CMSC 313 Lecture 24

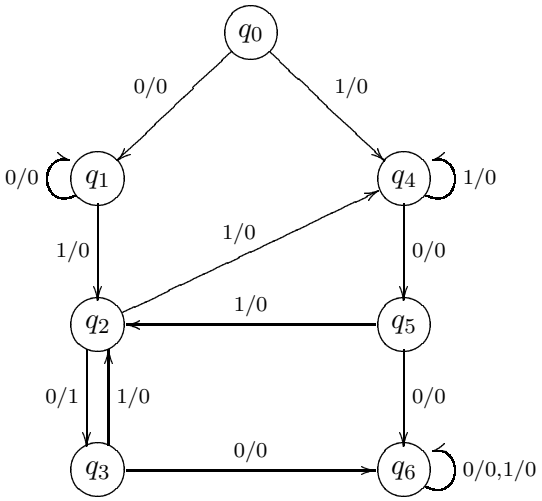
- DigSim Exercise 1 questions?
- Last Lecture:
  - J-K flip flops, edge-triggered flip-flops
  - Introduction to finite state machines
- Master-Slave J-K flip-flops & clocks
- Mealy vs Moore finite state machines
- Finite state machine design & simplification

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:

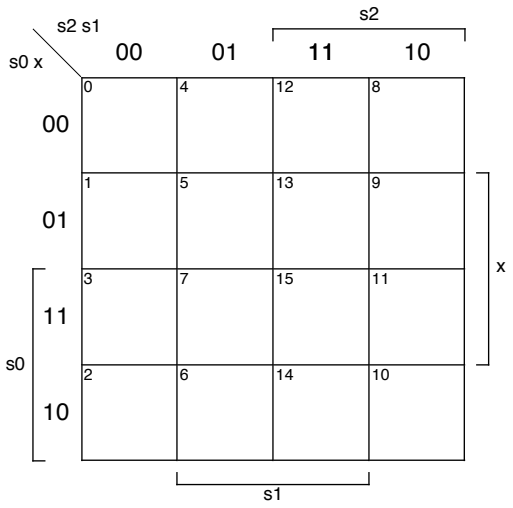
1. Make a copy of your truth-table and Karnaugh maps and submit the hard copy in class on Tuesday November 25.
2. Save your circuit as you did in the DigSim part of Homework 4. Submit the circuit file using the Unix `submit` command as in previous assignments. The submission name for this assignment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

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

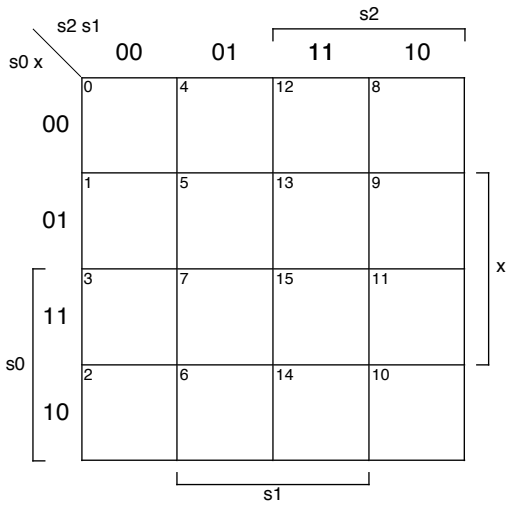
Name: _____

Truth table:

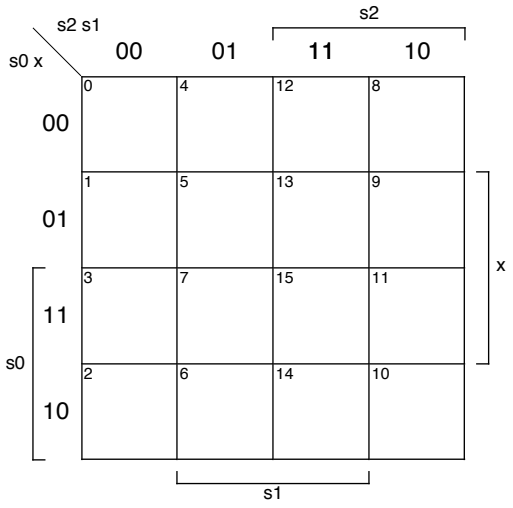
m	s_2	s_1	s_0	x	s_2'	s_1'	s_0'	z
0	0	0	0	0				
1	0	0	0	1				
2	0	0	1	0				
3	0	0	1	1				
4	0	1	0	0				
5	0	1	0	1				
6	0	1	1	0				
7	0	1	1	1				
8	1	0	0	0				
9	1	0	0	1				
10	1	0	1	0				
11	1	0	1	1				
12	1	1	0	0				
13	1	1	0	1				
14	1	1	1	0	d	d	d	d
15	1	1	1	1	d	d	d	d


