EE292: Fundamentals of ECE

Fall 2012
TTh 10:00-11:15 SEB 1242

Lecture 23
121120

http://www.ee.unlv.edu/~b1morris/ee292/

Outline

- Review
  - Combinatorial Logic
- Sequential Logic

Combinatorial Logic Circuits

- Combine logical variable inputs to produce a logic-variable output
- Logic can be specified by enumerating the output for all possible input combinations in a truth table

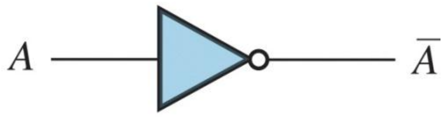
- Considered memoryless circuits
  - The output at a given time instant are only dependent on the input at the same time instant

Basic Logic Gates

- Inverter – NOT operation or complement of a variable
  - $\text{NOT}(A) = \bar{A}$

A	$\bar{A}$
0	1
1	0

(a) Truth table

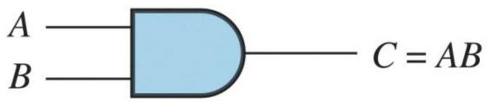


(b) Symbol for an inverter

- AND - computes the logical multiplication of input variables
  - $\text{AND}(A, B) = AB$

A	B	$C = AB$
0	0	0
0	1	0
1	0	0
1	1	1

(a) Truth table

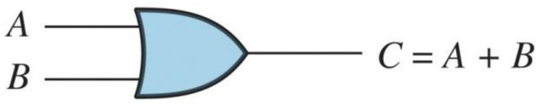


(b) Symbol for two-input AND gate

- OR - computes the logical addition of input variables
  - $\text{OR}(A, B) = A + B$

A	B	$C = A + B$
0	0	0
0	1	1
1	0	1
1	1	1

(a) Truth table

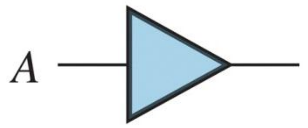


(b) Symbol for two-input OR gate

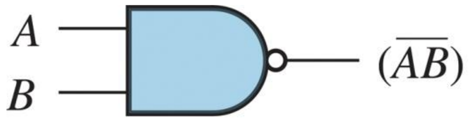
More Logic Gates

- In silicon chips, other gates are simpler to implement
- Buffer = NOT followed by NOT
  - Returns the same value
- NAND = AND followed by NOT
- NOR = OR followed by NOT
- XOR is the exclusive-OR gate
  - $\text{XOR}(A, B) = A \oplus B$

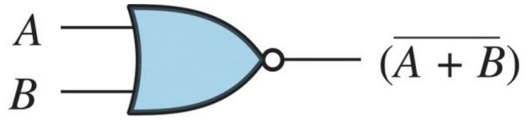
A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0



