Sequential Logic Design Principles.Clocked Synchronous State-Machine Synthesis (Continued)

Doru Todinca

Department of Computers
Politehnica University of Timisoara

Outline

Outline

Outline

First Example. The Problem

Design a clocked synchronous state machine with two inputs, A and B, and a single Z output that is 1 if:

- *A had the same value at each of the two previous clock ticks, or*
- *B has been 1 since the last time that the first condition was true.*

Otherwise, the output should be 0.

- The description does not look very clear at a first reading.
- At this point it can be useful to imagine a scenario and to draw a timing diagram for one or more sequences of inputs.
- Figure 1 contains such an example.

Timing Diagram for Example State Machine

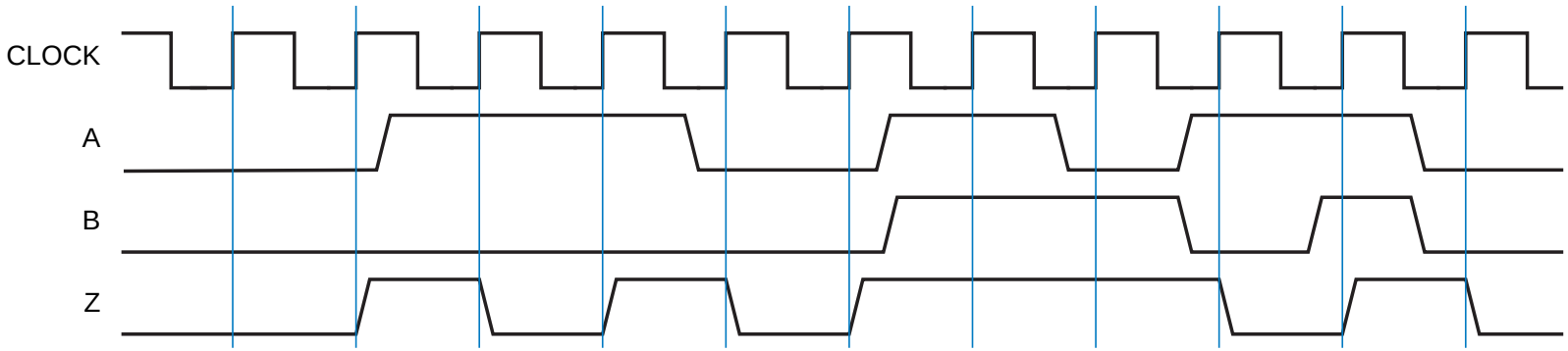


Figure 1 : Timing diagram for example state machine

The Timing Diagram and the State Table

- It's unlikely that the timing diagram will specify unambiguously the machine's behavior for all possible sequences of inputs, but it is a good starting point for discussion
- Also, the timing diagram helps us to imagine various scenarios, that we can verify again at the end of the design process
- The first step in a timing diagram is to construct a template
- In this example, we know from the word description that the machine is of Moore type
- Now, we can try to obtain a state table of the machine
- This process of building a state table will possibly take several iterations
- The evolution of the process is shown in figure 2

The Evolution of the State Table

- We provide two columns for the current state: one with the state's meaning, and the second one with the state label
- We provide one next-state column for each possible input combination
- Here we provide one single column for output values
- For a Mealy machine, the output values are written along with the next state values, for each input combination
- The order in which the input combinations are written is not important here, but we will simplify the derivation of excitation equation later if we write them in Karnaugh map-order

The Evolution of the State Table (Cont'd)

- The word description is not specific about what happens at the initialization (when the machine is first started), so we can assume that the machine enters an *initial state*, called INIT here (see fig 2 (a))
- The value of Z in the INIT state is 0, since there were *no* two inputs beforehand
- Next, we must fill the next-state entries for the INIT row
- The output Z cannot be 1 until we have seen at least two inputs on A, so we will provide two next states, A0 and A1, that "remember" the value of A on the previous clock ticks, as shown in fig 2 (b)
- In both A0 and A1 the output Z is 0, since the condition for a 1 hasn't been satisfied yet
- The meaning of A0 is "got $A = 0$ on the previous tick, $A \neq 0$ on the tick before that, and $B \neq 1$ at some time since the previous pair of equal A inputs"
- The meaning for A1 is similar

The Evolution of the State Table (Cont'd)

- In state A0, we know that A was 0 in the previous clock tick
  - If A is 0 again, we go to a new state OK with Z=1 (fig 2 (c))
  - If A is 1, then we don't have two equal inputs in a row, so we go to state A1 (to remember that we just got a 1)
- Similar, in state A1 we go to OK if we get a second 1 input in a row, or to A0 if we get a 0 (fig 2 (d))
- Once we are into the OK state, we can stay there as long as B=1, irrespective of the A input (see fig 3 (a))
- But if B=0 we have to look for two 1s or two 0s in a row on A again
- The problem is that the OK state itself doesn't tell us if the previous value of A was a 0 or a 1

The Evolution of the State Table (Cont'd)

- The solution is to split the OK state into two OK states: OK0 and OK1, that "remember" the previous A input (fig 3 (b))
- Hence, OK0 means that the previous value of A has been 0, and OK1, that the previous value of A has been 1
- All of the next states of OK0 and OK1 can be selected from the existing states, as shown in (c) and (d)
- Now we have achieved "closure" of the state table, meaning that the state table describes a *finite*-state machine
- As a final check, in figure 4 we repeat the timing diagram of figure 1, listing the states that should be visited according to the state table

Evolution of the State Table

(a)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT					0
	. . .					
	. . .					
	. . .					
				S*		

(b)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0					0
Got a 1 on A	A1					0
				S*		

(c)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK	OK	A1	A1	0
Got a 1 on A	A1					0
Got two equal A inputs	OK					1
				S*		

(d)

Meaning	S	A B: 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK	OK	A1	A1	0
Got a 1 on A	A1	A0	A0	OK	OK	0
Got two equal A inputs	OK					1
				S*		

Figure 2 : Evolution of a state table

Continued Evolution of the State Table

(a)

Meaning	S	A B = 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK	OK	A1	A1	0
Got a 1 on A	A1	A0	A0	OK	OK	0
Got two equal A inputs	OK	?	OK	OK	?	1
		S*				

(b)

Meaning	S	A B = 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0					1
Two equal, A=1 last	OK1					1
		S*				

(c)

Meaning	S	A B = 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0	OK0	OK0	OK1	A1	1
Two equal, A=1 last	OK1					1
		S*				

(d)

Meaning	S	A B = 00	01	11	10	Z
Initial state	INIT	A0	A0	A1	A1	0
Got a 0 on A	A0	OK0	OK0	A1	A1	0
Got a 1 on A	A1	A0	A0	OK1	OK1	0
Two equal, A=0 last	OK0	OK0	OK0	OK1	A1	1
Two equal, A=1 last	OK1	A0	OK0	OK1	OK1	1
		S*				

Figure 3 : Continued evolution of a state table

Timing Diagram and State Sequence for Example State Machine

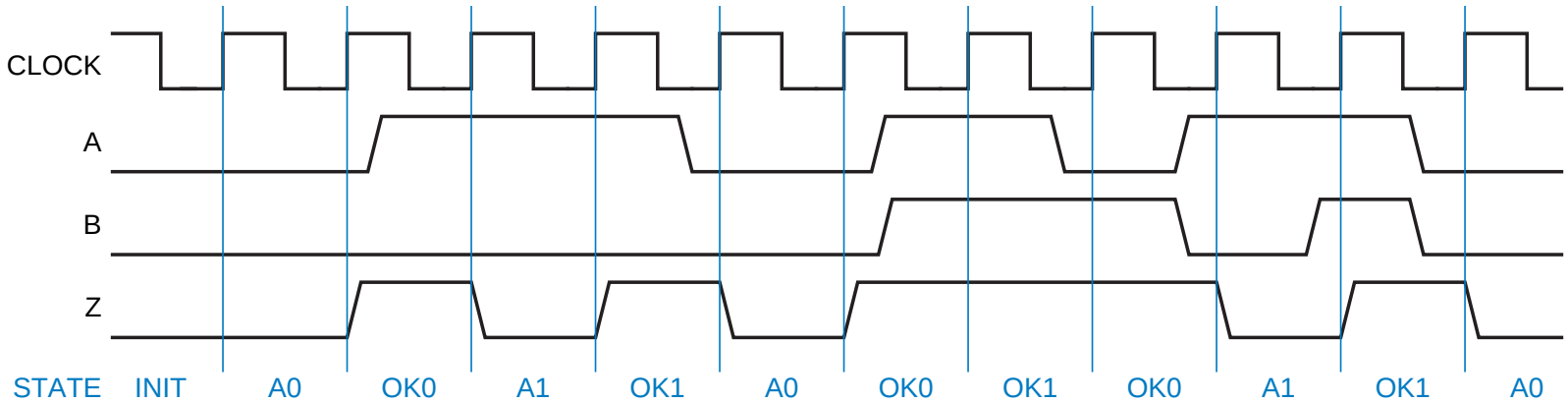


Figure 4 : Timing diagram and state sequence for example state machine

Realizing a Reliable Reset

- The hardware design of a state machine should ensure that it enters a known initial state on power-up
- Most systems have a RESET signal that is asserted during power-up
- The RESET signal is typically generated by an analog circuit
- During power-up, the circuit detects a voltage (e.g. 4.5 V) close to the power supply full voltage (5 V) with a certain delay (e.g. 100 ms) to ensure that all components have time to stabilize before the circuit "unresets" the system
- Such a circuit is TL7705 from Texas Instruments: it has an internal 4.5 V reference and uses an external resistor and capacitor to determine the "unreset" constant (e.g. 100 ms)

