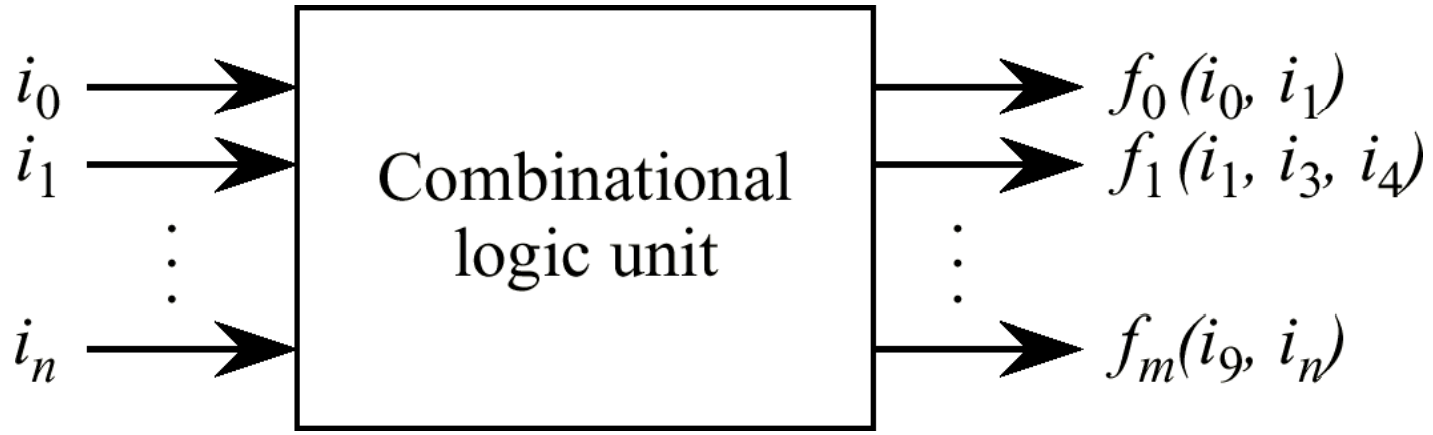
Computer Architecture

Digital Logic

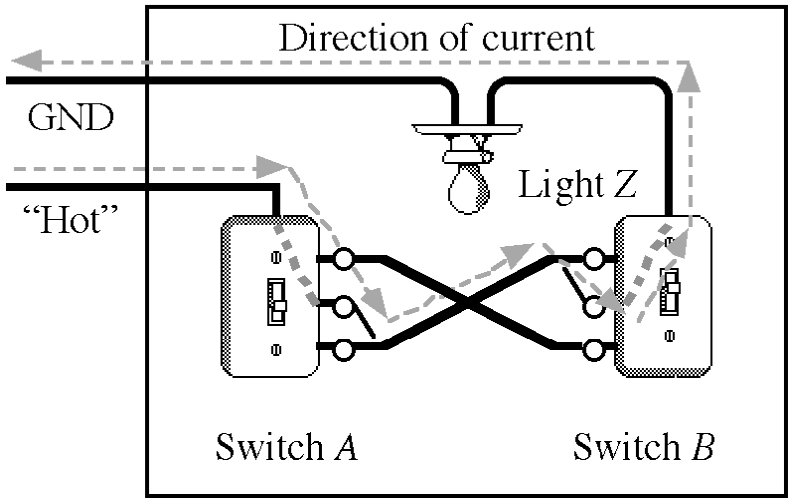
ARASH HABIBI LASHKARI
(April- 2010)

The Combinational Logic Unit



Truth Tables

Consider a room with two light switches. How must they work†?



Inputs		Output
A	*B*	*Z*
0	0	0
0	1	1
1	0	1
1	1	0

†Don't show this to your electrician, or wire your house this way. This circuit definitely violates the electric code. The practical circuit never leaves the lines to the light "hot" when the light is turned off. Can you figure how?

Alternate Assignments of Outputs to Switch Settings

- Logically identical truth table to the original, if the switches are configured up-side down.

Inputs		Output
A	*B*	*Z*
0	0	1
0	1	0
1	0	0
1	1	1

Truth Tables Showing All Possible Functions of Two Binary Variables

Inputs		Outputs							
A	B	*False*	*AND*	$A\overline{B}$	A	$\overline{A}B$	B	*XOR*	*OR*
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

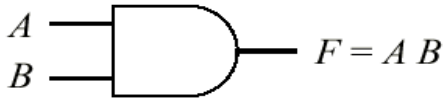
Inputs		Outputs							
A	B	*NOR*	*XNOR*	$\overline{B}$	$A+\overline{B}$	$\overline{A}$	$\overline{A}+B$	*NAND*	*True*
0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

- The more frequently used functions have names: AND, XOR, OR, NOR, XOR, and NAND. (Always use upper case spelling.)

Logic Gates and Their Symbols

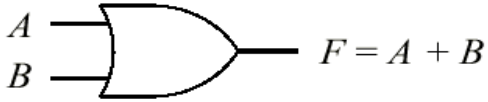
Logic symbols for AND, OR, buffer, and NOT Boolean functions

