Computer Architecture & Assembly Language

Chapter 3

DIGITAL CIRCUIT DESIGN

VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY
UNIVERSITY OF SCIENCE

Thái Hùng Văn
thvan@fit.hcmus.edu.vn

Objectives

- Understand the relationship between Boolean logic and digital computer circuits.
- Learn how to design simple logic circuits.
- Understand how digital circuits work together to form complex computer systems.

Outline

- Boolean Algebra
- Logic Gates
- Combinational Logic Circuits
- Sequential Logic Circuits
- Designing Circuits from Truth Table

Boolean Algebra

Introduction

- Computers, as we know them today, are implementations of **Boole**'s *Laws of Thought*.
- Boolean algebra is a mathematical system for the manipulation of variables that can have 1 of 2 values.
  - In formal logic, these values are "**true**" and "**false**."
  - In digital systems, these values are "**on**" and "**off**," **1** and **0**, or "high" and "low."
- Boolean expressions are created by performing operations on Boolean variables. Common Boolean operators include **AND**, **OR**, and **NOT**.

Boolean operator & Truth table

- A Boolean operator can be completely described using a truth table.
- The AND operator is also known as a Boolean product. The OR operator is the Boolean sum.
- The NOT operation is most often designated by an overbar. It is sometimes indicated by (~) or (¬)

X AND Y

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

X OR Y

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT X

X	$\overline{X}$
0	1
1	0

Boolean function & Truth table

- A Boolean function (BF) has:
  - At least one Boolean variable,
  - At least one Boolean operator, and
  - At least one input from the set {0,1}.
- It produces an output that is 0 or 1.
- The truth table for the BF:

$F(x,y,z) = x\bar{z}+y$

x	y	z	$\bar{z}$	$x\bar{z}$	$x\bar{z}+y$
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	0	1

- The NOT operator has highest priority, followed by AND and then OR

To make evaluation of the BF easier, the truth table contains extra columns to hold evaluations of subparts of the BF.

Boolean Functions & Circuits

- Digital computers contain circuits that implement BFs.
- The simpler that we can make a BF, the smaller the circuit that will result.
  - Simpler circuits are cheaper to build, consume less power, and run faster than complex circuits.
- With this in mind, we always want to reduce our BFs to their simplest form.
- There are a number of Boolean identities that help us to do this.

Boolean Laws

Identity Name	AND Form	OR Form
Identity Law	$1x = x$	$0 + x = x$
Null Law	$0x = 0$	$1 + x = 1$
Idempotent Law	$xx = x$	$x + x = x$
Inverse Law	$x\overline{x} = 0$	$x + \overline{x} = 1$

Identity Name	AND Form	OR Form
Absorption Law	$x(x+y) = x$	$x + xy = x$
DeMorgan's Law	$\overline{(xy)} = \overline{x} + \overline{y}$	$\overline{(x+y)} = \overline{x}\,\overline{y}$
Double Complement Law	$\overline{(\overline{x})} = x$	

Identity Name	AND Form	OR Form
Commutative Law	$xy = yx$	$x+y = y+x$
Associative Law	$(xy)z = x(yz)$	$(x+y)+z = x + (y+z)$
Distributive Law	$x+yz = (x+y)(x+z)$	$x(y+z) = xy+xz$

Simplifying Boolean expressions

- There are many ways of stating the same Boolean expression (BE)
  - These "synonymous" forms are *logically equivalent.*
  - Logically equivalent expressions have identical truth tables.
- In order to eliminate as much confusion as possible, designers express BF in ***standardized*** or ***canonical*** form.
- There are 2 canonical forms for BEs: sum-of-products & product-of-sums.
  - Recall the Boolean product is the AND operation and the Boolean sum is the OR operation.
- In the sum-of-products form, ANDed variables are ORed together.
  - For example: $F(x,y,z) = xy+yz + zx$
- In the product-of-sums form, ORed variables are ANDed together:
  - For example: $F(x,y,z) = (x+y)(y+z)(z+x)$

Sum-of-products form

- It is easy to convert a BF to sum-of-products form using its truth table.
- We are interested in the values of the variables that make the function true (=1).
- Using the truth table, we list the values of the variables that result in a true function value.
- Each group of variables is then ORed together.
- *The sum-of-products form for the function in example is:*

$$F(x,y,z) = \bar{x}y\bar{z}+\bar{x}yz+x\bar{y}\bar{z}+xy\bar{z}+xyz$$

$F(x,y,z) = x\bar{z}+y$

x	y	z	$x\bar{z}+y$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

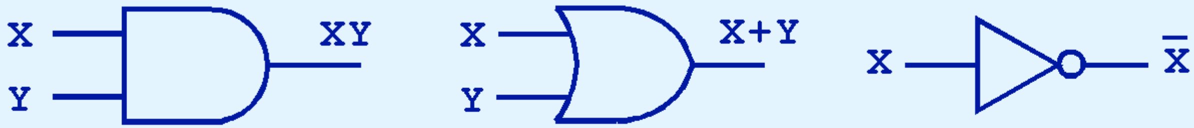
Logic Gates

Boolean Operators and Logic Gates

- In digital logic circuits, each Boolean operator can be represented by a logic gate. So, BFs can implement by logic gates.
- A gate is an electronic device that produces a result based on input values.
- A gate consists of 1 to 6 transistors, but when we design a digital circuit, think of it as a unit.
- Integrated circuits contain collections of gates suited to a particular purpose.