Realizing a Reliable Reset

- If we use discrete flip-flops with asynchronous *preset* and *clear* inputs, the RESET can be applied to these inputs to force the machine to the desired state
- If the asynchronous preset and clear are not available, or if we need a synchronous reset, then the RESET signal may be used as another input to the state machine and all the next state entries going to the desired state (e.g. $00\ldots00$) when RESET is asserted

Outline

State Minimization

- If the number of states of a finite state machine is not minimal, we can use formal procedure for state minimization
- These procedures aim to identify *equivalent states*
- Two states are *equivalent* if it is impossible to distinguish them by observing only the current and future *outputs* of the machine (and *not* the internal variables)
- A pair of equivalent states can be replaced by a single state
- Two states S1 and S2 are equivalent if they produce the same value at the machine outputs AND if, for each input combination, S1 and S2 must have either the same next state or equivalent next states
- An experienced designer can produce a machine with a minimal or near minimal number of states without using formal state minimization procedures
- Sometimes we can obtain a better or cheaper design if we increase the number of states
- A designer can do more to simplify a state machine during the *state-assignment* phase of the design

Outline

State Assignment

- Next step in the design process is to determine how many binary variables are required to represent the states in the state table and how to assign a binary combination to each named state
- A binary combination assigned to a particular state is called a *coded state*
- The total number of states in a machine with n state variables is 2^n
- Hence, the total number of flip-flops needed to code s states is $\lceil \log_2 s \rceil$ (the smallest integer greater than or equal to $\log_2 s$)

State Assignment

- The state/output table of our example is repeated in Table 7-6 (fig 5)
- There are 5 states, so it requires 3 flip-flops
- Three flip-flops provide $2^3 = 8$ states, so there will be $8 - 5 = 3$ unused states
- For the unused states, we can take either the minimal risk approach, or the minimal cost approach
- Now we will discuss the choices for coding the 5 states

State and Output Table for Example Problem

Table 7-6
State and output table for example problem.

	A B				
S	***00***	***01***	***11***	***10***	***Z***
INIT	A0	A0	A1	A1	0
A0	OK0	OK0	A1	A1	0
A1	A0	A0	OK1	OK1	0
OK0	OK0	OK0	OK1	A1	1
OK1	A0	OK0	OK1	OK1	1
			S*		

Figure 5 : State and output table for example problem

Possible State Assignments for Example Problem

- The simplest assignment of s coded states is to use the first s binary integers in binary counting order (first assignment column in Table 7-7, fig 6
- However, the simplest state assignment does not always lead to the simplest excitation equations, output equations, and resulting logic circuit
- The state assignment has a major effect on circuit cost, and it may interact with other factors, such as the choice of storage elements (D versus J-K flip-flops), and the realization for excitation and output logic (sum of products, product of sums, ad hoc design)
- If we want to choose the best assignment we have to try *all* assignments, but this is obviously not feasible !
- Most digital designers rely on experience and several practical guidelines for making reasonable state assignments

Guidelines for State Assignments

- Choose an initial coded state into which the machine can easily be forced at reset: $00\ldots00$ or $11\ldots11$
- Minimize the number of state variables that change at each transaction
- Maximize the number of state variables that don't change in a group of related states (a group of states in which most transactions stay in the group)
- Exploit symmetries in the problem specification and the corresponding symmetries in the state table:
  - If one group of states means almost the same thing as another, then, once we established an assignment for the first group, use a similar assignment, differing in only one bit, for the second group

Guidelines for State Assignments (Cont'd)

- If there are unused states, then choose the "best" of the available state-variable combinations to achieve the foregoing goals (don't limit the choice of coded states to the first s n bit integers)
- Decompose the set of variables into individual bits or fields, where each bit or field has a well-defined meaning with respect to the input effects or output behavior of the machine
- Consider using more than the minimum number of state variables to make a decomposed assignment possible

The Decomposed State Assignment

- Some of these guidelines are applied to the "decomposed" state assignment (in Table 7-7)
- The initial state is 000, which is easy to force by RESET (either synchronously or asynchronously)
- In this example, the INIT state is never reentered again (some other machines can have an "idle" state, that the machine enters at reset or when it has nothing in particular to do)
- After coding the INIT state, we take advantage of the fact that there are unused states
- Q1 is used to indicate if the machine is in the INIT state or not
- State bits Q2 and Q3 have individual meanings: Q3 gives the previous value of A and Q2 indicates that the conditions for a 1 output are satisfied in the current state
- We will continue the state machine design based on this assignment in later subsections

Other Possible State Assignments

- Another state machine assignment, that can be adapted to any state machine is *one-hot assignment*, shown in Table 7-7
- This assignment uses more than the minimum number of state variables: it uses one bit per state
- It is simple and usually leads to small excitation equations, since each flip-flop must be set to 1 for transitions into only one state
- If the machine has many states, then this assignment requires many more than the minimum number of flip-flops
- The last column in Table 7-7 is an "almost one-hot assignment", that uses the "no-hot" combination for the initial state
- This assignment makes sense, since the initial state is never revisited in this machine
- Also, it is easy to initialize all flip-flops to zero at power-up or reset

Possible State Assignments for Example Problem

State Name	Assignment			
	Simplest Q1–Q3	Decomposed Q1–Q3	One-hot Q1–Q5	Almost One-hot Q1–Q4
INIT	000	000	00001	0000
A0	001	100	00010	0001
A1	010	101	00100	0010
OK0	011	110	01000	0100
OK1	100	111	10000	1000

Table 7-7 Possible state assignments for the state machine in Table 7-6.

Figure 6 : Possible state assignments for the state machine in Table 7.6 (in figure 5)

Outline

Synthesis Using D Flip-Flops

- Figure 7 shows the transition and output table for this problem
- Here, coded states are substituted for named states in the state table
- For D flip-flops we can directly obtain the excitation table (in figure 8) from the transition and output table
- Actually, with D flip-flops, we can call the transition table *transition/excitation table*
- Then we transfer the information from the transition/excitation table to the excitation maps (figure 9 for minimal risk approach and figure 10 minimal cost approach)
- In this case we have to minimize five-variables Karnaugh-maps
- A 5-variable Karnaugh map is drawn as a pair of 4-variable Karnaugh maps where the cells in the same positions in the two maps are considered to be adjacent

Excitation equations for the Minimal Risk Approach

From the Karnaugh maps of figure 9, we find the excitation equations for minimal risk approach:

$$D1 = Q1 + Q2' \cdot Q3'$$
$$D2 = Q1 \cdot Q3' \cdot A' + Q1 \cdot Q3 \cdot A + Q1 \cdot Q2 \cdot B$$
$$D3 = Q1 \cdot A + Q2' \cdot Q3' \cdot A$$

The output Z is the sum of the minterms for the two coded states (111 and 110) in which Z is 1:

$$Z = Q1 \cdot Q2 \cdot Q3 + Q1 \cdot Q2 \cdot Q3' = Q1 \cdot Q2$$

Excitation equations for the Minimal Cost Approach

From the Karnaugh maps of figure 10, we find the excitation equations for minimal cost approach, that are simpler than in the minimal risk case:

$$D1 = 1$$
$$D2 = Q1 \cdot Q3' \cdot A' + Q3 \cdot A + Q2 \cdot B$$
$$D3 = A$$

The value of Z is "don't care" for the unused states, which lead to:

$$Z = Q2$$

Figure 11 shows the resulting logic diagram for the minimum cost approach

Transition and Output Table for Example Problem

	AB				
Q1 Q2 Q3	00	01	11	10	Z
000	100	100	101	101	0
100	110	110	101	101	0
101	100	100	111	111	0
110	110	110	111	101	1
111	100	110	111	111	1
	Q1* Q2* Q3*				

Table 7-8
Transition and output table for example problem.

Figure 7 : Transition and output table for example problem

Excitation and Output Table for Example Problem

Table 7-9
Excitation and output table for Table 7-8 using D flip-flops.

	A B				
Q1 Q2 Q3	***00***	***01***	***11***	***10***	***Z***
000	100	100	101	101	0
100	110	110	101	101	0
101	100	100	111	111	0
110	110	110	111	101	1
111	100	110	111	111	1
	D1 D2 D3				

