ECE 27000 S22 P 10

Chapter 9. State Machines

In a *combinational* circuit, the output values depend only on the current input values.

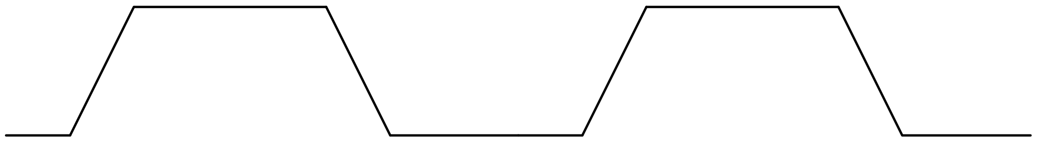
In a *sequential* circuit, the output values depend on the current as well as past input values.

The past input values determine the current *state* of the circuit.
The state is represented by a set of *state variables*.
Corresponding to each state variable there is one bit of memory.

The state, through the values of the state variables, contains enough information about the past to allow us to determine the output values of the circuit given its current input values.

The number of state variables in a digital logic circuit is finite.
For k state variables we have 2^k possible states (not all of them must be used).
A sequential circuit is sometimes called a *finite-state machine*.

For *synchronous* sequential circuits, state changes occur at times specified by a *clock* signal. The form of a clock signal:

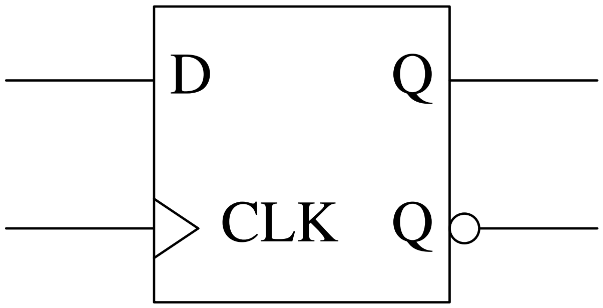


For an active high clock, state changes occur at the rising edge of the clock or when the clock is HIGH.

The *clock period* is the time between successive rising (falling) edges.
The *clock frequency* is the reciprocal of the period.
The *duty cycle* is the percentage of time that the clock signal is in its active level.

There are many types of memory elements for storing values of state variables.

Initially we will consider sequential circuits that use *positive-edge-triggered D flip-flop*s.
Logic symbol:



Operation:

D	CLK	Q	QN
0	L→H	0	1
1	L→H	1	0
x	0	last Q	last QN
x	1	last Q	last QN

It is convenient to describe the operation of the D flip-flop considering the current and next clock cycle,
before and after the next rising clock edge.

Q - current value of the state variable.
D - current value of the D input.
Q* - next value of the state variable, after the next rising edge of the clock.

D	Q	Q*
0	0	0
0	1	0
1	0	1
1	1	1

D	Q	Q*
0	x	0
1	x	1

Characteristic equation:
$Q* = D$

An S-R flip-flop:

S	R	Q	Q*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	unused
1	1	1	unused

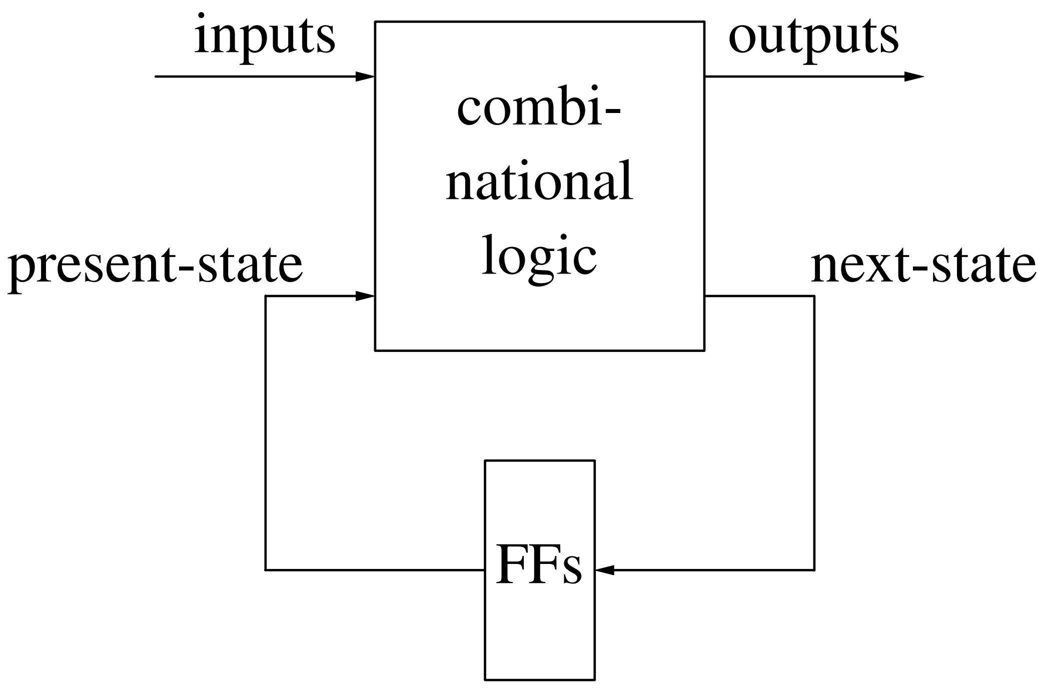
S	R	Q	Q*
0	0	0	0
0	0	1	1
0	1	x	0
1	0	x	1
1	1	x	unused

