CMSC 313 Lecture 25

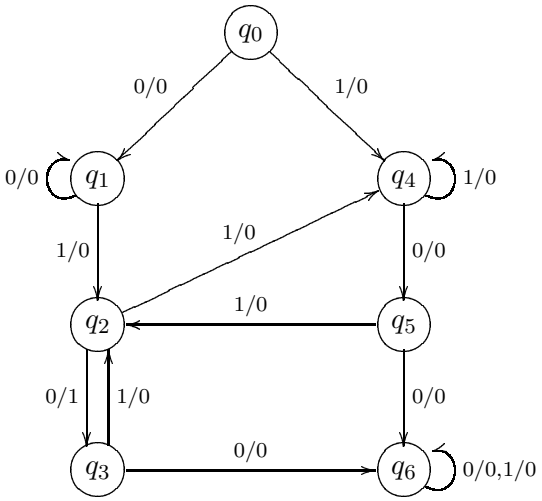
- DigSim Exercise 1 due
- State Reduction Algorithm
- State Assignment Heuristics
- Using J-K Flip-Flops (?)
- SCEQs

DigSim Assignment 1: A Finite State Machine

Due: November 25, 2003

Objective: The objective of this assignment is to implement a finite state machine using DigSim.

Assignment: Consider the finite state machine represented below as a transition diagram[1]:



This finite state machine starts in state q_0 and has one input bit and one output bit. The output bit is 1 if the input sequence up to the current point ends with 010 as long as the sequence 100 has never been seen. For example, the machine outputs 1 after reading 00011010, but outputs 0 after reading 110110010. (Verify for yourself that the transition diagram fits the description.)

Your assignment is to implement this finite state machine in DigSim. You must:

1. Use three D flip-flops to store the 7 states of the machine. State q_0 will be represented as 000, $q_1 = 001$, $q_2 = 010$, ..., $q_6 = 110$. The bit pattern 111 is not used.
2. Let s_2, s_1, s_0 be the state bits stored in the D flip-flops, x be the input bit and z be the output bit. Fill in the attached truth table for the next state bits s_2', s_1', s_0' and the output bit z.
3. For s_2', s_1', s_0' and z, use Karnaugh maps to obtain simplified SOP or POS Boolean formulas.
4. Implement the finite state machine in DigSim. You should study the "Sequence Detector" example in DigSim (use "Open example" under the File menu) for suggestions on the layout of your finite state machine.

Implementation notes:

- Label the switches and flip-flops in your circuit appropriately.
- If you need a 4-input OR gate, you need to use two layers of 2-input or 3-input OR gates to accomplish the same function. Ditto for 4-input AND gates.

[1] Adapted from *Contemporary Logic Design,* Randy H. Katz, Benjamin/Cummings Publishing, 1994.

- Make sure to leave time to debug your circuit. Note that to restart the finite state machine in the 000 state, you need to save the circuit and load it back into DigSim.
- The D flip-flops in DigSim are positive-edge triggered. To change the state of the flip-flop, change the input when the clock is low, then bring the clock from low to high. The input to the D flip-flop when the clock changes from low to high will be stored in the flip-flop.

What to submit:

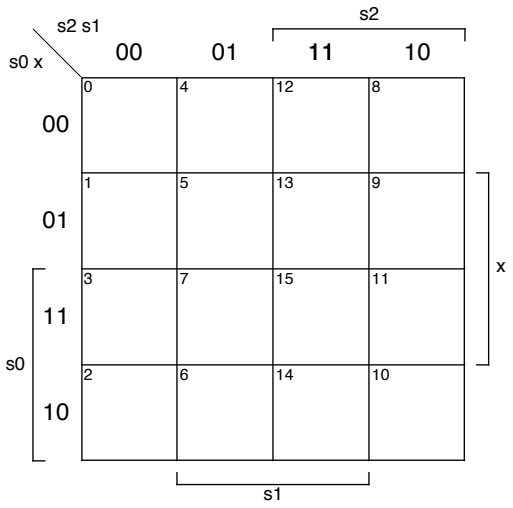
1. Make a copy of your truth-table and Karnaugh maps and submit the hard copy in class on Tuesday November 25.
2. Save your circuit as you did in the DigSim part of Homework 4. Submit the circuit file using the Unix `submit` command as in previous assignments. The submission name for this assignment is: `digsim1`. The UNIX command to do this should look something like:

```
submit cs313_0101 digsim1 fsm.sim
```

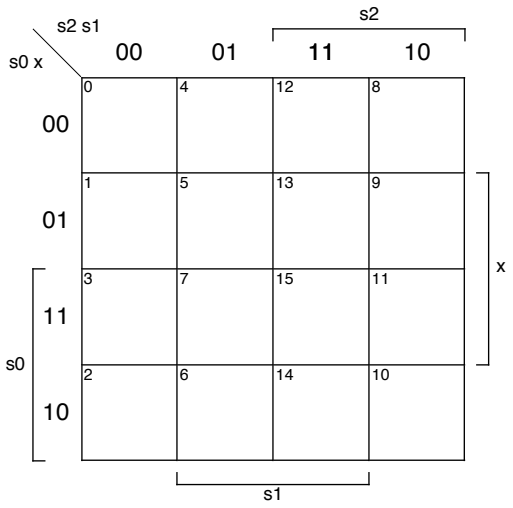
Name: _____

Truth table:

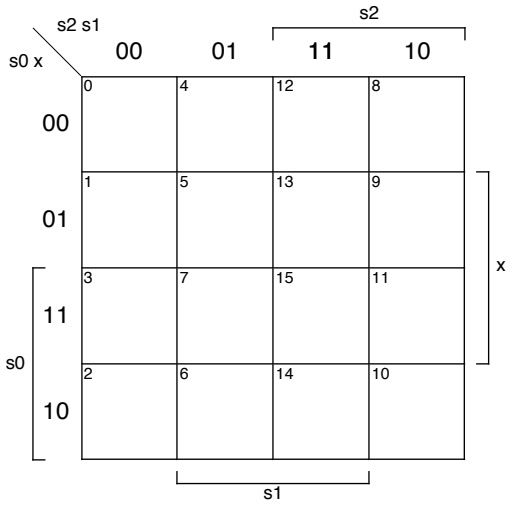
m	s_2	s_1	s_0	x	s_2'	s_1'	s_0'	z
0	0	0	0	0				
1	0	0	0	1				
2	0	0	1	0				
3	0	0	1	1				
4	0	1	0	0				
5	0	1	0	1				
6	0	1	1	0				
7	0	1	1	1				
8	1	0	0	0				
9	1	0	0	1				
10	1	0	1	0				
11	1	0	1	1				
12	1	1	0	0				
13	1	1	0	1				
14	1	1	1	0	d	d	d	d
15	1	1	1	1	d	d	d	d


s2 s1
s0 x
00
01
11
10
s2
s0
s1
x
0 4 12 8
1 5 13 9
3 7 15 11
2 6 14 10

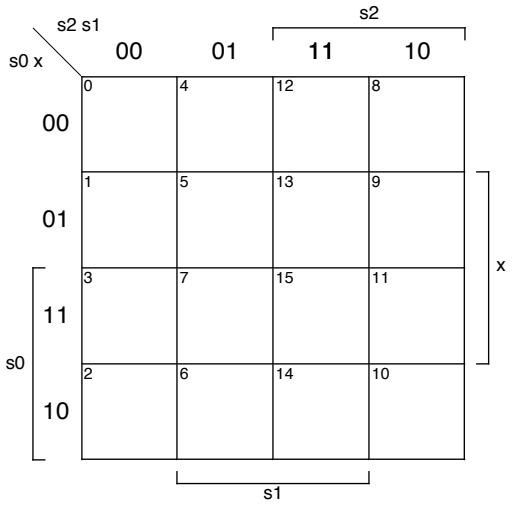
s2' =


s2 s1
s0 x
00
01
11
10
s2
s0
s1
x
0 4 12 8
1 5 13 9
3 7 15 11
2 6 14 10

s1' =


s2 s1
s0 x
00
01
11
10
s2
s0
s1
x
0 4 12 8
1 5 13 9
3 7 15 11
2 6 14 10

s0' =


s2 s1
s0 x
00
01
11
10
s2
s0
s1
x
0 4 12 8
1 5 13 9
3 7 15 11
2 6 14 10

z =

Last Time

- Master-slave J-K flip-flops with two-phase clock
- Mealy vs Moore finite state machines
- Vending machine example
- Sequence detector example

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

- **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 Algorithm

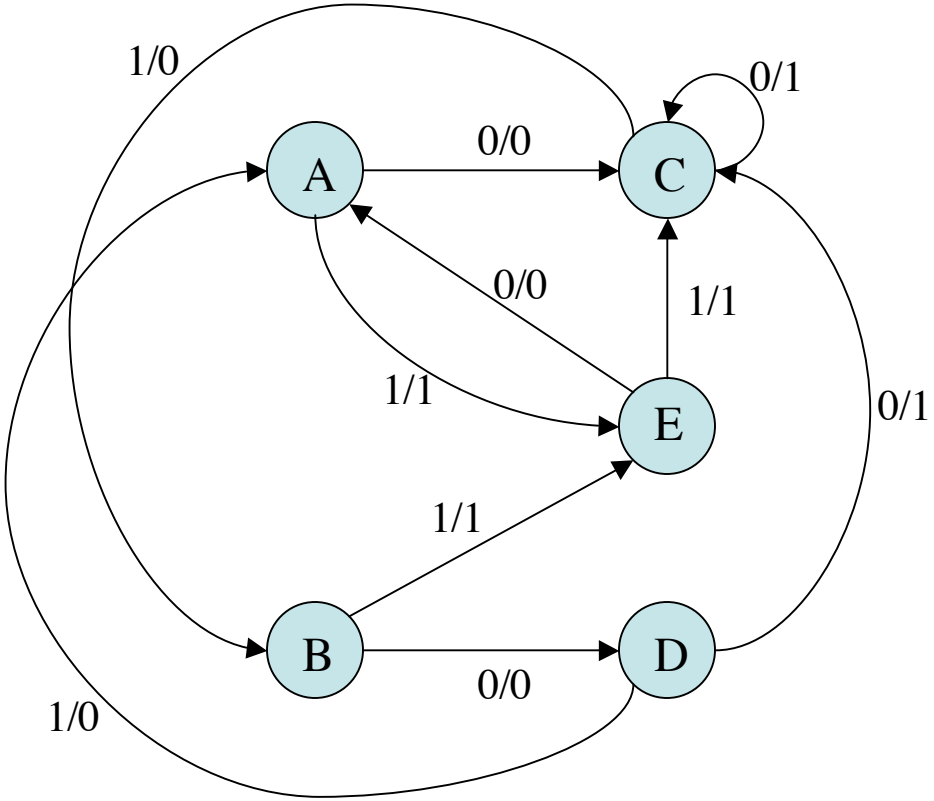
1. Use a 2-dimensional table — an entry for each pair of states.
2. Two states are "distinguished" if:

   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 Example: original transition diagram

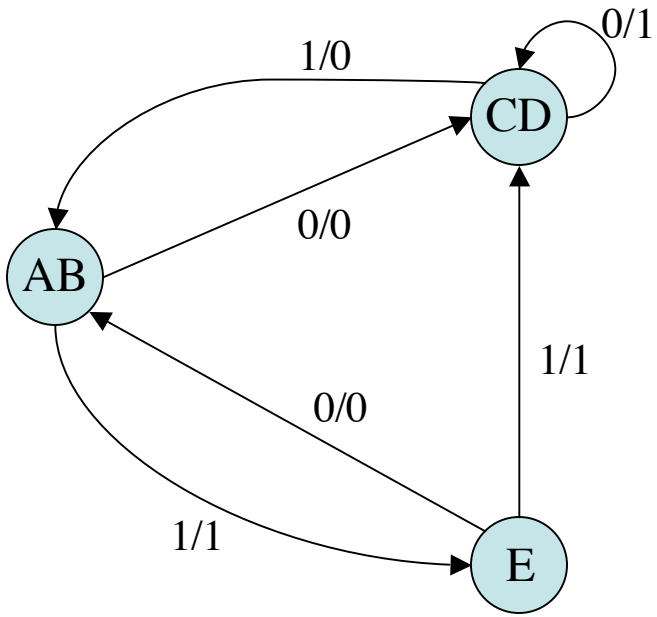


State Reduction Table

- 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.

The State Assignment Problem

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

Input / P.S.	X = 0	X = 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	X = 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

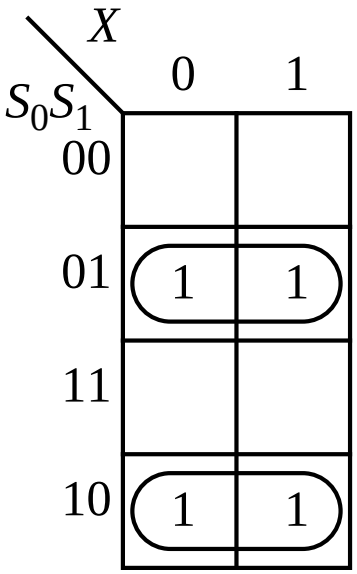
State assignment SA_0

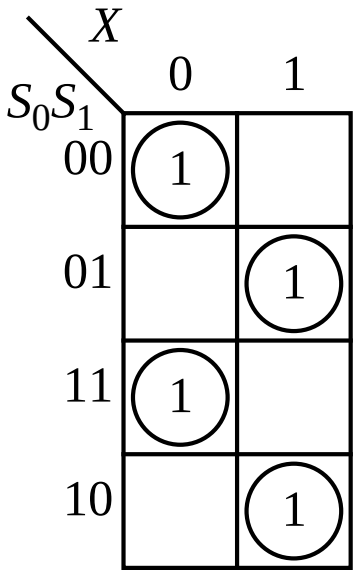
Input / S_0S_1	X = 0	X = 1
A: 00	01/1	00/1
B: 01	11/0	10/1
C: 11	11/0	10/0
D: 10	01/1	00/0

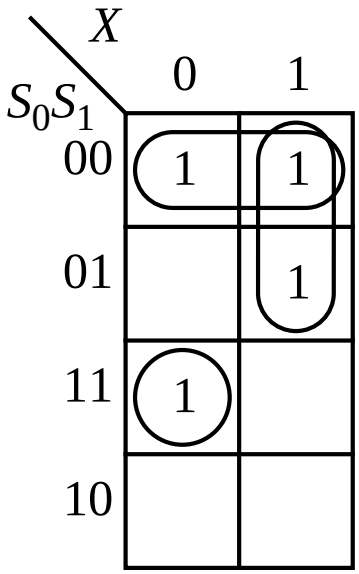
State assignment SA_1

State Assignment SA_0

- **Boolean equations for machine M_2 using state assignment SA_0.**







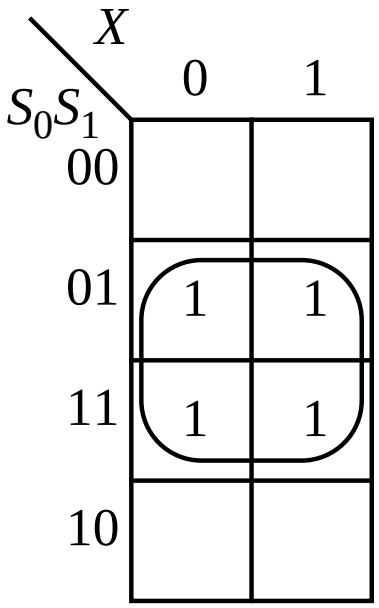
$$S_0 = \overline{S_0}S_1 + S_0\overline{S_1}$$

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

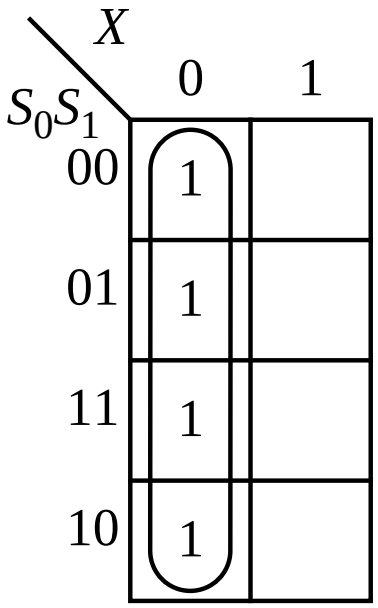
$$Z = \overline{S_0}\,\overline{S_1} + \overline{S_0}X + S_0S_1\bar{X}$$