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

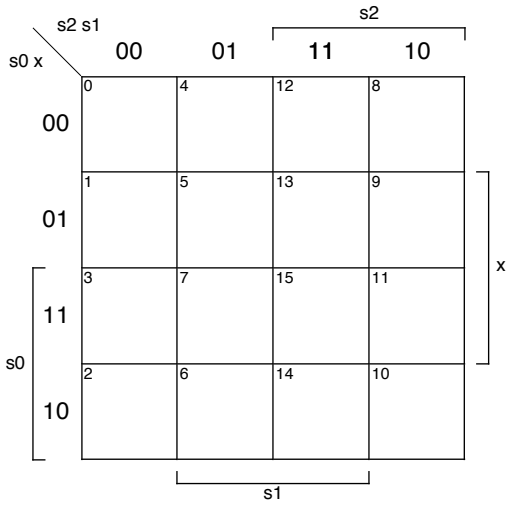
s2' =


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

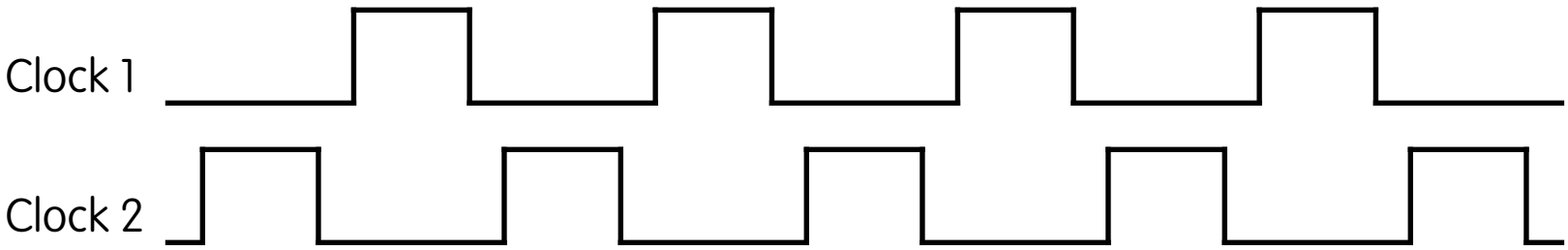
s0' =


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 =

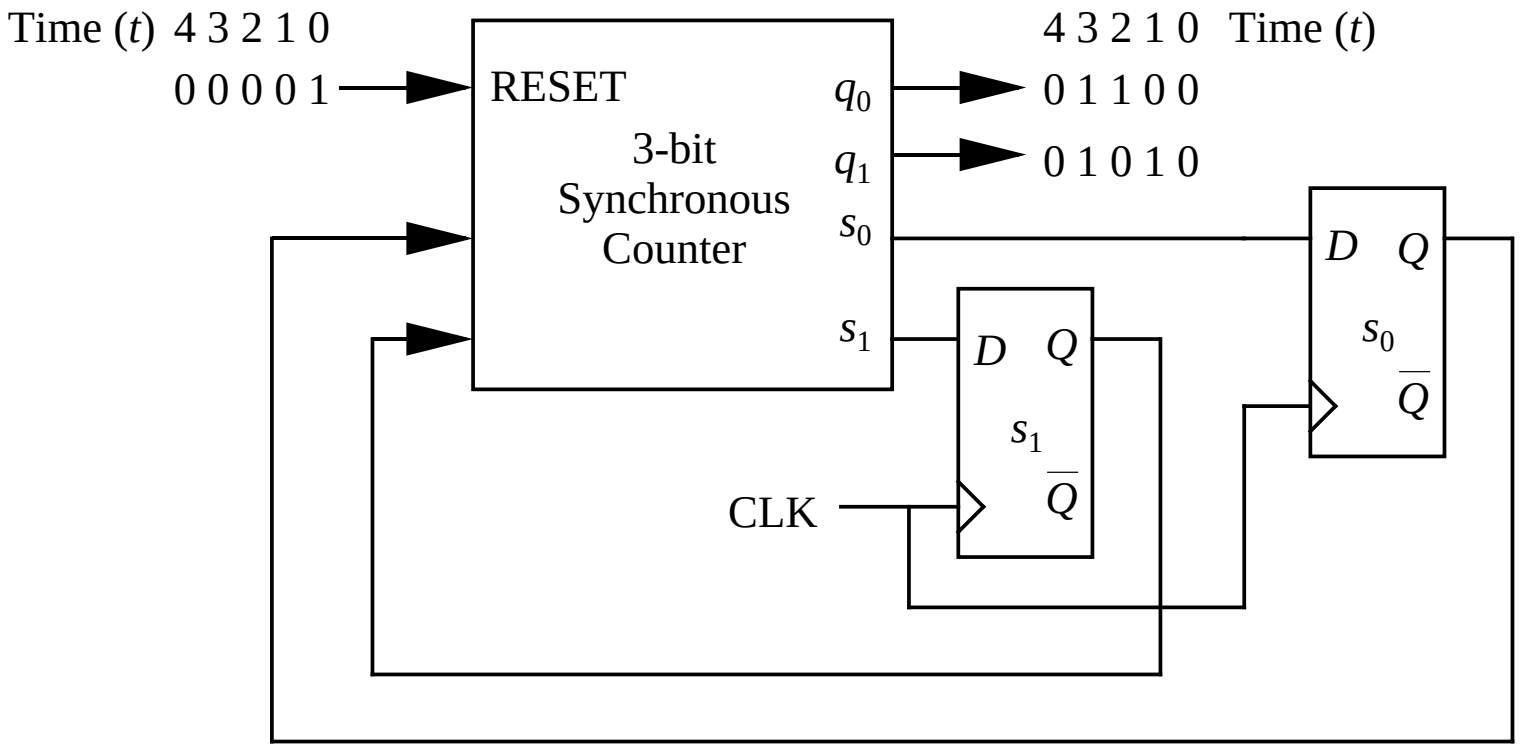
Master-Slave Flip-Flops Revisited

- **Master-slave J-K flip-flops**
  - ◇ **Should get rid of the endless toggle condition when J=1 & K=1**
  - ◇ **Clocking Problem 1: if master turns on before slave turns off, slave latches the input as soon as the clock goes high.**
  - ◇ **Clocking Problem 2: if slave turns on before master turns off, feeback of Q and ¬Q propagates through the master.**
- **Two-Phase Clocks: always turn off first!**

