
Digital Logic Design

ECEN 3233

Module 6-1: Introduction to Sequential Circuits

Louis G. Johnson

School of Electrical and Computer Engineering
Oklahoma State University

Fall 2007

Overview

- What is sequential logic?
- The D flip-flop
- Describing sequential circuits
 - state diagrams
 - state transition tables

Sequential Circuits

- Sequential circuits have “memory”.
- The output(s) depends on current input **and** past history of inputs – concept of “state”.
- “State” embodies all the information about the past needed to predict current output based on current input.
 - *State variables*, one or more bits of information.
- A “clock signal” synchronizes circuit operation.
 - State changes are synchronized with the clock

- Examples – a digital clock, a stoplight controller, an elevator controller, a digital vending machine controller

Motivation

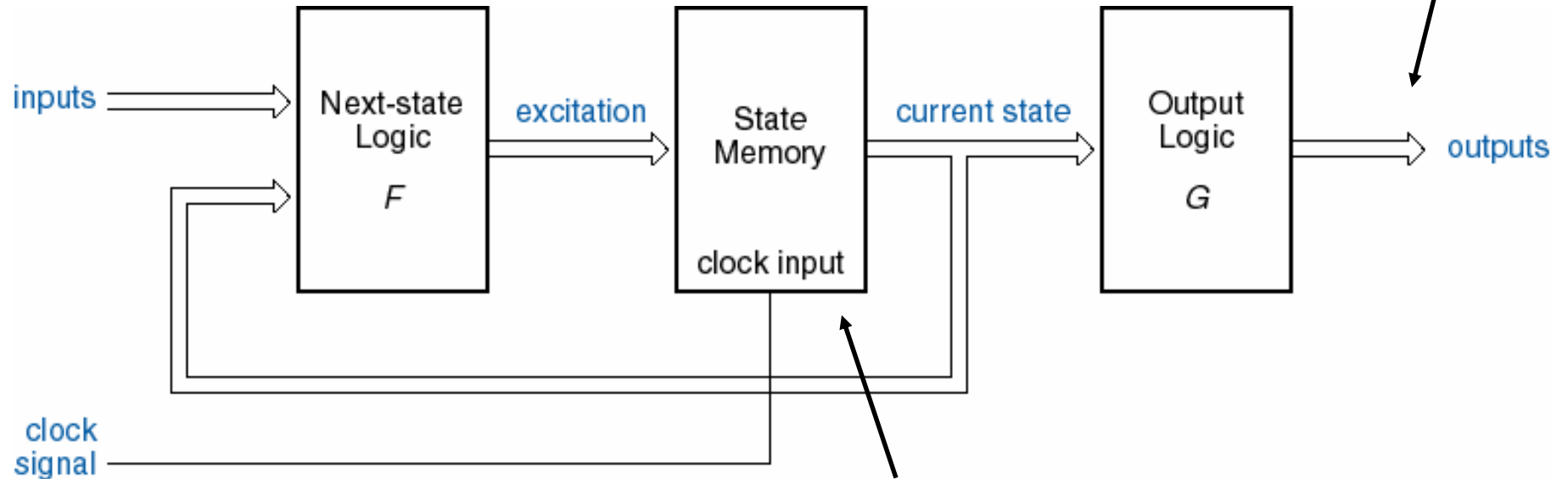
- **Counters: Sequential Circuits where State = Output**
 - A counter proceeds through a pre-defined sequence
 - The sequence repeats once the “end” is reached

- **Generalizes to Finite State Machines:**
 - Outputs are a Function of State (and Inputs)
 - Next States are Functions of State and Inputs
 - Used to implement circuits that control other circuits
 - “Decision Making” logic

State-Machine Structure (Moore Machine)

This is the type of state machine (Moore machine) that we usually design. The outputs are a function only of the present state (the Q outputs of the flip-flops). Mealy machines are harder to work with, but can be more efficient (require fewer flip-flops).