State Assignment SA_1

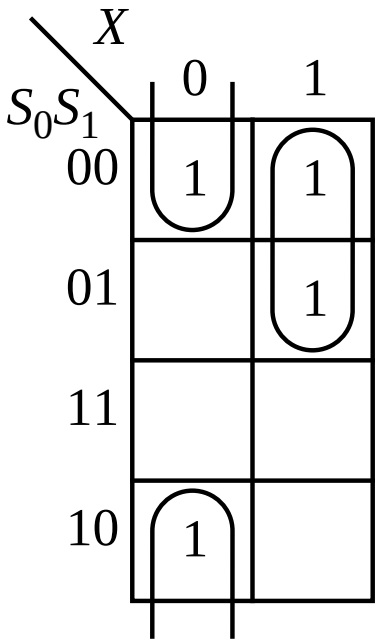
- **Boolean equations for machine M_2 using state assignment SA_1.**



$$S_0 = S_1$$



$$S_1 = \bar{X}$$

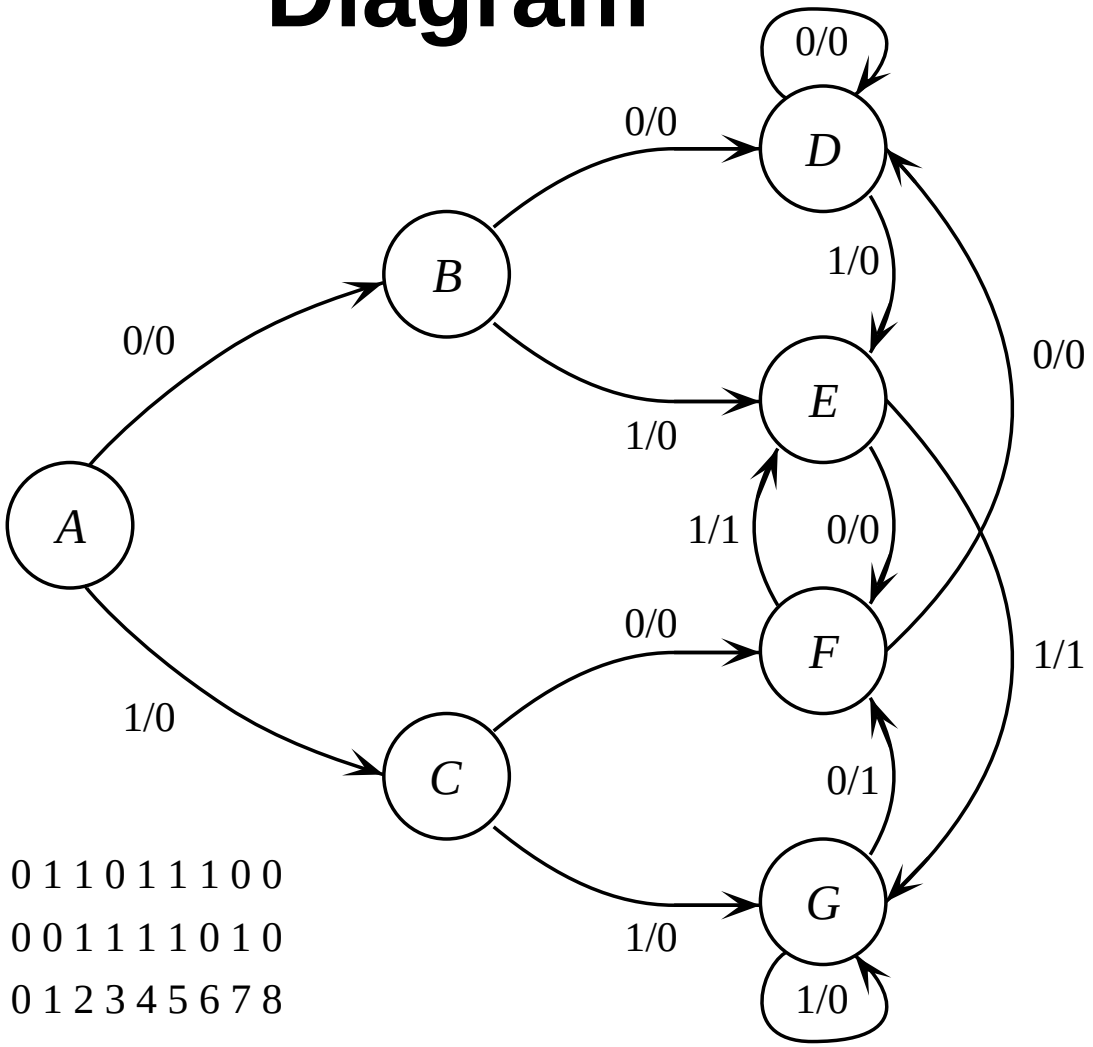


$$Z = \overline{S_1}\bar{X} + \overline{S_0}X$$

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.

