CMSC 313
COMPUTER ORGANIZATION
&
ASSEMBLY LANGUAGE PROGRAMMING

LECTURE 24, SPRING 2013

TOPICS TODAY

- **Example: Sequence Detector**
- **Finite State Machine Simplification**
    - Circuit Minimization
    - State Reduction
    - State Assignment
    - Choice of Flip Flop (not covered)

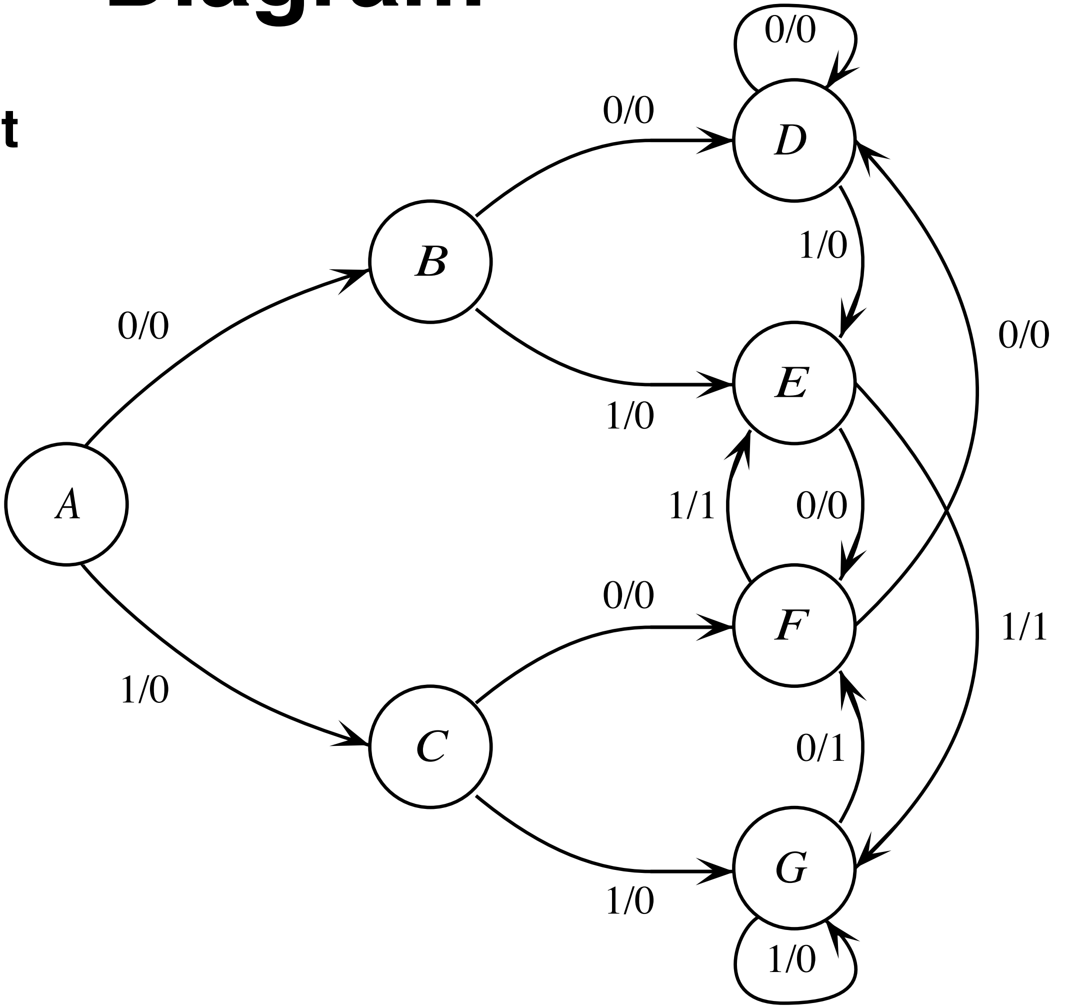
EXAMPLE:
SEQUENCE DETECTOR

Example: A Sequence Detector

- Example: Design a machine that outputs a 1 when exactly two of the last three inputs are 1.
- *e.g.* input sequence of 011011100 produces an output sequence of 001111010.
- Assume input is a 1-bit serial line.
- Use D flip-flops and 8-to-1 Multiplexers.
- Start by constructing a state transition diagram (next slide).

Sequence Detector State Transition Diagram

- **Design a machine that outputs a 1 when exactly two of the last three inputs are 1.**



Sequence Detector State Table

Input / Present state	X = 0	X = 1
A	B/0	C/0
B	D/0	E/0
C	F/0	G/0
D	D/0	E/0
E	F/0	G/1
F	D/0	E/1
G	F/1	G/0

Sequence Detector State Assignment

Input Present state	X	
	0	1
$S_2S_1S_0$	$S_2S_1S_0Z$	$S_2S_1S_0Z$
A: 000	001/0	010/0
B: 001	011/0	100/0
C: 010	101/0	110/0
D: 011	011/0	100/0
E: 100	101/0	110/1
F: 101	011/0	100/1
G: 110	101/1	110/0

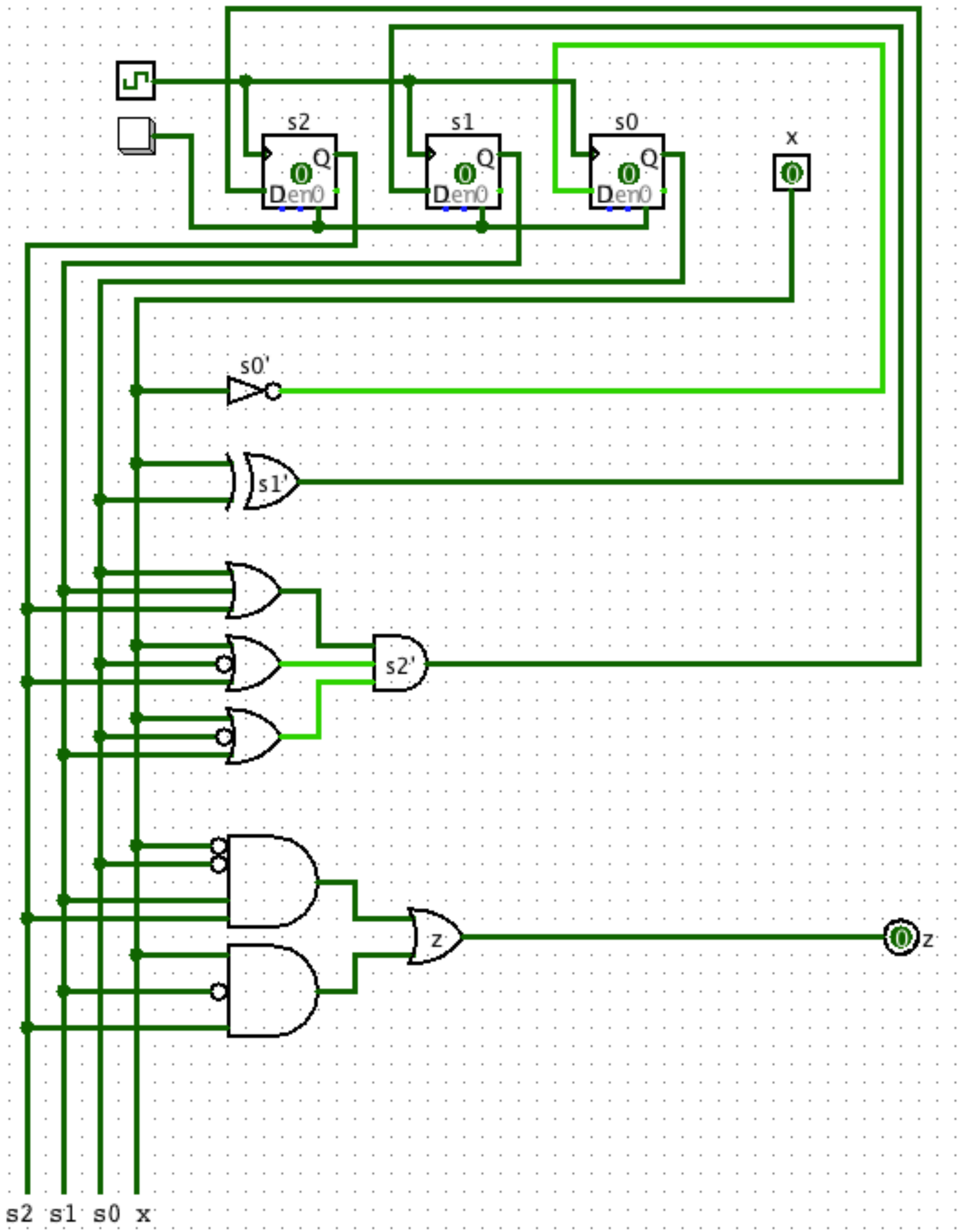
(a)

Input and state at time *t*				Next state and output at time *t*+1			
s_2	s_1	s_0	x	s_2	s_1	s_0	z
0	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0
0	0	1	0	0	1	1	0
0	0	1	1	1	0	0	0
0	1	0	0	1	0	1	0
0	1	0	1	1	1	0	0
0	1	1	0	0	1	1	0
0	1	1	1	1	0	0	0
1	0	0	0	1	0	1	0
1	0	0	1	1	1	0	1
1	0	1	0	0	1	1	0
1	0	1	1	1	0	0	1
1	1	0	0	1	0	1	1
1	1	0	1	1	1	0	0
1	1	1	0	d	d	d	d
1	1	1	1	d	d	d	d

(b)

Sequence Dectector

Output 1 when EXACTLY two of last three bits are 1

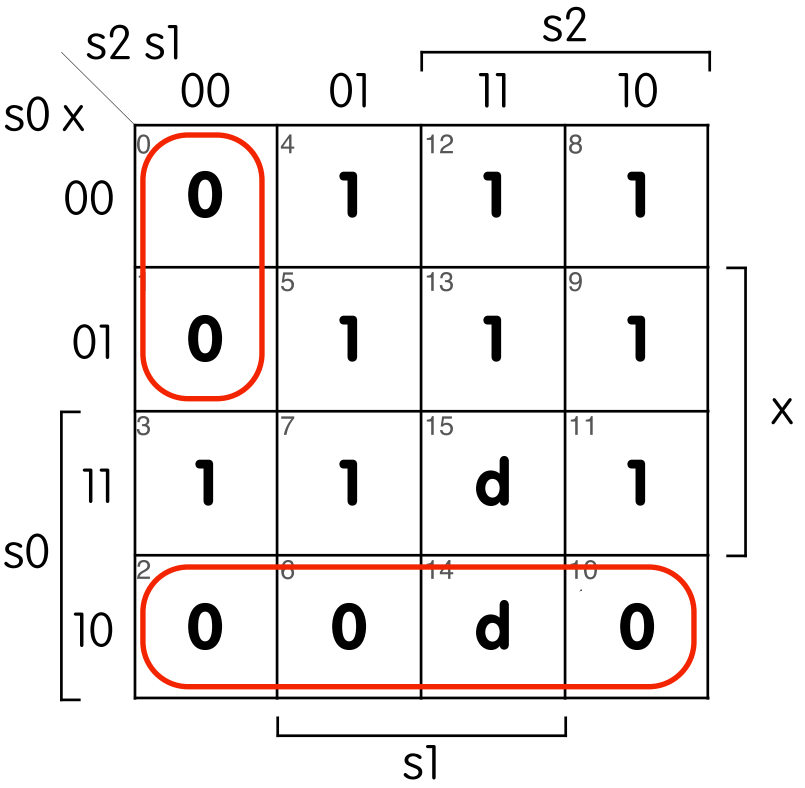


FINITE STATE MACHINE SIMPLIFICATION

CIRCUIT MINIMIZATION

Sequence Detector

	s2	s1	s0	x	s2'	s1'	s0'	z
0	0	0	0	0	0	0	1	0
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	1	0
3	0	0	1	1	1	0	0	0
4	0	1	0	0	1	0	1	0
5	0	1	0	1	1	1	0	0
6	0	1	1	0	0	1	1	0
7	0	1	1	1	1	0	0	0
8	1	0	0	0	1	0	1	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	0	1	1	0
11	1	0	1	1	1	0	0	1
12	1	1	0	0	1	0	1	1
13	1	1	0	1	1	1	0	0
14	1	1	1	0	d	d	d	d
15	1	1	1	1	d	d	d	d