Characteristic equation:

$Q^* = S + R' \cdot Q$

9.2. State-Machine Structure and Analysis

The general form of a synchronous finite state-machine is the following:



In a Mealy machine:
next-state = F(present-state, input)
output = G(present-state, input)

In a Moore machine:
next-state = F(present-state, input)
output = G(present-state)

A special type of Moore machine is obtained when the states are encoded such that the state variables can serve as outputs.

This ensures that outputs are available as early as possible, and do not need to be computed through combinational logic (other than wires).

Analysis of State Machines

Steps in the analysis of synchronous finite-state machines:

Determine the next-state and output functions F and G.

First obtain the functions in terms of the flip-flop inputs (for example, obtain functions for S and R, or for D).

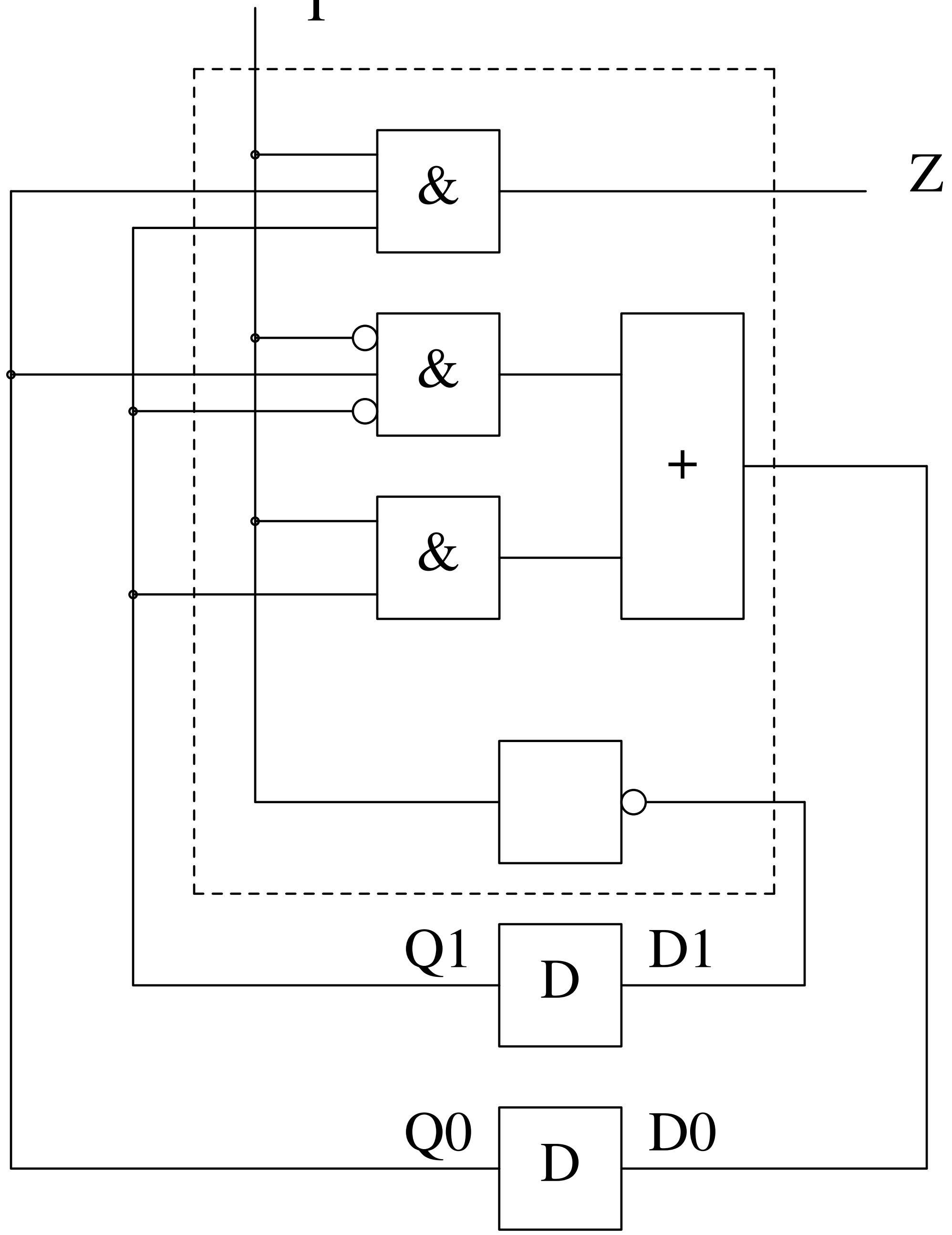
Then substitute the functions into the characteristic equations of the flip-flops in order to compute the next-states of the flip-flops.