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1



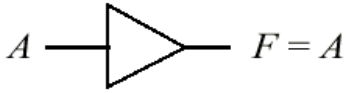
AND

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1



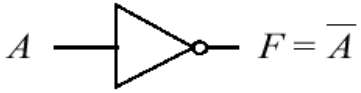
OR

A	F
0	0
1	1



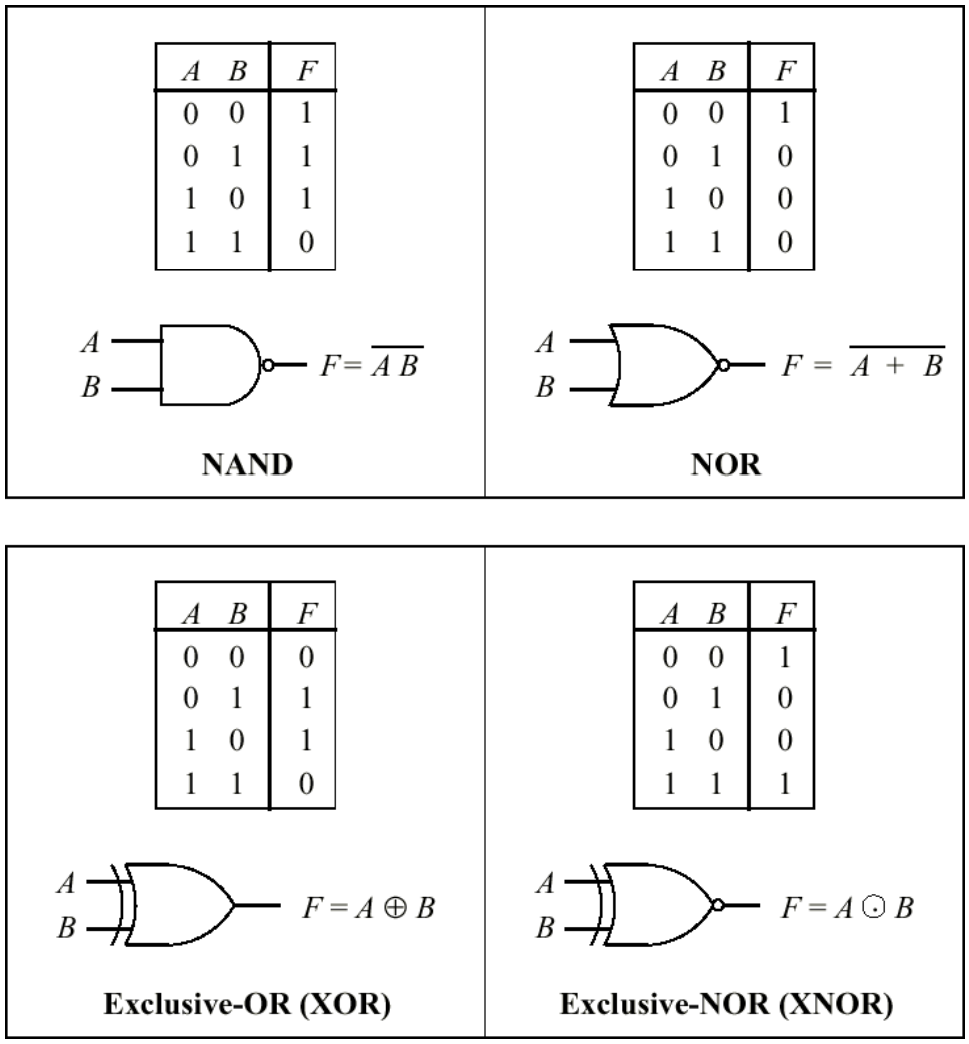
Buffer

A	F
0	1
1	0

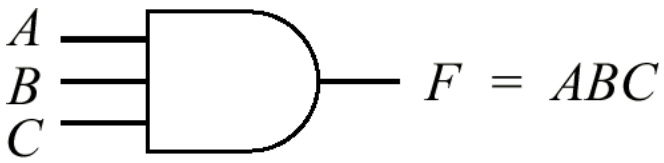


NOT (Inverter)

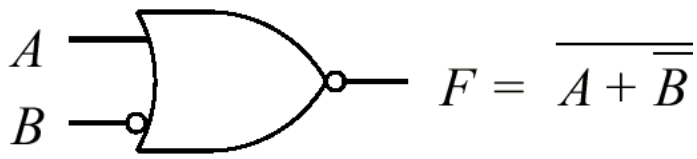
Logic symbols for NAND, NOR, XOR, and XNOR Boolean functions



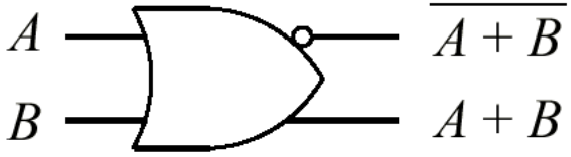
Variations of Basic Logic Gate Symbols



(a)



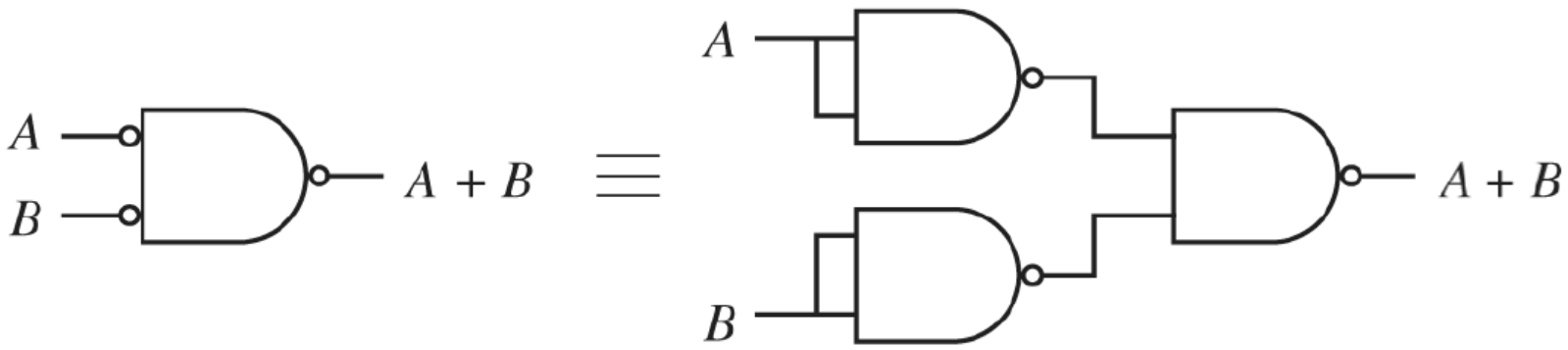
(b)



(c)

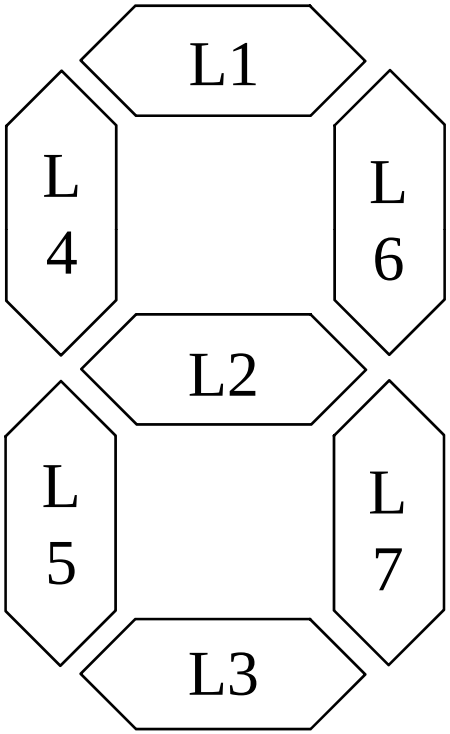
(a) 3 inputs **(b) A Negated input** **(c) Complementary outputs**

NAND Gates Can Implement AND and OR Gates



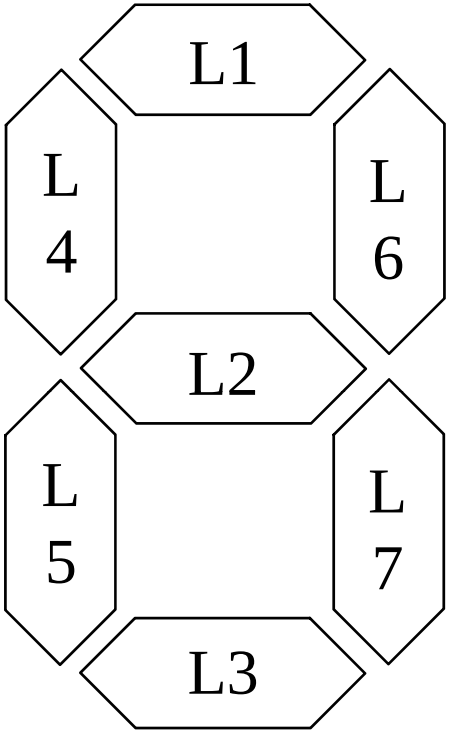
Inverted inputs to a NAND gate are implemented with NAND gates.