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

Present state \ Input	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

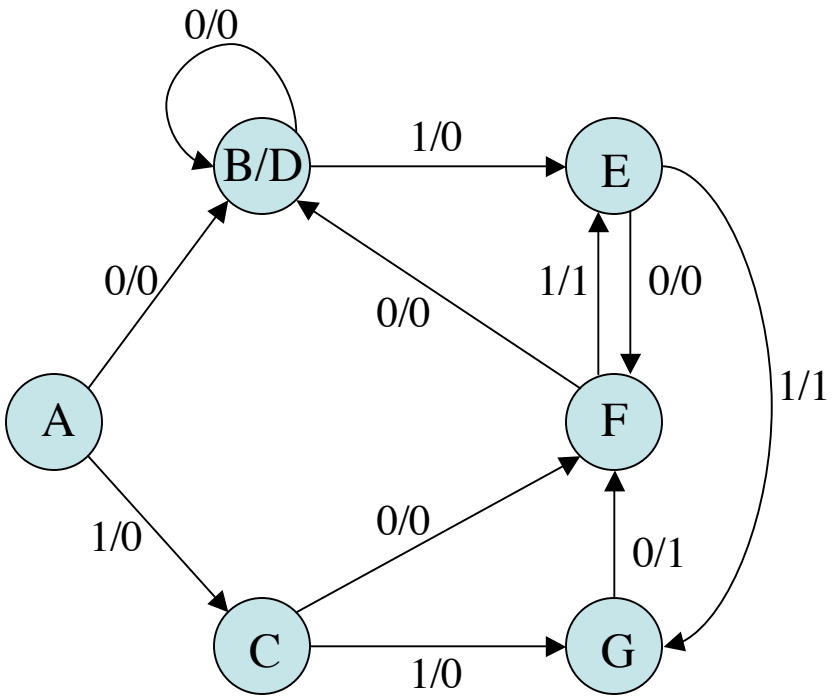
Sequence Detector State Reduction Table

	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

Present state \ Input	X = 0	X = 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

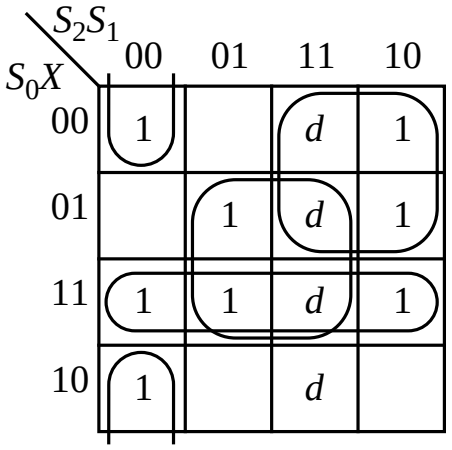


Sequence Detector State Assignment

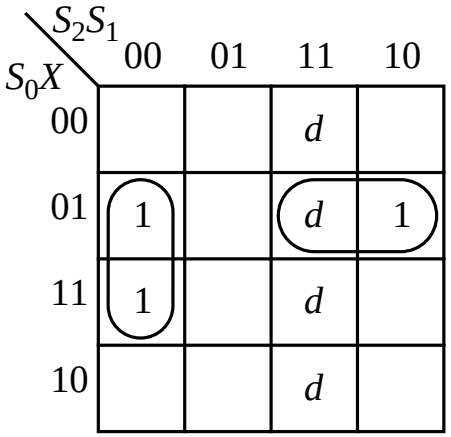
Present state \ Input	X = 0	X = 1
$S_2S_1S_0$	$S_2S_1S_0Z$	$S_2S_1S_0Z$
A': 000	001/0	010/0
B': 001	001/0	011/0
C': 010	100/0	101/0
D': 011	100/0	101/1
E': 100	001/0	011/1
F': 101	100/1	101/0

Sequence Detector K-Maps

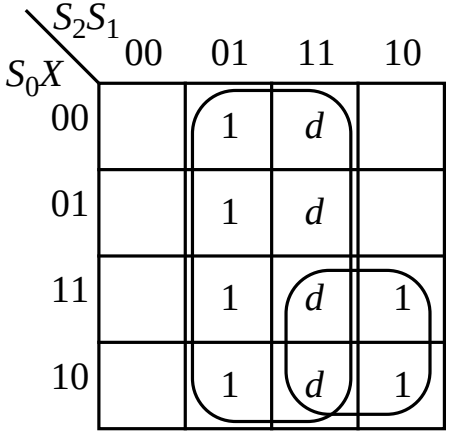
- **K-map reduction of next state and output functions for sequence detector.**



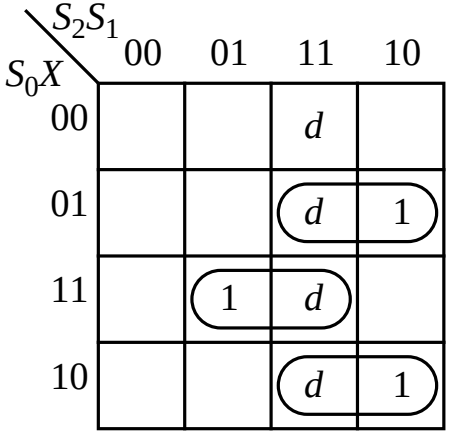
$$S_0 = \overline{S_2}\,\overline{S_1}\,\bar{X} + S_0X + S_2\overline{S_0} + S_1X$$



