedge Clock); {Left,Right,Hazard} = 3'b101;   // after three cycles
repeat (6) @(negedge Clock); {Left,Right,Hazard} = 3'b000;

repeat (6) @(negedge Clock); {Left,Right,Hazard} = 3'b010;   // initiate Right then interrupt
with Hazard
repeat (3) @(negedge Clock); {Left,Right,Hazard} = 3'b011;   // after three cycles
repeat (6) @(negedge Clock); {Left,Right,Hazard} = 3'b000;
repeat (6) @(negedge Clock);
$stop;
end


endmodule
```