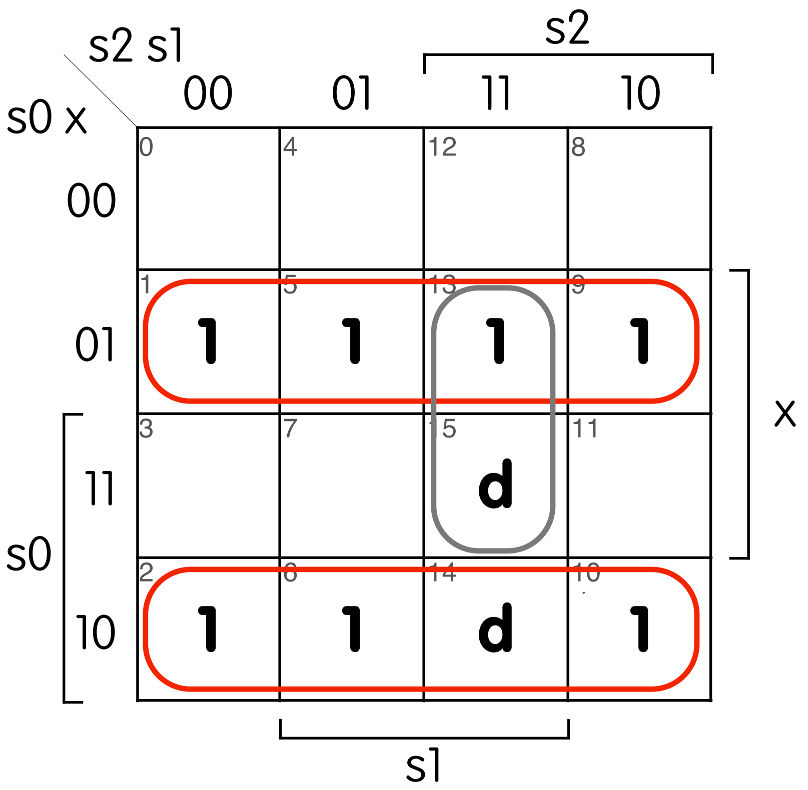


s0 x \ s2 s1	00	01	11	10
00	0	1	1	1
01	0	1	1	1
11	1	1	d	1
10	0	0	d	0

$$s2' = (\overline{s0} + x)(s2 + s1 + s0)$$

Sequence Detector

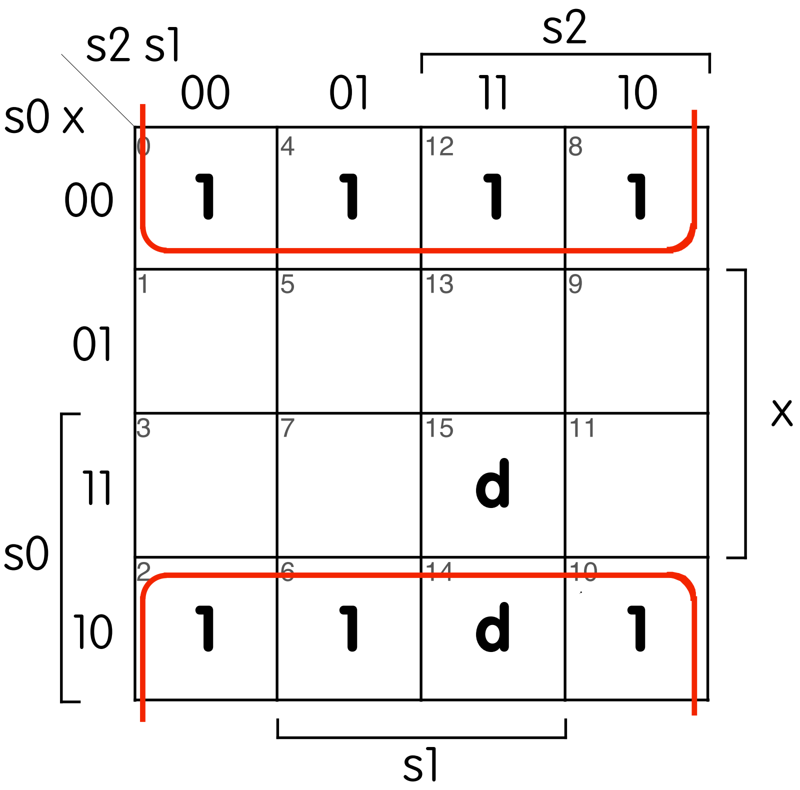
	s2	s1	s0	x	s2'	s1'	s0'	z
0	0	0	0	0	0	0	1	0
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	1	0
3	0	0	1	1	1	0	0	0
4	0	1	0	0	1	0	1	0
5	0	1	0	1	1	1	0	0
6	0	1	1	0	0	1	1	0
7	0	1	1	1	1	0	0	0
8	1	0	0	0	1	0	1	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	0	1	1	0
11	1	0	1	1	1	0	0	1
12	1	1	0	0	1	0	1	1
13	1	1	0	1	1	1	0	0
14	1	1	1	0	d	d	d	d
15	1	1	1	1	d	d	d	d



$$s1' = \overline{s0}\ x + s0\ \overline{x} = s0 \text{ xor } x$$

Sequence Detector

	s2	s1	s0	x	s2'	s1'	s0'	z
0	0	0	0	0	0	0	1	0
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	1	0
3	0	0	1	1	1	0	0	0
4	0	1	0	0	1	0	1	0
5	0	1	0	1	1	1	0	0
6	0	1	1	0	0	1	1	0
7	0	1	1	1	1	0	0	0
8	1	0	0	0	1	0	1	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	0	1	1	0
11	1	0	1	1	1	0	0	1
12	1	1	0	0	1	0	1	1
13	1	1	0	1	1	1	0	0
14	1	1	1	0	d	d	d	d
15	1	1	1	1	d	d	d	d

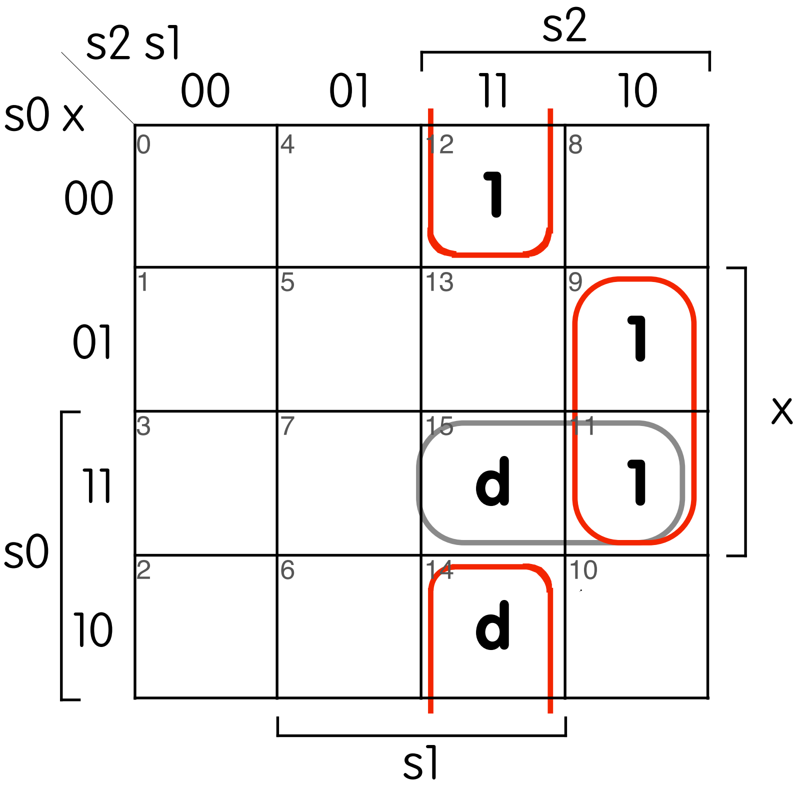


s0 x \ s2 s1	00	01	11	10
00	1	1	1	1
01				
11			d	
10	1	1	d	1

$$s0' = \overline{x}$$

Sequence Detector

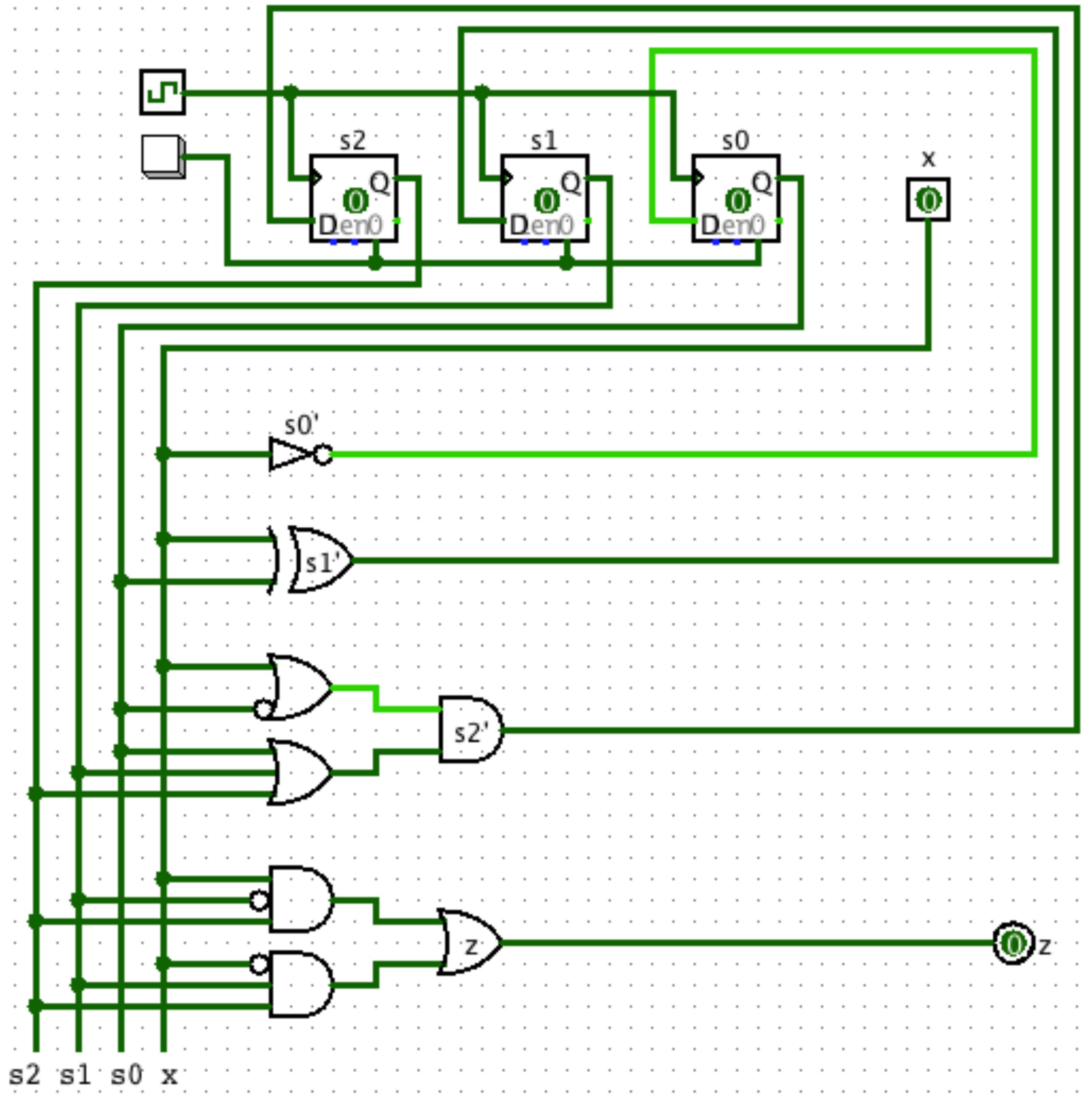
	s2	s1	s0	x	s2'	s1'	s0'	z
0	0	0	0	0	0	0	1	0
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	1	0
3	0	0	1	1	1	0	0	0
4	0	1	0	0	1	0	1	0
5	0	1	0	1	1	1	0	0
6	0	1	1	0	0	1	1	0
7	0	1	1	1	1	0	0	0
8	1	0	0	0	1	0	1	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	0	1	1	0
11	1	0	1	1	1	0	0	1
12	1	1	0	0	1	0	1	1
13	1	1	0	1	1	1	0	0
14	1	1	1	0	d	d	d	d
15	1	1	1	1	d	d	d	d



$$z = s2\ \overline{s1}\ x + s2\ s1\ \overline{x}$$

Sequence Dectector (optimized)

Output 1 when EXACTLY two of last three bits are 1



Notes on K-maps

- Also works for POS
- Takes 2^n time for formulas with n variables
- Only optimizes two-level logic
    - ◇ Reduces number of terms, then number of literals in each term
- Assumes inverters are free
- Does not consider minimizations across functions
- Circuit minimization is generally a hard problem
- Quine-McCluskey can be used with more variables
- CAD tools are available if you are serious

Karnaugh Maps

- **Implicant:** rectangle with 1, 2, 4, 8, 16 ... 1's
- **Prime Implicant:** an implicant that cannot be extended into a larger implicant
- **Essential Prime Implicant:** the only prime implicant that covers some 1
- **K-map Algorithm (not from M&H):**

  1. Find ALL the prime implicants. Be sure to check every 1 and to use don't cares.

  2. Include all essential prime implicants.

  3. Try all possibilities to find the minimum cover for the remaining 1's.

CIRCUIT MINIMIZATION IS HARD

- **Unix systems store passwords in encrypted form.**
  - User types x, system computes f(x) and looks for f(x) in a file
- **Suppose we use 64-bit passwords and I want to find the password x such that f(x) = y.**
- **Let $g_i(x)$ = 0 if f(x) = y and the i^{th} bit of x is 0.**
  1 otherwise

- **If the i^{th} bit of x is 1, then $g_i(x)$ outputs 1 for every x and $g_i(x)$ has a very, very simple circuit.**
- **If you can simplify every circuit quickly, then you can crack passwords quickly.**

Simplifying Finite State Machines

- State Reduction: equivalent FSM with fewer states
- State Assignment: choose an assignment of bit patterns to states (e.g., A is 010) that results in a smaller circuit
- Choice of flip-flops: use D flip-flops, J-K flip-flops or a T flip-flops? a good choice could lead to simpler circuits.