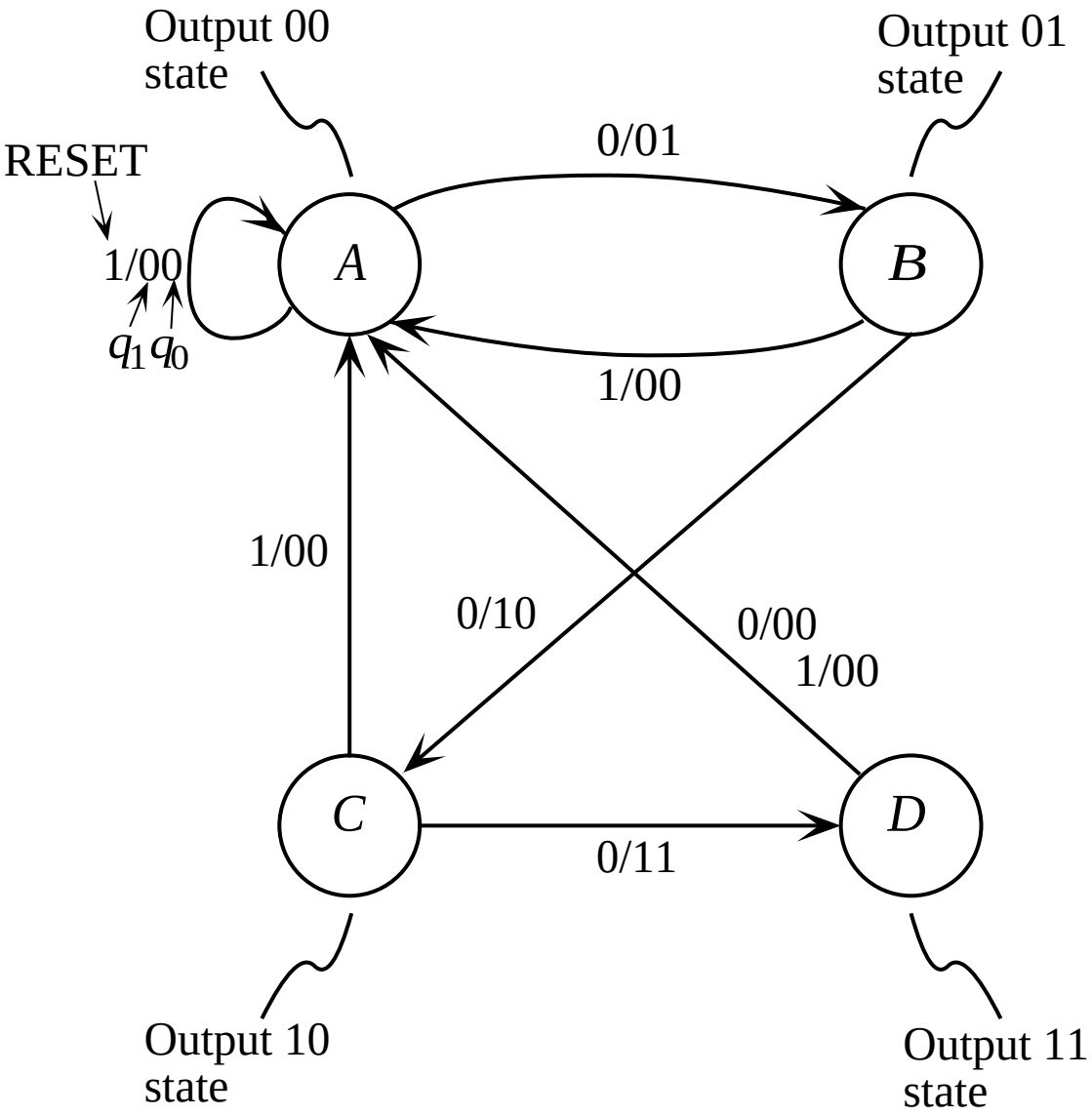


Example: Modulo-4 Counter

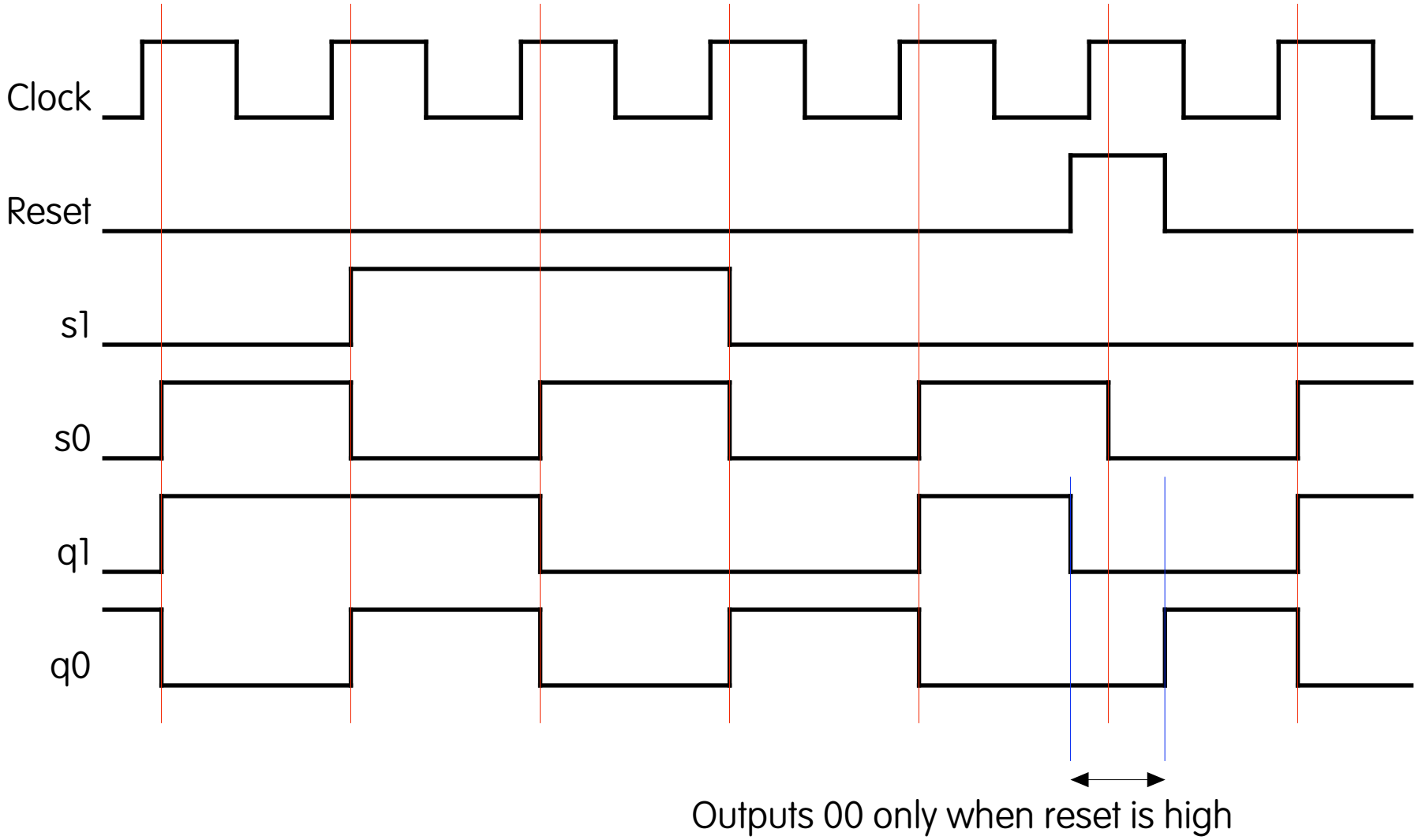
- **Counter has a clock input (CLK) and a RESET input.**
- **Counter has two output lines, which take on values of 00, 01, 10, and 11 on subsequent clock cycles.**