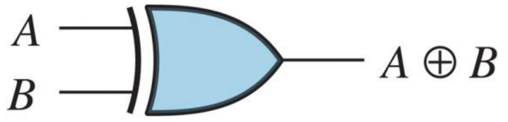
(d) Buffer



(a) NAND gate



(b) NOR gate



(c) XOR gate

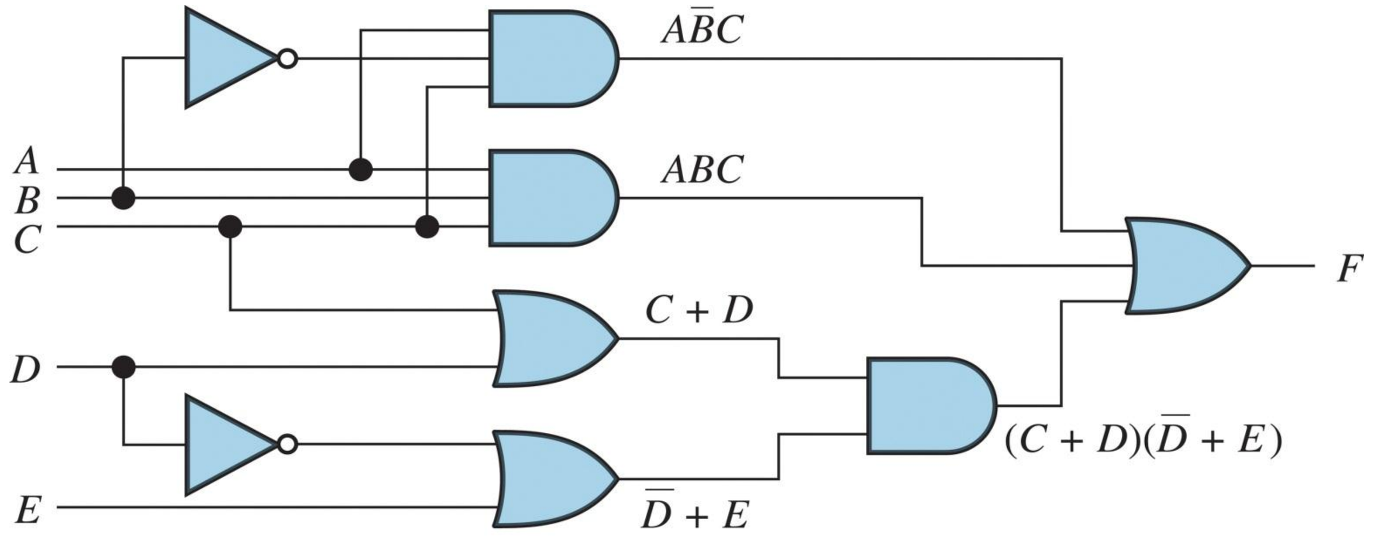
Boolean Algebra

- Mathematical theory of logical variables
- Use basic AND, OR, and NOT relationships to prove a Boolean expression
  - Can generate a truth table to specify the output relationship for all possible input values

- De Morgan's Laws
  - Provides a way to convert an AND relationship into an OR relationship and vice versa
  - $ABC = \overline{\bar{A} + \bar{B} + \bar{C}}$
  - $(A + B + C) = \overline{\bar{A}\bar{B}\bar{C}}$

Implementation of Boolean Expressions

- A logical variable can be composed of Boolean relationships
  - AND, OR, NOT, etc.
- Gate level implementation is straightforward
- Example
- $F = A\bar{B}C + ABC + (C + D)(\bar{D} + E)$



Simplifying Boolean Expression

- Find simpler equivalent expressions by manipulation of equation and Boolean relations

- Example
- $F = A\bar{B}C + ABC + (C + D)(\bar{D} + E)$
- $F = A\bar{B}C + ABC + (C\bar{D} + CE + D\bar{D} + DE)$
- $F = A\bar{B}C + ABC + (C\bar{D} + CE + 0 + DE)$
- $F = AC(\bar{B} + B) + (C\bar{D} + CE + 0 + DE)$
- $F = AC(1) + (C\bar{D} + CE + 0 + DE)$
- $F = C(A + \bar{D} + E) + DE$

Synthesis of Logic

- Require methods to convert logic circuit specifications into a practical gate level implementation
  - Often the logic is specified in a natural language
- Example: automatic windshield wipers
  - When it is raining (input 1) and cloudy (input 2) the wipers should be on (output)
- Example: 3 Logical variable input and 1 output
  - Enumerate all possible input values and specify the corresponding output
  - A, B, C input and D output

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

Sum-of-Products Implementation

- Find all output rows that have a 1 output
  - Determine AND relationship between inputs
- OR the AND terms from each row

A	B	C	D		**AND Term**
0	0	0	1		$\bar{A}\bar{B}\bar{C}$
0	0	1	0		
0	1	0	1		$\bar{A}B\bar{C}$
0	1	1	0		
1	0	0	0		
1	0	1	0		
1	1	0	1		$AB\bar{C}$
1	1	1	1		ABC

$$D = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + AB\bar{C} + ABC$$

Product-of-Sums Implementation

- Find all output rows that have a 0 output
  - Determine OR relationship between inputs
- AND the OR terms from each row

A	B	C	D		**OR Term**
0	0	0	1		
0	0	1	0		$A + B + \bar{C}$
0	1	0	1		
0	1	1	0		$A + \bar{B} + \bar{C}$
1	0	0	0		$\bar{A} + B + C$
1	0	1	0		$\bar{A} + B + \bar{C}$
1	1	0	1		
1	1	1	1		

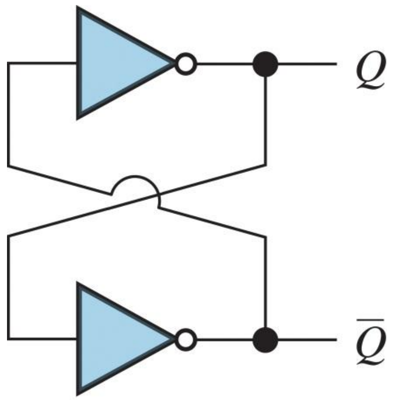
$$D = (A + B + \bar{C})(A + \bar{B} + \bar{C})(\bar{A} + B + C)(\bar{A} + B + \bar{C})$$

Sequential Logic

- Combinatorial logic output is only dependent on input at the given time
- Sequential logic has outputs that are dependent not only on current input but past input as well
  - The circuits have “memory”
- Often times sequential circuits use a clock signal to regulate when the output should change
  - These are called synchronous circuits
  - Asynchronous circuits are able to change as soon as inputs change (no clock signal is required)

The Flip-Flop

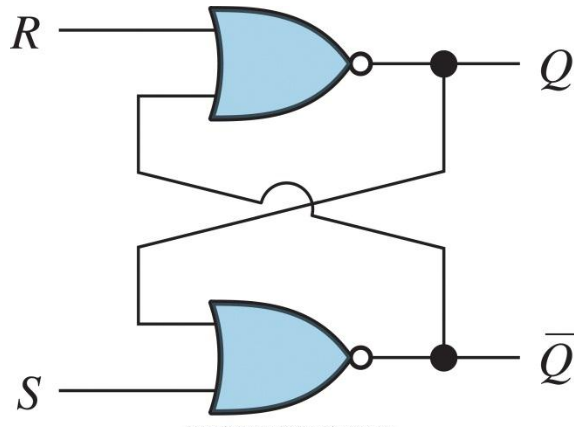
- This is the basic building block for sequential circuits
- A flip-flop has two allowable “states” of operation
  - It is able to store a bit of information
- When Q is high
  - $\bar{Q}$ will be low
- When Q is low
  - $\bar{Q}$ will be high
- Coupled inverter configuration ensures the state does not change



- Once Q is set, it does not change
- How can we set the flip-flop state?