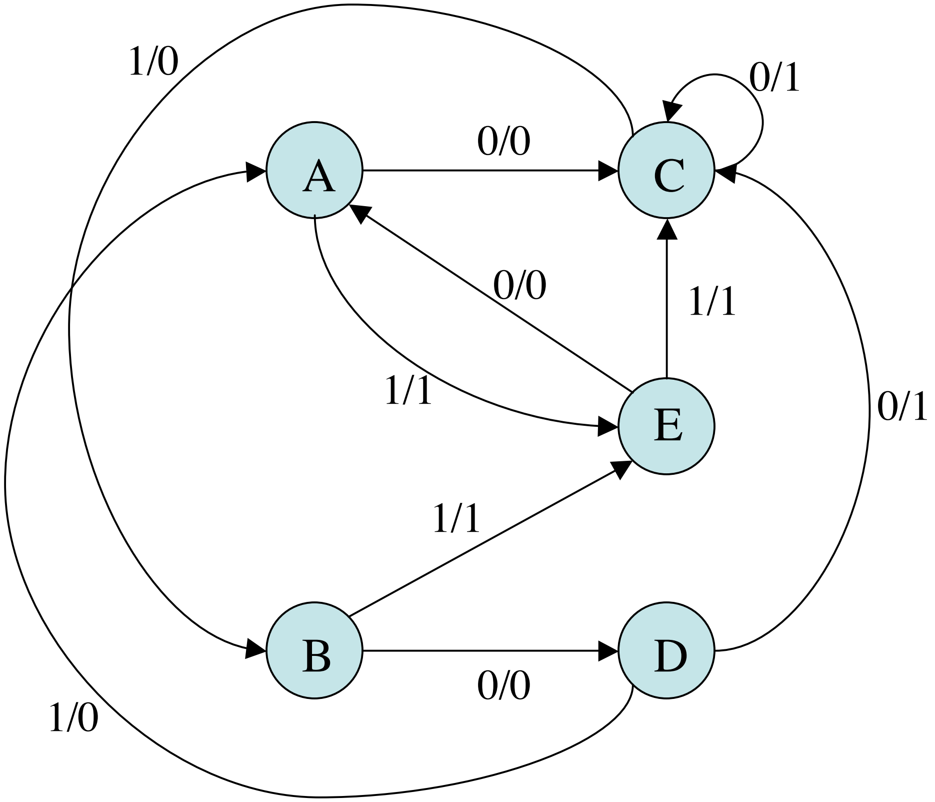
STATE REDUCTION

State Reduction

- **Description of state machine M_0 to be reduced.**

Present state \ Input	X = 0	X = 1
A	C/0	E/1
B	D/0	E/1
C	C/1	B/0
D	C/1	A/0
E	A/0	C/1

State Reduction Example: original transition diagram



State Reduction Algorithm

1. Use a 2-dimensional table — an entry for each pair of states.
2. Two states are "distinguished" if:

   a. States X and Y of a finite state machine M are distinguished if there exists an input r such that the output of M in state X reading input r is different from the output of M in state Y reading input r.

   b. States X and Y of a finite state machine are distinguished if there exists an input r such that M in state X reading input r goes to state X', M in state Y reading input r goes to state Y' and we already know that X' and Y' are distinguished states.

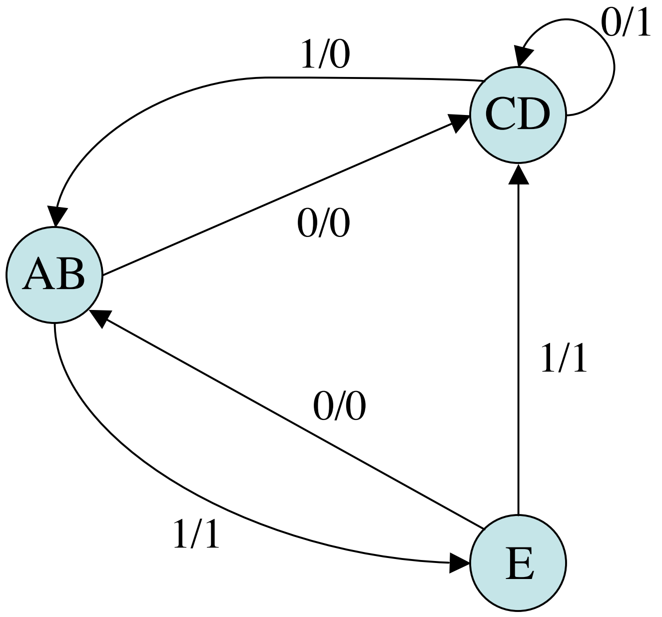
3. For each pair (X,Y), check if X and Y are distinguished using the definition above.
4. At the end of the algorithm, states that are not found to be distinguished are in fact equivalent.

State Reduction Table

- An **x** entry indicates that the pair of states are known to be distinguished.
- A & B are equivalent, C & D are equivalent

	A	B	C	D	E
A			x	x	x
B			x	x	x
C					x
D					x
E					

State Reduction Example: reduced transition diagram



State Reduction Algorithm Performance

- As stated, the algorithm takes $O(n^4)$ time for a FSM with n states, because each pass takes $O(n^2)$ time and we make at most $O(n^2)$ passes.
- A more clever implementation takes $O(n^2)$ time.
- The algorithm produces a FSM with the fewest number states possible.
- Performance and correctness can be proven.

STATE ASSIGNMENT

The State Assignment Problem

- **Two state assignments for machine M_2.**

Input P.S.	X	
	0	1
A	$B/1$	$A/1$
B	$C/0$	$D/1$
C	$C/0$	$D/0$
D	$B/1$	$A/0$

Machine M_2

Input S_0S_1	X	
	0	1
A: 00	01/1	00/1
B: 01	10/0	11/1
C: 10	10/0	11/0
D: 11	01/1	00/0

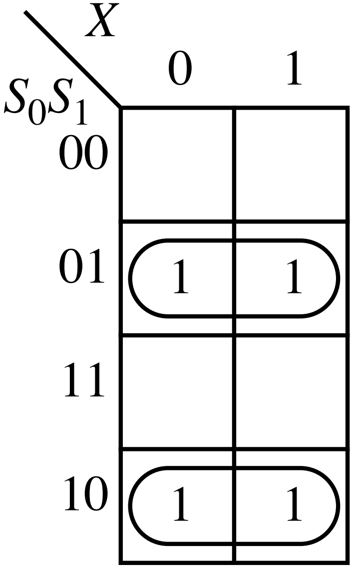
State assignment SA_0

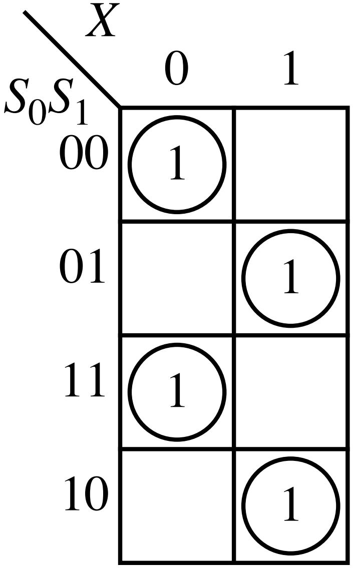
Input S_0S_1	X	
	0	1
A: 00	01/1	00/1
B: 01	11/0	10/1
C: 11	11/0	10/0
D: 10	01/1	00/0

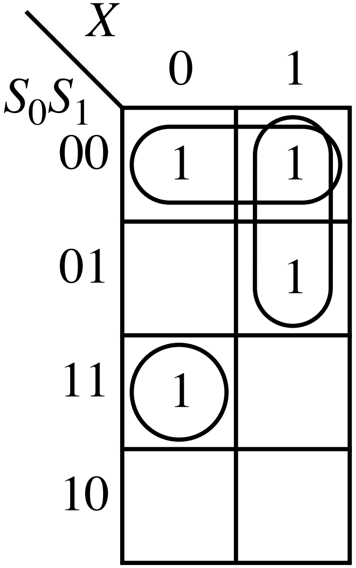
State assignment SA_1

State Assignment SA$_0$

- **Boolean equations for machine *M*$_2$ using state assignment SA$_0$.**







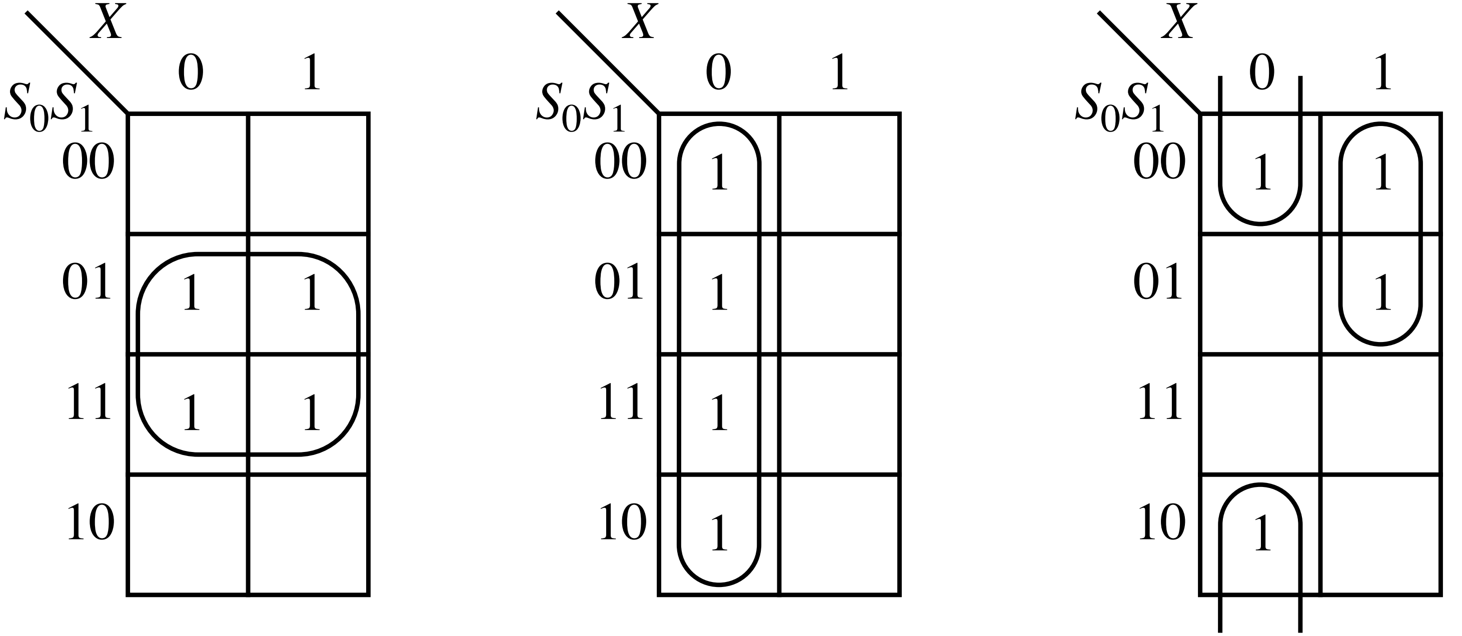
$$S_0 = \overline{S_0}S_1 + S_0\overline{S_1}$$

$$S_1 = \overline{S_0}\,\overline{S_1}\,\overline{X} + \overline{S_0}S_1X + S_0S_1\overline{X} + S_0\overline{S_1}X$$