State Transition Diagram for Mod-4 Counter

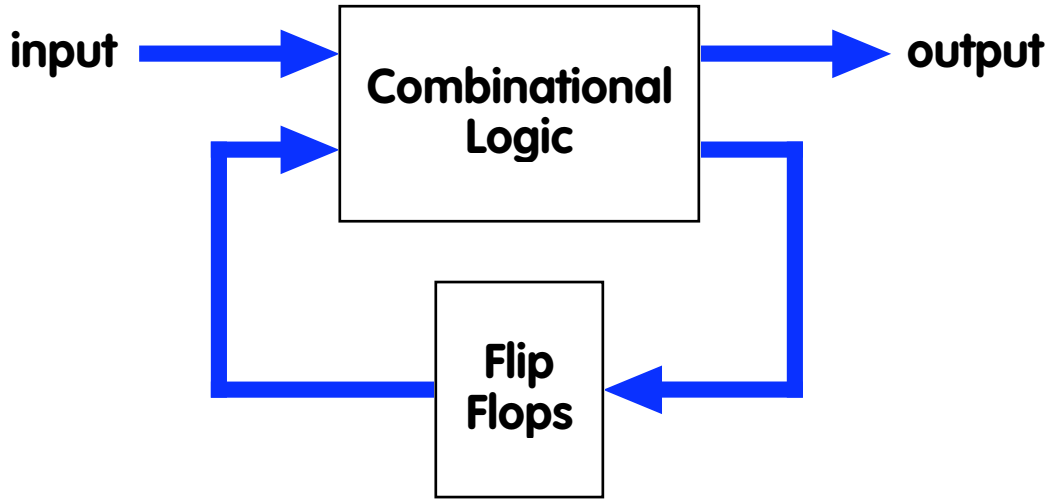


Mod 4 Counter Timing

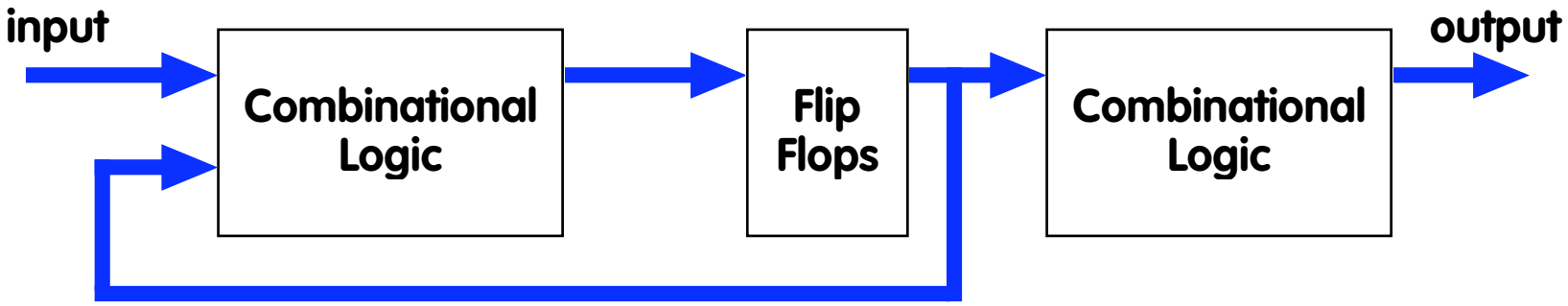


Mealy vs Moore Finite State Machines

- **Mealy: output depends on input and state bits**



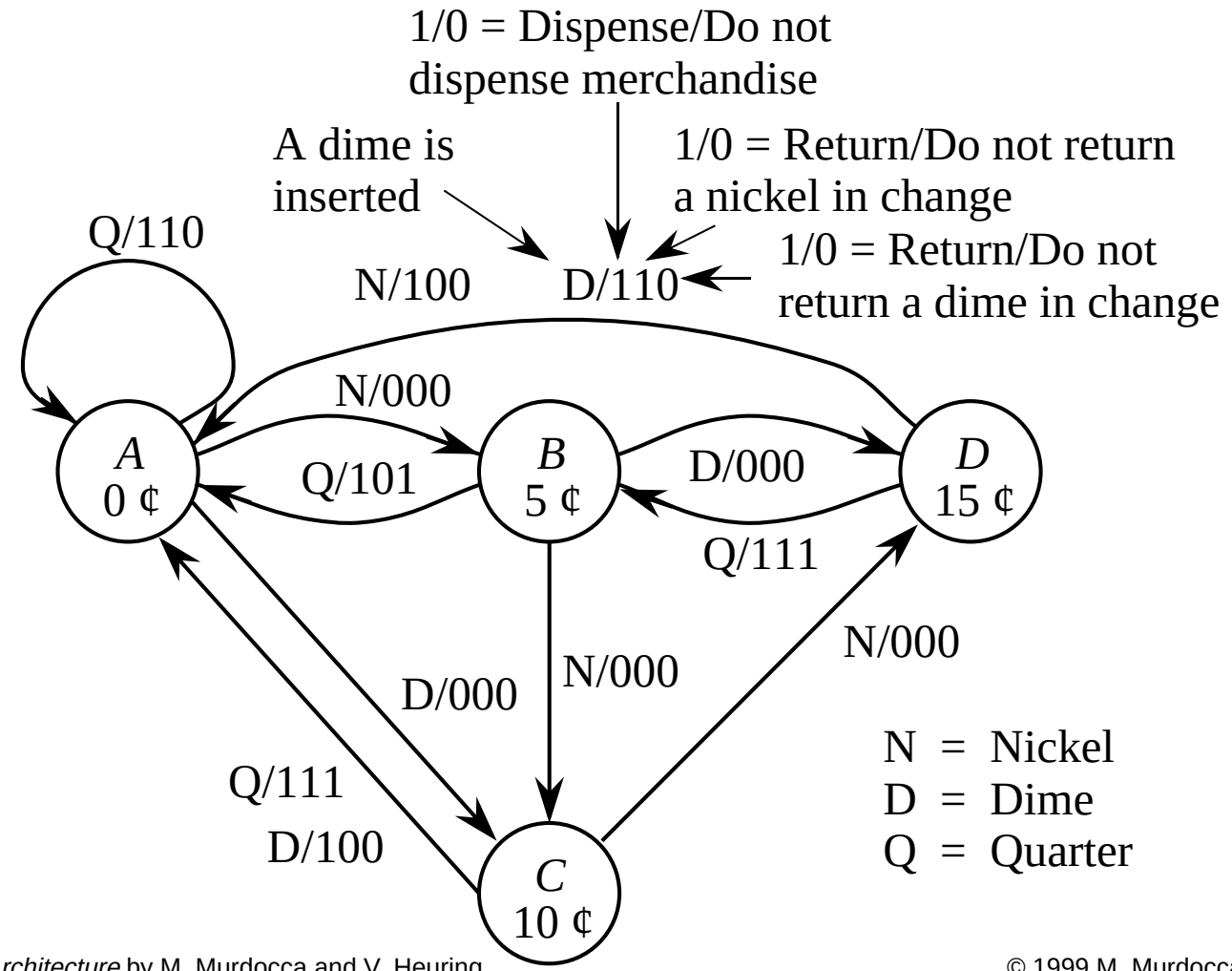
- **Moore: output depends only on state bits**



Example: A Vending Machine Controller

- **Example: Design a finite state machine for a vending machine controller that accepts nickels (5 cents each), dimes (10 cents each), and quarters (25 cents each). When the value of the money inserted equals or exceeds twenty