Use F and G to construct a *transition table* that specifies the next-state and output for every combination of present-state and input.

Assign symbolic names to the states and obtain a *state table*.

Draw a state diagram.

Example of Analysis:



$D0 = I' \cdot Q0 \cdot Q1' + I \cdot Q1$
$D1 = I'$
$Z = I \cdot Q0 \cdot Q1$

Using $Q^* = D$:

$Q0^* = I' \cdot Q0 \cdot Q1' + I \cdot Q1$
$Q1^* = I'$
$Z = I \cdot Q0 \cdot Q1$

A transition and output table:

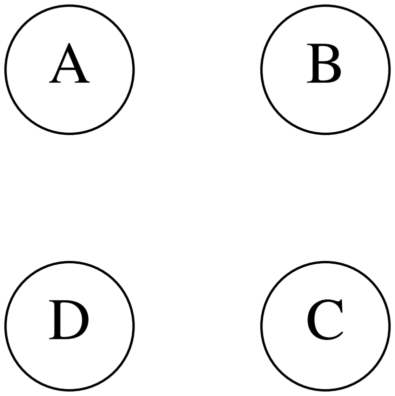
		Q0*Q1*		Z	
Q0	Q1	I=0	I=1	I=0	I=1
0	0	01	00	0	0
0	1	01	10	0	0
1	0	11	00	0	0
1	1	01	10	0	1

A state-table:

A - 00, B - 01, C - 10, D - 11

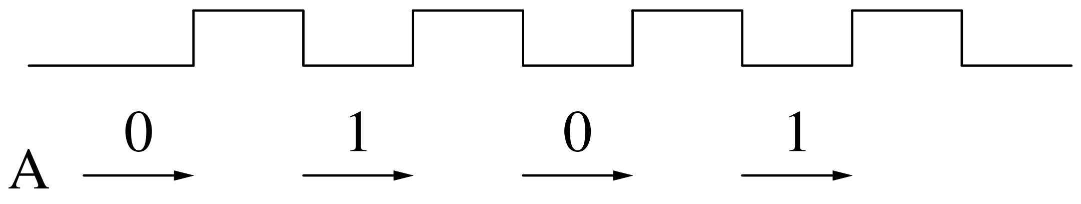
	S*		Z	
S	I=0	I=1	I=0	I=1
A	B	A	0	0
B	B	C	0	0
C	D	A	0	0
D	B	C	0	1

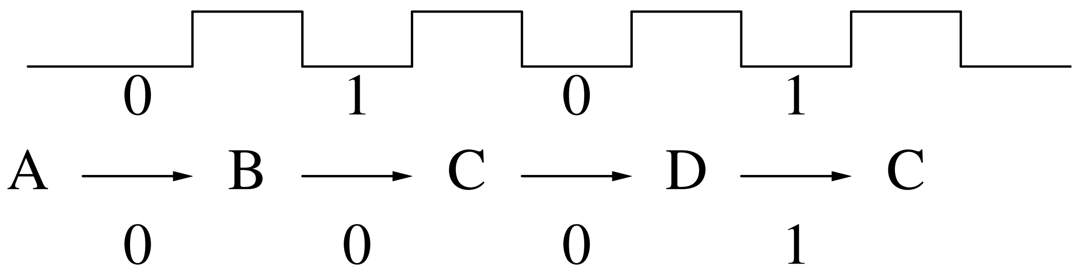
A state diagram:



	S*,Z	
S	I=0	I=1
A	B,0	A,0
B	B,0	C,0
C	D,0	A,0
D	B,0	C,1

The response of the circuit to the input sequence (0101) starting from state A:




0
1
0
1
A
B
C
D
C
0
0
0
1

9.3. State-Machine Design with State Tables

Given a functional description of the finite-state machine (a specification, typically in words), the synthesis (or design) steps are:

Find a state table.

Check if redundant states exist and eliminate them (state minimization).

Select the number of state variables. Assign combinations of state variables to the states (state assignment).

Derive transition and output tables.

Select a flip-flop type. Write an excitation table that specifies the next-state functions in terms of the flip-flop inputs.

Derive logic equations for the next-state variables and the outputs from the excitation table.

Perform logic minimization.

Example: Design a single-input, single-output sequence detector that produces an output 1 every time the sequence 0101 is detected on the input, and an output 0 at all other times.

input	0	0	1	0	1	0	1	0
output	0	0	0	0	1	0	1	0

The state table will have the following form:

		S^*, Z	
meaning	S	$X = 0$	$X = 1$
	A	$\cdots$	$\cdots$
	B	$\cdots$	$\cdots$
	$\cdots$	$\cdots$	$\cdots$

We need to associate a meaning with the states based on the function we need to implement.

For the start state A, the appropriate meaning is - no part of the sequence 0101 has been applied.

In state A, under input 0, go to state B -
the first symbol of the sequence 0101 has been applied.

In state A, under input 1, remain in state A -
no part of the sequence 0101 has been applied.

		S^*, Z	
meaning	S	$X = 0$	$X = 1$
-	A	$B, 0$	$A, 0$
0	B		

In state B (0):
Under input 0, stay in state B.
Under input 1, go to state C -
the first two symbols of the sequence 0101 have been applied.

		S^*, Z	
meaning	S	$X = 0$	$X = 1$
-	A	$B, 0$	$A, 0$
0	B	$B, 0$	$C, 0$
01	C		

In state C (01):
Under input 0, go to state D -
the first three symbols of the sequence 0101 have been applied.
Under input 1, go to state A -
no symbol of the sequence has been applied.

		S^*, Z	
meaning	S	$X = 0$	$X = 1$
-	A	$B, 0$	$A, 0$
0	B	$B, 0$	$C, 0$
01	C	$D, 0$	$A, 0$
010	D		

In state D (010):
Under input 0, go to state B.
Under input 1, produce an output 1.
The last four input symbols are 0101.
Go to the state corresponding to the prefix 01 of the sequence - state C.

meaning	S	S^*, Z	
		$X = 0$	$X = 1$
-	A	$B, 0$	$A, 0$
0	B	$B, 0$	$C, 0$
01	C	$D, 0$	$A, 0$
010	D	$B, 0$	$C, 1$

$$A \xrightarrow[0]{0} B \xrightarrow[0]{1} C \xrightarrow[0]{0} D \xrightarrow[1]{1} C \xrightarrow[0]{0} D \xrightarrow[1]{1} C \rightarrow \cdots$$

To identify non-overlapping appearances of 0101:
From state D with input 1, instead of going to C (which remembers that 01 has been applied), go to state A.

	S^*, Z	
S	$X = 0$	$X = 1$
A	$B, 0$	$A, 0$
B	$B, 0$	$C, 0$
C	$D, 0$	$A, 0$
D	$B, 0$	$A, 1$

$$A \xrightarrow[0]{0} B \xrightarrow[0]{1} C \xrightarrow[0]{0} D \xrightarrow[1]{1} A \xrightarrow[0]{0} B \xrightarrow[0]{1} C \rightarrow \cdots$$

Example: When a certain serial binary communication channel is operating correctly, all blocks of 0s are of even length and all blocks of 1s are of odd length. Show the state table of a machine which will produce an output $Z = 1$ whenever a discrepancy from the above pattern is detected.