$$Z = \overline{S_0}\,\overline{S_1} + \overline{S_0}X + S_0S_1\overline{X}$$

State Assignment SA_1

- **Boolean equations for machine M_2 using state assignment SA_1.**



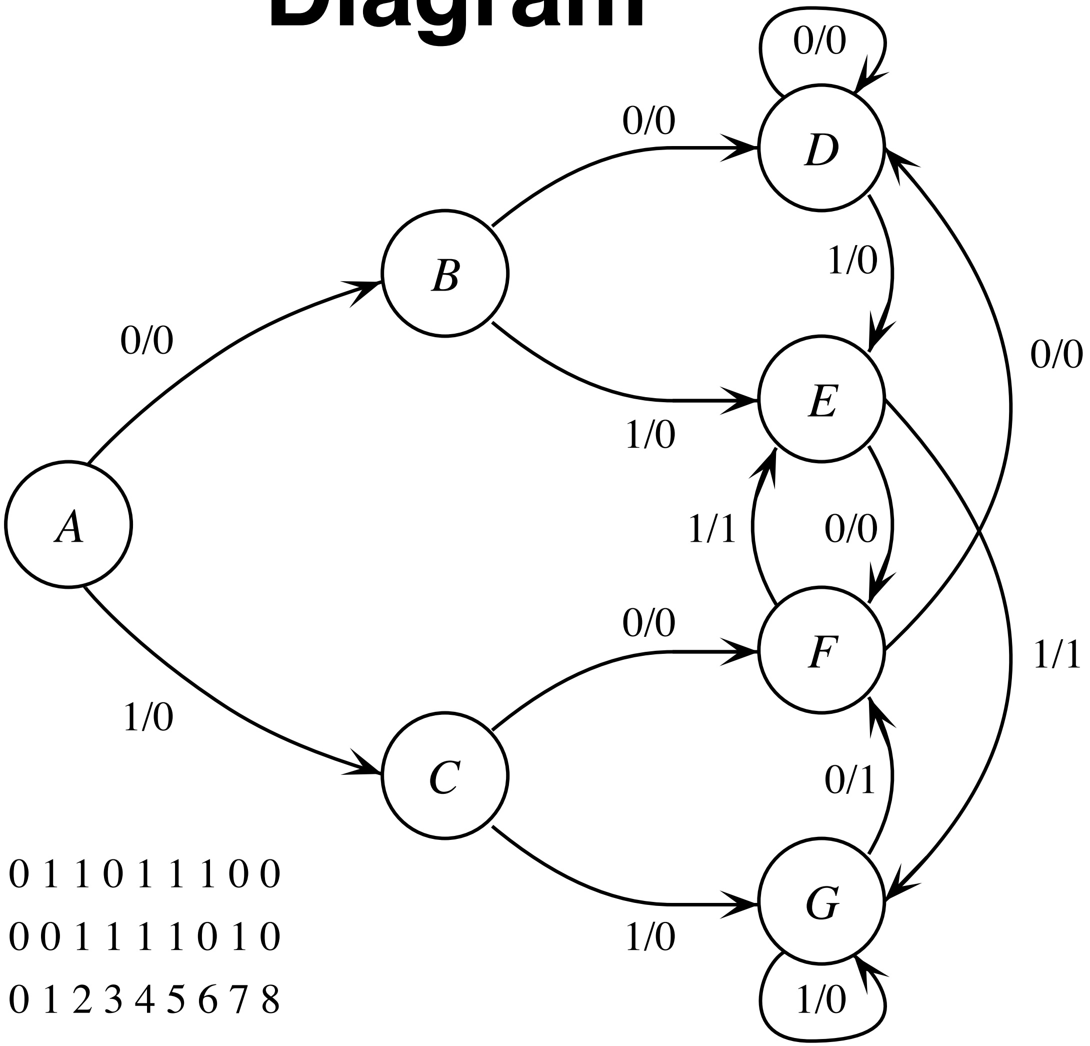
$S_0 = S_1$ $\quad$ $S_1 = \bar{X}$ $\quad$ $Z = \overline{S_1}\bar{X} + \overline{S_0}X$

State Assignment Heuristics

- No known efficient alg. for best state assignment
- Some heuristics (rules of thumb):
  - ◇ The initial state should be simple to reset — all zeroes or all ones.
  - ◇ Minimize the number of state variables that change on each transition.
  - ◇ Maximize the number of state variables that don't change on each transition.
  - ◇ Exploit symmetries in the state diagram.
  - ◇ If there are unused states (when the number of states s is not a power of 2), choose the unused state variable combinations carefully. (Don't just use the first s combination of state variables.)
  - ◇ Decompose the set of state variables into bits or fields that have well-defined meaning with respect to the input or output behavior.
  - ◇ Consider using more than the minimum number of states to achieve the objectives above.

APPLY
STATE REDUCTION
& STATE ASSIGNMENT
TO SEQUENCE DETECTOR

Sequence Detector State Transition Diagram



Input: 0 1 1 0 1 1 1 0 0

Output: 0 0 1 1 1 1 0 1 0

Time: 0 1 2 3 4 5 6 7 8

Sequence Detector State Table

Input Present state	X 0	 1
A	$B/0$	$C/0$
B	$D/0$	$E/0$
C	$F/0$	$G/0$
D	$D/0$	$E/0$
E	$F/0$	$G/1$
F	$D/0$	$E/1$
G	$F/1$	$G/0$

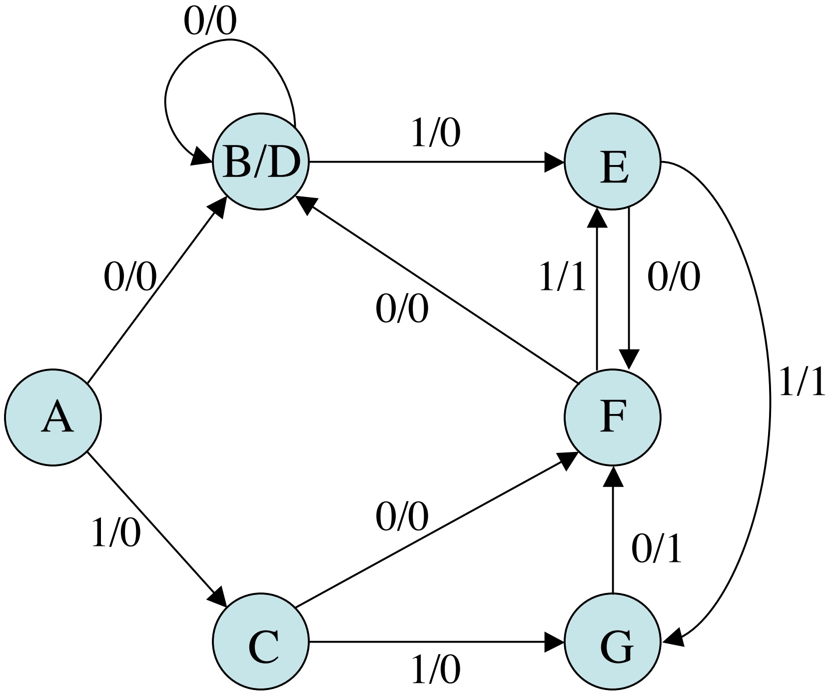
Sequence Detector State Reduction Table

	A	B	C	D	E	F	G
A	☒	x	x	x	x	x	x
B	☒	☒	x		x	x	x
C	☒	☒	☒	x	x	x	x
D	☒	☒	☒	☒	x	x	x
E	☒	☒	☒	☒	☒	x	x
F	☒	☒	☒	☒	☒	☒	x
G	☒	☒	☒	☒	☒	☒	☒

Sequence Detector Reduced State Table

Input Present state	X	
	0	1
$A: A'$	$B'/0$	$C'/0$
$BD: B'$	$B'/0$	$D'/0$
$C: C'$	$E'/0$	$F'/0$
$E: D'$	$E'/0$	$F'/1$
$F: E'$	$B'/0$	$D'/1$
$G: F'$	$E'/1$	$F'/0$

6-State Sequence Detector

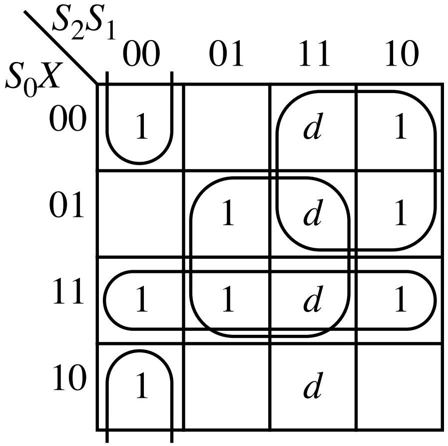


Sequence Detector State Assignment

Input Present state	X	
	0	1
$S_2S_1S_0$	$S_2S_1S_0Z$	$S_2S_1S_0Z$
A': 000	001/0	010/0
B': 001	001/0	011/0
C': 010	100/0	101/0
D': 011	100/0	101/1
E': 100	001/0	011/1
F': 101	100/1	101/0

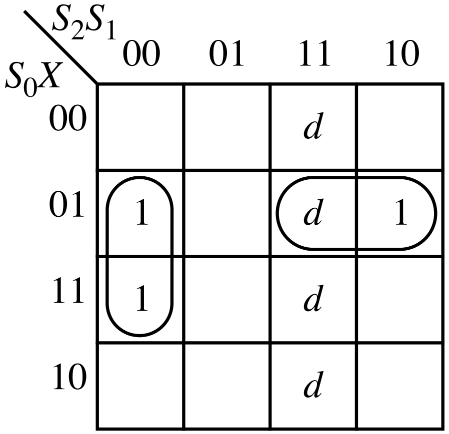
Sequence Detector K-Maps

- **K-map reduction of next state and output functions for sequence detector.**



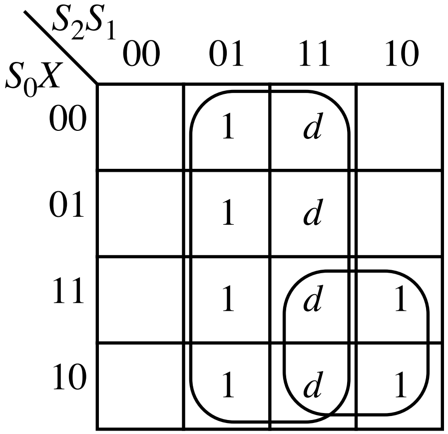
S_0X \ S_2S_1	00	01	11	10
00	1		d	1
01		1	d	1
11	1	1	d	1
10	1		d	

$$S_0 = \overline{S_2}\,\overline{S_1}\bar{X} + S_0X + S_2\overline{S_0} + S_1X$$



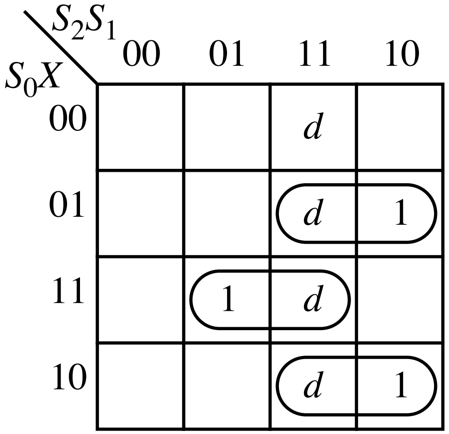
S_0X \ S_2S_1	00	01	11	10
00			d	
01	1		d	1
11	1		d	
10			d	

$$S_1 = \overline{S_2}\,\overline{S_1}X + S_2\overline{S_0}X$$



S_0X \ S_2S_1	00	01	11	10
00		1	d	
01		1	d	
11		1	d	1
10		1	d	1

$$S_2 = S_2S_0 + S_1$$



S_0X \ S_2S_1	00	01	11	10
00			d	
01			d	1
11		1	d	
10			d	1

$$Z = S_2\overline{S_0}X + S_1S_0X + S_2S_0\bar{X}$$

Improved Sequence Detector?