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



$$S_2 = S_2S_0 + S_1$$



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

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

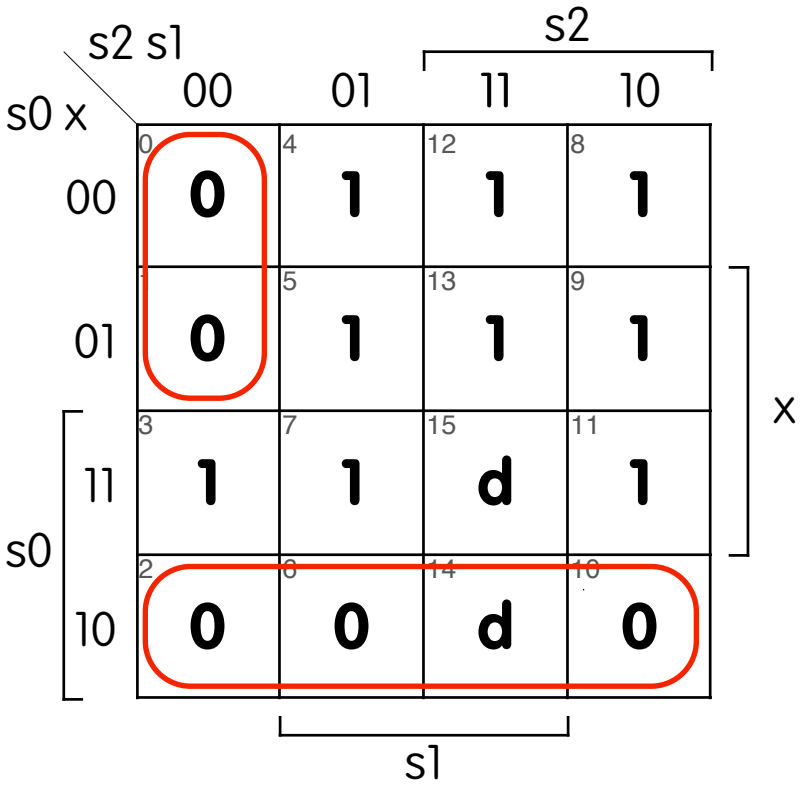
new 6-state

	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

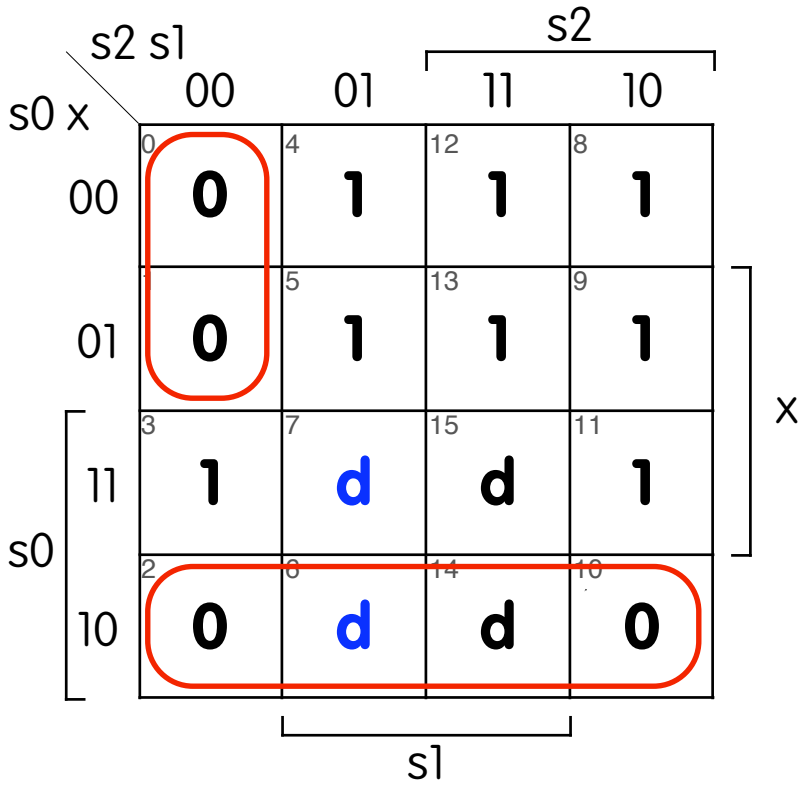
6-State Sequence Detector

7-state



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