cents, the machine vends the item and returns change if any, and waits for next transaction.**
- **Implement with PLA and D flip-flops.**

Vending Machine State Transition Diagram



Vending Machine State Table and State Assignment

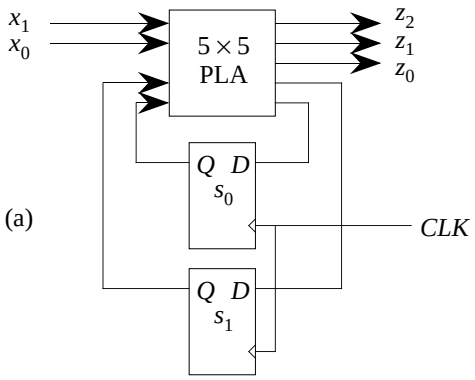
P.S. \ Input	N 00	D 01	Q 10
A	*B*/000	*C*/000	*A*/110
B	*C*/000	*D*/000	*A*/101
C	*D*/000	*A*/100	*A*/111
D	*A*/100	*A*/110	*B*/111

(a)

P.S. \ Input	N x_1x_0 00	D x_1x_0 01	Q x_1x_0 10
s_1s_0	s_1s_0 / $z_2z_1z_0$		
A:00	01/000	10/000	00/110
B:01	10/000	11/000	00/101
C:10	11/000	00/100	00/111
D:11	00/100	00/110	01/111

(b)