output depends on state only

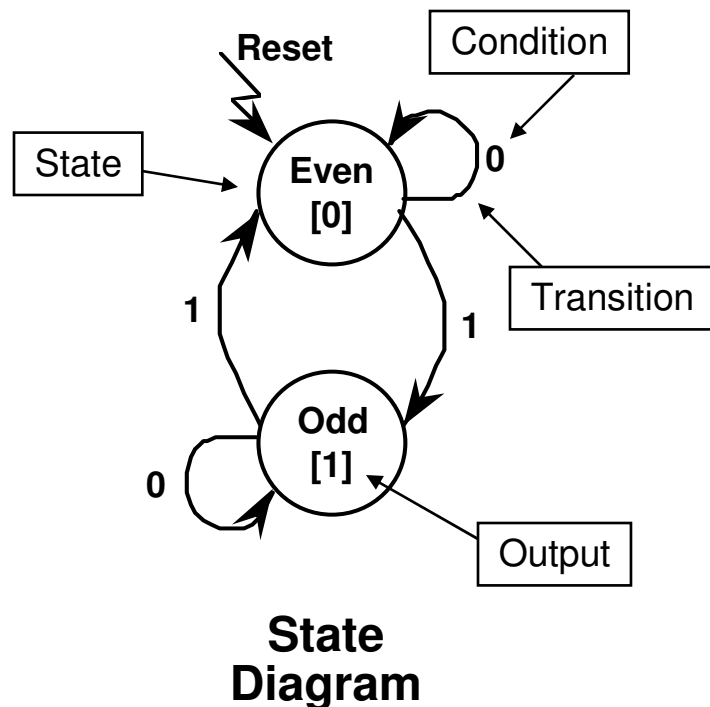


typically edge-triggered D flip-flops

Concept of the State Machine

Example: Odd Parity Checker

Assert output whenever input bit stream has odd # of 1's



Present State	Input	Next State	Output
Even	0	Even	0
Even	1	Odd	0
Odd	0	Odd	1
Odd	1	Even	1

Symbolic State Transition Table

Present State	Input	Next State	Output
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

Encoded State Transition Table

The State Diagram is a graphical version of the State Transition Table. Draw a State Diagram, and label it, to aid in understanding what your state machine is supposed to do. Both the State Table and State Diagram illustrate how the state machine moves from one state to another, the input conditions that are necessary to make the transition, and the outputs associated with each state.