- Formulas from the 7-state FSM:

$$s2' = (\overline{s0} + x)(s2 + s1 + s0)$$

$$s1' = \overline{s0}\ x + s0\ \overline{x} = s0 \text{ xor } x$$

$$s0' = \overline{x}$$

$$z = s2\ \overline{s1}\ x + s2\ s1\ \overline{x}$$

- Formulas from the 6-state FSM:

$$s2' = s2\ s0 + s1$$

$$s1' = \overline{s2}\ \overline{s1}\ x + s2\ \overline{s0}\ x$$

$$s0' = \overline{s2}\ \overline{s1}\ \overline{x} + s0\ x + s2\ \overline{s0} + s1\ x$$

$$z = s2\ \overline{s0}\ x + s1\ s0\ x + s2\ s0\ \overline{x}$$

Sequence Detector State Assignment

7-state

	s2	s1	s0	x	s2'	s1'	s0'	z
0	0	0	0	0	0	0	1	0
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	1	0
3	0	0	1	1	1	0	0	0
4	0	1	0	0	1	0	1	0
5	0	1	0	1	1	1	0	0
6	0	1	1	0	0	1	1	0
7	0	1	1	1	1	0	0	0
8	1	0	0	0	1	0	1	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	0	1	1	0
11	1	0	1	1	1	0	0	1
12	1	1	0	0	1	0	1	1
13	1	1	0	1	1	1	0	0
14	1	1	1	0	d	d	d	d
15	1	1	1	1	d	d	d	d

A = 000
B = 001
C = 010
D = 011
E = 100
F = 101
G = 110

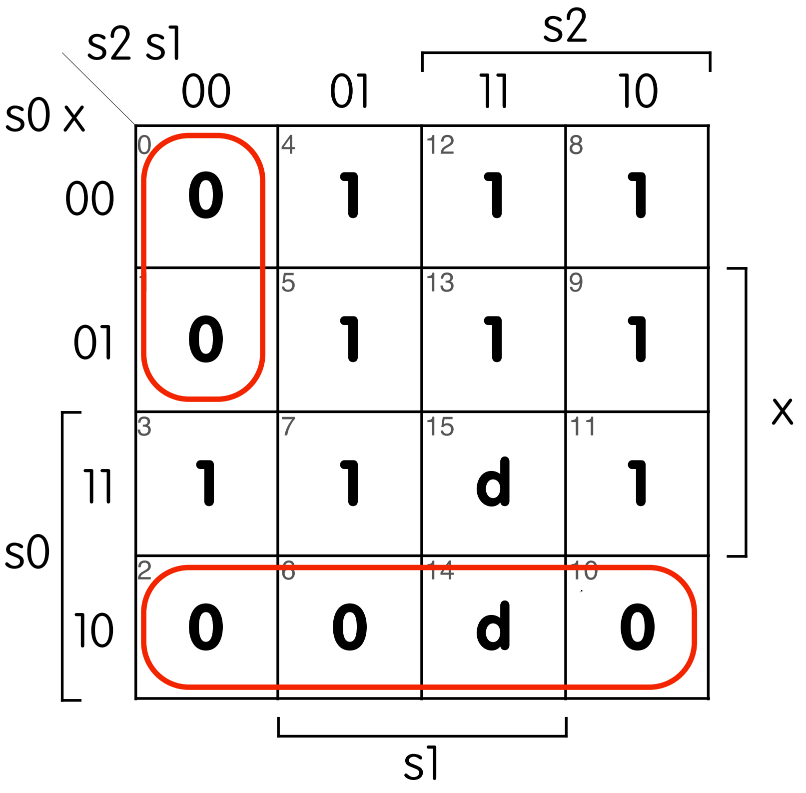
new 6-state

	s2	s1	s0	x	s2'	s1'	s0'	z
0	0	0	0	0	0	0	1	0
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	0	1	0
3	0	0	1	1	1	0	0	0
4	0	1	0	0	1	0	1	0
5	0	1	0	1	1	1	0	0
6	0	1	1	0	d	d	d	d
7	0	1	1	1	d	d	d	d
8	1	0	0	0	1	0	1	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	0	0	1	0
11	1	0	1	1	1	0	0	1
12	1	1	0	0	1	0	1	1
13	1	1	0	1	1	1	0	0
14	1	1	1	0	d	d	d	d
15	1	1	1	1	d	d	d	d

A = 000
B/D = 001
C = 010
~~D = 011~~
E = 100
F = 101
G = 110

6-State Sequence Detector

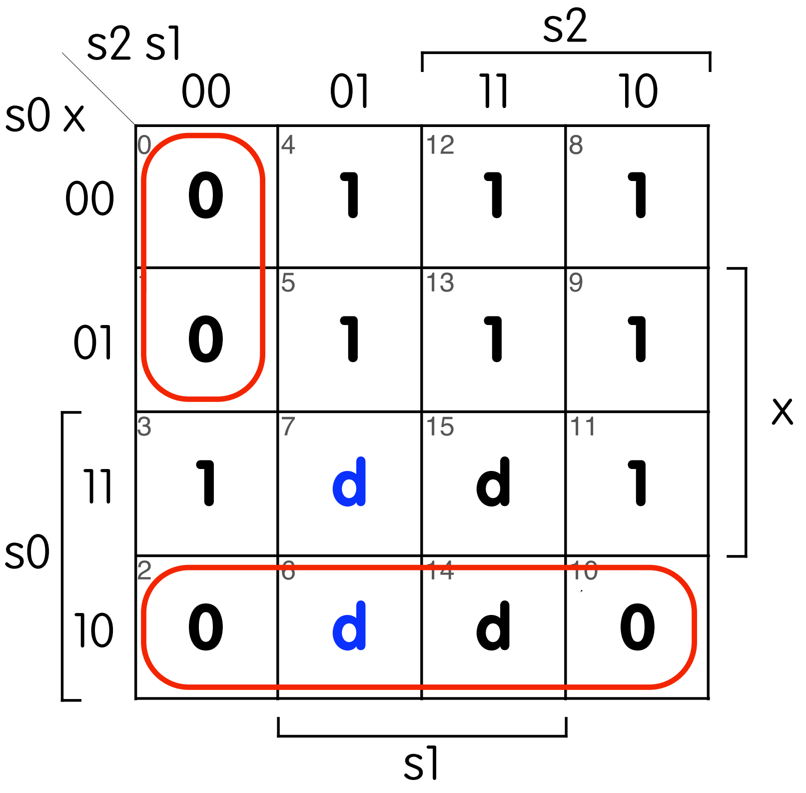
7-state



s0 x \ s2 s1	00	01	11	10
00	0	1	1	1
01	0	1	1	1
11	1	1	d	1
10	0	0	d	0

$$s2' = (\overline{s0} + x)(s2 + s1 + s0)$$

new 6-state

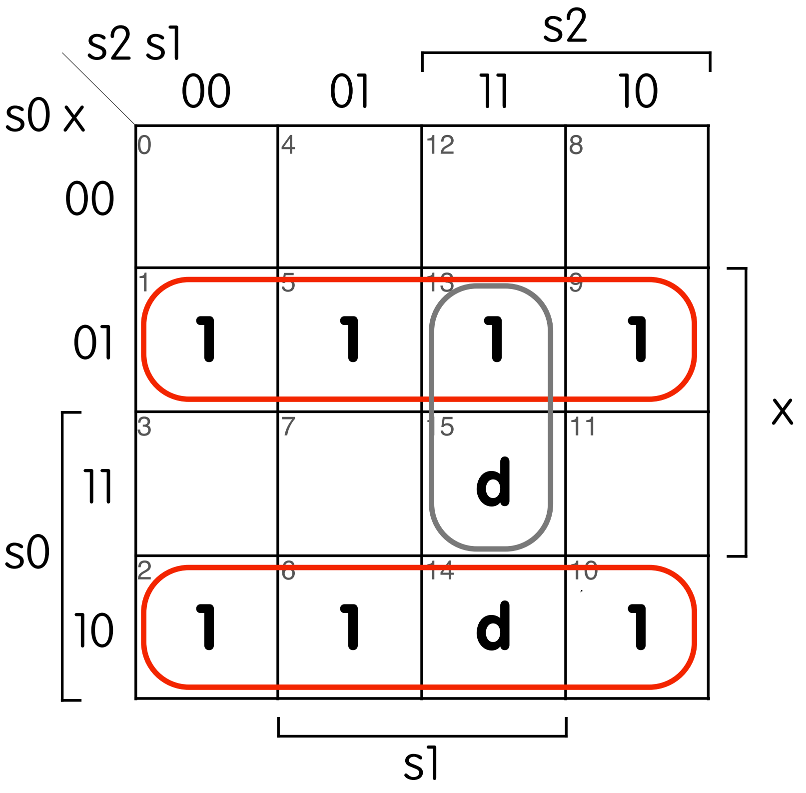


s0 x \ s2 s1	00	01	11	10
00	0	1	1	1
01	0	1	1	1
11	1	d	d	1
10	0	d	d	0

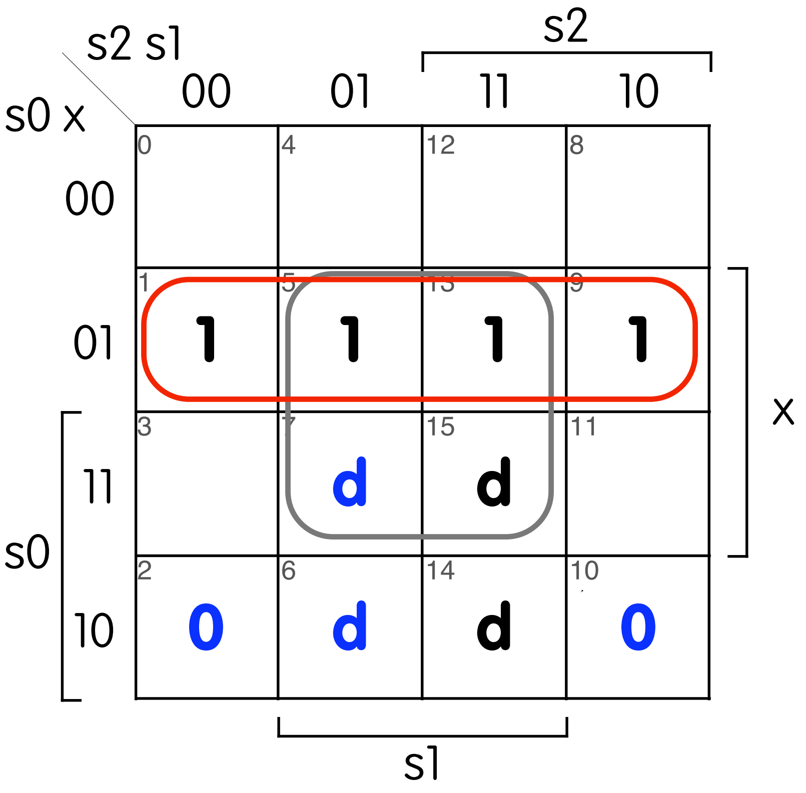
$$s2' = (\overline{s0} + x)(s2 + s1 + s0)$$