SR Flip-Flops

- Set-reset (SR) flip-flop allows control of the state
- When R and S are low
  - This behaves as the coupled inverters
  - Remains in "set state" (Q set stays high)
- When S is high and R is low
  - Top NOR is an inverter and Q is forced high and $\bar{Q}$ low
- When R is high and S is low
  - Bottom NOR is an inverter and $\bar{Q}$ is forced high and Q is low

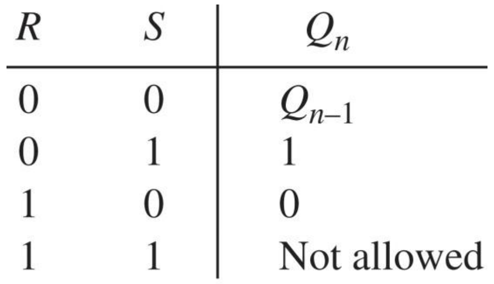


- R and S cannot both be set at the same time

SR Flip-Flop Truth Table

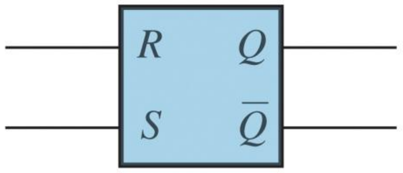
- S = Set
- R = Reset

- Truth Table



R	S	Q_n
0	0	Q_{n-1}
0	1	1
1	0	0
1	1	Not allowed

(a) Truth table



(b) Circuit symbol

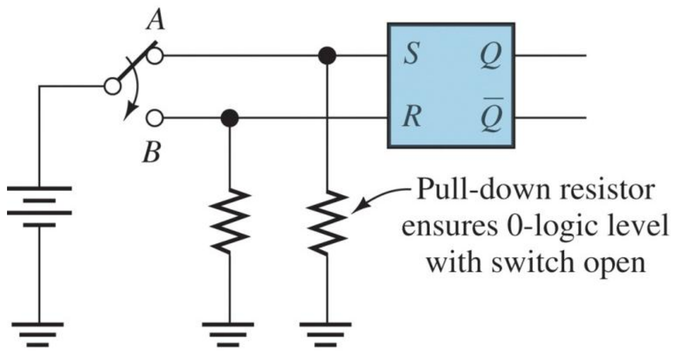
- The circuit remember what was the last input

- Q_n - represents the output at a time n
- Q_{n-1} - represents the output at an earlier time $n - 1$
  - Time $n - 1$ is the time when either S or R was last high

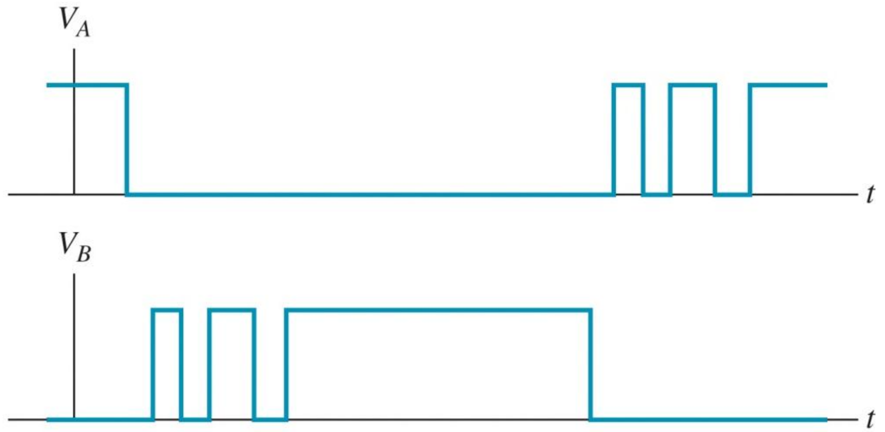
- Notice that set and reset inputs are not allowed to be high at the same time
  - Circuit cannot operate in this condition