Basic Logic Gates

- The three simplest gates are the AND, OR, and NOT gates.



X AND Y

X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

X OR Y

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT X

X	$\overline{X}$
0	1
1	0

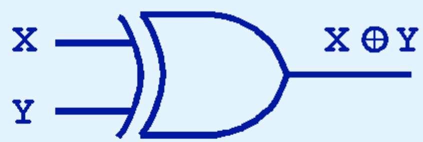
- They correspond directly to their respective Boolean operations, as you can see by their truth tables.

Logic Gates - XOR

- Another very useful gate is the exclusive OR (XOR) gate, with the special symbol ⊕ for the representation
- The output of the XOR operation is true only when the values of the inputs differ.

X XOR Y

X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

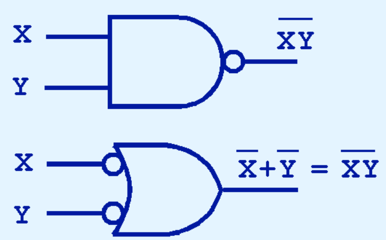


Logic Gates - NAND & NOR

NAND and NOR are very important gates, known as universal gates

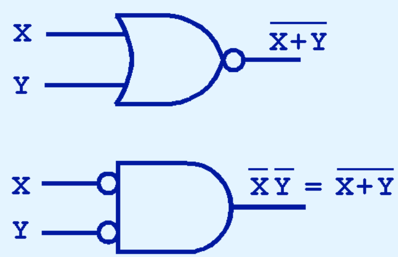
X NAND Y

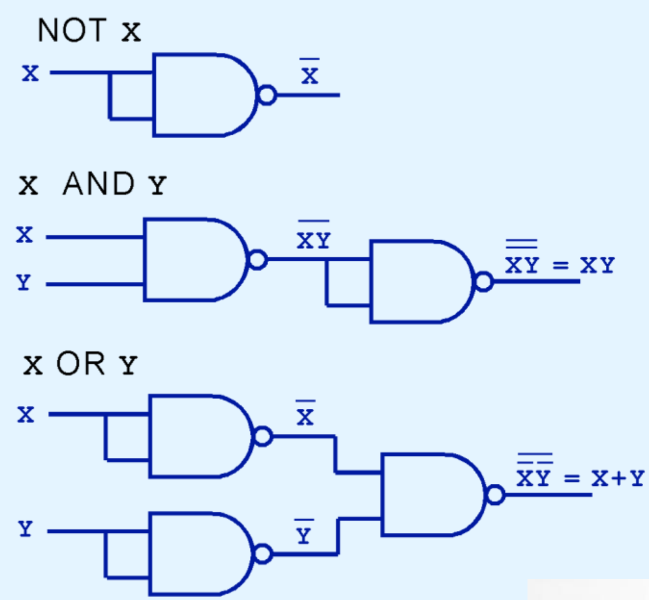
X	Y	X NAND Y
0	0	1
0	1	1
1	0	1
1	1	0

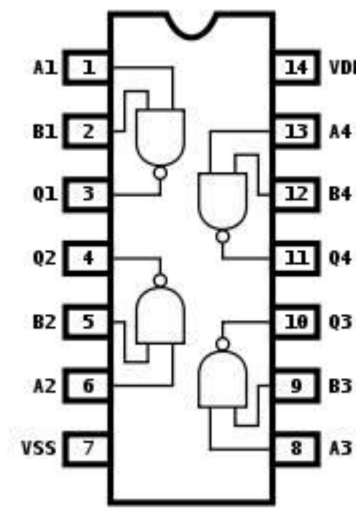


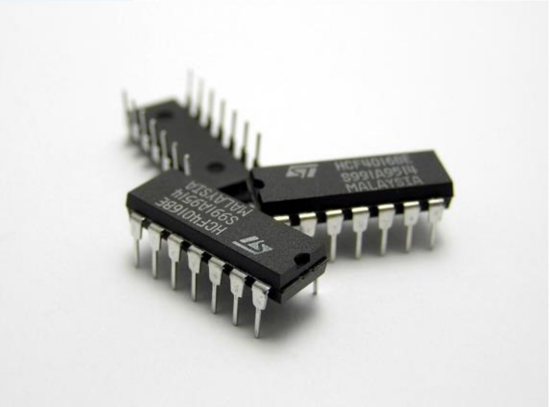
X NOR Y

X	Y	X NOR Y
0	0	1
0	1	0
1	0	0
1	1	0





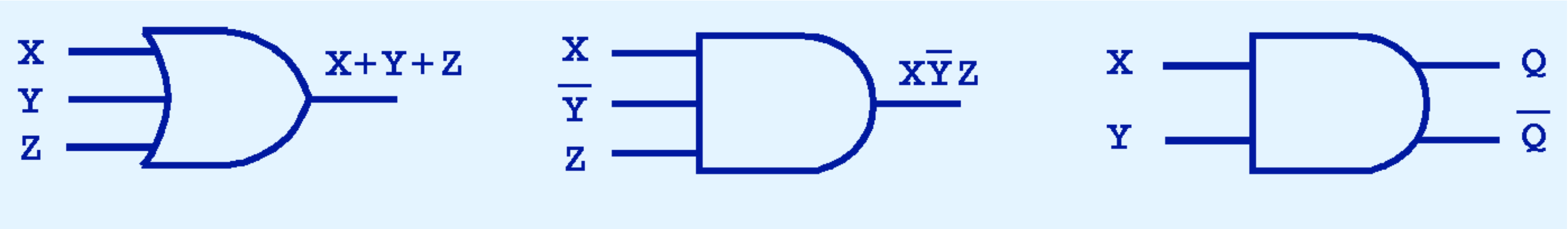




AND, OR gates: 3 transistors.
NAND, NOR gates: only 2 transistors!

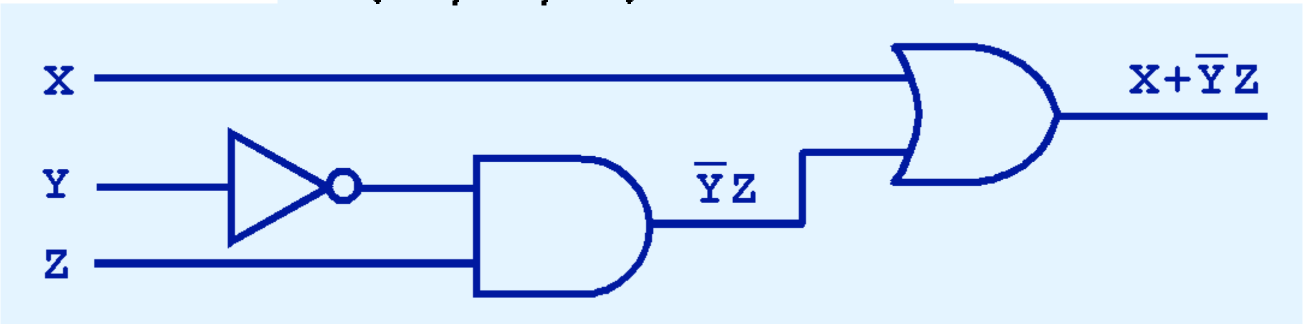
Expansion Gates and Simple Circuit

- Expansion gates can have multiple inputs and more than one output.



- The circuit below implements the BF:

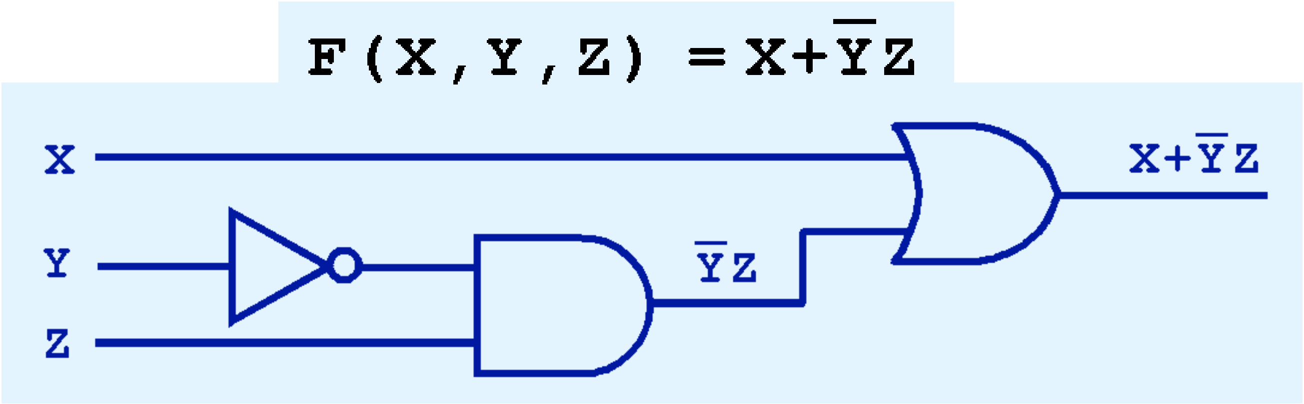
$$F(X,Y,Z) = X+\overline{Y}Z$$



Combinational Logic Circuits

Concepts

- The circuit that implements the above Boolean function F(x,y,z) is a *combinational logic* circuit (CLC)