Example: Seven Segment Display

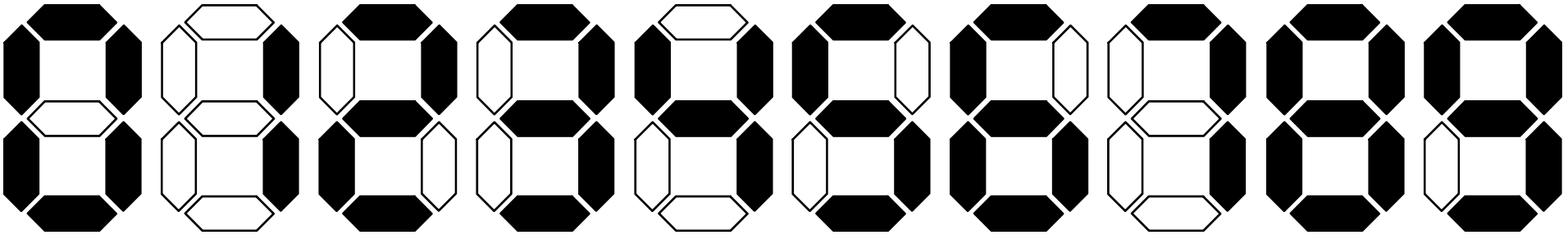


B3	B2	B1	B0	Val
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9

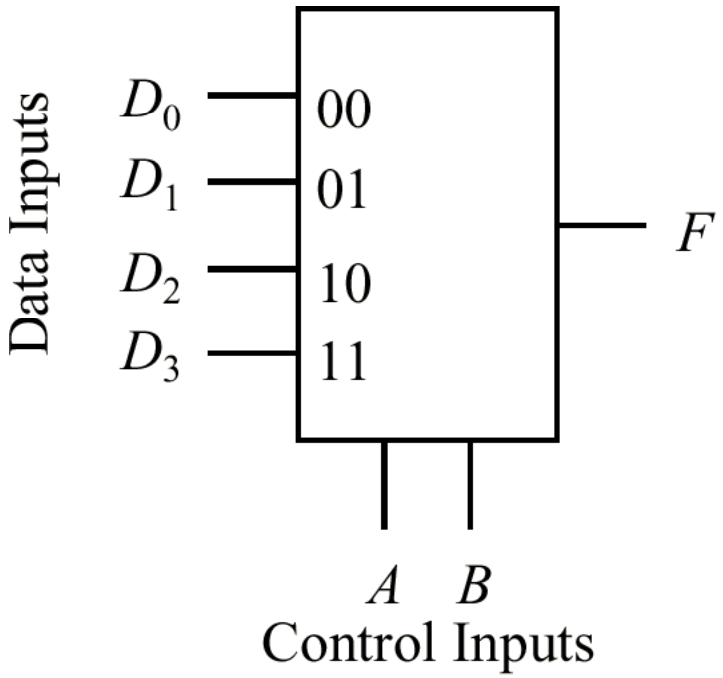
Example (cont.)



B3	B2	B1	B0	Val	L1	L2	L3	L4	L5	L6	L7
0	0	0	0	0	1	0	1	1	1	1	1
0	0	0	1	1	0	0	0	0	0	1	1
0	0	1	0	2	1	1	1	0	1	1	0
0	0	1	1	3	1	1	1	0	0	1	1
0	1	0	0	4	0	1	0	1	0	1	1
0	1	0	1	5	1	1	1	1	0	0	1
0	1	1	0	6	1	1	1	1	1	0	1
0	1	1	1	7	1	0	0	0	0	1	1
1	0	0	0	8	1	1	1	1	1	1	1
1	0	0	1	9	1	1	1	1	0	1	1