6-State Sequence Detector

7-state



new 6-state

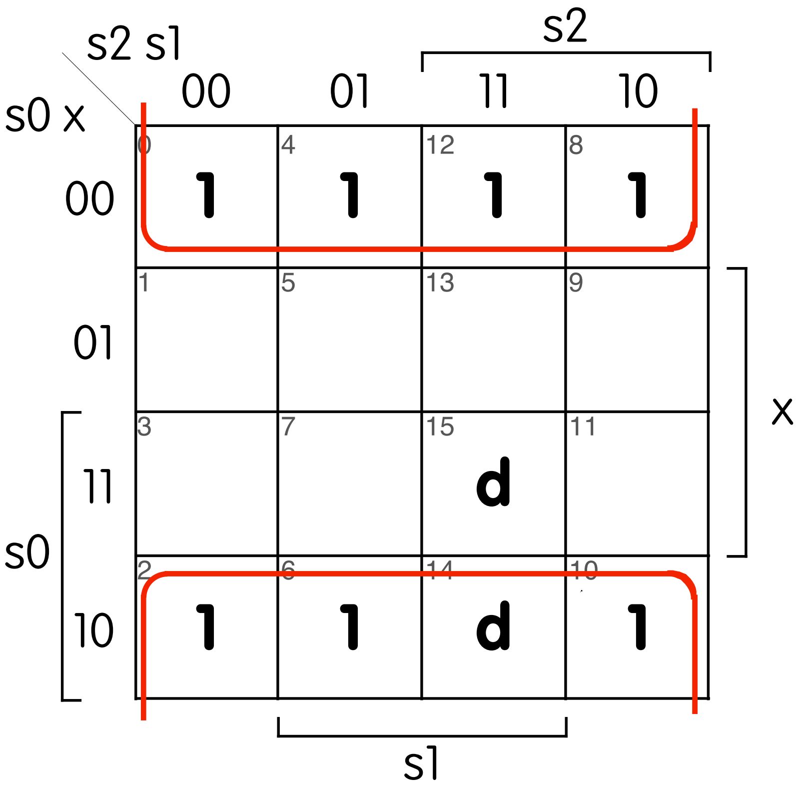


$s1' = \overline{s0}\ x + s0\ \overline{x}$

$s1' = \overline{s0}\ x$

6-State Sequence Detector

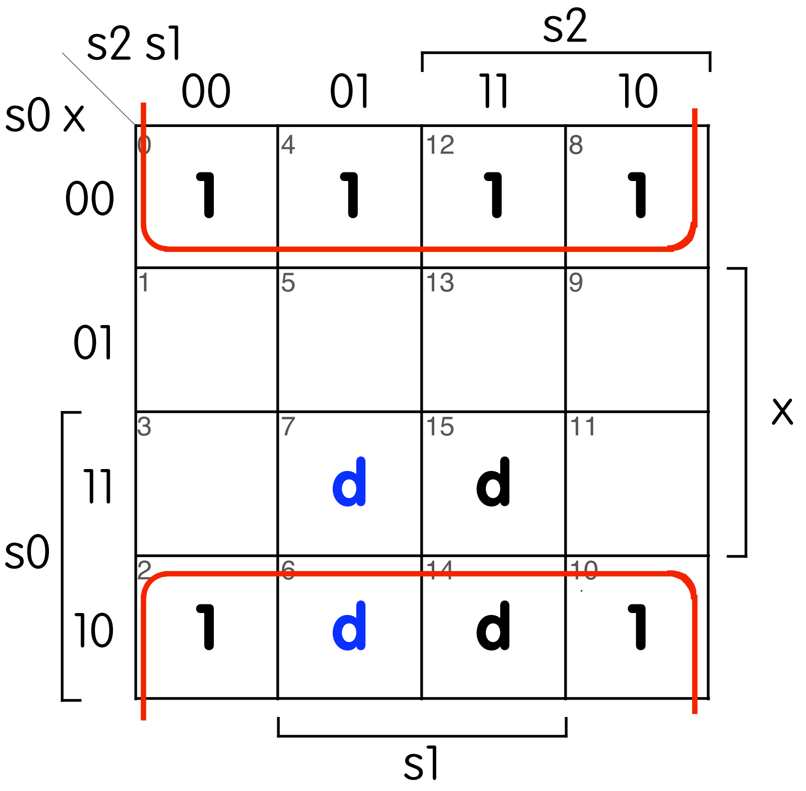
7-state



s0 x \ s2 s1	00	01	11	10
00	1	1	1	1
01				
11			d	
10	1	1	d	1

s0′ = $\overline{x}$

new 6-state

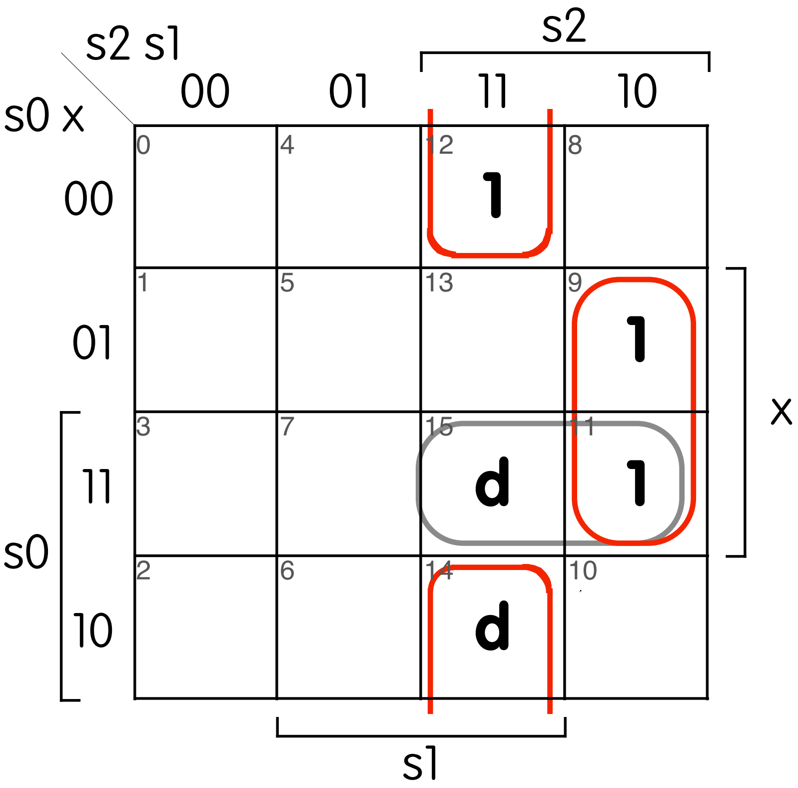


s0 x \ s2 s1	00	01	11	10
00	1	1	1	1
01				
11		d	d	
10	1	d	d	1

s0′ = $\overline{x}$

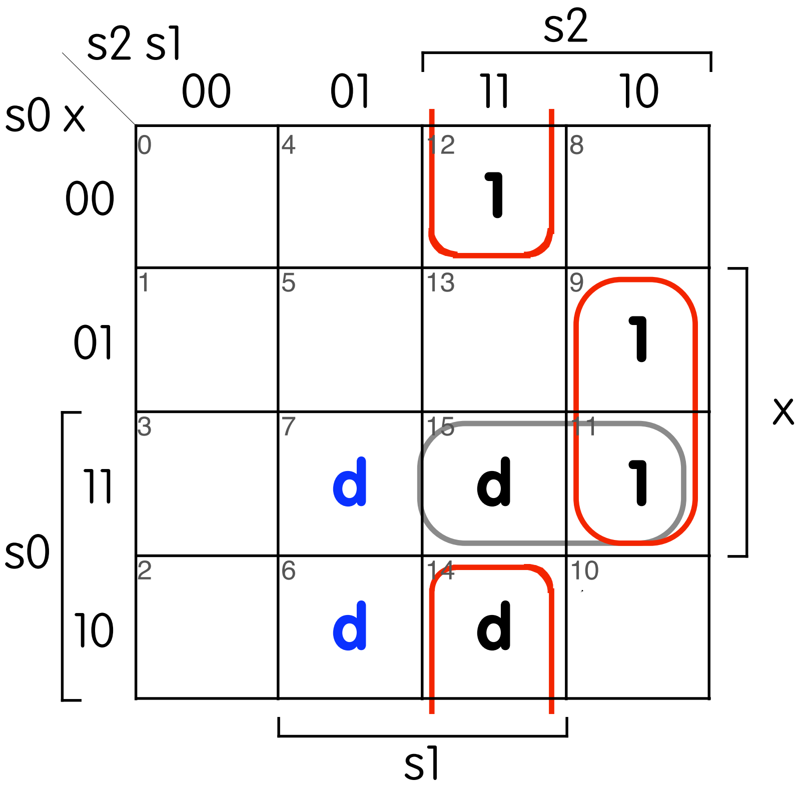
6-State Sequence Detector

7-state



$$z = s2\ \overline{s1}\ x + s2\ s1\ \overline{x}$$

new 6-state



$$z = s2\ \overline{s1}\ x + s2\ s1\ \overline{x}$$

Improved Sequence Detector

- **Textbook formulas for the 6-state FSM:**

$$s2' = s2\ s0 + s1$$

$$s1' = \overline{s2}\ \overline{s1}\ x + s2\ \overline{s0}\ x$$

$$s0' = \overline{s2}\ \overline{s1}\ \overline{x} + s0\ x + s2\ \overline{s0} + s1\ x$$

$$z = s2\ \overline{s0}\ x + s1\ s0\ x + s2\ s0\ \overline{x}$$

- **New formulas for the 6-state FSM:**

$$s2' = (\overline{s0} + x)(s2 + s1 + s0)$$

$$s1' = \overline{s0}\ x$$

$$s0' = \overline{x}$$

$$z = s2\ \overline{s1}\ x + s2\ s1\ \overline{x}$$

CHOICE OF FLIP FLOP (NOT COVERED)

Excitation Tables

- **Each table shows the settings that must be applied at the inputs at time *t* in order to change the outputs at time *t*+1.**

S-R flip-flop

Q_t	Q_{t+1}	S	R
0	0	0	0
0	1	1	0
1	0	0	1
1	1	0	0

D flip-flop

Q_t	Q_{t+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

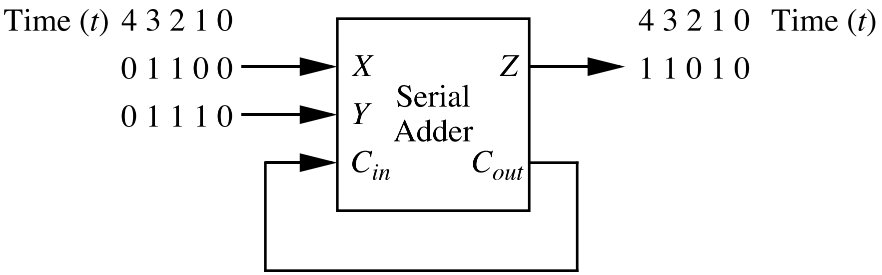
J-K flip-flop

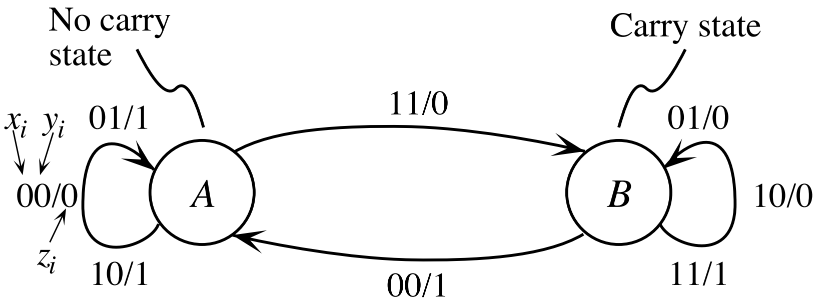
Q_t	Q_{t+1}	J	K
0	0	0	d
0	1	1	d
1	0	d	1
1	1	d	0

T flip-flop

Q_t	Q_{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Serial Adder





- **State transition diagram, state table, and state assignment for a serial adder.**

Input Present state	XY			
	00	01	10	11
A	A/0	A/1	A/1	B/0
B	A/1	B/0	B/0	B/1

Next state Output

Input Present state (S_t)	XY			
	00	01	10	11
A:0	0/0	0/1	0/1	1/0
B:1	0/1	1/0	1/0	1/1

Serial Adder Next-State Functions

- **Truth table showing next-state functions for a serial adder for D, S-R, T, and J-K flip-flops. Shaded functions are used in the example.**

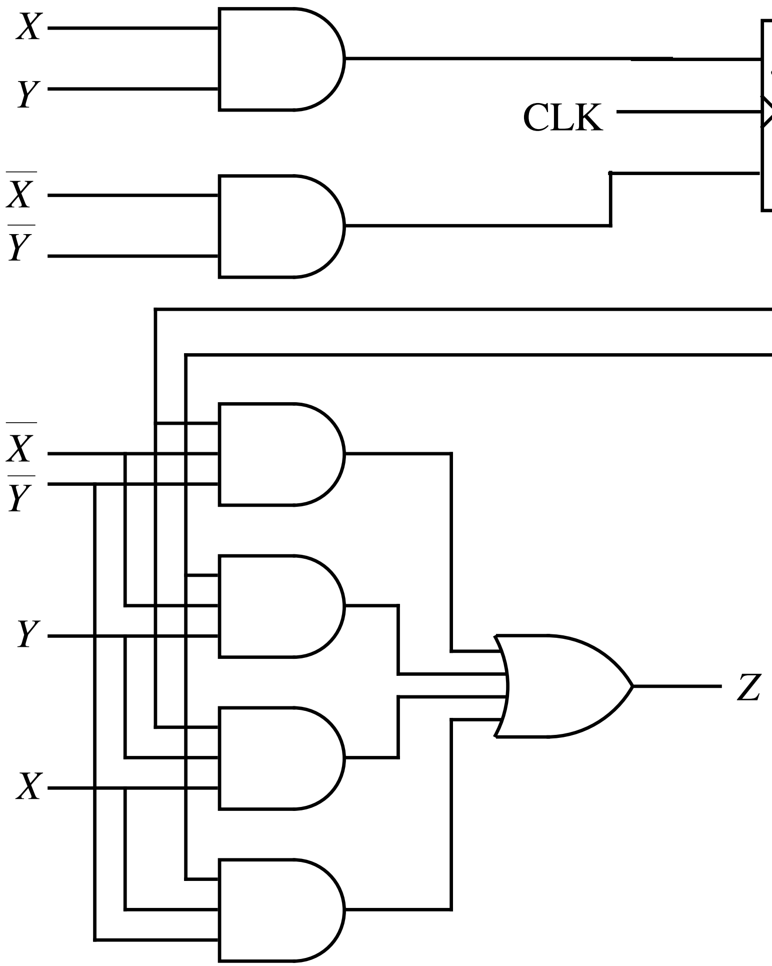
		Present State		(Set)	(Reset)				
X	Y	S_t	D	S	R	T	J	K	Z
0	0	0	0	0	0	0	0	d	0
0	0	1	0	0	1	1	d	1	1
0	1	0	0	0	0	0	0	d	1
0	1	1	1	0	0	0	d	0	0
1	0	0	0	0	0	0	0	d	1
1	0	1	1	0	0	0	d	0	0
1	1	0	1	1	0	1	1	d	0
1	1	1	1	0	0	0	d	0	1