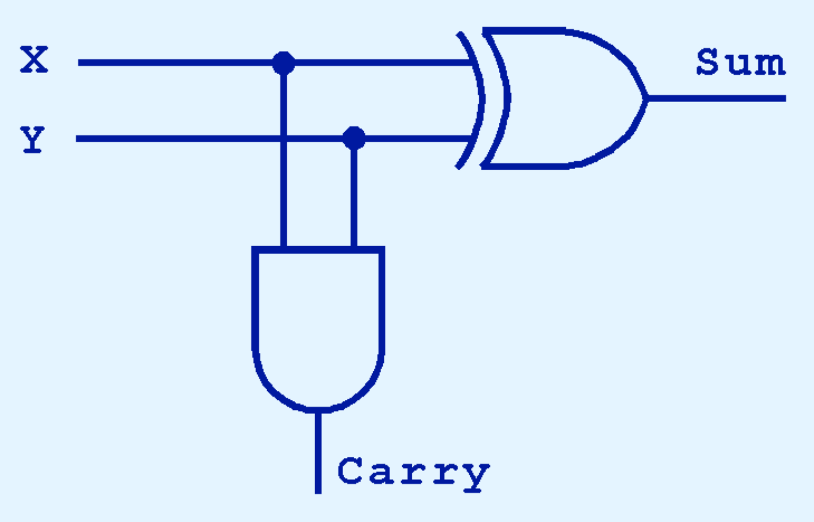


- CLCs produce a specified output (almost) at the instant when input values are applied.

CLC - Haft Adder

- CLCs give us many useful devices.
- A simplest CLC is the ***half adder*** (HA), which finds the sum of 2 bits.
- We can gain some insight as to the construction of a HA by looking at its truth table, shown at the left.
- The sum can be found using the XOR operation and the carry using the AND operation

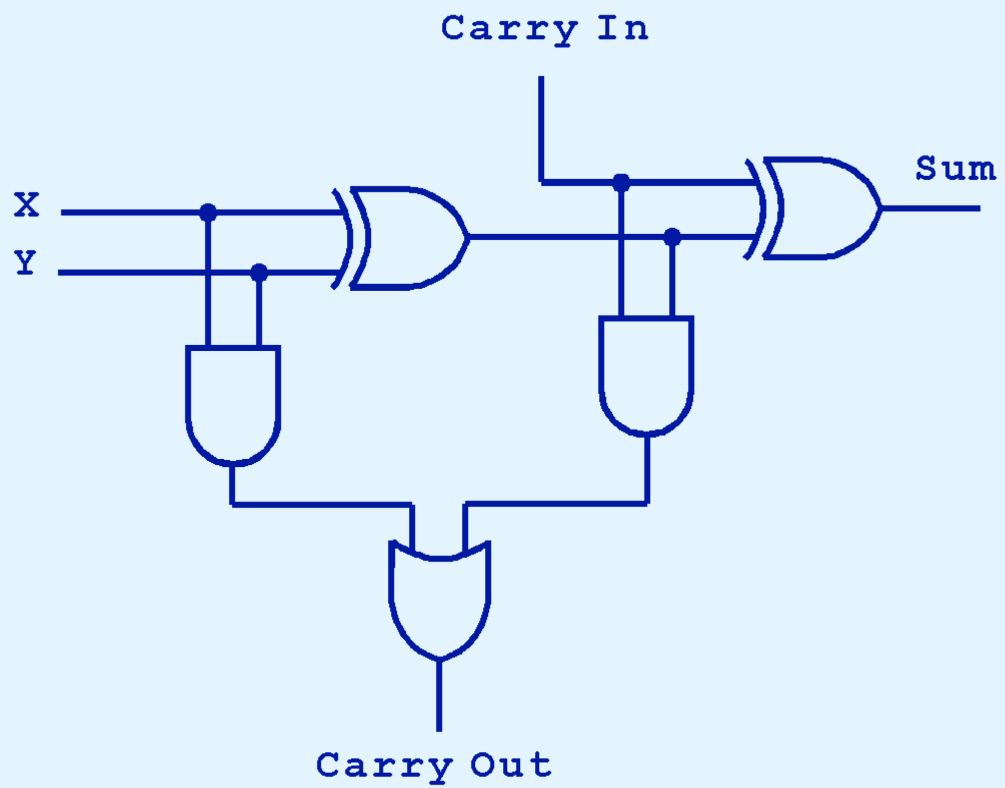
Inputs		Outputs	
X	Y	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



CLC - Full Adder

The ***full adder*** (FA) finds the sum of 3 bits. The truth table is shown at the left. We can combine 2 HAs to make a FA