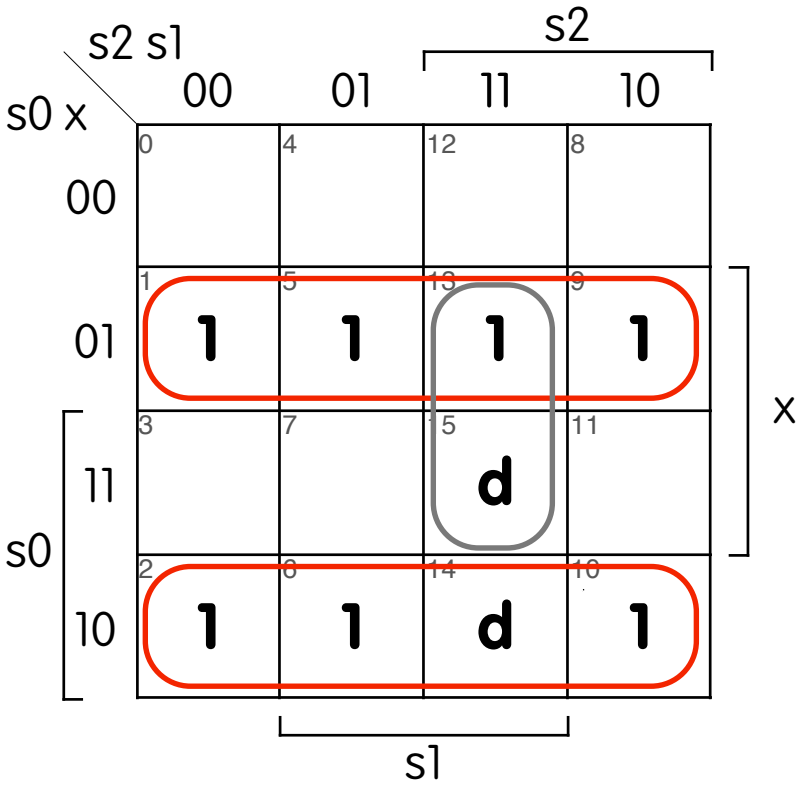
new 6-state



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

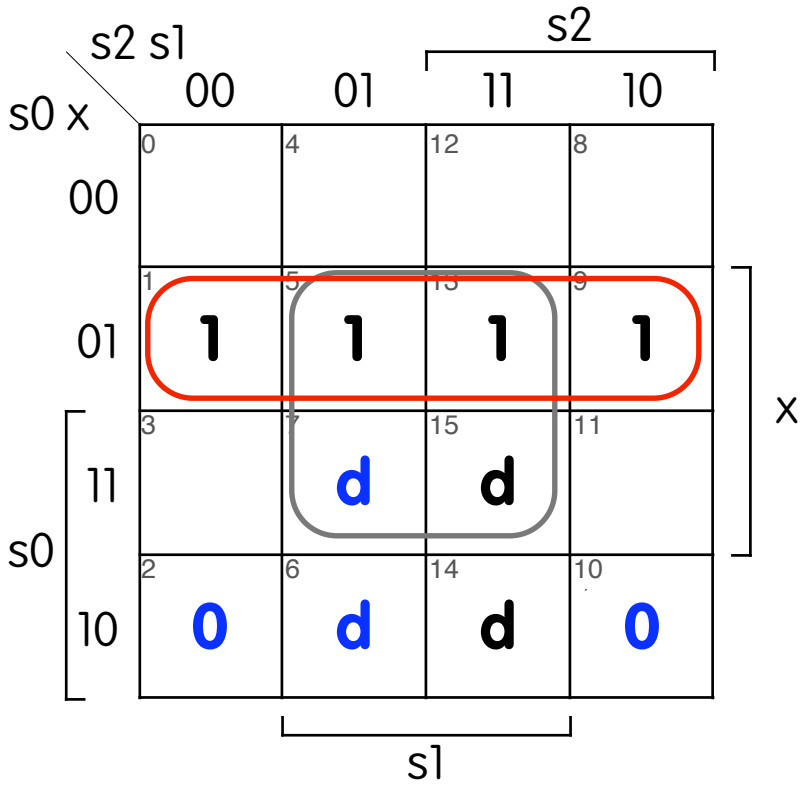
6-State Sequence Detector

7-state



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

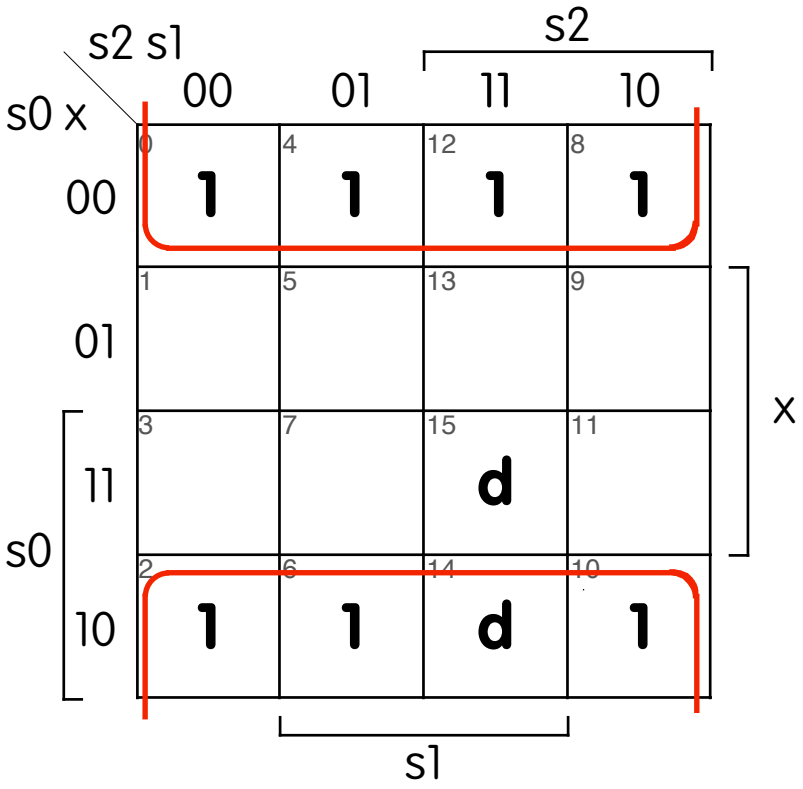
new 6-state



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

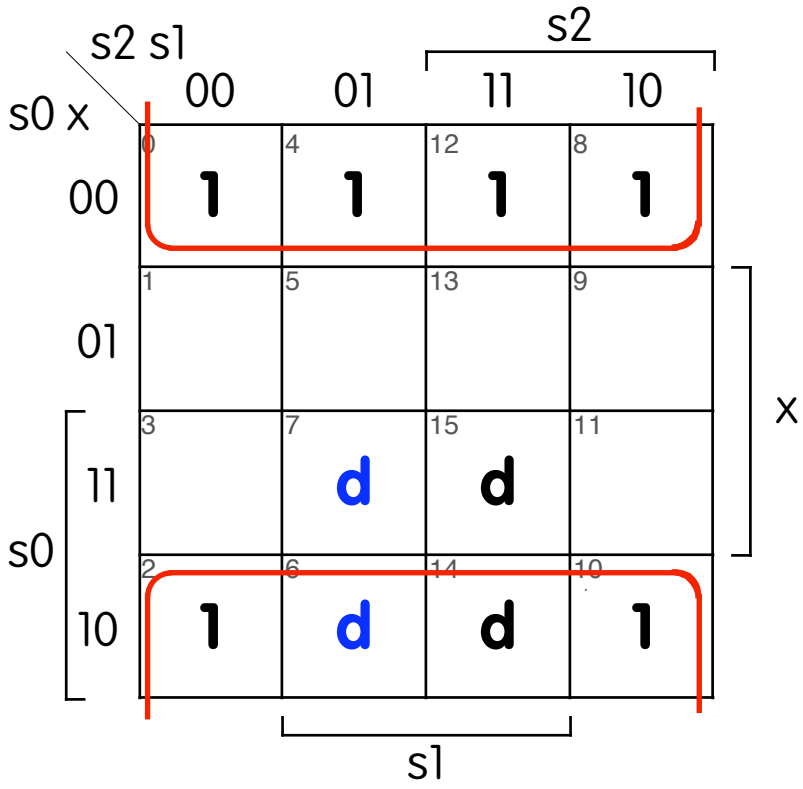
6-State Sequence Detector

7-state



$s0' = \overline{x}$