J-K Flip-Flop Serial Adder Circuit

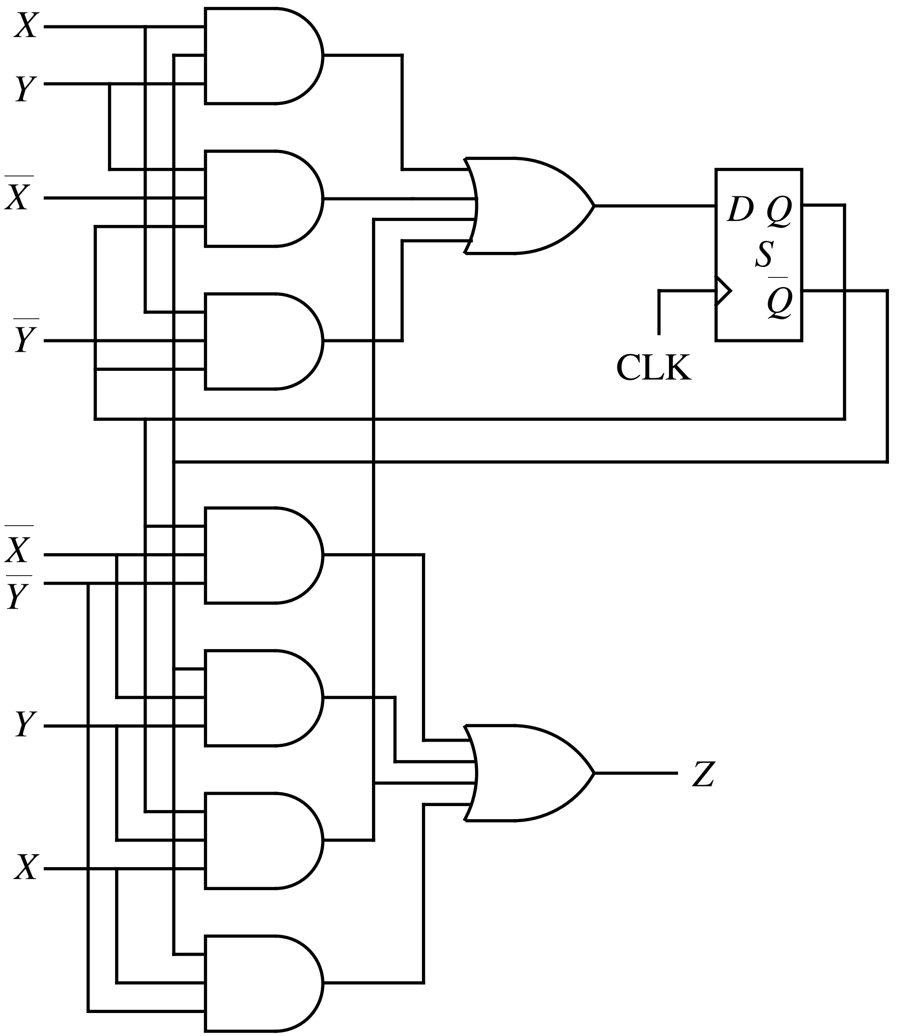
$$J = XY$$

$$K = \overline{X}\,\overline{Y}$$

$$Z = \overline{X}\,\overline{Y}S + \overline{X}Y\overline{S} + XYS + X\overline{Y}\,\overline{S}$$

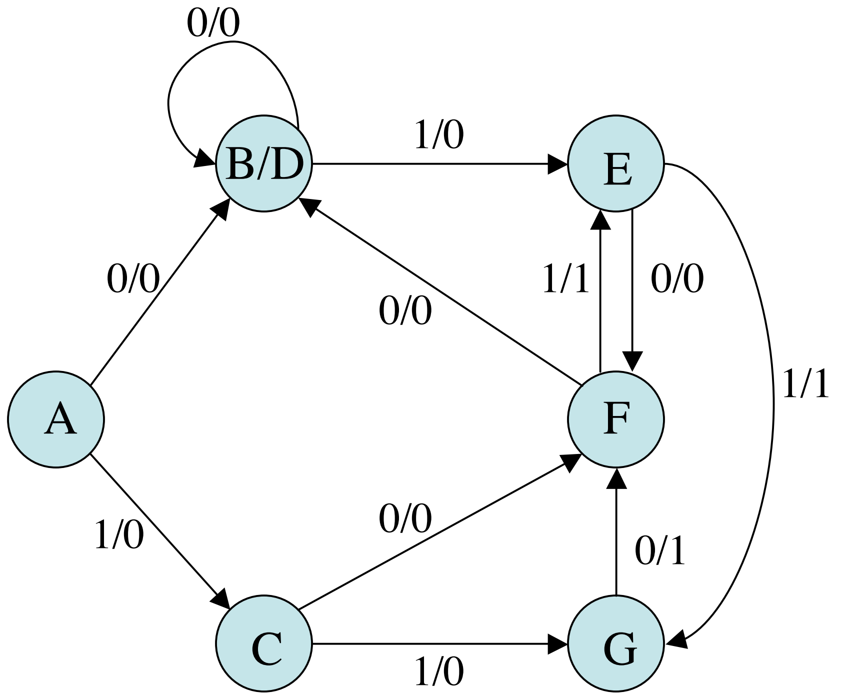


D Flip-Flop Serial Adder Circuit



CONSIDER
FLIP FLOP CHOICE
IN SEQUENCE DETECTOR
(NOT COVERED)

6-State Sequence Detector



Sequence Detector State Assignment

7-state

	s2	s1	s0	x	s2'	s1'	s0'	z
0	0	0	0	0	0	0	1	0
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	1	0
3	0	0	1	1	1	0	0	0
4	0	1	0	0	1	0	1	0
5	0	1	0	1	1	1	0	0
6	0	1	1	0	0	1	1	0
7	0	1	1	1	1	0	0	0
8	1	0	0	0	1	0	1	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	0	1	1	0
11	1	0	1	1	1	0	0	1
12	1	1	0	0	1	0	1	1
13	1	1	0	1	1	1	0	0
14	1	1	1	0	d	d	d	d
15	1	1	1	1	d	d	d	d

A = 000
B = 001
C = 010
D = 011

E = 100
F = 101
G = 110

new 6-state

	s2	s1	s0	x	s2'	s1'	s0'	z
0	0	0	0	0	0	0	1	0
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	0	1	0
3	0	0	1	1	1	0	0	0
4	0	1	0	0	1	0	1	0
5	0	1	0	1	1	1	0	0
6	0	1	1	0	d	d	d	d
7	0	1	1	1	d	d	d	d
8	1	0	0	0	1	0	1	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	0	0	1	0
11	1	0	1	1	1	0	0	1
12	1	1	0	0	1	0	1	1
13	1	1	0	1	1	1	0	0
14	1	1	1	0	d	d	d	d
15	1	1	1	1	d	d	d	d

A = 000
B/D = 001
C = 010
~~D = 011~~

E = 100
F = 101
G = 110

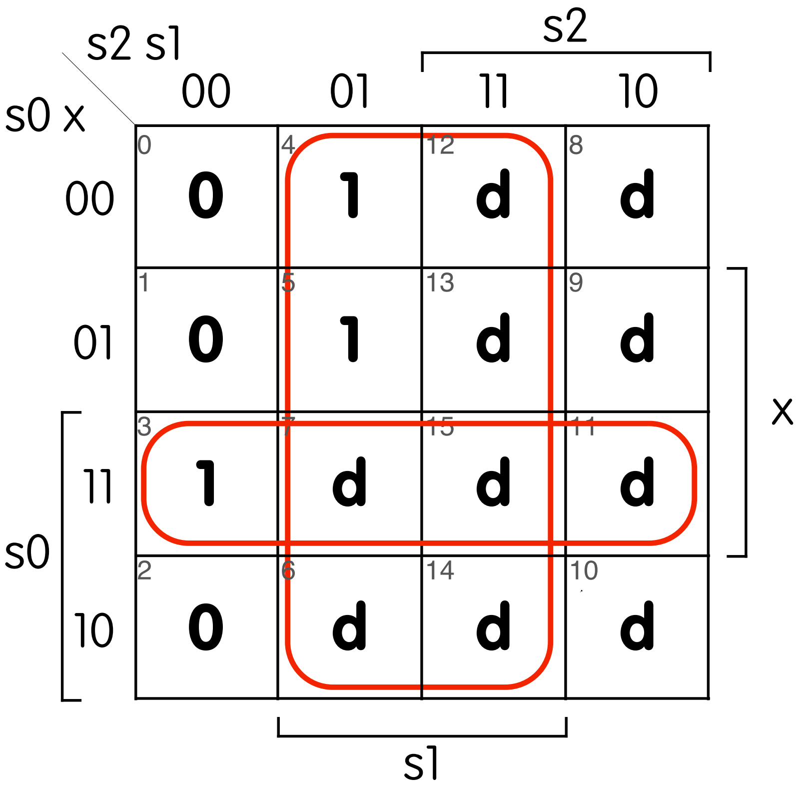
6-State Sequence Detector

	s2	s1	s0	x	s2'	s1'	s0'	z	j2	k2	j1	k1	j0	k0
0	0	0	0	0	0	0	1	0	0	*d*	0	*d*	1	*d*
1	0	0	0	1	0	1	0	0	0	*d*	1	*d*	0	*d*
2	0	0	1	0	0	0	1	0	0	*d*	0	*d*	*d*	0
3	0	0	1	1	1	0	0	0	1	*d*	0	*d*	*d*	1
4	0	1	0	0	1	0	1	0	1	*d*	*d*	1	1	*d*
5	0	1	0	1	1	1	0	0	1	*d*	*d*	0	0	*d*
6	0	1	1	0	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*
7	0	1	1	1	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*
8	1	0	0	0	1	0	1	0	*d*	0	0	*d*	1	*d*
9	1	0	0	1	1	1	0	1	*d*	0	1	*d*	0	*d*
10	1	0	1	0	0	0	1	0	*d*	1	0	*d*	*d*	0
11	1	0	1	1	1	0	0	1	*d*	0	0	*d*	*d*	1
12	1	1	0	0	1	0	1	1	*d*	0	*d*	1	1	*d*
13	1	1	0	1	1	1	0	0	*d*	0	*d*	0	0	*d*
14	1	1	1	0	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*
15	1	1	1	1	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*

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

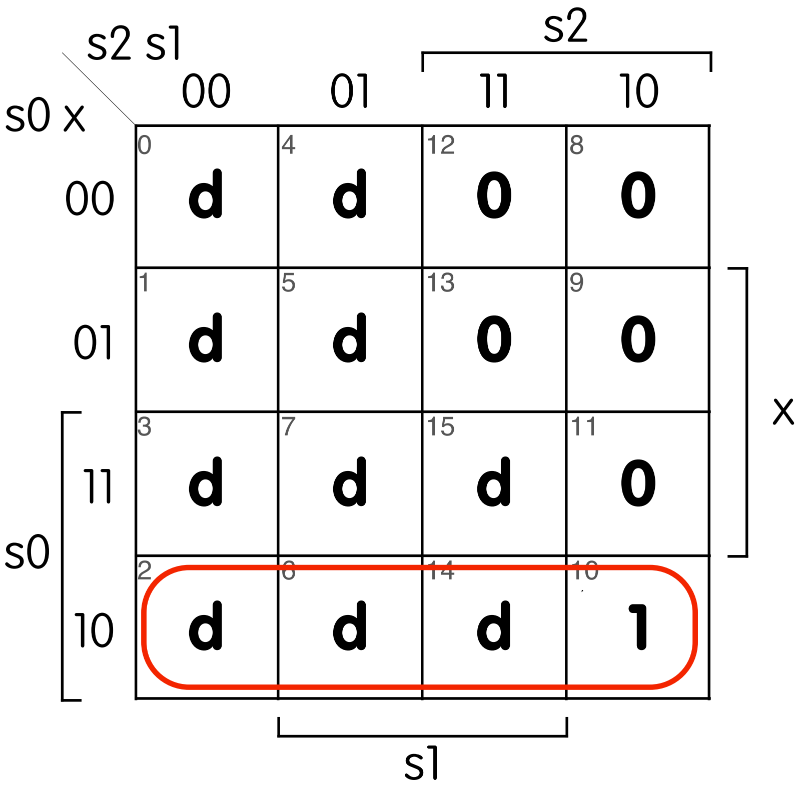
6-State Sequence Detector

J2



s0 x \ s2 s1	00	01	11	10
00	0	1	d	d
01	0	1	d	d
11	1	d	d	d
10	0	d	d	d

K2



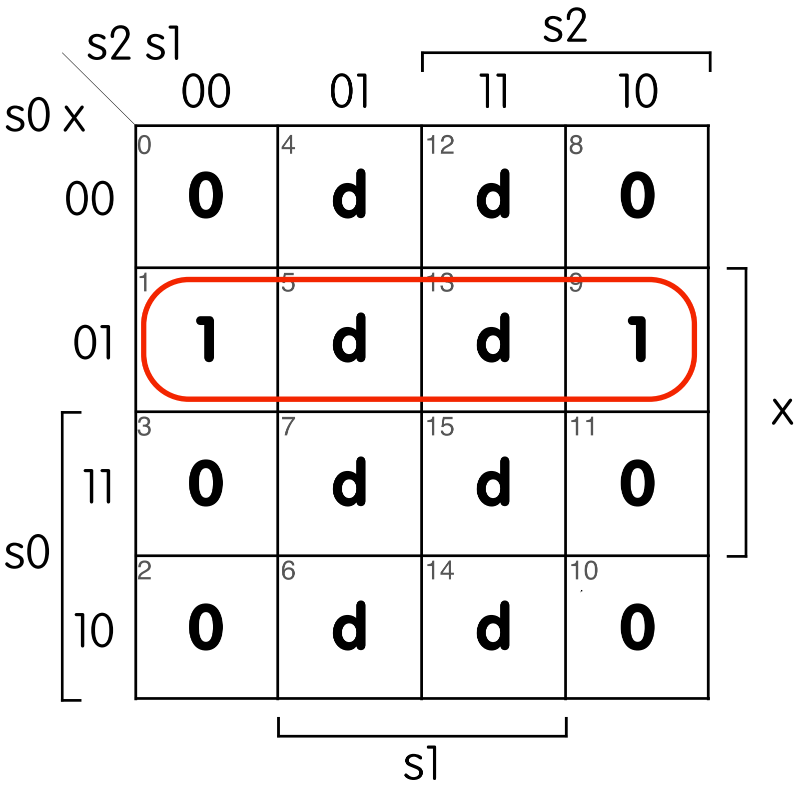
s0 x \ s2 s1	00	01	11	10
00	d	d	0	0
01	d	d	0	0
11	d	d	d	0
10	d	d	d	1