Figure 8 : Excitation and output table for Table 7-8 (fig 7) using D flip-flops

Excitation Maps for Example State Machine, Minimum Risk Approach

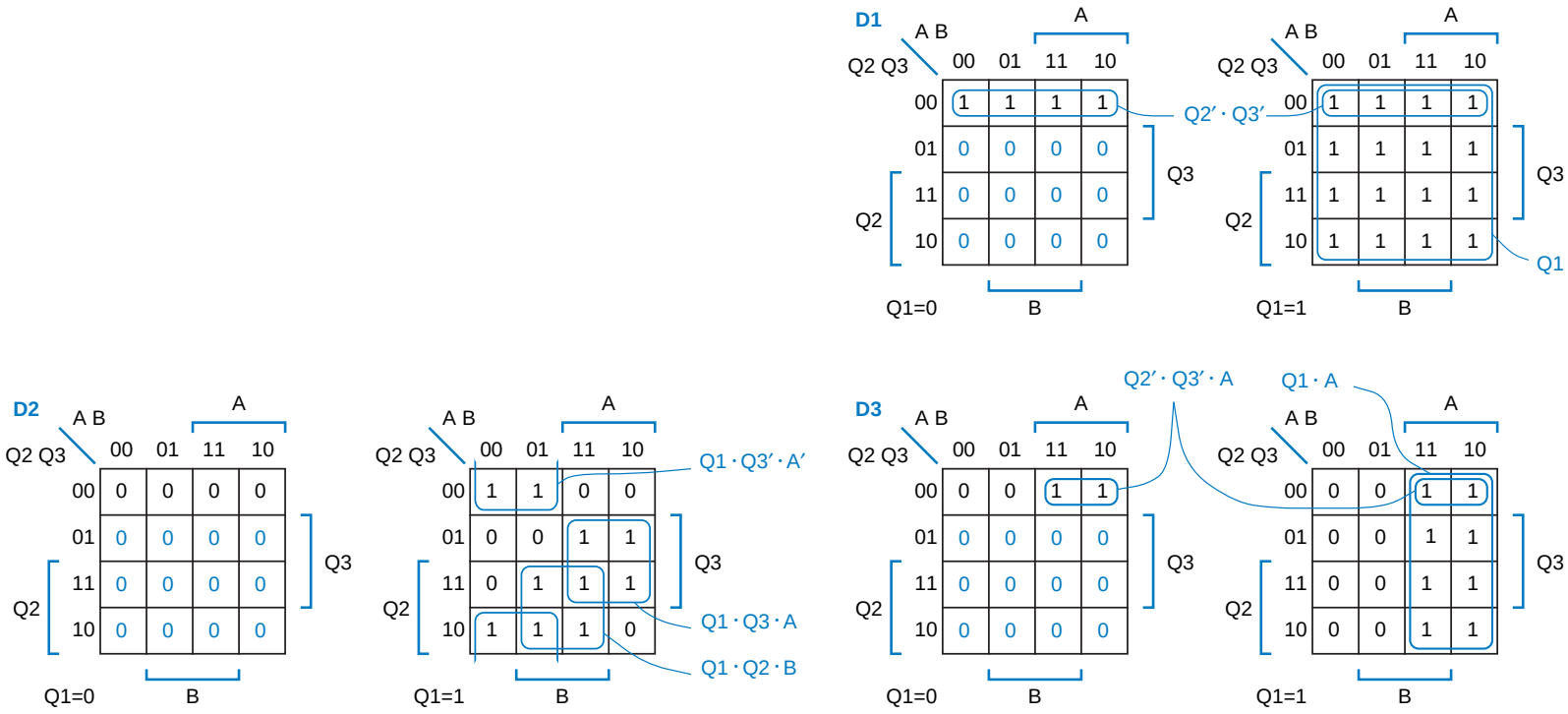


Figure 9 : Excitation maps for D1, D2, and D3 assuming that unused states go to state 000

Excitation Maps for Example State Machine, Minimum Cost Approach

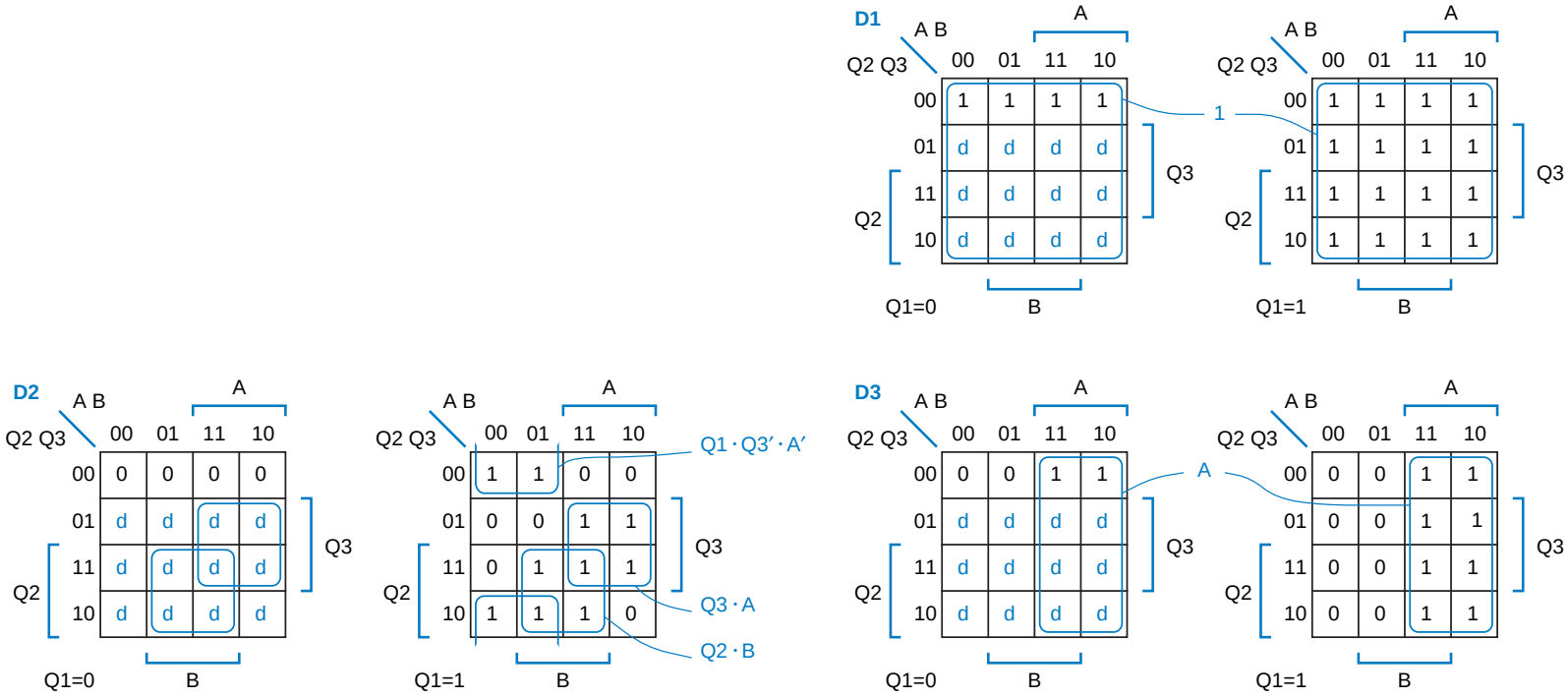


Figure 10 : Excitation maps for D1, D2, and D3 assuming that next states of unused statesare “don’t-cares”

Logic Diagram for Example State Machine, Minimum Cost Approach

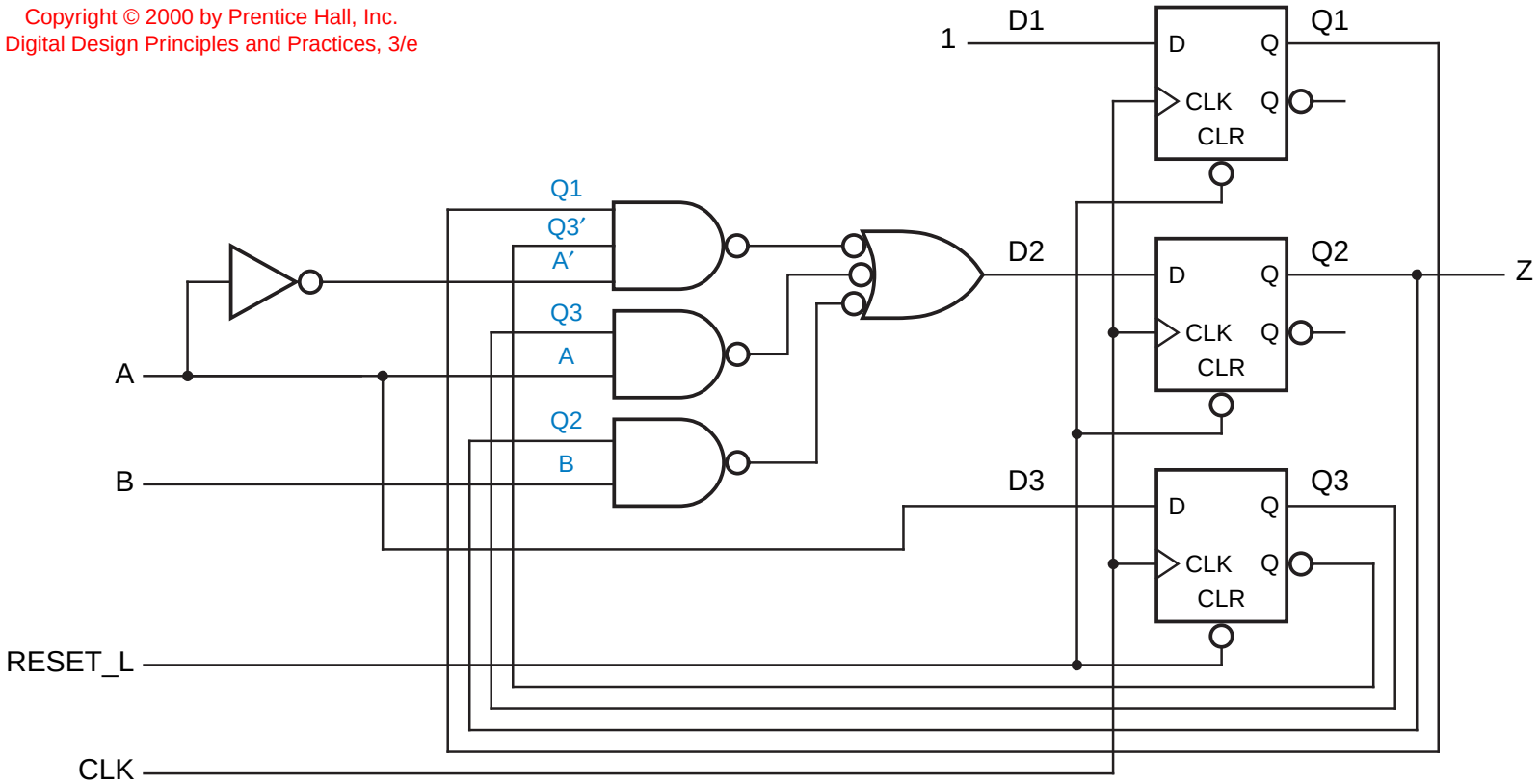


Figure 11 : Logic diagram resulting from figure 10

Outline

Synthesis Using J-K Flip-Flops

- Compared to D flip-flops, the big difference occurs in the derivation of an excitation table from the transition table
- We need an *application table*, which expresses the required values of J and K (as functions of Q and Q*) in order to change the flip-flop's state from Q to Q*
- To obtain a J-K excitation table, the designer must look at both the current and desired next value of each state bit in the transition table and substitute the corresponding pair of J and K values from the application table
- For our problem, the excitation table obtained in this way is shown in Table 7-11, fig 13
- This information is transferred to Karnaugh maps in figure 14 for the minimum risk approach.