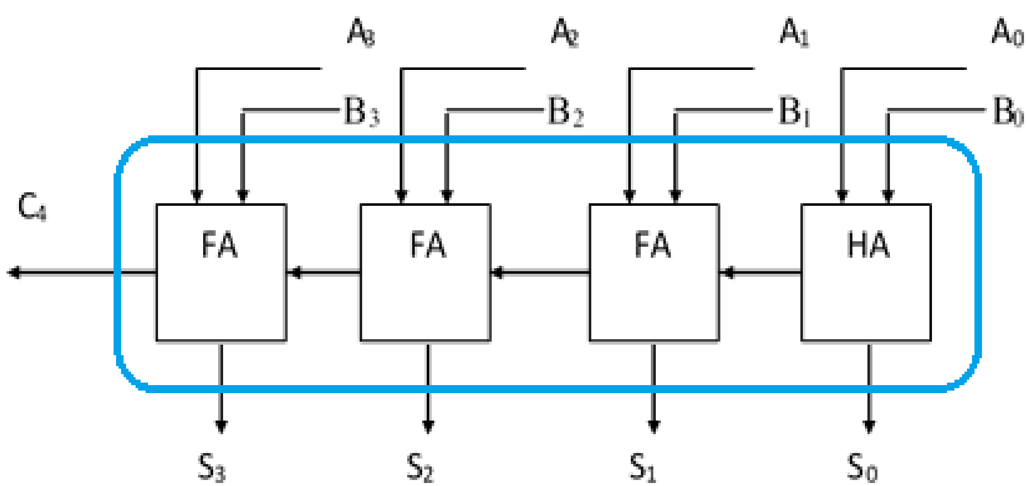
Inputs			Outputs	
X	Y	Carry In	Sum	Carry Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



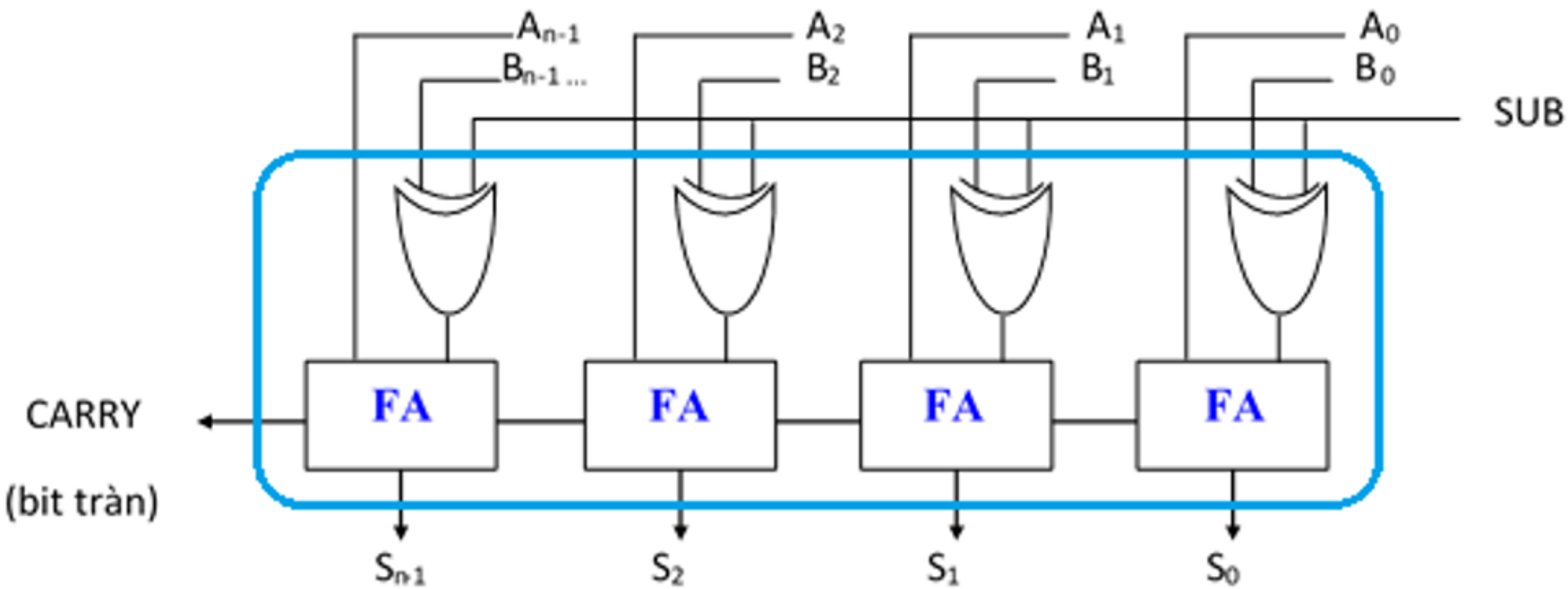
CLC - Add Circuit

- FAs can connected in series to make an Add_Circuit
- The carry bit “ripples” from one adder to the next; hence, this configuration is called a *ripple-carry adder*.