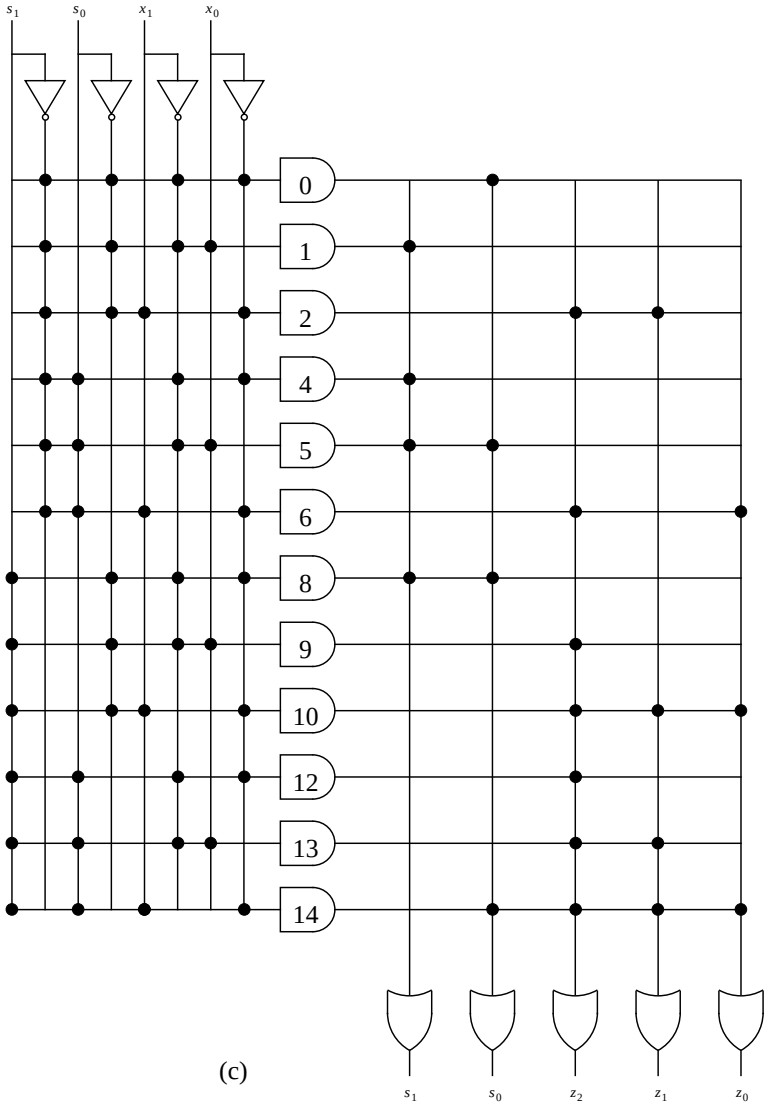
PLA Vending Machine Controller



(a)

Base 10 equivalent	Present state s_1	s_0	Coin x_1	x_0	Next state s_1	s_0	Dispense z_2	Return nickel z_1	Return dime z_0
0	0	0	0	0	0	1	0	0	0
1	0	0	0	1	1	0	0	0	0
2	0	0	1	0	0	0	1	1	0
3	0	0	1	1	d	d	d	d	d
4	0	1	0	0	1	0	0	0	0
5	0	1	0	1	1	1	0	0	0
6	0	1	1	0	0	0	1	0	1
7	0	1	1	1	d	d	d	d	d
8	1	0	0	0	1	1	0	0	0
9	1	0	0	1	0	0	1	0	0
10	1	0	1	0	0	0	1	1	1
11	1	0	1	1	d	d	d	d	d
12	1	1	0	0	0	0	1	0	0
13	1	1	0	1	0	0	1	1	0
14	1	1	1	0	0	1	1	1	1
15	1	1	1	1	d	d	d	d	d

(b)



(c)

Simplifying Finite State Machines

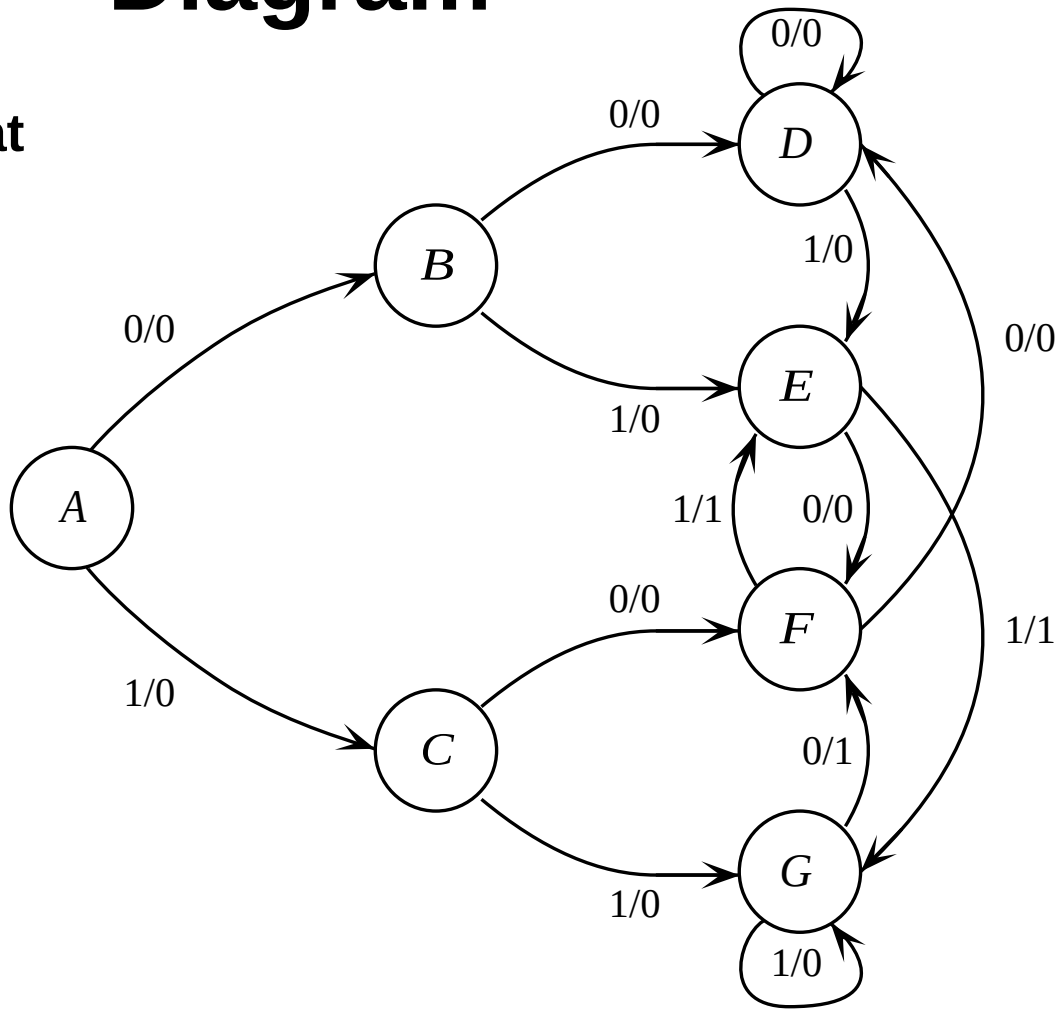
- State Reduction: equivalent FSM with fewer states
- State Assignment: choose an assignment of bit patterns to states (e.g., A is 010) that results in a smaller circuit
- Choice of flip-flops: use D flip-flops, J-K flip-flops or a T flip-flops? a good choice could lead to simpler circuits.