Excitation Equations for J-K Flip-Flops

From the Karnaugh maps we obtain the following excitation equations for the minimal risk approach:

$$J1 = Q2' \cdot Q3'$$
$$K1 = 0$$
$$J2 = Q1 \cdot Q3' \cdot A' + Q1 \cdot Q3 \cdot A$$
$$K2 = Q1' + Q3' \cdot A \cdot B' + Q3 \cdot A' \cdot B'$$
$$J3 = Q2' \cdot A + Q1 \cdot A$$
$$K3 = Q1' + A'$$

For the minimal cost approach, it result the following excitation equations:

$$J1 = 1$$
$$K1 = 0$$
$$J2 = Q1 \cdot Q3' \cdot A' + Q3 \cdot A$$
$$K2 = Q3' \cdot A \cdot B' + Q3 \cdot A' \cdot B'$$
$$J3 = A$$
$$K3 = A'$$

The state encoding of the J-K circuit is the same as for D circuit, so the output equation is the same:
$Z = Q1 \cdot Q2$ for the minimal risk approach and
$Z = Q2$ for the minimal cost approach.

A logic diagram corresponding to the minimal-cost equations is shown in figure 15. The circuit has two more gates than the minimal-cost D circuit !!

Application Table for J-K Flip-Flops

Q	*Q**	*J*	*K*
0	0	0	d
0	1	1	d
1	0	d	1
1	1	d	0

Table 7-10
Application table for J-K flip-flops.

Figure 12 : Application table for J-K flip-flops

Excitation and Output Table for the Example State Machine

	A B				
Q1 Q2 Q3	**00**	**01**	**11**	**10**	Z
000	1d, 0d, 0d	1d, 0d, 0d	1d, 0d, 1d	1d, 0d, 1d	0
100	d0, 1d, 0d	d0, 1d, 0d	d0, 0d, 1d	d0, 0d, 1d	0
101	d0, 0d, d1	d0, 0d, d1	d0, 1d, d0	d0, 1d, d0	0
110	d0, d0, 0d	d0, d0, 0d	d0, d0, 1d	d0, d1, 1d	1
111	d0, d1, d1	d0, d0, d1	d0, d0, d0	d0, d0, d0	1
	J1 K1, J2 K2, J3 K3				

Table 7-11
Excitation and output table for the state machine of Table 7-8, using J-K flip-flops.

Figure 13 : Excitation and output table for the state machine of table 7-8 (fig 7)

Excitation Maps for Example State Machine, Minimum Risk Approach

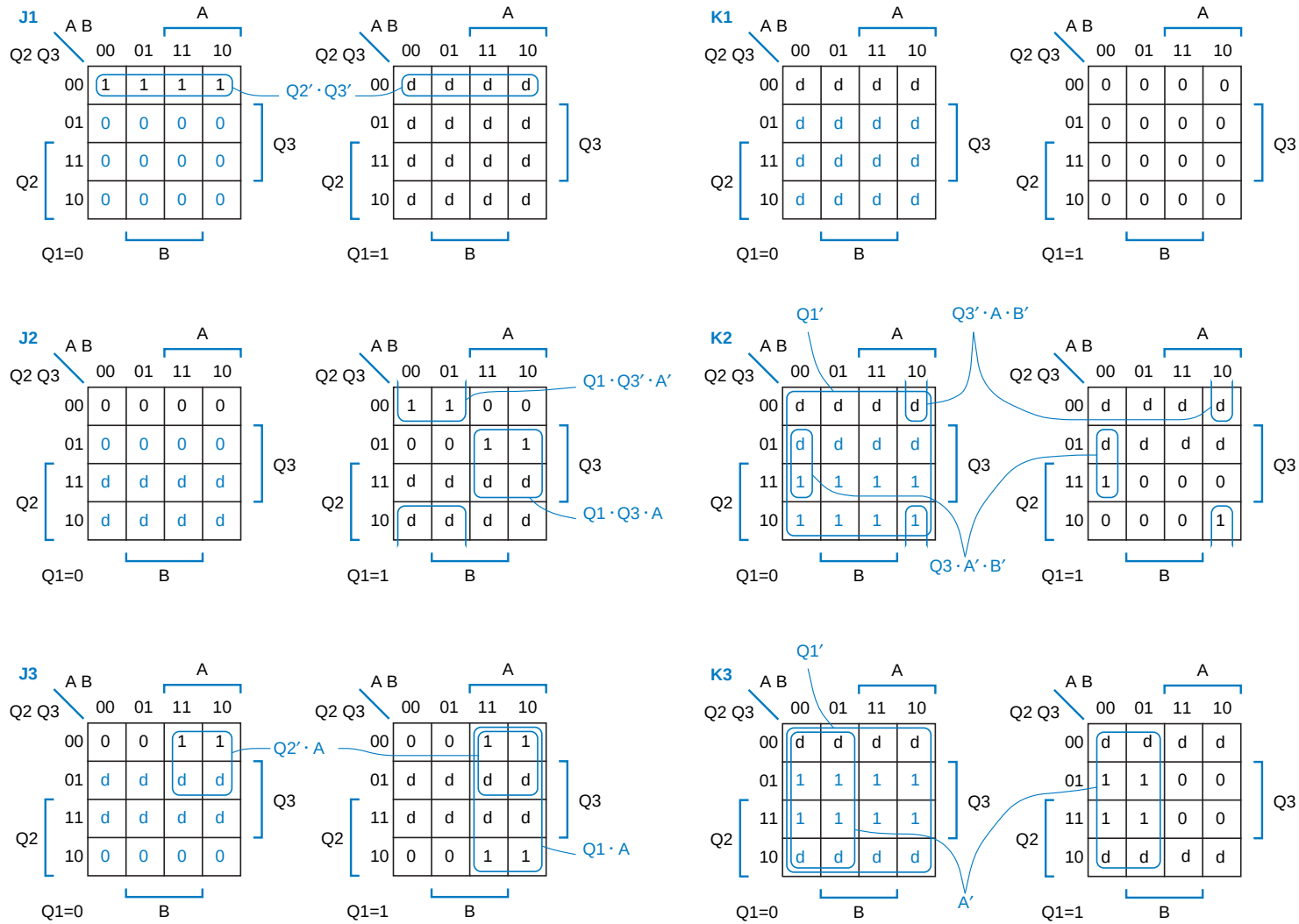


Figure 14 : Excitation maps for J1, K1, J2, K2, J3, and K3 assuming that unused states go to state 000