	A_3	A_2	A_1	A_0
+	B_3	B_2	B_1	B_0
C_4	S_3	S_2	S_1	S_0



CLC - Subtract Circuit

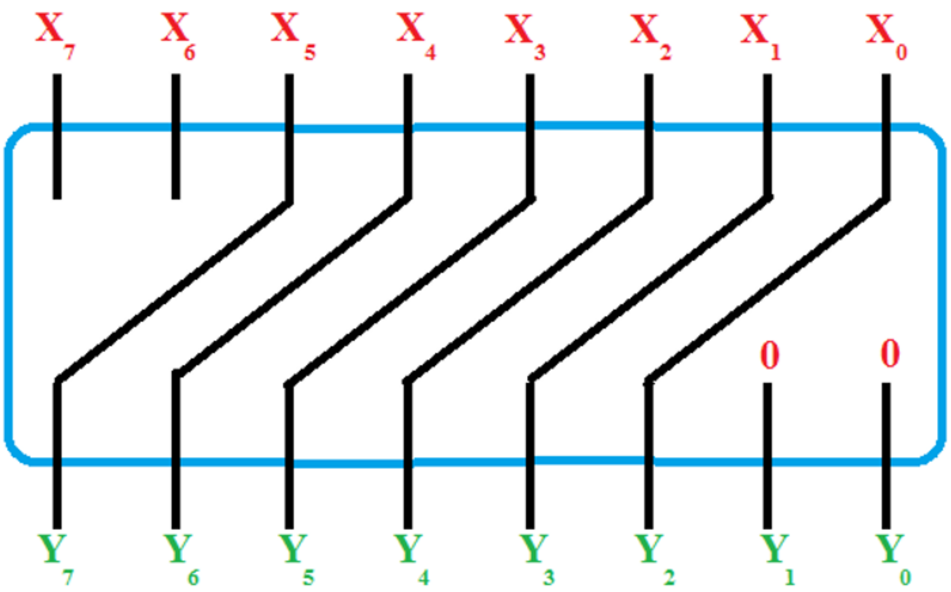


SUB = 0 ⇔ S= A + B

SUB = 1 ⇔ S = A – B

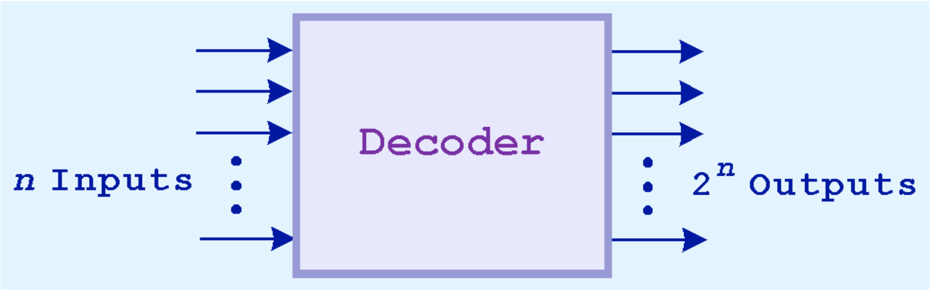
CLC - Shift Circuit

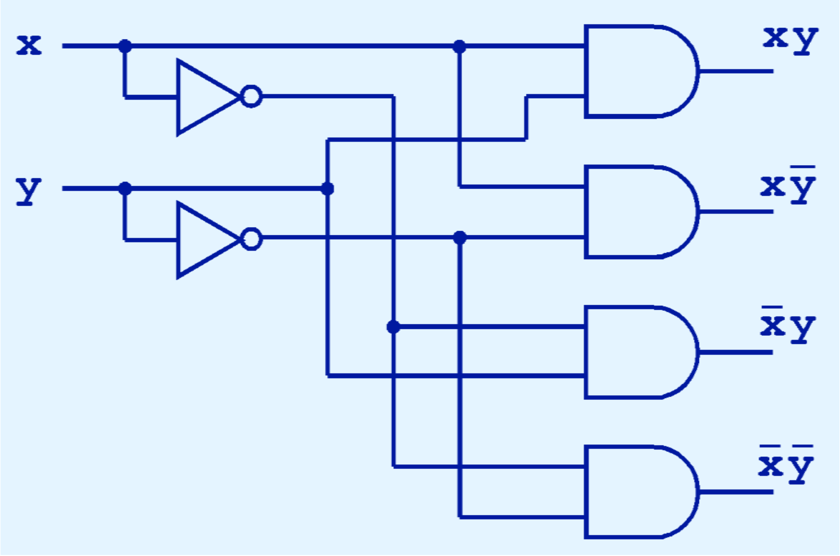
Y=4X | X and Y are 8-bit integers



CLC - Decoder

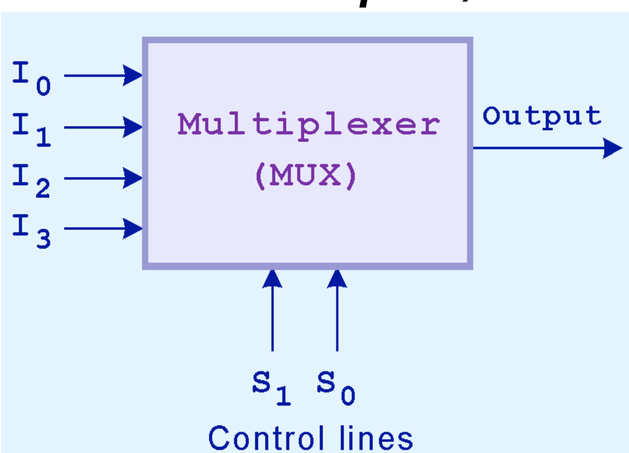
- Decoders are another important type of CLC. They are useful in selecting a memory location according a value placed on the address lines of a memory bus.
- Address decoders with *n* inputs can select any of 2^n locations. It is shown at the left.
- *For example, a 2-to-4 decoder is at the right:*