Input	0	0	1	0	0	0	1
Output	0	0	0	0	0	0	1

Input	1	1	0	1	1	0	0
Output	0	0	0	1	0	1	0

It is possible to consider states that represent:
start
single 0
two 0s
three 0s
. . .
one 1
two 1s
three 1s
. . .

However:
there is no known upper bound on the length of a block, and
there is no need to distinguish between different even/odd numbers of 0s/1s.

The states should represent:
initial
even block of 0s
odd block of 0s
even block of 1s
odd block of 1s

		S^*, Z	
meaning	S	$X = 0$	$X = 1$
initial	$START$	$B, 0$	$C, 0$
even block of 0s	A	$B, 0$	$C, 0$
odd block of 0s	B	$A, 0$	$C, 1$
odd block of 1s	C	$B, 0$	$D, 0$
even block of 1s	D	$B, 1$	$C, 0$

START and *A* are identical.
It is possible to remove one of them.

		S^*, Z	
	S	$X = 0$	$X = 1$
even block of 0s	A	$B, 0$	$C, 0$
odd block of 0s	B	$A, 0$	$C, 1$
odd block of 1s	C	$B, 0$	$D, 0$
even block of 1s	D	$B, 1$	$C, 0$

We could have avoided state START through the following arguments:
Initially we have zero 0s and zero 1s.
Zero is an even number.
We can start from an even block of 0s or an even block of 1s.
If we start from an even block of 1s and the first input is 0, the output will be 1.
To avoid it, we start from an even block of 0s.

State Minimization

Two states S_i and S_j of a machine M are *distinguishable* if and only if there exists at least one finite input sequence X, such that when X is applied to M starting from state S_i it produces a different output sequence than when X is applied to M starting from state S_j.

States S_i and S_j of machine M are said to be *equivalent* if and only if for every possible input sequence, the same output sequence is produced regardless of whether M is in state S_i or S_j.

The goal of state minimization is to identify, and then eliminate, equivalent states.

A state minimization procedure considers increasingly long input sequences to distinguish the states of the machine.

For length 0:
no states are distinguished.

For length $k \geq 1$:
it partitions the states into subsets.
Within each subset the states cannot be distinguished by a sequence of length k.

It is possible to prove that the process will terminate after considering a finite sequence length (equal to the number of states).
The states that could not be distinguished before termination are equivalent.

Example:

S	S^*, z $X = 0$	 $X = 1$
A	$E, 0$	$D, 1$
B	$F, 0$	$D, 0$
C	$E, 0$	$B, 1$
D	$F, 0$	$B, 0$
E	$C, 0$	$F, 1$
F	$B, 0$	$C, 0$

$P_0 = (ABCDEF)$

$P_1 = (ACE)(BDF)$

P_2:

$ACE \overset{0}{\rightarrow} EEC$.

E and C are in the same block of P_1.

$ACE \overset{1}{\rightarrow} DBF$.

D, B and F are in the same block of P_1.

$BDF \overset{0}{\rightarrow} FFB$.

F and B are in the same block of P_1.

$BDF \xrightarrow{1} DBC$.

D and B are in the same block of P_1.

C is in a different block.

Therefore, both B and D can be distinguished from F by a sequence of length 2:

$$B \overset{1}{\underset{0}{\rightarrow}} D \overset{1}{\underset{0}{\rightarrow}} B$$

$$D \overset{1}{\underset{0}{\rightarrow}} B \overset{1}{\underset{0}{\rightarrow}} D$$

$$F \overset{1}{\underset{0}{\rightarrow}} C \overset{1}{\underset{1}{\rightarrow}} B$$

$P_2 = (ACE)(BD)(F)$

P_3:

$ACE \overset{0}{\rightarrow} EEC$.

E and C are in the same block of P_2.

$ACE \overset{1}{\rightarrow} DBF$.

D and B are in the same block of P_2.

F is in a separate block.

This divides ACE into (AC) and (E).

$BD \overset{0}{\rightarrow} FF$.

$BD \overset{1}{\rightarrow} DB$.

$P_3 = (AC)(E)(BD)(F)$

P_4:

$AC \overset{0}{\rightarrow} EE$.

$AC \overset{1}{\rightarrow} DB$.

$BD \overset{0}{\rightarrow} FF$.

$BD \overset{1}{\rightarrow} DB$.

$P_4 = (AC)(E)(BD)(F)$

Since $P_4 = P_3$, no additional states can be distinguished.