Logic Diagram for Example State Machine, Minimum Cost Approach

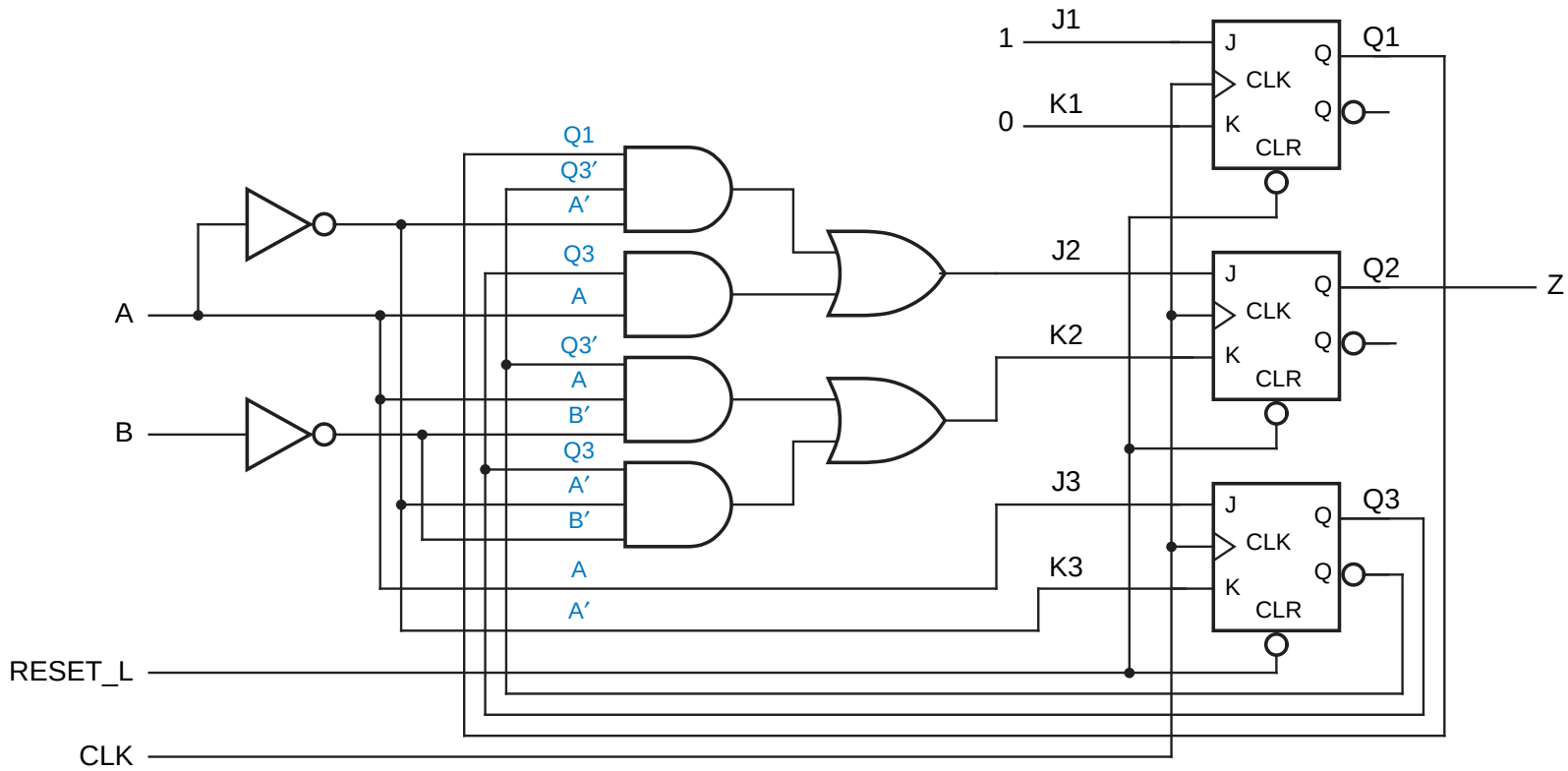


Figure 15 : Logic diagram for example state machine, using J-K flip-flops and minimal-cost excitation logic

Outline

"Combination Lock" State Machine Example

Design a clocked synchronous state machine with one input, X, and two outputs, UNLK and HINT. The UNLK output should be 1 if and only if X is 0 and the sequence of inputs received on X at the preceding clock ticks was 0110111. The HINT output should be 1 if and only if the current value of X is the correct one to move the machine closer to being in the "unlocked" state (with UNLK=1).

It is apparent from the word description that it is a Mealy machine:

- The UNLK depends on both the past history of inputs and on X's current value
- HINT depends on the state and on the current X

State and Output Table for the Combination Lock Machine

- A state and output table for the combination lock machine is presented in Table 7-14, fig 16
- In initial state A we assume that we have received no inputs in the required sequence and we are looking for the first 0 in the sequence
- If we get an 1, we stay into state A
- If we get a 0, we move into state B
- In state B we are looking for a 1, and move to state C if we get it, or stay into state B if we don't get it (since the 0 we just received might be the first 0 in the required sequence
- In each successive state, we move on to the next state if we get the correct input, and we go back to A or B if we get the wrong input
- An exception occurs in state G: if we get the wrong input (a 0), the previous three inputs might still turn out to be the first three inputs of the required sequence, so we go back to E instead of B

Combination Lock Machine

- In state H, we have received the required sequence, so we set UNLK to 1 if X is 0
- In each state, we set HINT to 1 for the value of X that moves us closer to state H
- There are 8 states, so we can code them with 3 variables
- We assign the states in binary counting order, to obtain the transition/excitation table in Table 7-15, figure 17
- The corresponding Karnaugh maps for D1, D2 and D3 are shown in figure 18
- The output values are transfered from the transition/excitation and output table to the set of Karnaugh maps from figure 19

Excitations and Output Equations

From figure 18 we read the following excitation equations:

$$D1 = Q1 \cdot Q2' \cdot X + Q1' \cdot Q2 \cdot Q3 \cdot X' + Q1 \cdot Q2 \cdot Q3'$$
$$D2 = Q2' \cdot Q3 \cdot X + Q2 \cdot Q3' \cdot X$$
$$D3 = Q1 \cdot Q2' \cdot Q3' + Q1 \cdot Q3 \cdot X' + Q2' \cdot X' + Q3' \cdot Q1' \cdot X' + Q2 \cdot Q3' \cdot X$$

From figure 19 we can read the output equations:

$$UNLK = Q1 \cdot Q2 \cdot Q3 \cdot X'$$
$$HINT =$$
$$Q1' \cdot Q2' \cdot Q3' \cdot X' + Q1 \cdot Q2' \cdot X + Q2' \cdot Q3 \cdot X + Q2 \cdot Q3 \cdot X' + Q2 \cdot Q3' \cdot X$$

State and Output Table for the Combination Lock Machine

Table 7-14
State and output table for combination-lock machine.

Meaning	*S*	*X* = *0*	*X* = *1*
Got zip	A	B, 01	A, 00
Got 0	B	B, 00	C, 01
Got 01	C	B, 00	D, 01
Got 011	D	E, 01	A, 00
Got 0110	E	B, 00	F, 01
Got 01101	F	B, 00	G, 01
Got 011011	G	E, 00	H, 01
Got 0110111	H	B, 11	A, 00
		S*, UNLK HINT	

Figure 16 : State and output table for combination lock machine

Transition/Excitation Table for the Combination Lock Machine

	X	
Q1 Q2 Q3	***0***	***1***
000	001, 01	000, 00
001	001, 00	010, 01
010	001, 00	011, 01
011	100, 01	000, 00
100	001, 00	101, 01
101	001, 00	110, 01
110	100, 00	111, 01
111	001, 11	000, 00
	Q1* Q2* Q3*, UNLK HINT	

Table 7-15
Transition/excitation table for combination-lock machine.