Debouncing a Switch

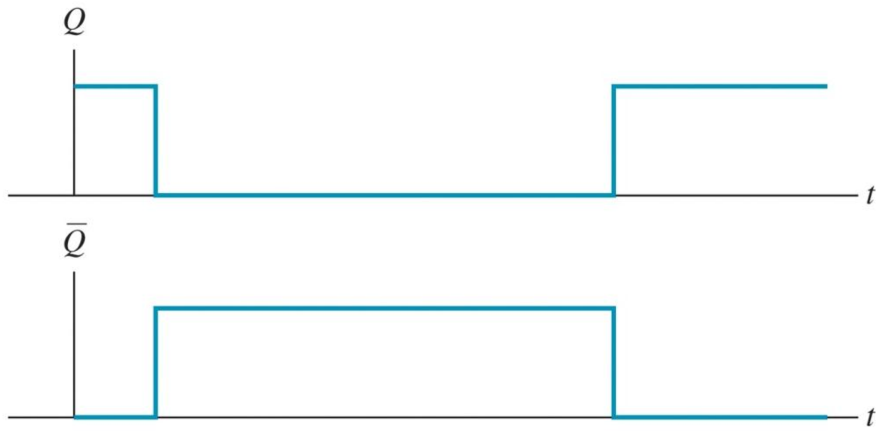
- Physical switches do not make perfect contact when moving from one position to another
  - The voltage across the switch is said to bounce
- An SR flip-flop can be used to debounce the switch
  - Make a "good" contact and prevent voltage from bounching



(a) Circuit diagram



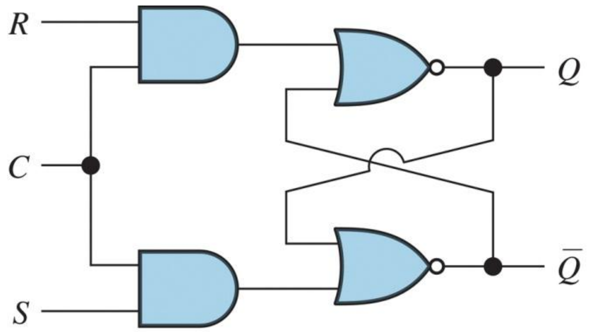
- Debounced output



(b) Waveforms

Clocked SR Flip-Flop

- The previous SR flip-flop state changes as soon as either *S* or *R* is set
- Often we would like to have control on when the state is allowed to change
  - Want a synchronous circuit
- Add a clock signal to the flip-flop
  - Only when the clock is high s the output allowed to change

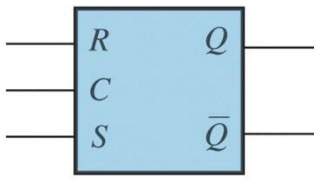


(a) Circuit diagram

- AND gates prevent *R* or *S* from reaching the flip-flop unless the clock *C* is high

R	S	C	Q_n
0	0	×	Q_{n-1}
0	1	1	1
1	0	1	0
1	1	1	Not allowed
×	×	0	Q_{n-1}

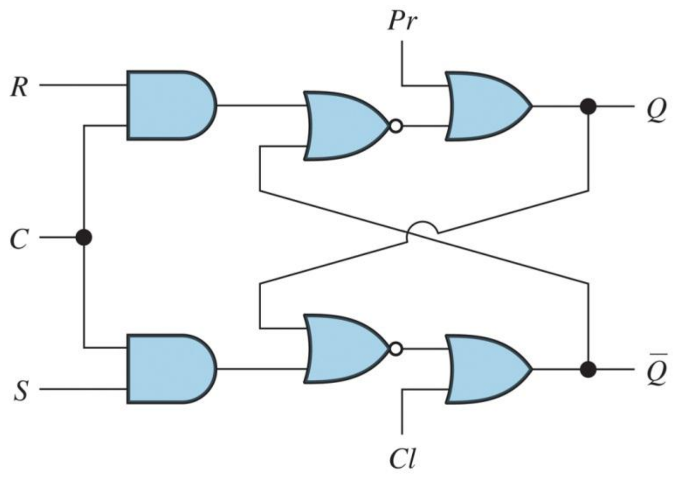
(b) Truth table



(c) Circuit symbol

Clocked SR with Asynchronous Input

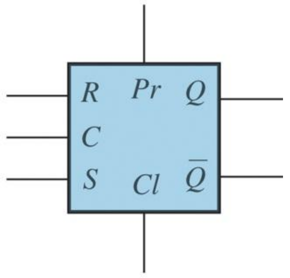
- Clocked set and reset functionality with asynchronous set and reset as well



(a) Circuit diagram

Pr	Cl	R	S	C	Q_n
0	0	0	0	×	Q_{n-1}
0	0	0	1	1	1
0	0	1	0	1	0
×	×	1	1	1	Not allowed
0	1	×	×	×	0
1	0	×	×	×	1
1	1	×	×	×	Not allowed