The equivalence classes of the machine:
(AC), (E), (BD) and (F).

Original machine:

S	S^*, z $X = 0$	$X = 1$
A	$E, 0$	$D, 1$
B	$F, 0$	$D, 0$
C	$E, 0$	$B, 1$
D	$F, 0$	$B, 0$
E	$C, 0$	$F, 1$
F	$B, 0$	$C, 0$

Reduced machine:
$(AC) \rightarrow a$, $(E) \rightarrow b$, $(BD) \rightarrow c$ and $(F) \rightarrow d$.

S	S^*, z $X = 0$	$X = 1$
a	$b, 0$	$c, 1$
b	$a, 0$	$d, 1$
c	$d, 0$	$c, 0$
d	$c, 0$	$a, 0$

State Assignment

For a state table with n states we need at least $\lceil \log_2 n \rceil$ state variables.
It is possible to use more if it leads to a simplification of the logic.
Typically the smallest number of state variables yields the simplest logic.

The state assignment can have a significant effect on the cost of the logic. Different state assignments will result in different costs.

Example: The sequence detector for the sequence 0101

	S^*, z	
S	$X = 0$	$X = 1$
A	$B, 0$	$A, 0$
B	$B, 0$	$C, 0$
C	$D, 0$	$A, 0$
D	$B, 0$	$C, 1$

State assignment 1:

			Q0* Q1* , Z	
	Q0	Q1	X=0	X=1
A	0	0	01, 0	00, 0
B	0	1	01, 0	11, 0
C	1	1	10, 0	00, 0
D	1	0	01, 0	11, 1

With D flip-flops, D0 = Q0* and D1 = Q1*.

$$D0 = Q0* = X' \cdot Q0 \cdot Q1 + X \cdot Q0' \cdot Q1 +$$
$$+X \cdot Q0 \cdot Q1'.$$
$$D1 = Q1* = X' \cdot Q0' + Q0 \cdot Q1' + Q0' \cdot Q1$$
$$Z = X \cdot Q0 \cdot Q1'$$

This state assignment results in 18 literals.

State assignment 2:

			Q0* Q1* ,Z	
	Q0	Q1	X=0	X=1
A	0	0	01,0	00,0
B	0	1	01,0	10,0
C	1	0	11,0	00,0
D	1	1	01,0	10,1

$D0 = Q0* = X' \cdot Q0 \cdot Q1' + X \cdot Q1.$
$D1 = Q1* = X'.$
$Z = X \cdot Q0 \cdot Q1.$

This state assignment results in 9 literals.

The total number of state assignments for a machine with n states and using k bits for state encoding is $\dfrac{2^k!}{(2^k - n)!}$.

The first class of state encoding procedures tries to help logic minimization procedures to obtain smaller implementations.

The basic property they use is that if two states are given adjacent state encodings, then it is possible to minimize their next-state and output functions.
Adjacent - differ in the value of a single state variable.

Example:

	S^*, z
S	Xi
0000 A	C 0011
0001 B	C 0011

A different approach to state encoding is based on reducing the dependence among the state variables through decomposition.

In general,

$$Qi^* = f_i(Q0, Q1, \cdots, Qk-1, \\ X0, X1, \cdots, Xn-1).$$

If we can ensure that Qi * will depend only on a subset of $Q0, Q1, \cdots, Qk-1$, we are likely to reduce the number of literals in its implementation.

Unused States

With k state variables it is possible to encode $n \leq 2^k$ states.
If $n < 2^k$, there are $2^k - n$ unused states.
The unused states can be used in one of two ways:

Direct all the state-transitions from the unused states into the initial state, or into an idle, error or safe state.

Specify the unused state-transitions to minimize the logic.

Example: Design a finite-state machine with two inputs, X and Y, and one output, Z.
The output should be 1 if the number of 1 inputs on X and Y is a multiple of three, and 0 otherwise.

input	01	10	00	01
output	0	0	0	1

input	11	11	11	01	01	10
output	0	0	1	0	0	1

It is convenient to assign to state Si the meaning:
The number of 1 inputs, modulo 3, is equal to i.

	S*,Z			
S	XY=00	XY=01	XY=10	XY=11
S0	S0,1	S1,0	S1,0	S2,0
S1	S1,0	S2,0	S2,0	S0,1
S2	S2,0	S0,1	S0,1	S1,0

A Moore machine for the same problem:

	S*				
S	XY=00	XY=01	XY=10	XY=11	Z
S0	S0	S1	S1	S2	1
S1	S1	S2	S2	S0	0
S2	S2	S0	S0	S1	0

The output is 1 after the machine enters state S0, one clock cycle after the number of 1 inputs reaches 3.