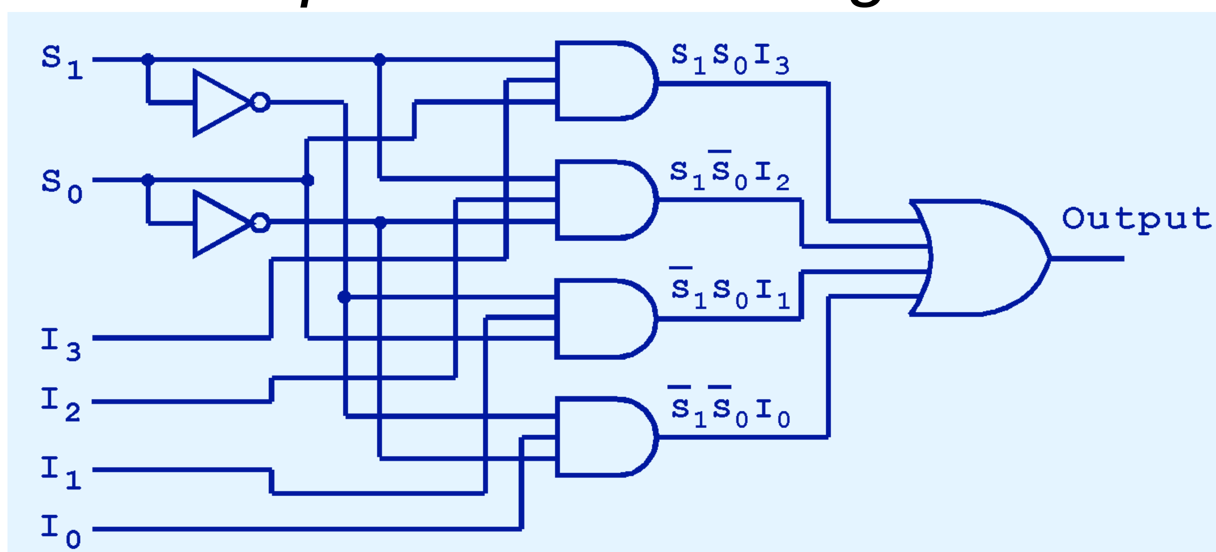




CLC - Multiplexer

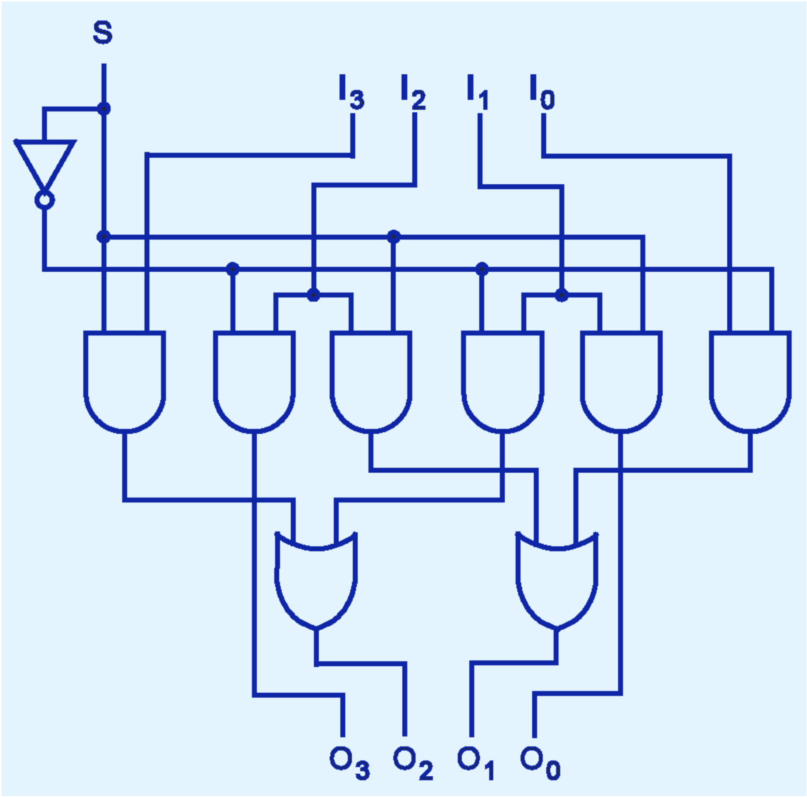
- A multiplexer does just the opposite of a decoder. It selects a single output from several inputs.
- The particular input chosen for output is determined by the value of the multiplexer’s control lines.
- To be able to select among *n* inputs, $\log_2 n$ control lines are needed. It is shown at the left.
- *For example, a 4-to-1 multiplexer is at the right:*





CLC - Decoder & Multiplexer

This shifter moves the bits of a nibble one position to the left or right.