(b) Truth table

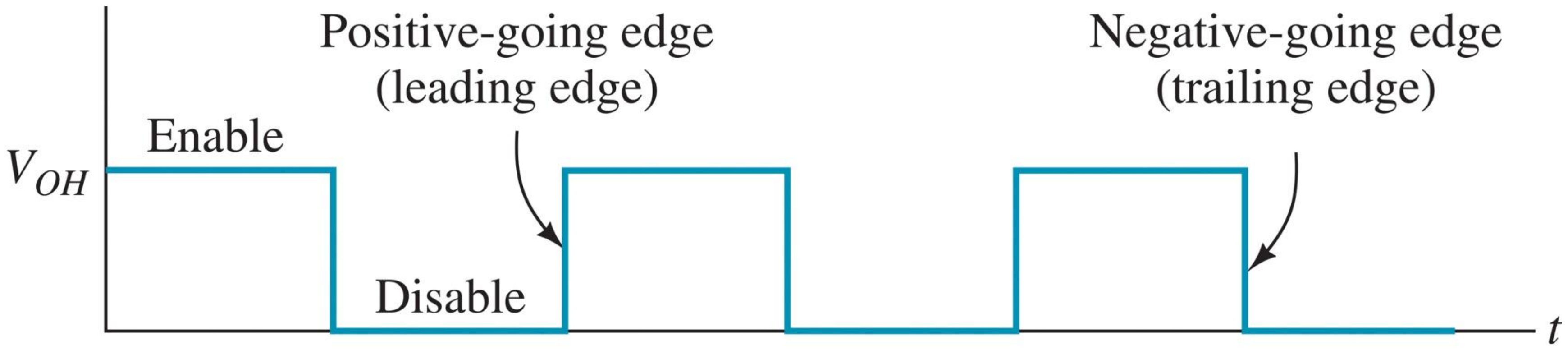


(c) Circuit symbol

- Add OR gates at $Q\bar{Q}$ outputs to automatically set or reset
  - Notice that the clocked S and R cannot be high at the same time and neither can the asynchronous preset Pr and clear Cl

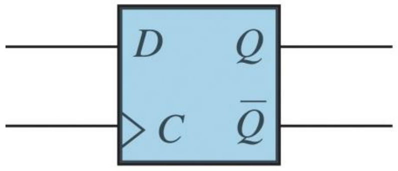
Edge-Triggered Circuits

- The clocked SR flip-flop uses the clock signal as an enable signal
  - When the clock is high the circuit is allowed to change
- Edge-triggered circuits only respond at the time when the clock changes between low and high
  - Positive-edge-triggered – low to high transition
    - Known as the leading edge
  - Negative-edge-triggered – high to low transition
    - Known as the training edge



D Flip-Flop

- The delay (D) flip-flop is edge-triggered to take make the output the same as the input right before the clock transition



(a) Circuit symbol

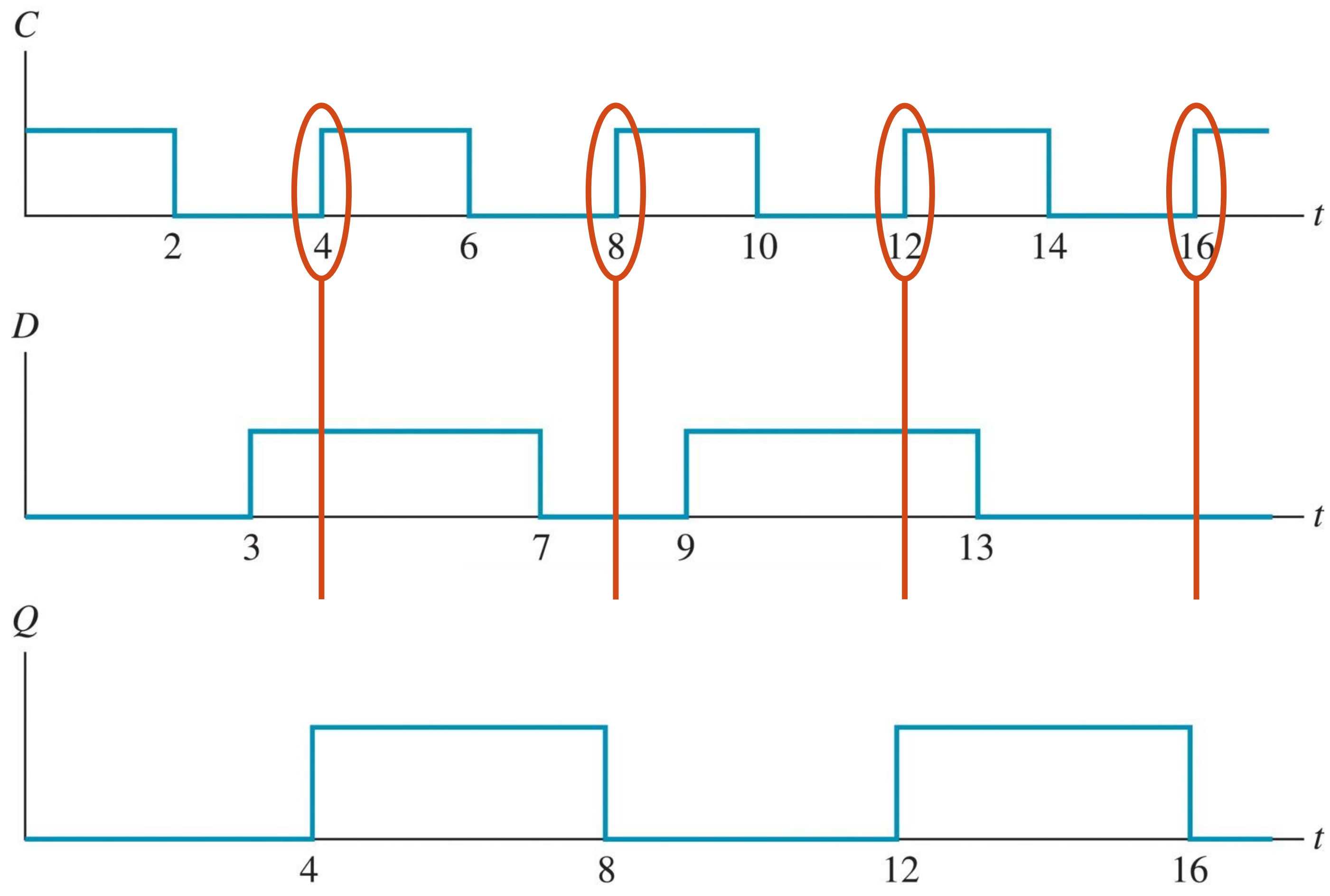
C	D	Q_n
0	$\times$	Q_{n-1}
1	$\times$	Q_{n-1}
$\uparrow$	0	0
$\uparrow$	1	1