Concept of the State Machine

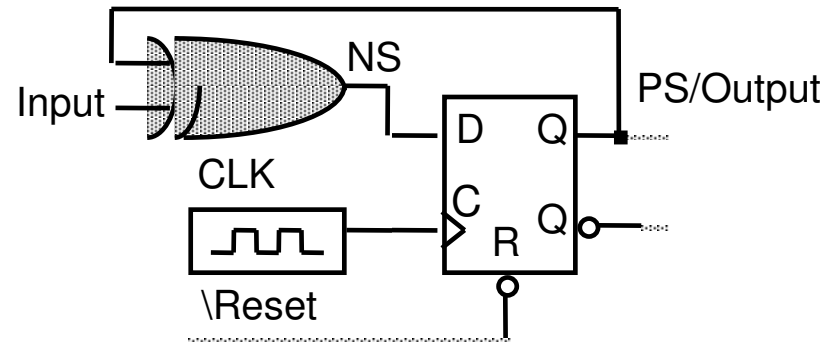
Example: Odd Parity Checker

D flip-flop: $Q \leftarrow D$ after clock is applied

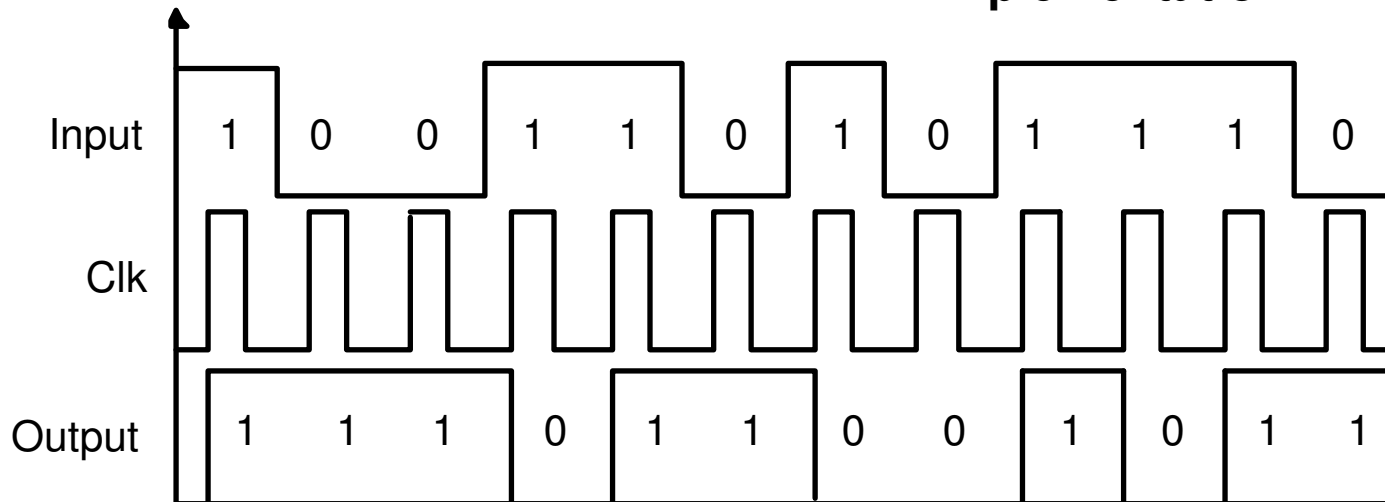
(Characteristic Equation)

Next State/Output Functions:

$$NS = PS \text{ xor } INPUT; \quad OUT = PS$$



D FF Implementation



Timing Behavior: Input 1 0 0 1 1 0 1 0 1 1 1 0

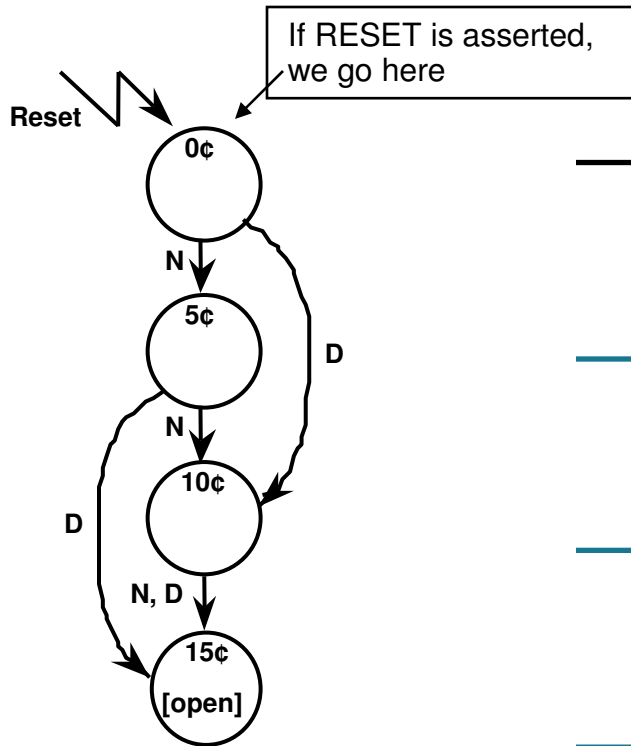
Concept of State Machine

- **Clocking event causes state/outputs to transition, based on inputs and present state**
- **Inputs must be stable before clocking event, or unexpected behavior may result**
- **After propagation delay, Next State entered and Outputs are stable**
- **Some signals are “asynchronous”**
 - they take effect immediately
 - they don't depend on the clock
 - example: \Reset