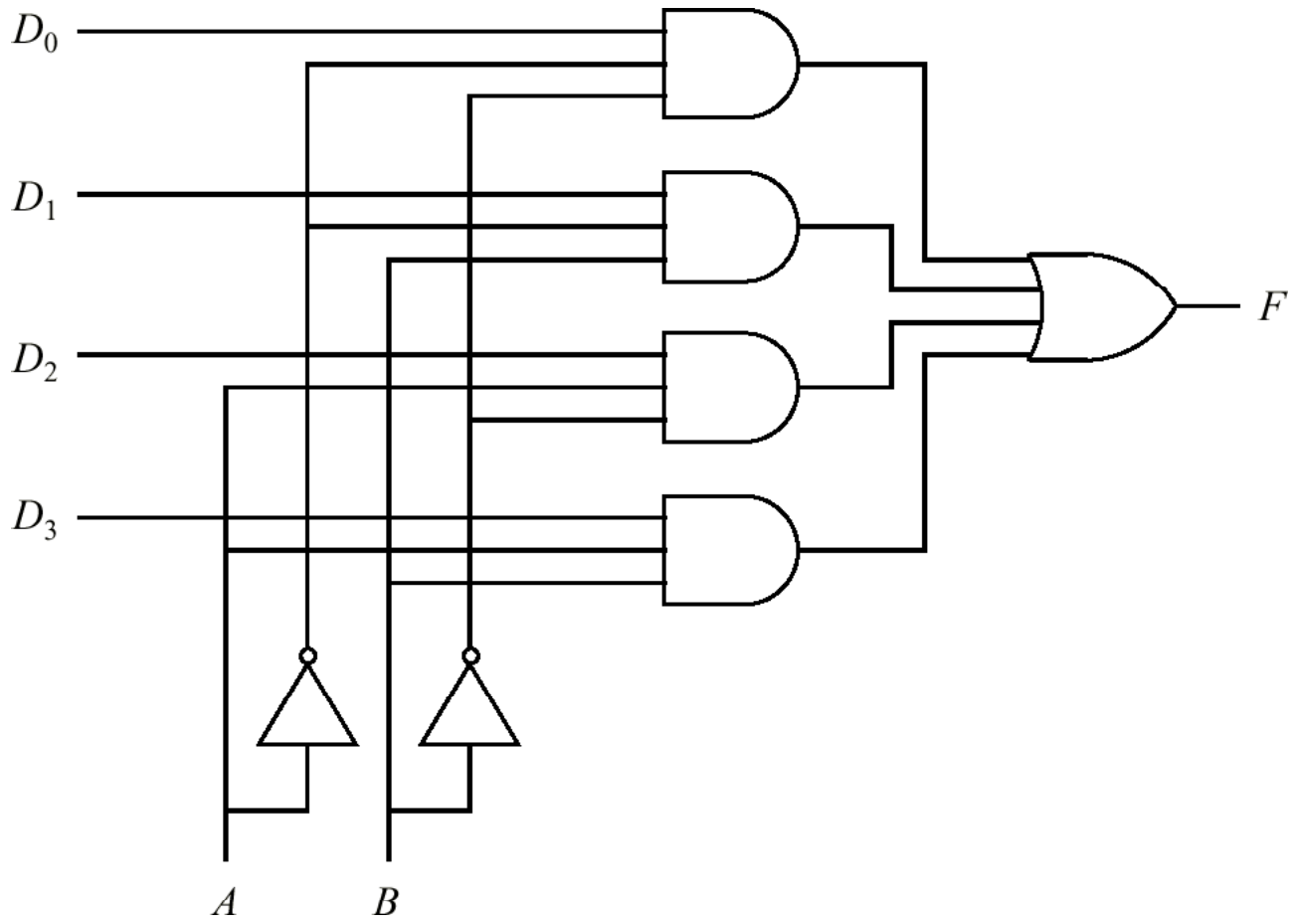
The Multiplexer



A	B	F
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

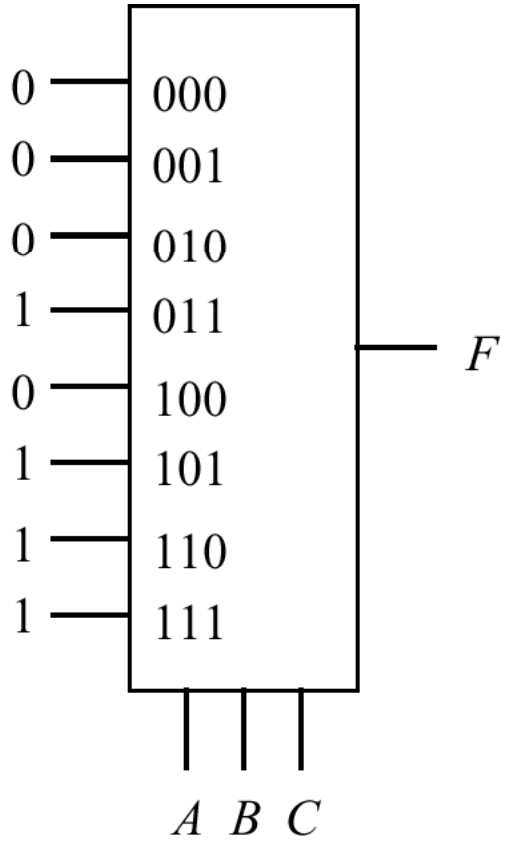
$$F = \overline{A}\,\overline{B}\,D_0 + \overline{A}\,B\,D_1 + A\,\overline{B}\,D_2 + A\,B\,D_3$$

Gate-Level Layout of Multiplexer



Implementing the Majority Function with an 8-1 Mux

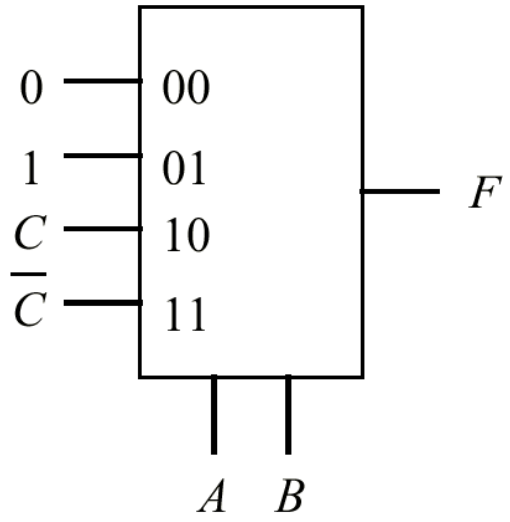
A	B	C	M
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



Principle: Use the mux select to pick out the selected minterms of the function.

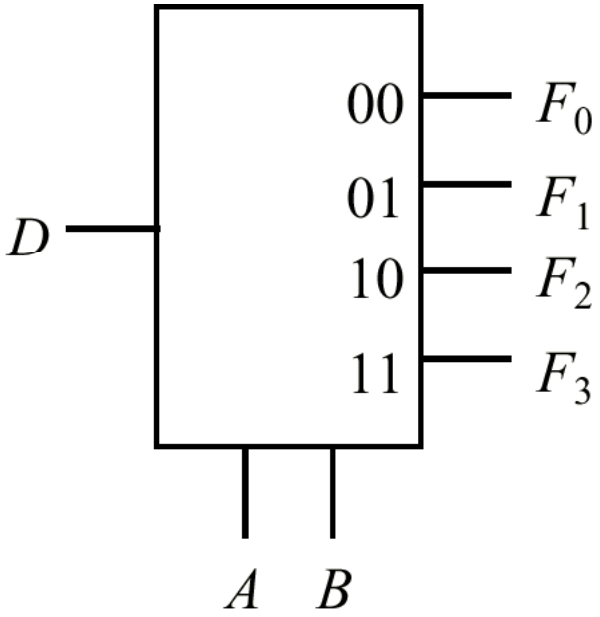
Efficiency: Using a 4-1 Mux to Implement the Majority Function

A	B	C	F	
0	0	0	0	0
0	0	1	0	
0	1	0	1	1
0	1	1	1	
1	0	0	0	C
1	0	1	1	
1	1	0	1	$\overline{C}$
1	1	1	0	



Principle: Use the A and B inputs to select a pair of minterms. The value applied to the MUX input is selected from {0, 1, C, C} to pick the desired behavior of the minterm pair.

The Demultiplexer (DEMUX)

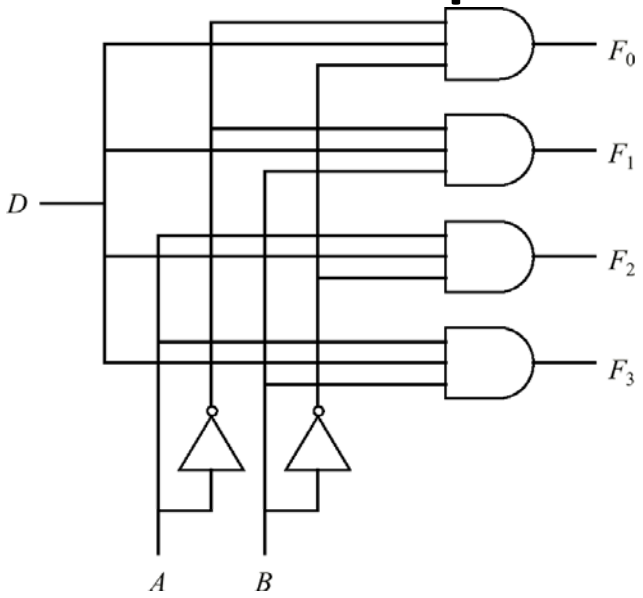


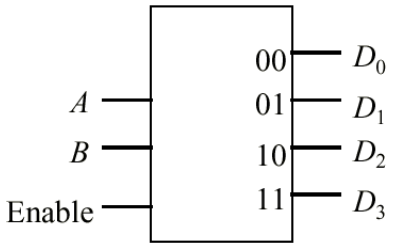
$$F_0 = D\,\overline{A}\,\overline{B} \qquad F_2 = D\,A\,\overline{B}$$