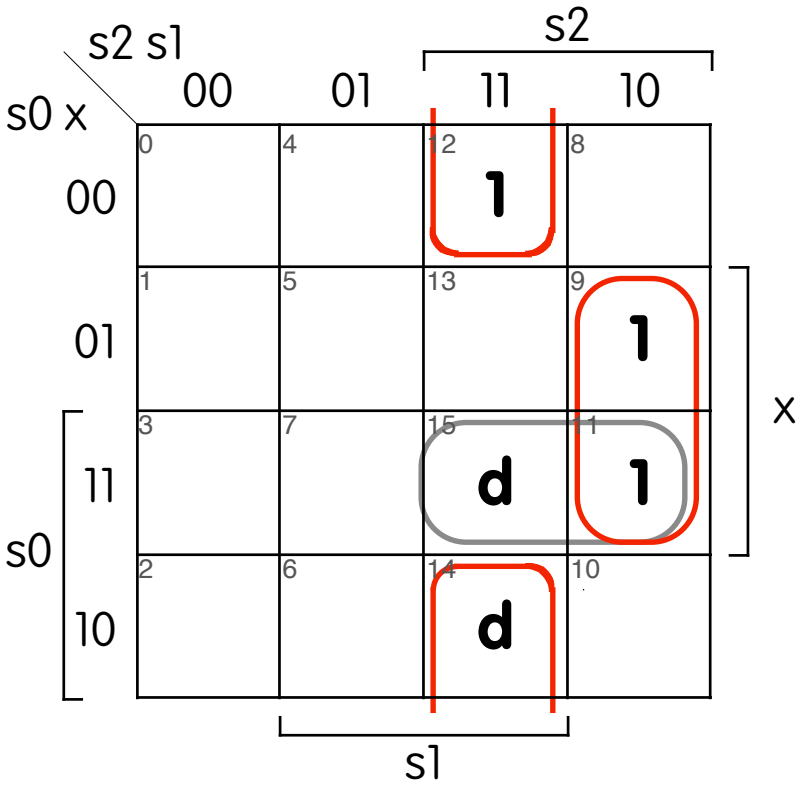
new 6-state



$s0' = \overline{x}$

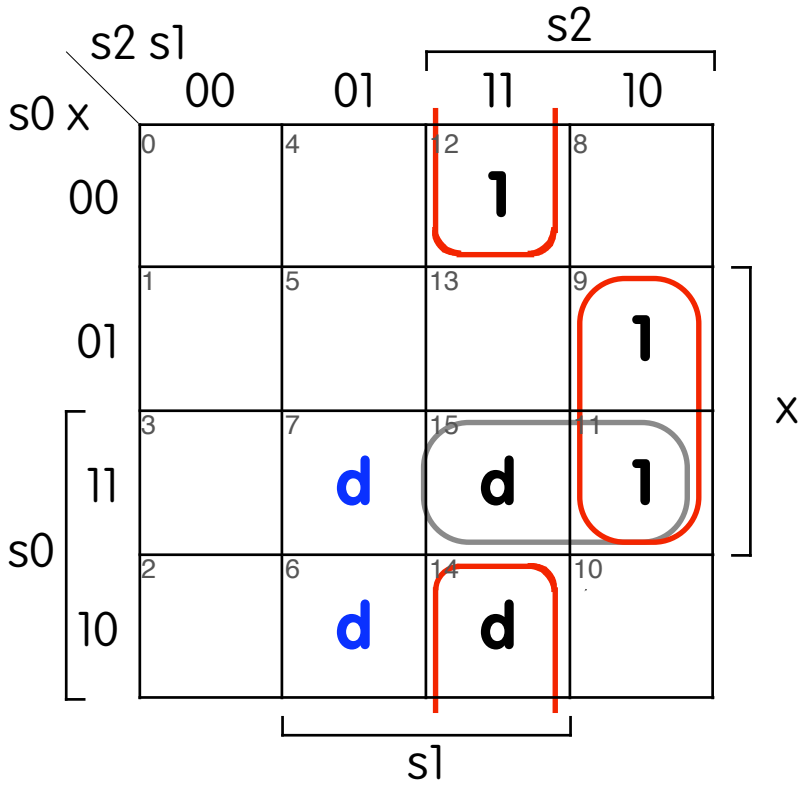
6-State Sequence Detector

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:**

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

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

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

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

Next Time

- more finite state machine design