Describing Sequential Circuits

- State table
 - For each current-state, specify next-states as function of inputs
 - For each current-state, specify outputs as function of inputs
- State diagram
 - Graphical version of state table

Vending Machine Example



VEND the item once at least \$0.15 has been inserted using any combination of nickels and dimes

Present State	Inputs		Next State	Output Open
	D	N		
0¢	0	0	0¢	0
	0	1	5¢	0
	1	0	10¢	0
	1	1	X	X
5¢	0	0	5¢	0
	0	1	10¢	0
	1	0	15¢	0
	1	1	X	X
10¢	0	0	10¢	0
	0	1	15¢	0
	1	0	15¢	0
	1	1	X	X
15¢	X	X	15¢	1

Symbolic State Table or Symbolic State Transition Table

The State Transition Table is a tabular version of the State Diagram. We begin by using descriptive names (mnemonics, etc.) for the states to aid in understanding. Finally, we *encode* the state transition table by assigning binary values to the states (next page), replacing the names we started with. The binary values are the actual Q values for the FFs in the circuit. The Q's define the states. We assign don't cares (X) for input combinations that cannot occur (i.e., we cannot insert both a nickel and dime at the same time).

Vending Machine Example

State Encoding

Present State		Inputs		Next State		Output
Q ₁	Q ₀	D	N	D ₁	D ₀	Open
0	0	0	0	0	0	0
		0	1	0	1	0
		1	0	1	0	0
		1	1	X	X	X
0	1	0	0	0	1	0
		0	1	1	0	0
		1	0	1	1	0
		1	1	X	X	X
1	0	0	0	1	0	0
		0	1	1	1	0
		1	0	1	1	0
		1	1	X	X	X
1	1	0	0	1	1	1
		0	1	1	1	1
		1	0	1	1	1
		1	1	X	X	X

Assign binary state encodings*

Encoded State Transition Table

*For example, if we are in state [10] then \$0.10 has been inserted. If we are in state [11] then at least \$0.15 has been inserted and we assert the VEND signal to release the item. State [00] is the reset (starting) state.

If D flip-flops are used, the Next State columns represent the *excitation* that must be applied to the D inputs to cause the flip-flops to transition to the desired next state upon application of the clock signal.