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

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 Assignment

Present state \ Input	X = 0	X = 1
	$S_2S_1S_0Z$	$S_2S_1S_0Z$
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

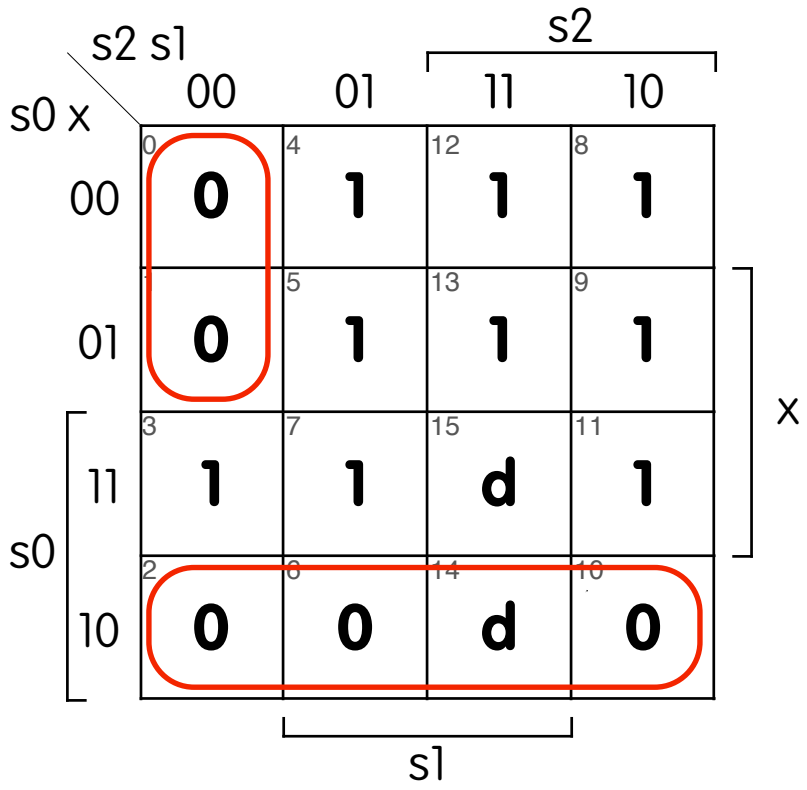
(a)

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 Detector

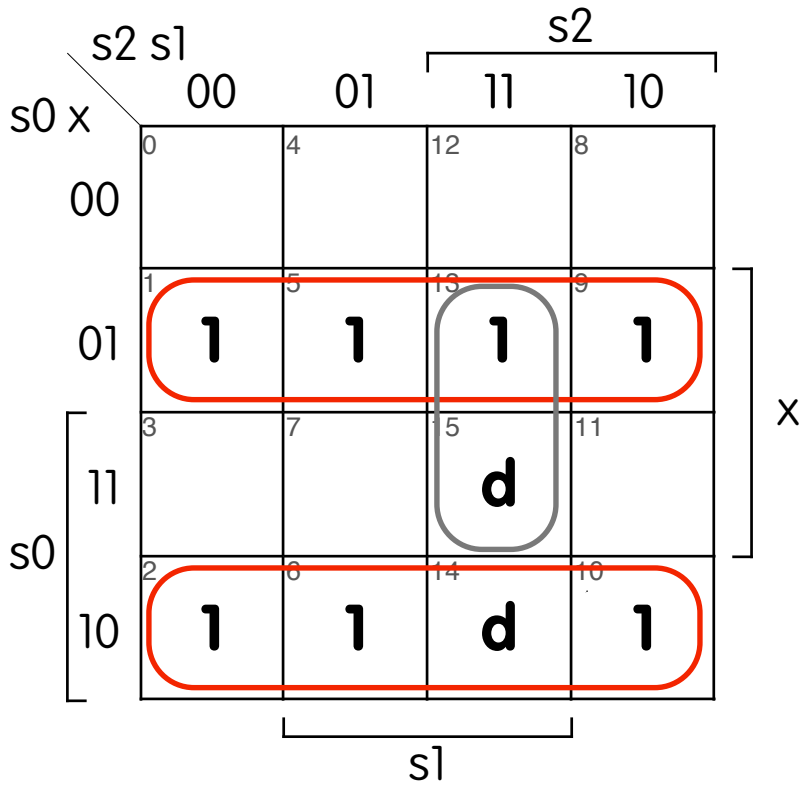
	s2	s1	s0	x	s2'	s1'	s0'	z
0	0	0	0	0	0	0	1	0
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	1	0
3	0	0	1	1	1	0	0	0
4	0	1	0	0	1	0	1	0
5	0	1	0	1	1	1	0	0
6	0	1	1	0	0	1	1	0
7	0	1	1	1	1	0	0	0
8	1	0	0	0	1	0	1	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	0	1	1	0
11	1	0	1	1	1	0	0	1
12	1	1	0	0	1	0	1	1
13	1	1	0	1	1	1	0	0
14	1	1	1	0	d	d	d	d
15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