$$F_1 = D\,\overline{A}\,B \qquad F_3 = D\,A\,B$$

D	A	B	F_0	F_1	F_2	F_3
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

The Demultiplexer is a Decoder with an Enable Input



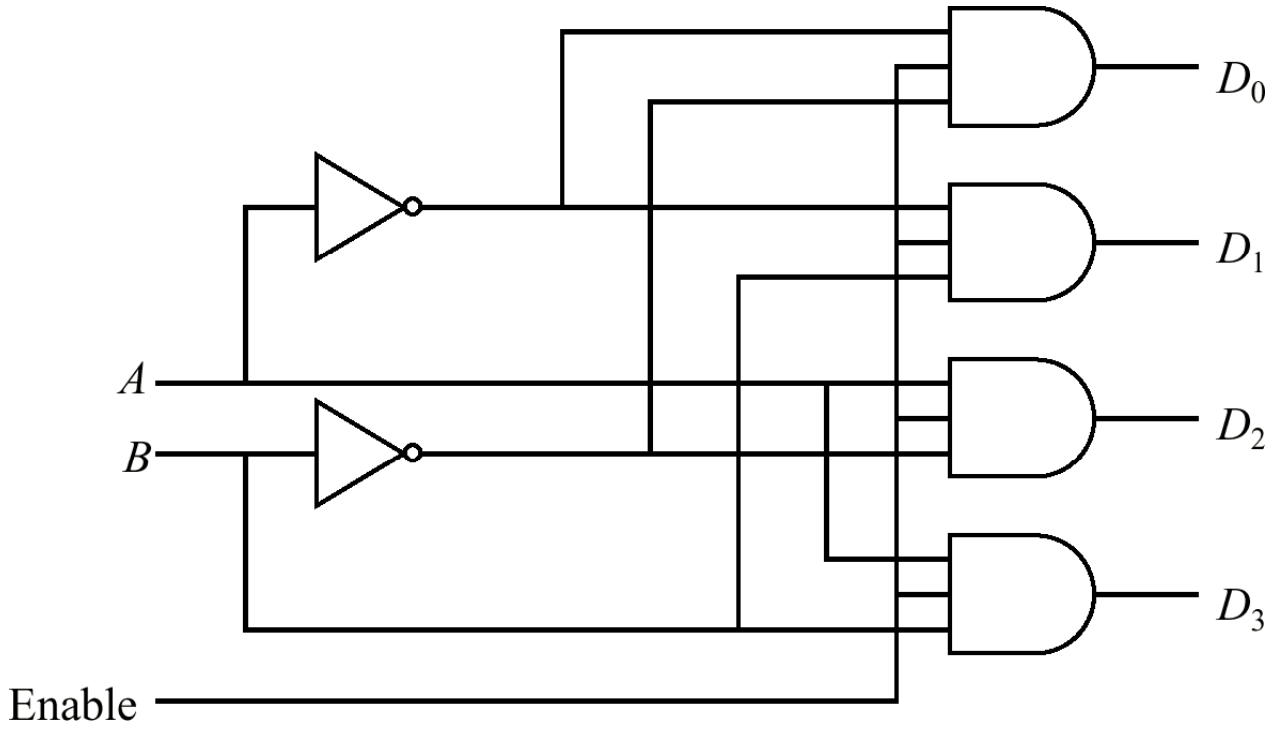


Enable = 1						
A	B	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Enable = 0						
A	B	D_0	D_1	D_2	D_3
0	0	0	0	0	0
0	1	0	0	0	0
1	0	0	0	0	0
1	1	0	0	0	0

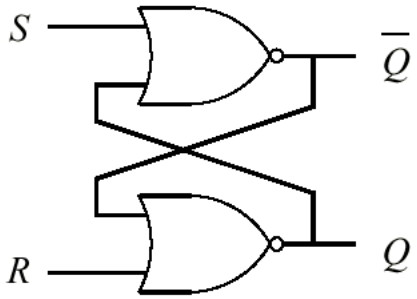
$D_0 = \overline{A}\,\overline{B}$ $D_1 = \overline{A}\,B$ $D_2 = A\,\overline{B}$ $D_3 = A\,B$

A 2-to-4 Decoder

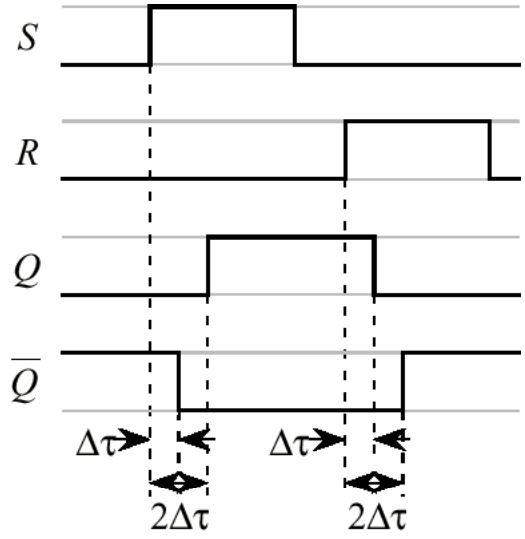


S-R Flip-Flop

- The S-R flip-flop is an active high (positive logic) device.

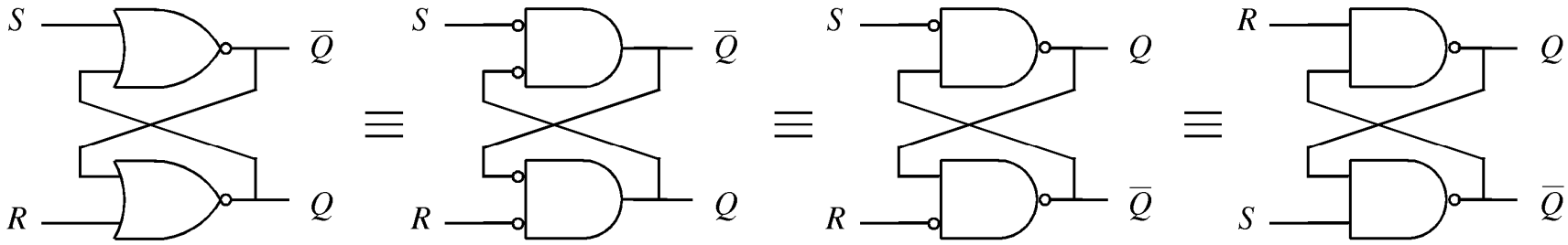


Q_t	S_t	R_t	Q_{i+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	(disallowed)
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	(disallowed)