$$Q \leftarrow D$$

We often write this as

$$Q^+ \leftarrow D$$

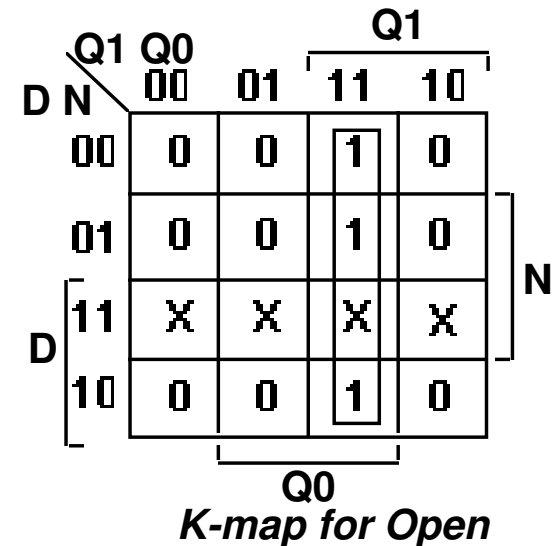
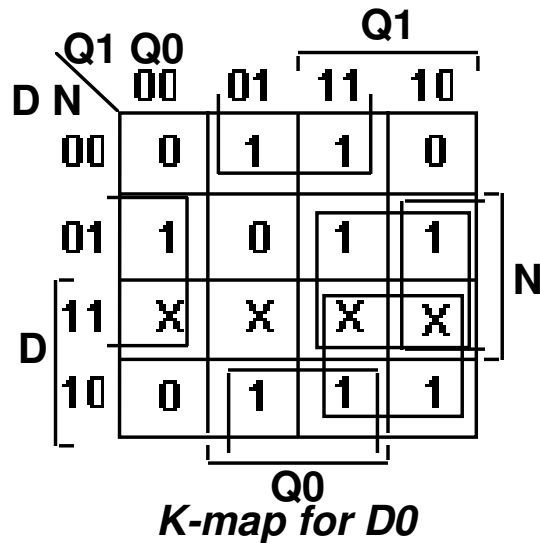
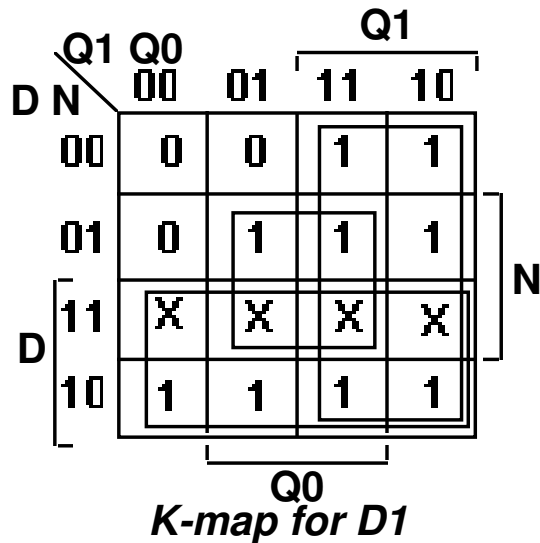
or just

$$Q^+ = D$$

To distinguish the Next State from the Present State.

Vending Machine Example

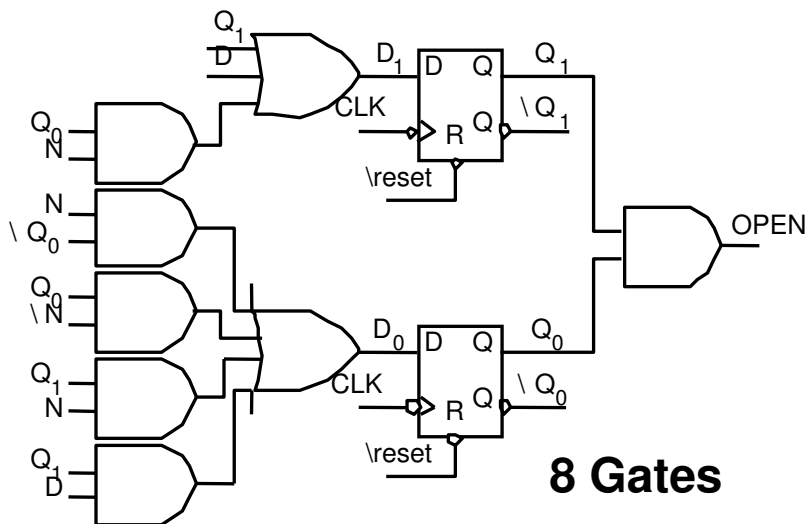
Determine minimized excitation equations based on desired transitions



$$D1 = Q1 + D + Q0 N$$

$$D0 = N \overline{Q0} + \overline{Q0} \overline{N} + Q1 N + Q1 D$$

$$OPEN = Q1 Q0$$



The present state (Q1 and Q0) and the nickel (N) and dime (D) signals are the inputs. We must design logic that takes these inputs and computes the excitation logic for the two D FF inputs (D1 and D0) necessary to produce the next state values (Q1+ and Q0+). The K-maps are used to determine the minimized logic for this operation. The circuit will now move to the correct next state given the present state (Q1 and Q0) and inputs N and D.