We can use two state variables with one unused state.
Let us use the state assignment:
S0 - 00, S1 - 01, S2 - 11, S3 - 10.

The transition and excitation table for D flip-flops:

	D0 D1,Z			
Q0 Q1	XY=00	XY=01	XY=10	XY=11
00	00,1	01,0	01,0	11,0
01	01,0	11,0	11,0	00,1
11	11,0	00,1	00,1	01,0
10	xx,x	xx,x	xx,x	xx,x

D0:

X Y

Q0 Q1	00	01	11	10
00			1	
01		1		1
11	1			
10	x	x	x	x

$$D0 = Q0 \cdot X' \cdot Y' + Q0' \cdot Q1 \cdot X' \cdot Y + \\ Q1' \cdot X \cdot Y + Q0' \cdot Q1 \cdot X \cdot Y'.$$

D1:

Q0 Q1 \ X Y	00	01	11	10
00		1	1	1
01	1	1		1
11	1		1	
10	x	x	x	x

$$D1 = Q1 \cdot X' \cdot Y' + Q0' \cdot X' \cdot Y + Q1' \cdot Y + \\ Q0' \cdot X \cdot Y' + Q0 \cdot X \cdot Y.$$

Z:

Q0 Q1 \ X Y	00	01	11	10
00	1			
01			1	
11		1		1
10	x	x	x	x

$$Z = Q1' \cdot X' \cdot Y' + Q0 \cdot X' \cdot Y +$$
$$+Q0' \cdot Q1 \cdot X \cdot Y + Q0 \cdot X \cdot Y'$$

Example of decomposition:

	S^*, Z	
S	$X = 0$	$X = 1$
A	$A, 0$	$D, 1$
B	$A, 0$	$C, 0$
C	$C, 0$	$B, 0$
D	$C, 0$	$A, 1$

Considering next-states for pairs of present-states: $AB \to AA, CD$; $CD \to CC, AB$.

Suppose that we assign a state variable $Q0$ such that $Q0 = 0$ for $\overline{AB}$ and $Q0 = 1$ for $\overline{CD}$ (or vice versa).
We say in this case that $Q0$ is assigned based on the partition $\{\overline{AB}, \overline{CD}\}$.

$D0$ will depend only on the values of $Q0$ and X.
$D0 = D0(X, Q0)$.
This is likely to be simpler than
$D0 = D0(X, Q0, Q1)$.

	S^*, Z	
S	$X = 0$	$X = 1$
A	$A, 0$	$D, 1$
B	$A, 0$	$C, 0$
C	$C, 0$	$B, 0$
D	$C, 0$	$A, 1$

Considering next-states for different pairs of present-states:
$AC \rightarrow AC, BD$; $BD \rightarrow AC, AC$.

Suppose that we assign a state variable $Q1$ such that $Q1 = 0$ for $\overline{AC}$ and $Q1 = 1$ for $\overline{BD}$ (or vice versa).
We say that $Q1$ is assigned based on the partition $\{\overline{AC}, \overline{BD}\}$.

$D1$ will depend only on the values of $Q1$ and X.
$D1 = D1(X, Q1)$.
This is likely to be simpler than
$D1 = D0(X, Q0, Q1)$.

We need to verify that this would be a complete state assignment.
$Q0 = 0$ for $\overline{AB}$ and $Q0 = 1$ for $\overline{CD}$.
$Q1 = 0$ for $\overline{AC}$ and $Q1 = 1$ for $\overline{BD}$

S	Q0	Q1
A	0	0
B	0	1
C	1	0
D	1	1

Transition and excitation tables using D flip-flops:

		D0 D1, Z	
Q0	Q1	X=0	X=1
0	0	00,0	11,1
0	1	00,0	10,0
1	0	10,0	01,0
1	1	10,0	00,1

$D0 = X' \cdot Q0 + X \cdot Q0'$
$D1 = X \cdot Q1'$
$Z = X \cdot Q0' \cdot Q1' + X \cdot Q0 \cdot Q1$

Reducing the output dependence:

S	S^*, Z $X = 0$	$X = 1$
A	$B, 1$	$D, 0$
B	$A, 0$	$C, 1$
C	$D, 0$	$A, 1$
D	$C, 1$	$B, 0$

If we assign a state variable Q based on the partition $\{\overline{AD}, \overline{BC}\}$, the output will depend only on Q.