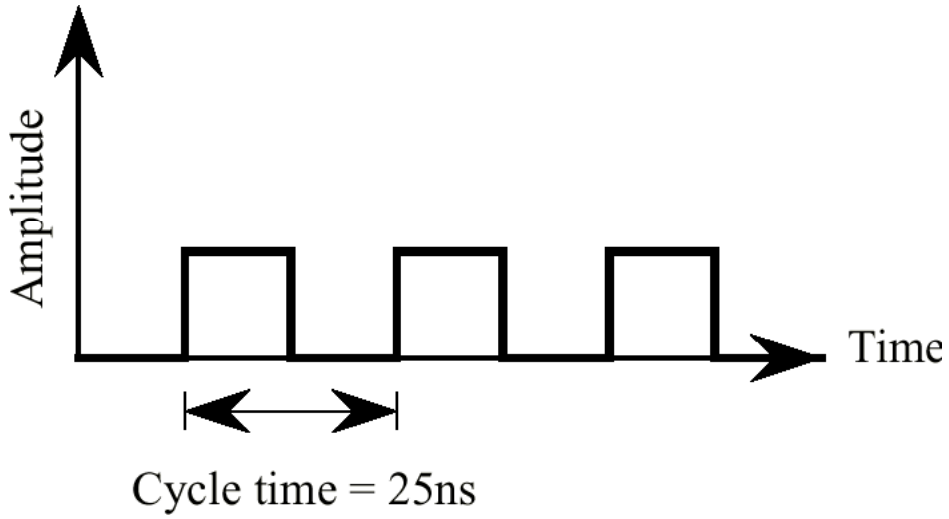
Timing Behavior

NAND Implementation of S-R Flip-Flop



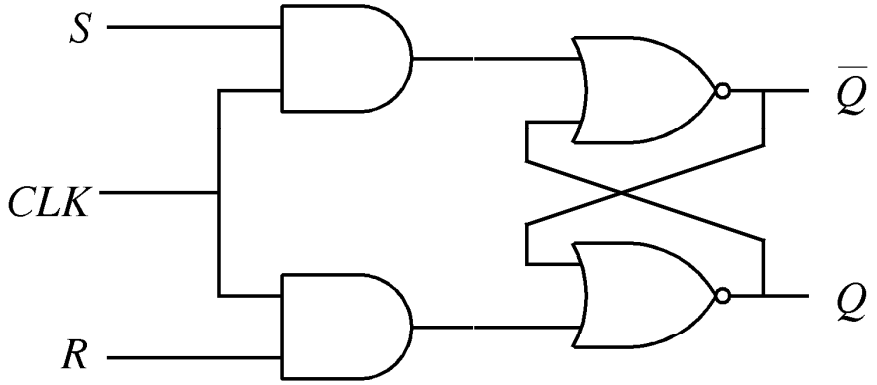
- A NOR implementation of an S-R flip-flop is converted into a NAND implementation.

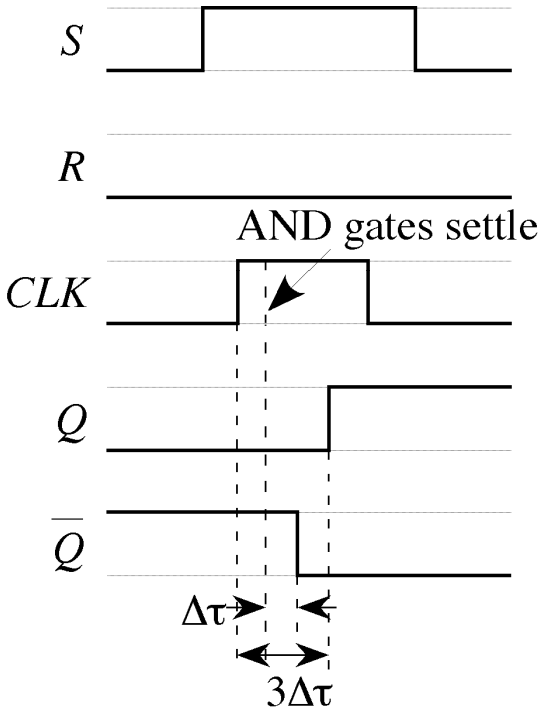
A Clock Waveform: The Clock Paces the System



- In a positive logic system, the “action” happens when the clock is high, or positive. The low part of the clock cycle allows propagation between subcircuits, so that the signals settle at their correct values when the clock next goes high.