(b) Truth table
indicates a transition
from low to high

- The triangle by the clock signal C indicates it is positive-edge-triggered
  - Up arrow in truth table indicates rising edge

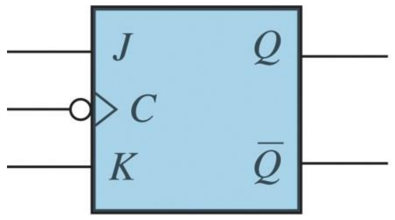
Example: D Flip-Flop

- Positive-edge-triggered



JK Flip-Flop

- Similar operation to the SR flip-flop
  - Except when *J* and *K* are both high, the output state *Q* will toggle



(a) Circuit symbol

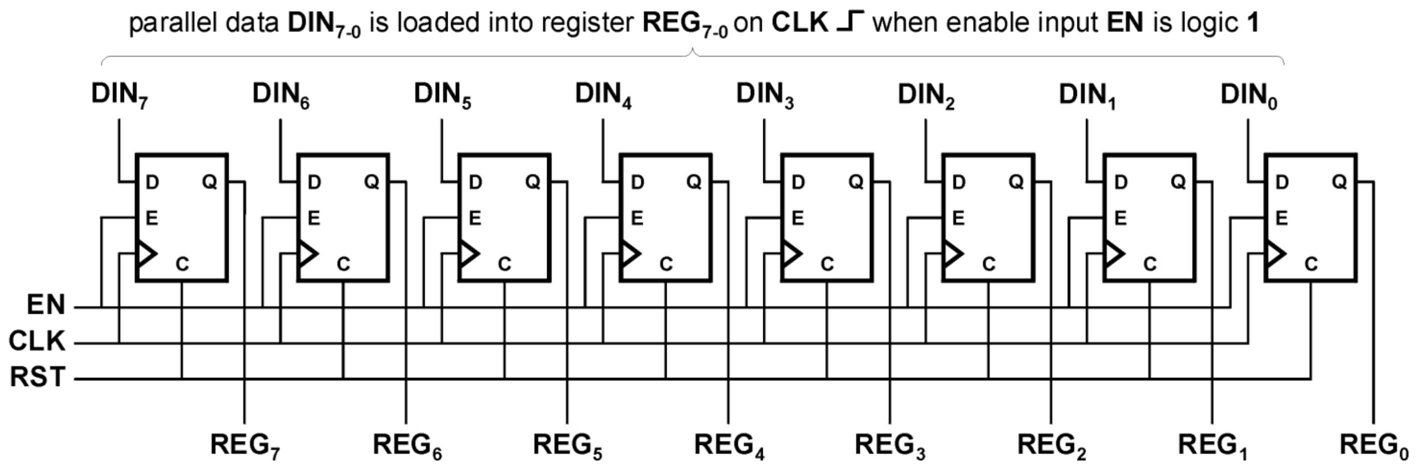
C	J	K	Q_n	Comment
0	×	×	Q_{n-1}	Memory
1	×	×	Q_{n-1}	Memory
↓	0	0	Q_{n-1}	Memory
↓	0	1	0	Reset
↓	1	0	1	Set
↓	1	1	$\overline{Q}_{n-1}$	Toggle

(b) Truth table
indicates a transition
from low to high

- Notice this is a negative-edge-triggered
  - Triangle with a preceding invert bubble