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

Sequence Detector

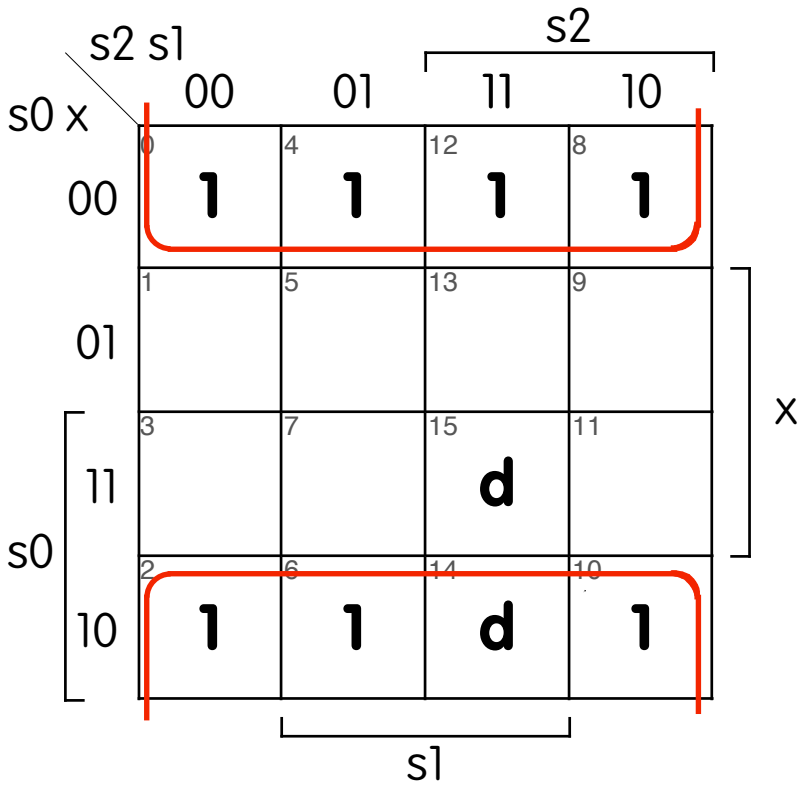
	s2	s1	s0	x	s2'	s1'	s0'	z
0	0	0	0	0	0	0	1	0
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	1	0
3	0	0	1	1	1	0	0	0
4	0	1	0	0	1	0	1	0
5	0	1	0	1	1	1	0	0
6	0	1	1	0	0	1	1	0
7	0	1	1	1	1	0	0	0
8	1	0	0	0	1	0	1	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	0	1	1	0
11	1	0	1	1	1	0	0	1
12	1	1	0	0	1	0	1	1
13	1	1	0	1	1	1	0	0
14	1	1	1	0	d	d	d	d
15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



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

Sequence Detector

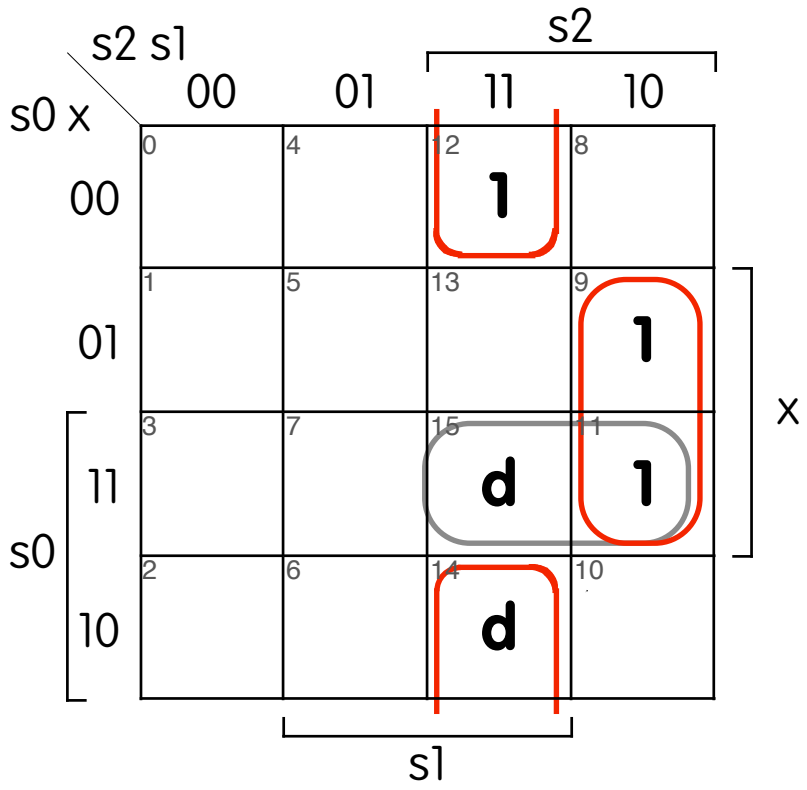
	s2	s1	s0	x	s2'	s1'	s0'	z
0	0	0	0	0	0	0	1	0
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	1	0
3	0	0	1	1	1	0	0	0
4	0	1	0	0	1	0	1	0
5	0	1	0	1	1	1	0	0
6	0	1	1	0	0	1	1	0
7	0	1	1	1	1	0	0	0
8	1	0	0	0	1	0	1	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	0	1	1	0
11	1	0	1	1	1	0	0	1
12	1	1	0	0	1	0	1	1
13	1	1	0	1	1	1	0	0
14	1	1	1	0	d	d	d	d
15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



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

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



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

Next Time

- more finite state machine design