Clocked S-R Flip-Flop

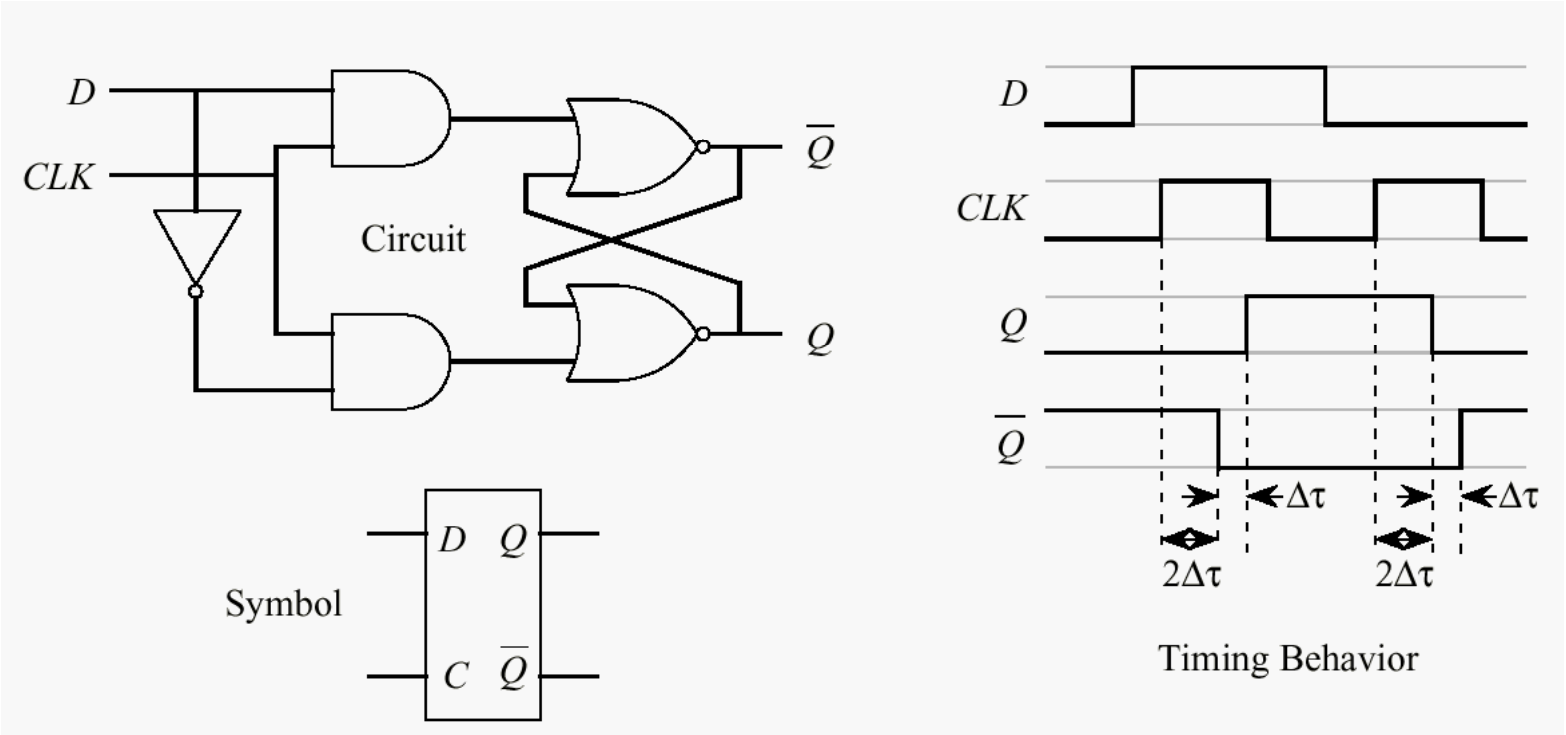




Timing Behavior

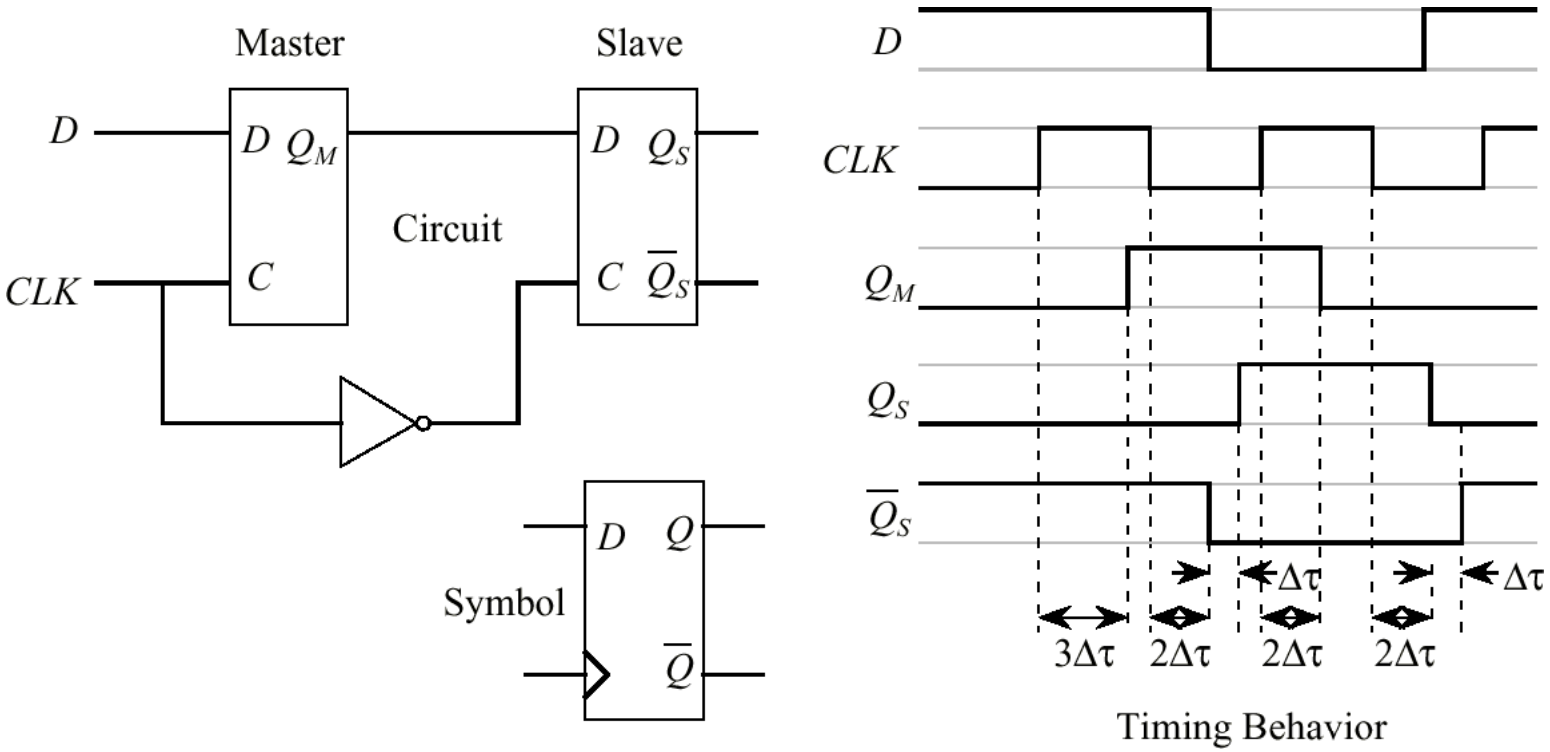
- The clock signal, CLK, enables the S and R inputs to the flip-flop.

Clocked D Flip-Flop



- The clocked D flip-flop, sometimes called a latch, has a potential problem: If D changes while the clock is high, the output will also change. The Master-Slave flip-flop (next slide) addresses this problem.

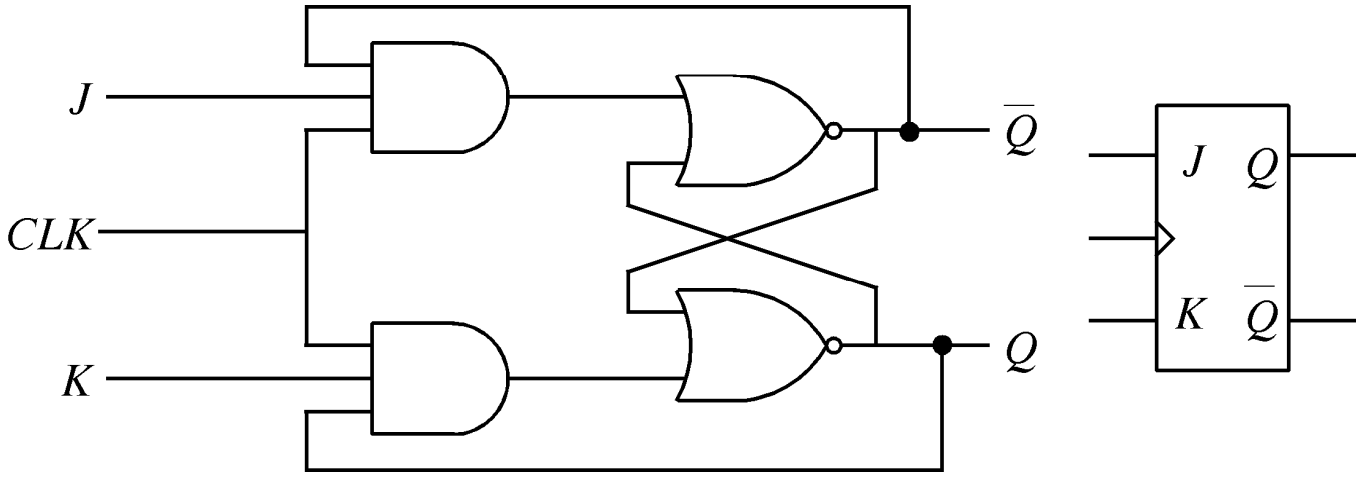
Master-Slave Flip-Flop



- The rising edge of the clock loads new data into the master, while the slave continues to hold previous data. The falling edge of the clock loads the new master data into the slave.