Registers

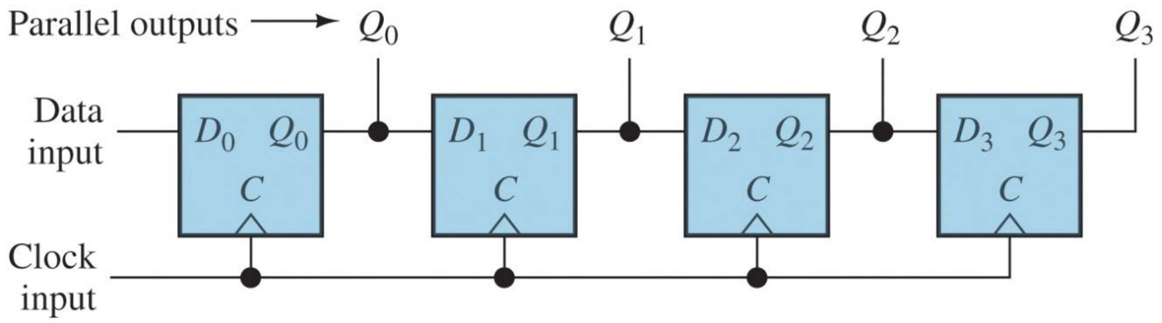
- A flip-flop is able to store a single bit
- A register is an array of flip-flops used to store a digital word
    - A hexadecimal number requires 4 bits so 4 flip-flops are required to internally store the hex number



http://enpub.fulton.asu.edu/cse517/Lab3.html

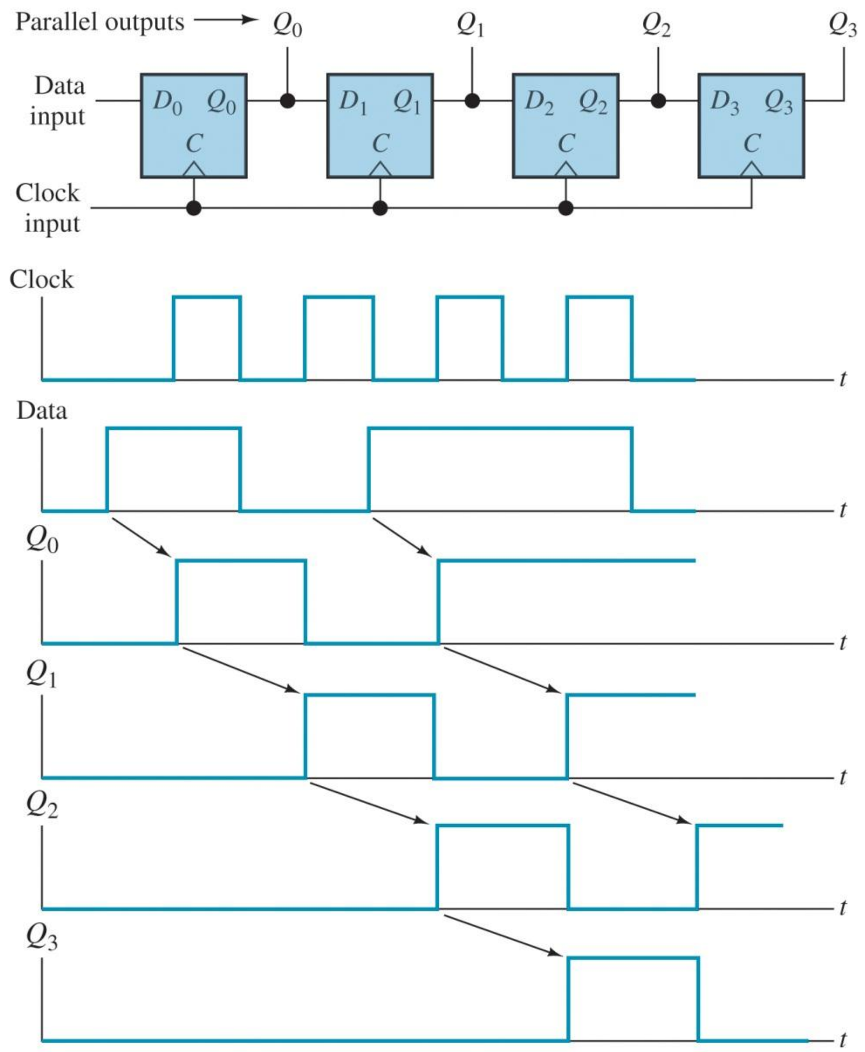
Serial-In Parallel-Out Shift Register

- Serial-in – implies bit are presented to the register one at a time (in a sequence)
- Parallel-out – implies the contents of the register (all the bits) can be accessed at the same time