Figure 17 : transition and excitation table for combination lock machine

Excitation Maps for Combination-Lock Machine

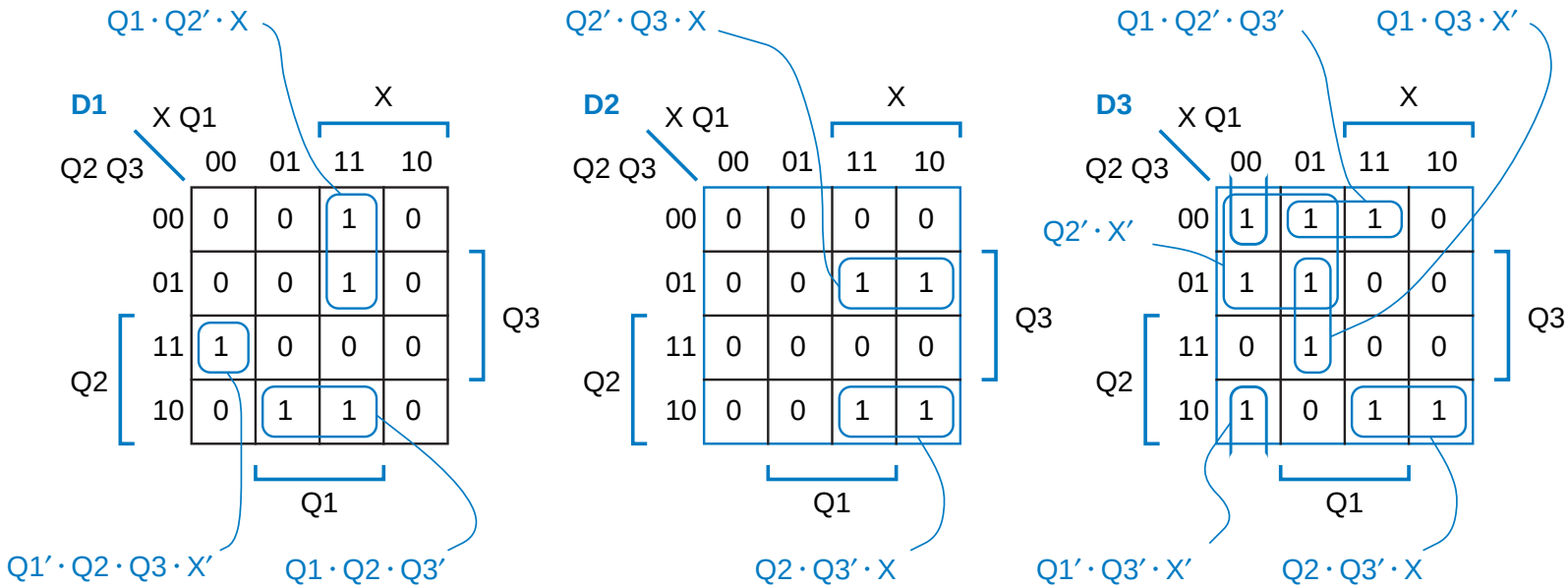


Figure 18 : Excitation maps for D1, D2, and D3 in combination-lock machine

Karnaugh Maps for Output Functions in Combination-Lock Machine

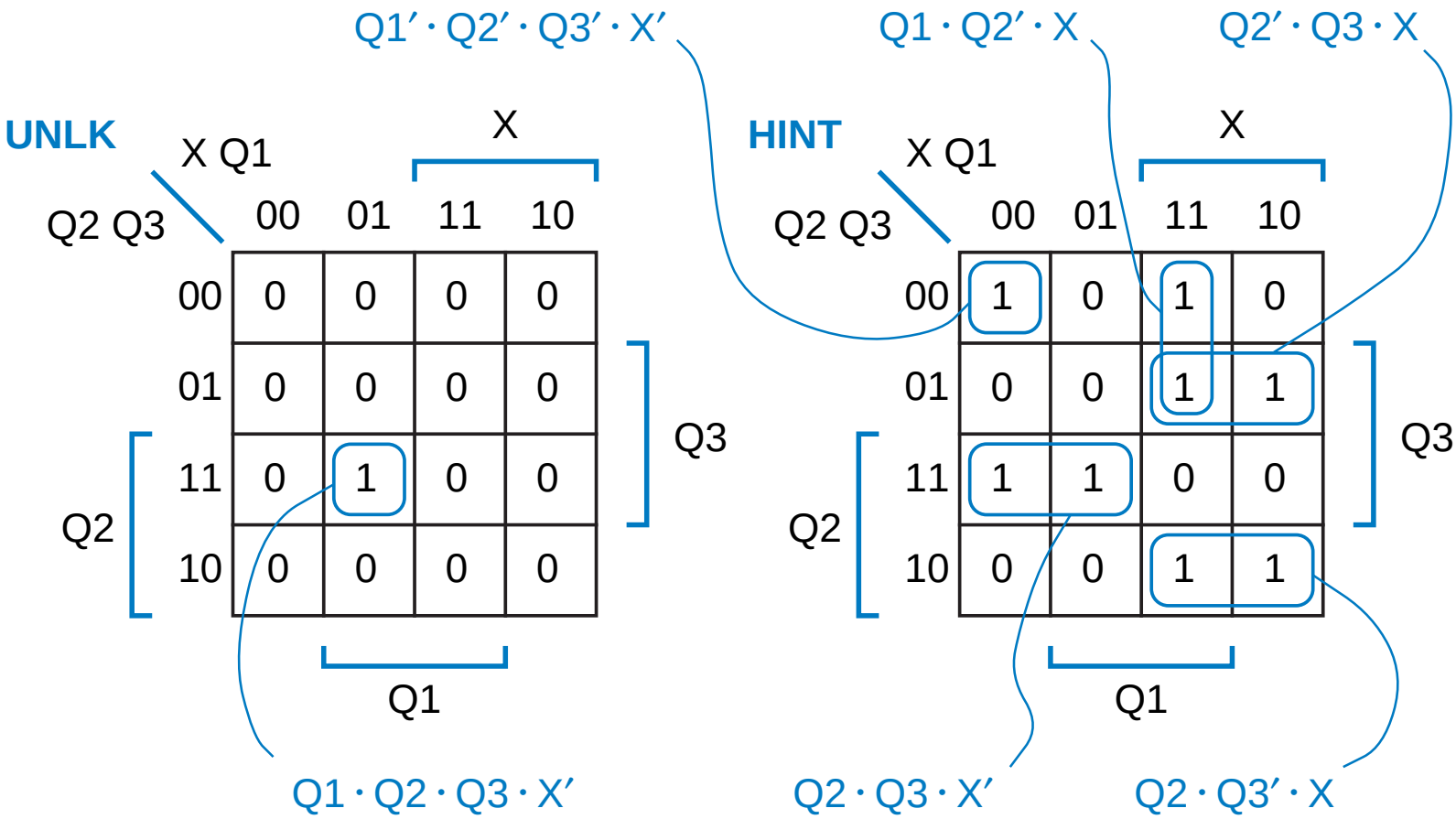


Figure 19 : Karnaugh maps for output functions UNLK and HINT in combination-lock machine

Outline

Designing State Machines Using State Diagrams

- State diagrams offer a graphical approach to state machines design which is suited for small to medium sized state machines
- Designing a state diagram is much like designing a state table
- However, there is one fundamental difference between a state diagram and a state table:
- This difference makes state diagram design simpler, but more error-prone than the state-table design
  - A state table is an exhaustive listing of the next states for each state/input combination. No ambiguity is possible
  - A state diagram contains a set of arcs labeled with transition expressions. Only one transition expression is required per arc, even if there are many inputs. When a state diagram is constructed there is no guarantee that the transition expressions written on the arcs leaving a particular state cover all input combinations exactly once

Designing State Machines Using State Diagrams

- In an improperly constructed (i.e. ambiguous) state diagram, some state/input combinations way have no next state specified, which is generally undesirable
- Other states may have multiple next states for the same state/input combination, which is wrong !
- Thus, considerable care must be taken in the design of state diagrams
- We will show an example

Example of State Machine Design Using State Diagrams

- We want to design a state machine that controls the tail lights of a car Ford Thunderbird from 1965, shown in fig 20
- There are three lights on each side, and for turns they operate in sequence to show the turning direction, as illustrated in fig 21
- The state machine has two input signals, LEFT and RIGHT, that carry the driver's request for a left or a right turn
- It also has an emergency-flasher input, HAZ, that request the tail lights to be operated in hazard mode (all six lights flashing on and off in unison)
- We also assume the existence of a free running clock whose frequency equals the desired flashing rate for the lights