Clocked J-K Flip-Flop

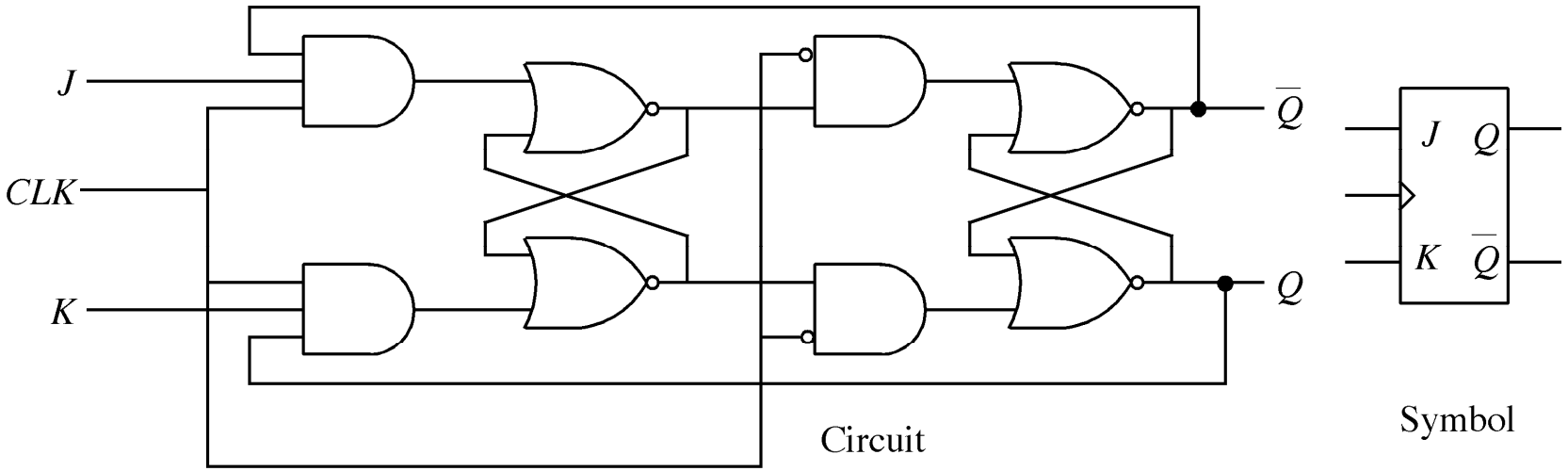
- The J-K flip-flop eliminates the disallowed S=R=1 problem of the S-R flip-flop, because Q enables J while Q' disables K, and vice-versa.
- However, there is still a problem. If J goes momentarily to 1 and then back to 0 while the flip-flop is active and in the reset state, the flip-flop will "catch" the 1. This is referred to as "1's catching."
- The J-K Master-Slave flip-flop (next slide) addresses this problem.



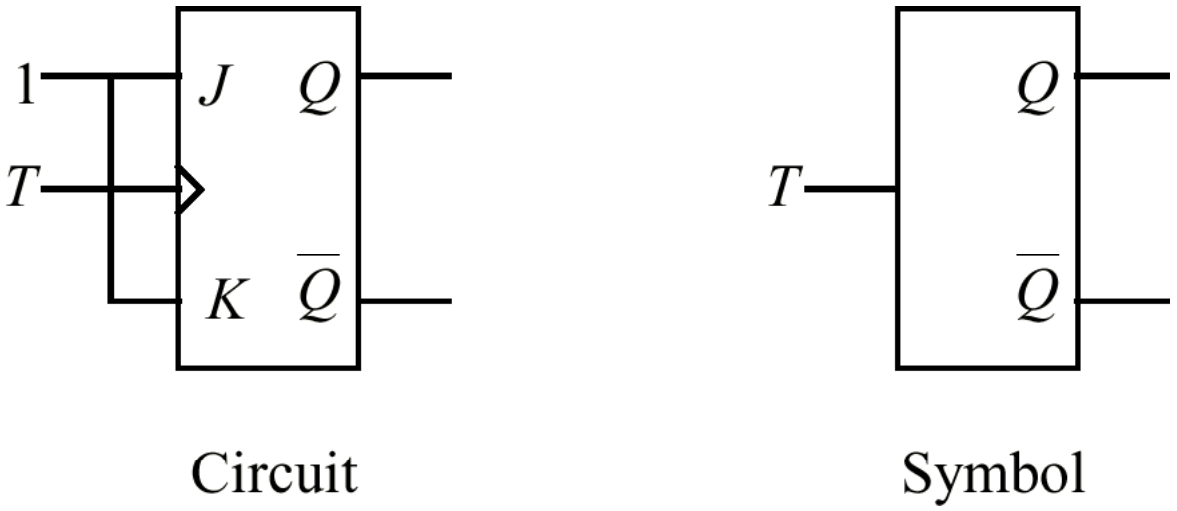
Circuit

Symbol

Master-Slave J-K Flip-Flop



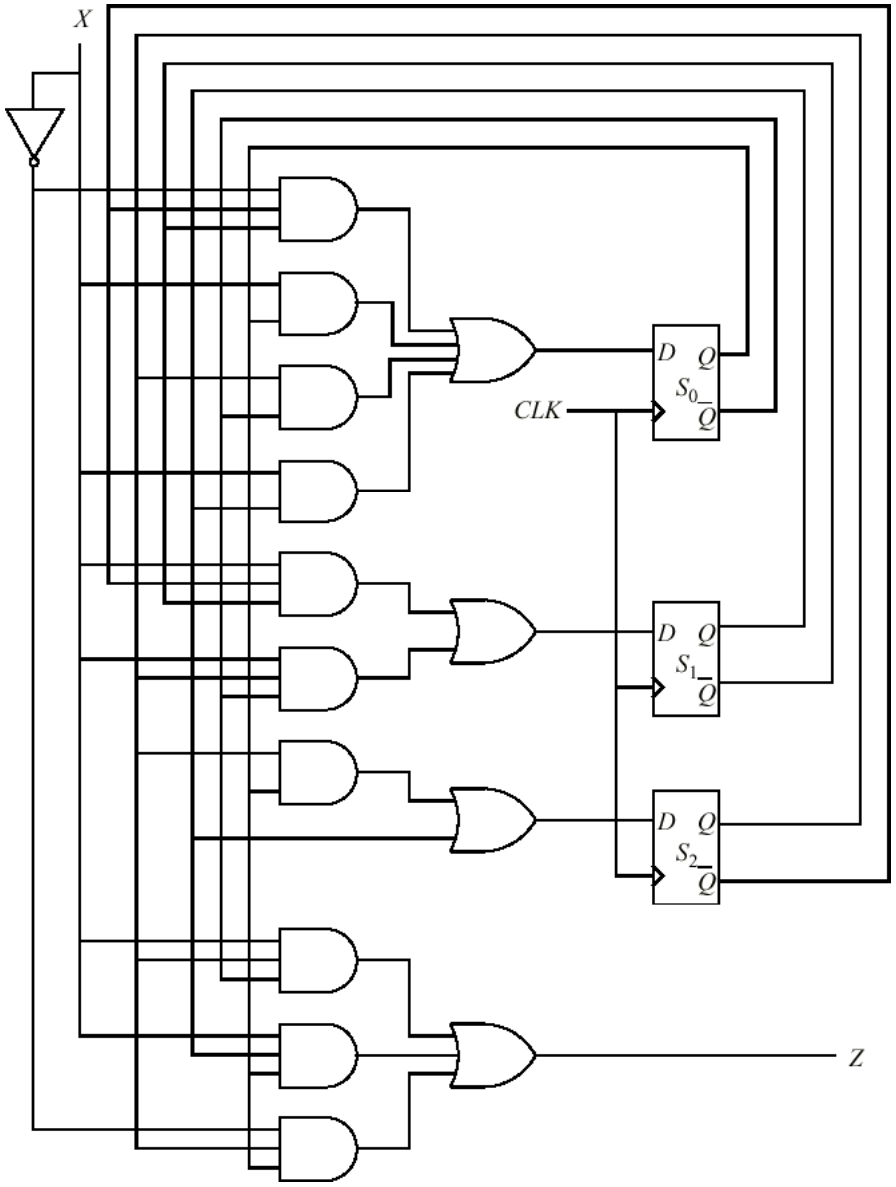
Clocked T Flip-Flop



Circuit Symbol

- The presence of a constant 1 at J and K means that the flip-flop will change its state from 0 to 1 or 1 to 0 each time it is clocked by the T (Toggle) input.

Sequence Detector Circuit



Questions