$J2 = s1 + s0\ x$

$K2 = s0\ \overline{x}$

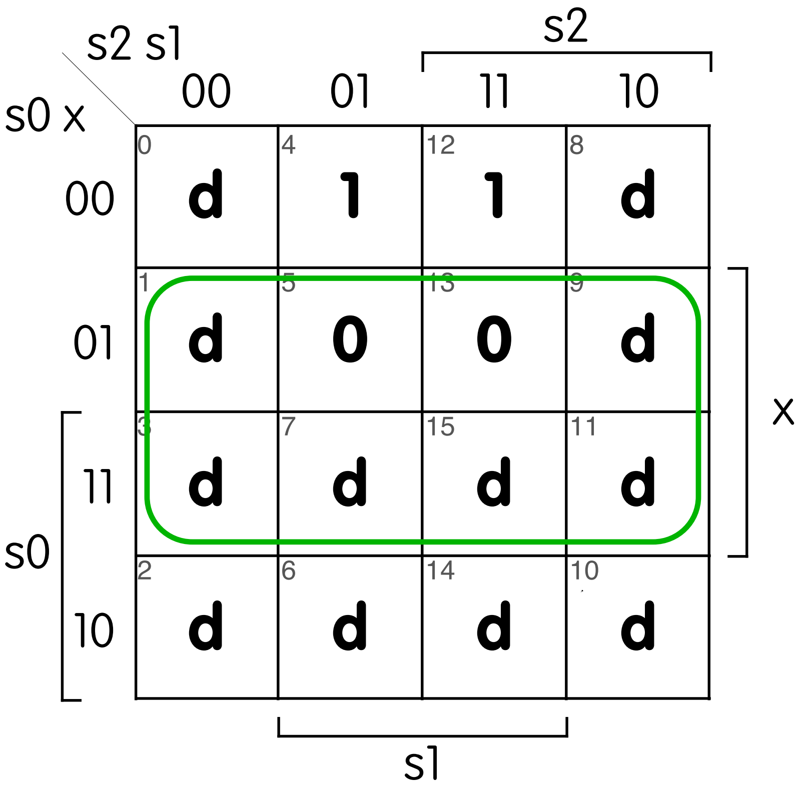
6-State Sequence Detector

J1



s0 x \ s2 s1	00	01	11	10
00	0	d	d	0
01	1	d	d	1
11	0	d	d	0
10	0	d	d	0

K1



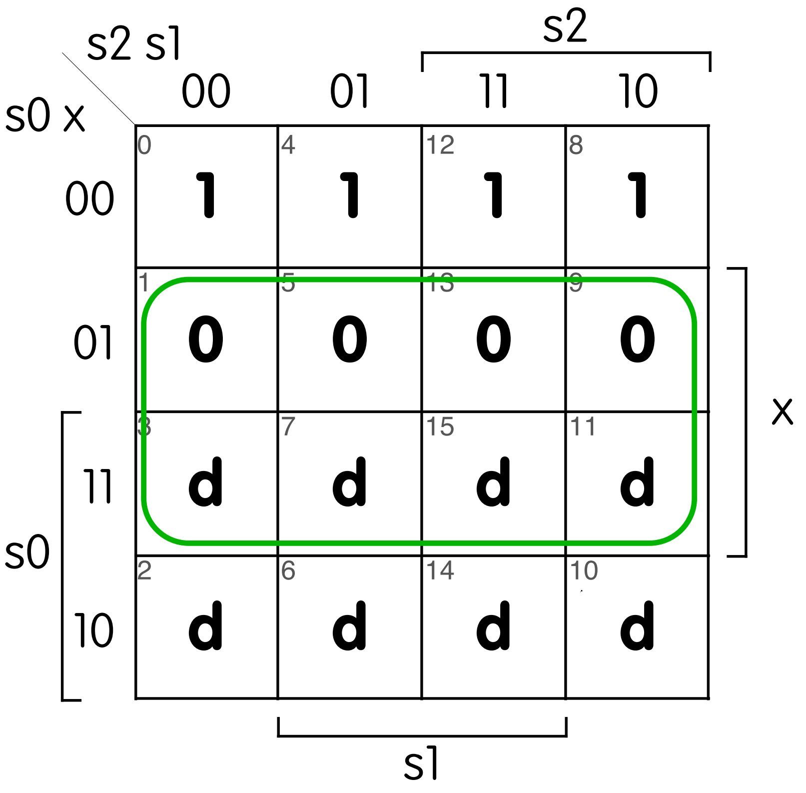
s0 x \ s2 s1	00	01	11	10
00	d	1	1	d
01	d	0	0	d
11	d	d	d	d
10	d	d	d	d

$J1 = \overline{s0}\ x$

$K1 = \overline{x}$

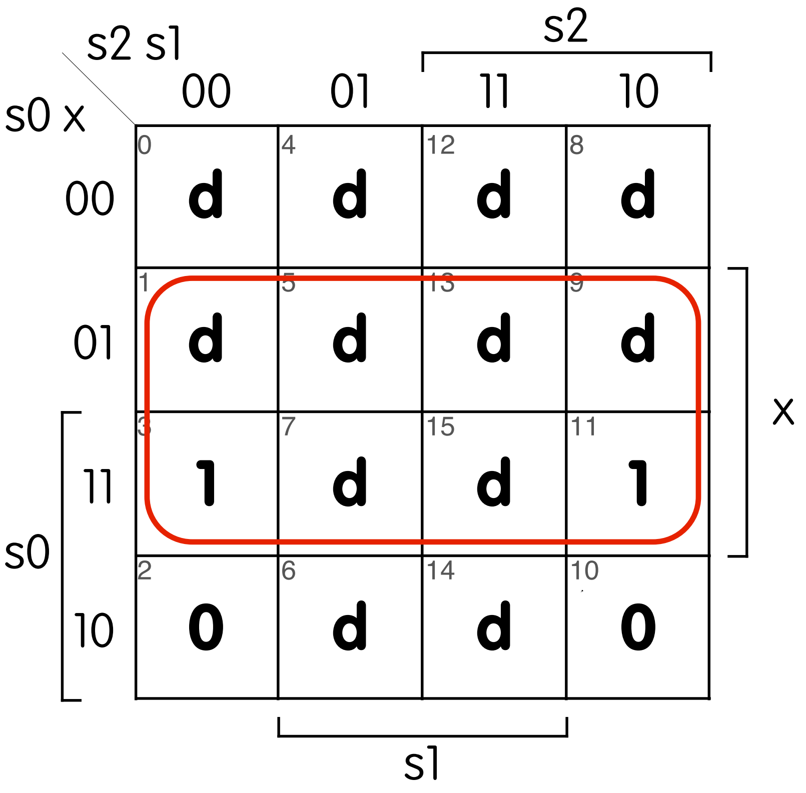
6-State Sequence Detector

J0



s0 x \ s2 s1	00	01	11	10
00	1	1	1	1
01	0	0	0	0
11	d	d	d	d
10	d	d	d	d

K0



s0 x \ s2 s1	00	01	11	10
00	d	d	d	d
01	d	d	d	d
11	1	d	d	1
10	0	d	d	0

$J0 = \overline{x}$

$K0 = x$

Improved Sequence Detector

- **Formulas for the 6-state FSM with D Flip-flops:**

$s2' = (\overline{s0} + x)(s2 + s1 + s0)$

$s1' = \overline{s0}\ x$

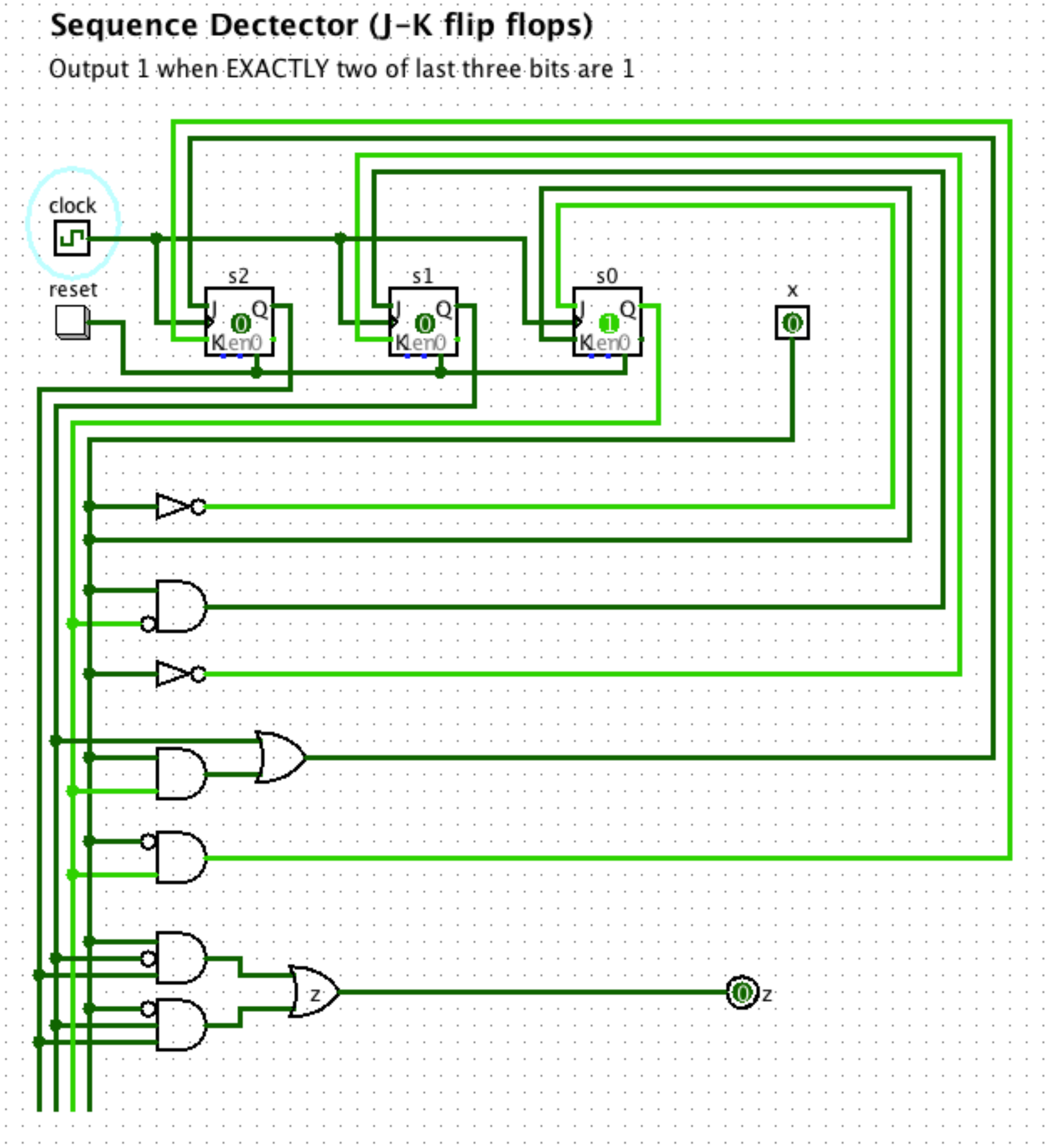
$s0' = \overline{x}$

- **Formulas for the 6-state FSM with J-K Flip-flops:**

$J2 = s1 + s0\ x$ $K2 = s0\ \overline{x}$

$J1 = \overline{s0}\ x$ $K1 = \overline{x}$

$J0 = \overline{x}$ $K0 = x$



Sequence Dectector (J-K flip flops)

Output 1 when EXACTLY two of last three bits are 1

NEXT TIME

- A 2-bit CPU