S	Z $X = 0$	$X = 1$
$AD\ Q = 0$	1	0
$BC\ Q = 1$	0	1

$Z = Q' \cdot X' + Q \cdot X$

Example: An up-counter

			Q_2*	Q_1*	Q_0*
Q_2	Q_1	Q_0	D_2	D_1	D_0
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

$$D_0 = Q_0'$$

$$D_1 = Q_1 \oplus Q_0$$

$$\begin{aligned} D_2 &= Q_2' \cdot Q_1 \cdot Q_0 + Q_2 \cdot (Q_1' + Q_0') \\ &= Q_2' \cdot (Q_1 \cdot Q_0) + Q_2 \cdot (Q_1 \cdot Q_0)' \\ &= Q_2 \oplus (Q_1 \cdot Q_0) \end{aligned}$$

Example: A down-counter

Q_2	Q_1	Q_0	D_2	D_1	D_0
0	0	0	1	1	1
0	0	1	0	0	0
0	1	0	0	0	1
0	1	1	0	1	0
1	0	0	0	1	1
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0

$$D_0 = Q_0'$$

$$D_1 = (Q_1 \oplus Q_0)'$$

$$\begin{aligned} D_2 &= Q_2' \cdot Q_1' \cdot Q_0' + Q_2 \cdot (Q_1 + Q_0) \\ &= Q_2' \cdot (Q_1 + Q_0)' + Q_2 \cdot (Q_1 + Q_0) \\ &= Q_2 \oplus (Q_1 + Q_0)' \end{aligned}$$

9.6. State-Machine Design with Verilog

	AB				
S	00	01	11	10	Z
S0	S1	S1	S2	S2	0
S1	S3	S3	S2	S2	0
S2	S1	S1	S4	S4	0
S3	S3	S3	S4	S2	1
S4	S1	S3	S4	S4	1

```
module VrFSM(CLOCK,A,B,Z);
    input CLOCK,A,B;
    output reg Z;
    reg [2:0] Sreg,Snext;
    parameter [2:0] S0=3'b000,
                    S1=3'b001,
                    S2=3'b010,
                    S3=3'b011,
                    S4=3'b100;

    /* create state memory */
    always @(posedge CLOCK)
        Sred <= Snext;
```

```
/* create next-state logic */
always @(A,B,Sreg)
begin
    case(Sreg)
        S0: if(A==0) Snext=S1;
            else Snext=S2;
        S1: if(A==0) Snext=S3;
            else Snext=S2;
        S2: if(A==0) Snext=S1;
            else Snext=S4;
        S3: if(A==0) Snext=S3;
            else if(B==0) Snext=S4;
            else Snext=S2;
        S4: if((A==0)&&(B==0)) Snext=S1;
            else if((A==0)&&(B==1)) Snext=S3;
            else Snext=S4;
        default: Snext=S0;
    endcase
end
```

```
/* create output logic */
always @(Sreg)
    case(Sreg)
        S0,S1,S2: Z=0;
        S3,S4: Z=1;
        default: Z=0;
    endcase
endmodule
```

The previous state table was designed for the following specification.
Design a state machine with two inputs, A and B, and a single output Z that is 1 if:
A had the same value at each of the two previous clock cycles, or
B has been 1 since the last time the first condition was true.

		AB				
meaning	S	00	01	11	10	Z
initial	S0	S1	S1	S2	S2	0
A0	S1	S3	S3	S2	S2	0
A1	S2	S1	S1	S4	S4	0
A00	S3	S3	S3	S4	S2	1
A11	S4	S1	S3	S4	S4	1

$$S0 \overset{0x}{\rightarrow} S1 \overset{0x}{\rightarrow} S3 \overset{0x}{\rightarrow} S3 \overset{11}{\rightarrow} S4 \overset{01}{\rightarrow} S3$$
$$S0 \overset{0x}{\rightarrow} S1 \overset{0x}{\rightarrow} S3 \overset{10}{\rightarrow} S2$$
$$S0 \overset{0x}{\rightarrow} S1 \overset{0x}{\rightarrow} S3 \overset{11}{\rightarrow} S4 \overset{1x}{\rightarrow} S4$$