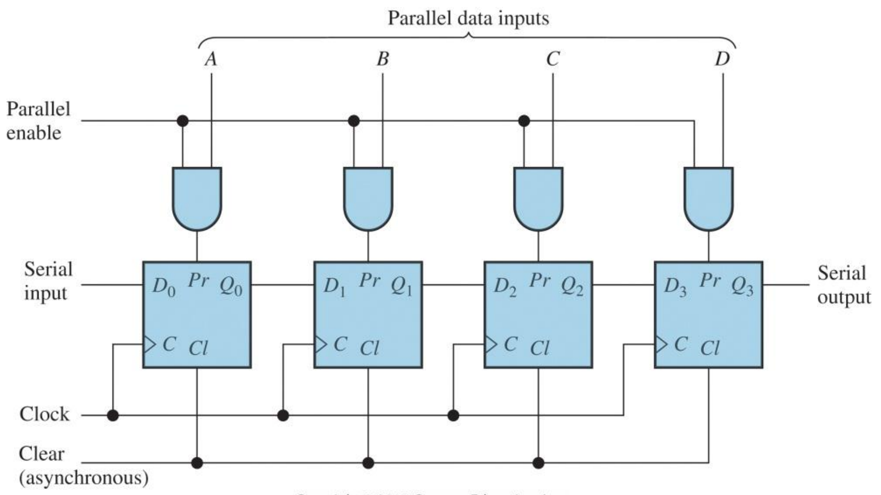
Serial-In Parallel-Out Operation

- Data is presented to a single input at the “front” of the word
- At each clock transition the data is shifted from one flip-flop to the next
- After 4 clock cycles, the full word is available to be read in the register



Parallel-In Serial Out Shift Register

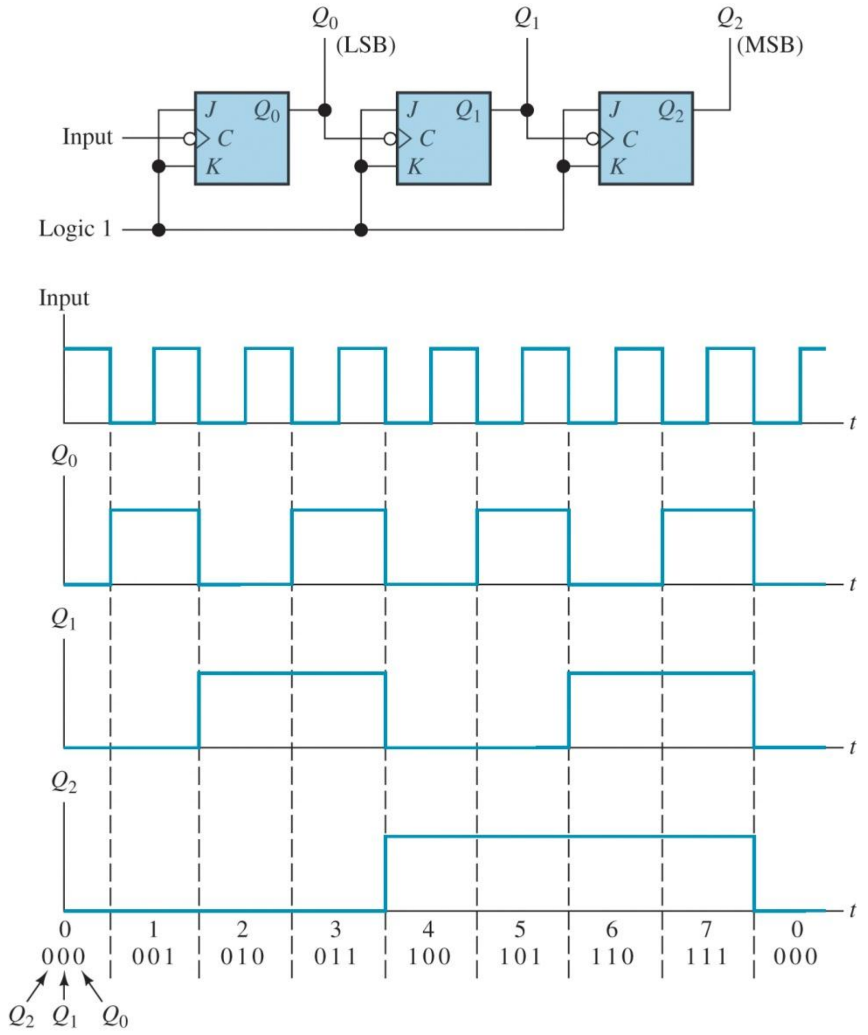
- Parallel data is provided as input and the output is serial data
  - E.g. transmission of data on a telephone line
- The register can be cleared asynchronously to initialize the register
- Data is set on inputs and a parallel enable signal asynchronously loads the data
- The output is serially transmitted from the last flip-flop stage



Counters

- Circuit used to count the pulses in an input signal
  - Often the number of clocks are the signal of interest

- Inputs of JK flip-flop are tied together and high
  - Causes the output state to toggle with each clock cycle
  - Notice this is negative-edge-triggered
- Output of one JK flip-flop is the clock input of the next stage
  - The word $Q_2Q_1Q_0$ is the binary representation of the count



Minimization of Logic Circuits

- We saw you can minimize logic variable representations using Boolean algebra but this can be tedious and prone to error

- The Karnaugh map (K-map) (Ch 7.5) is a principled method for determining the minimum representation
  - This method is usually only practical for 4 variables (maybe 5 or 6)
  - We will not cover this in class but it is an interesting read