T-Bird Tail Lights

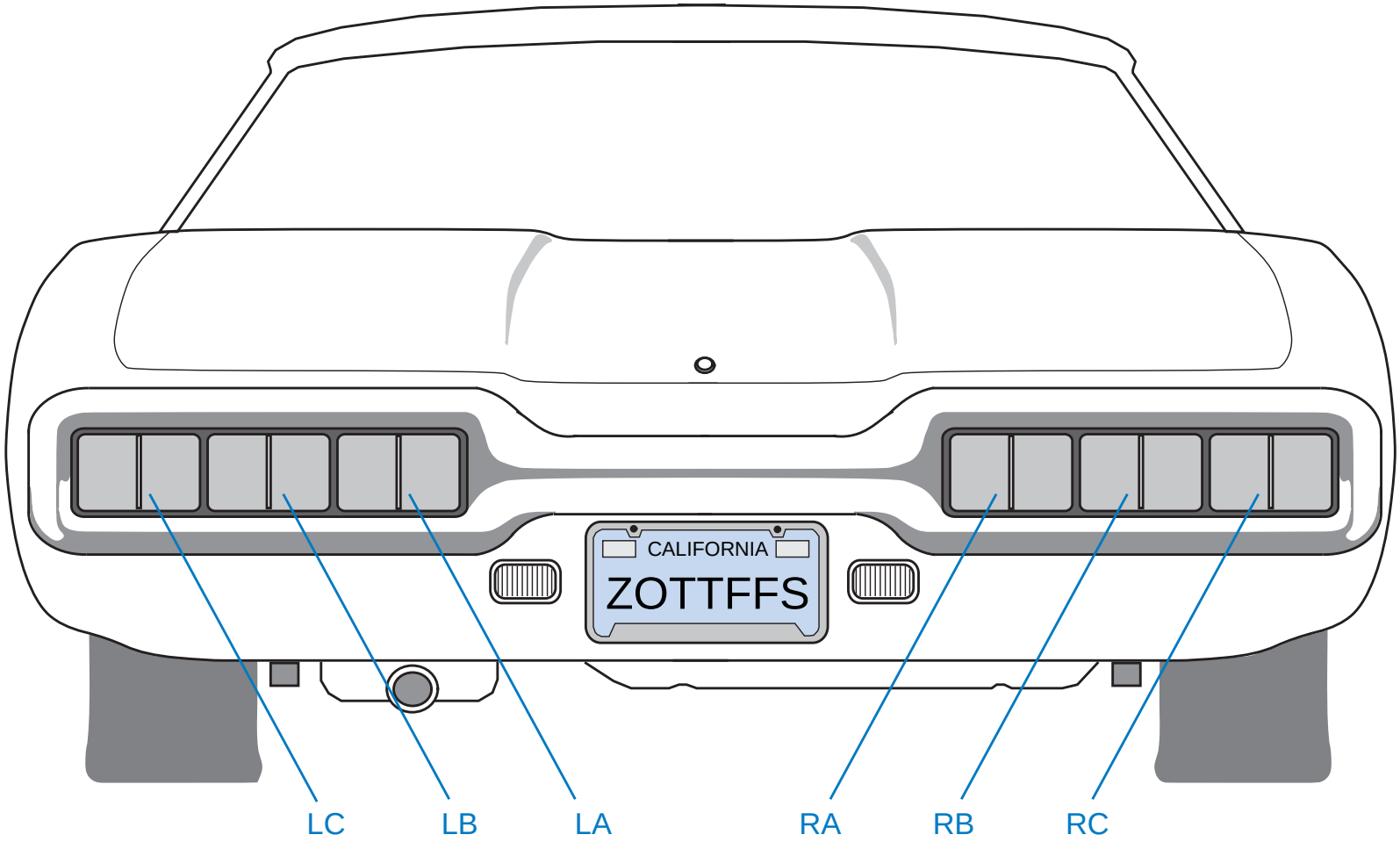


Figure 20 : T-bird tail lights

Flashing Sequence for T-Bird Tail Lights

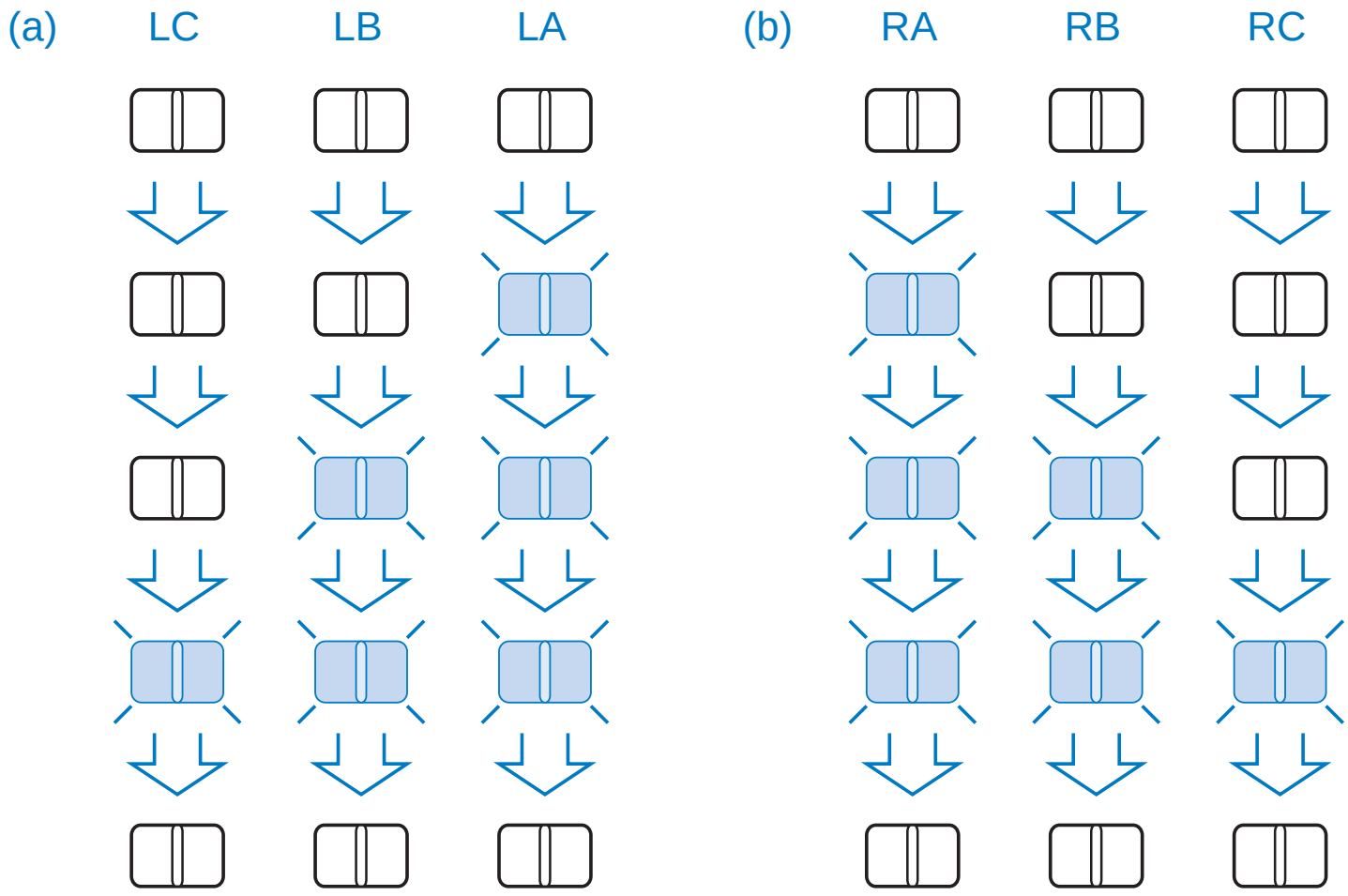


Figure 21 : Flashing sequence for T-bird tail lights: (a) left turn; (b) right turn

Initial State Diagram and Output Table for T-Bird Tail Lights

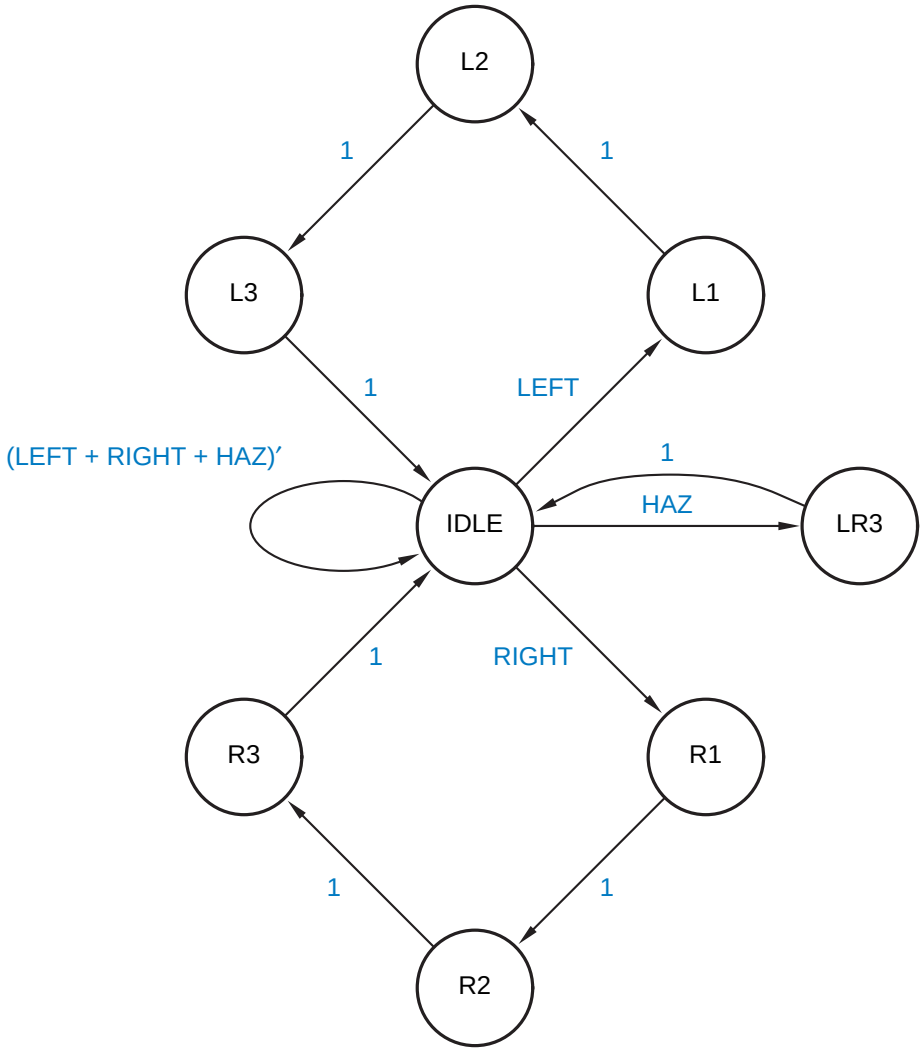
- We will design a Moore machine: the state alone determines which lights are on and which are off
- For a left turn, the state machine should cycle through four states in which the righthand lights are off and 0, 1, 2, or 3 of the lefthand lights are on
- A similar situation takes place for a right turn
- In hazard mode, only two states are required: all lights on and all lights off
- Figure 22 shows our first attempt of state diagram
- A common IDLE state is defined where all lights are off
- When a left turn is requested, the machine goes through three states in which 1, 2, and 3 of the lefthand lights are on, and then back to IDLE
- Right turn works similarly

Initial State Diagram and Output Table for T-Bird Tail Lights

- In the hazard mode, the machine cycles back and forth between IDLE state and a state in which all six lights are on (labeled LR3)
- The figure 22 contains an output table instead of writing output values on the diagram
- We can write output equations from the output table, if we let each state name represent a logic expression that is 1 only in that state:

$$LA = L1 + L2 + L3 + LR3$$
$$RA = R1 + R2 + R3 + LR3$$
$$LB = L2 + L3 + LR3$$
$$RB = R2 + R3 + LR3$$
$$LC = L3 + LR3$$
$$RC = R3 + LR3$$

Initial State Diagram and Output Table for T-Bird Tail Lights



Output Table

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

Figure 22 : Initial state diagram and output table for T-bird tail lights

Corrected State Diagram for T-Bird Tail Lights

- There is a big problem with the state diagram of figure 22: it does not properly handle multiple inputs asserted simultaneously
- What happens if LEFT and HAZ (or RIGHT and HAZ) are asserted ?
- According to the state diagram, the machine goes to two states, L1 and LR3, which is impossible
- In reality the machine should have only one next state, which could be L1, LR3, or a third state
- The problem is fixed in figure 23, where the HAZ input is given priority
- Also, if LEFT and RIGHT are asserted simultaneously, the situation is treated as a hazard request (the driver is confused and needs help)
- The new state diagram is unambiguously because the transition expressions on the arcs leaving each state are mutually exclusive and all-inclusive

Corrected State Diagram for T-Bird Tail Lights

- It means that, for each state, no two expressions are 1 for the same combination, and some expression is 1 for every input combination
- These conditions can be confirmed algebraically by performing two steps:
  1. *Mutual exclusion.* For each state, show that the logical product of each possible pair of transition expressions on arcs leaving the state is 0. If there are n arcs, then there are $n(n-1)/2$ logical products to evaluate (for each state)
  2. *All inclusion.* For each state, show that the logical sum of the transition expression on all arcs leaving that state is 1
- In this example the verification for unambiguity is trivial except for IDLE state, which has 4 transitions leaving it
- We can list all 8 possible combinations of the three variables HAZ, LEFT and RIGHT and verify the conditions

Corrected State Diagram for T-Bird Tail Lights

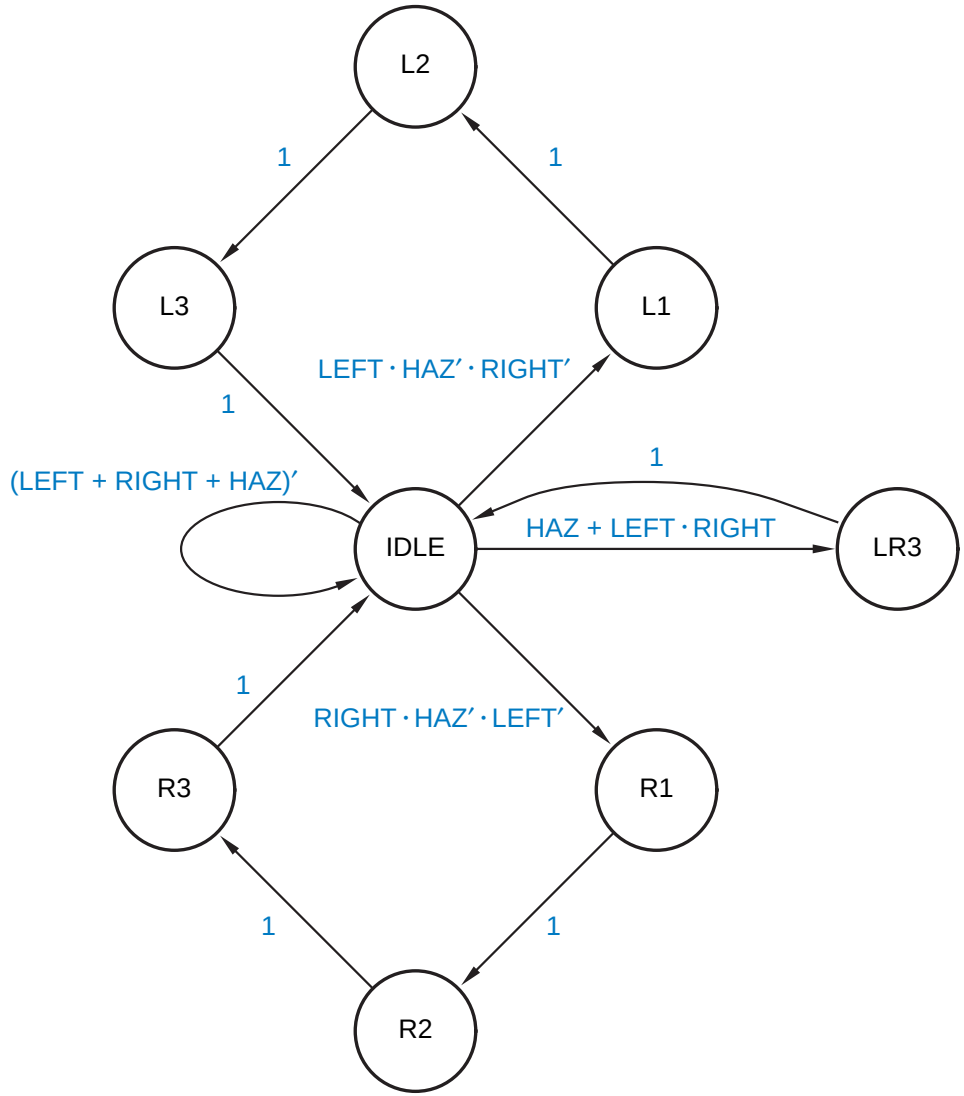


Figure 23 : Corrected state diagram for T-bird tail lights

Enhanced State Diagram for T-Bird Tail Lights

- There is one more problem with the state diagram: for a left- or right-turn, the cycle of 4 states cannot be interrupted by a HAZ input
- This problem is corrected in the enhanced version of the diagram, from fig 24
- For the synthesis of the state machine we have to code the 8 states, for which we need three flip-flops
- We use the state assignment from Table 7-16, fig 25
- Next step is to write a sort of transition table: we call the new format a *transition list* because it has one row for each transition or arc in the state diagram (Table 7-17, fig 26)
- The *transition equations* that define each next state variable (i.e., Q2*, Q1*, Q0*) can be obtained from the transition list by logical addition of all the expressions (current state and transition expression) for the rows in which the next state variable is 1

Transition Equations

- First, we have to develop a set of *transition equations* from the transition list
- A *transition equation* defines each next-state variable $V*$ in terms of the current state and input
- The transition list can be viewed as a sort of hybrid truth table in which the state variable combinations from the current state are listed explicitly and the input combinations are listed algebraically
- When we read a $V*$ column (e.g. $Q2*$, $Q1*$, or $Q0*$ in Table 7-17) we find a sequence of 0s and 1s, indicting the values of $V*$ for various (all, if we worked correctly) state/input combinations
- We define a row's *"transition p-term"* for each row of the transition list that contain a 1 in the $V*$ column
- The row's *"transition p-term"* is the product of the current's state minterm and the transition expression.

Transition Equations

Thus, a transition equation for a next-state variable $V*$ can be written using a sort of hybrid canonical sum:

$$V* = \sum_{\text{transition-list rows where V*=1}} (\text{transition p\_term})$$

The transition equation for $Q2*$ will be:

$Q2* =$

$Q2' \cdot Q1' \cdot Q0' \cdot (HAZ + LEFT \cdot RIGHT) + Q2' \cdot Q1' \cdot Q0' \cdot (RIGHT \cdot HAZ' \cdot LEFT') + Q2' \cdot Q1' \cdot Q0 \cdot HAZ + Q2' \cdot Q1 \cdot Q0 \cdot HAZ + Q2 \cdot Q1' \cdot Q0 \cdot HAZ' + Q2 \cdot Q1' \cdot Q0 \cdot HAZ + Q2 \cdot Q1 \cdot Q0 \cdot HAZ' + Q2 \cdot Q1 \cdot Q0 \cdot HAZ$

Similar equations can be written for $Q1*$ and $Q0*$.

The equations can be simplified using algebraic methods or, better, using CAD tools and HDLs.

Enhanced State Diagram for T-Bird Tail Lights

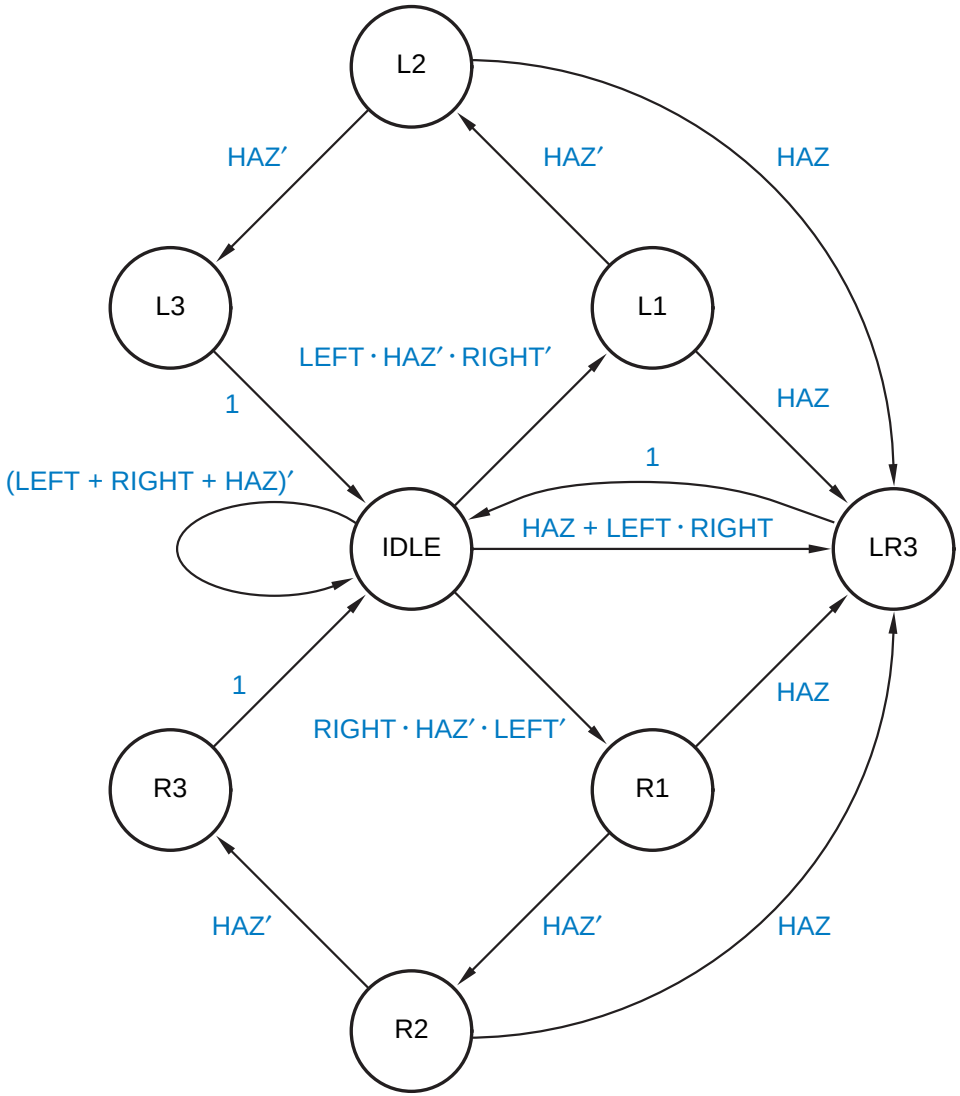


Figure 24 : Enhanced state diagram for T-bird tail lights

State Assignment for T-bird Tail Lights State Machine

Table 7-16
State assignment for T-bird tail-lights state machine.

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

Figure 25 : State assignment for T-bird tail-lights state machine

Transition List for T-bird Tail Lights State Machine

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

Table 7-17 Transition list for T-bird tail-lights state machine.

Figure 26 : Transition list for T-bird tail-lights state machine