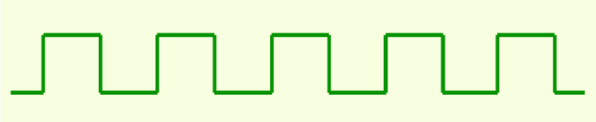


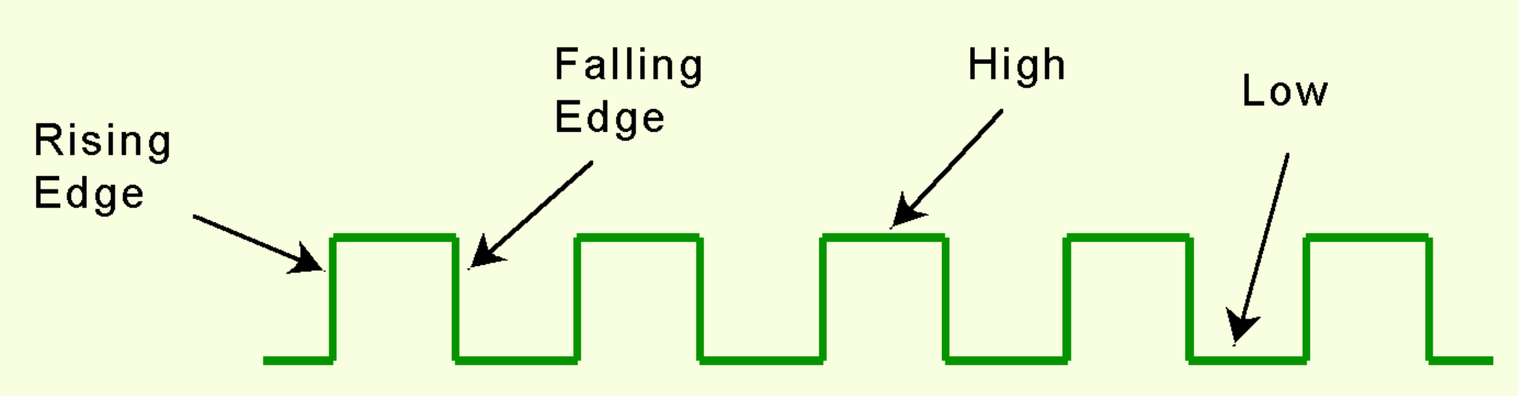
Sequential Logic Circuits

Concepts

- CLCs are perfect for situations when we require the immediate application of a BF to a set of inputs.
- There are other times, we need a circuit to change its value with consideration to its current state as well as its inputs. These circuits must “remember” their current state.
- *Sequential logic circuits* (SLCs) provide this functionality.

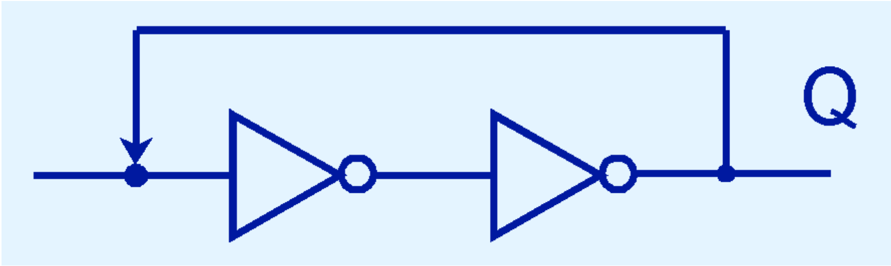
SLC - Clocks

- As the name implies, SLCs require a means by which events can be sequenced.
- State changes are controlled by clocks. “clock” is a special circuit that sends electrical pulses
- Clocks produce electrical waveforms: 
- State changes occur in SLC only when the clock ticks.
- Circuits can change state on the rising, falling edge, or when the clock pulse reaches its highest voltage



SLC - Feed back

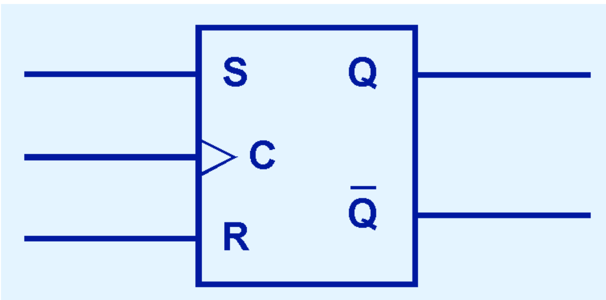
- You can see how feedback works by examining the most basic sequential logic components, the SR flip-flop.
  - The “SR” stands for set/reset.
- The internals of an SR flip-flop are shown below, along with its block diagram

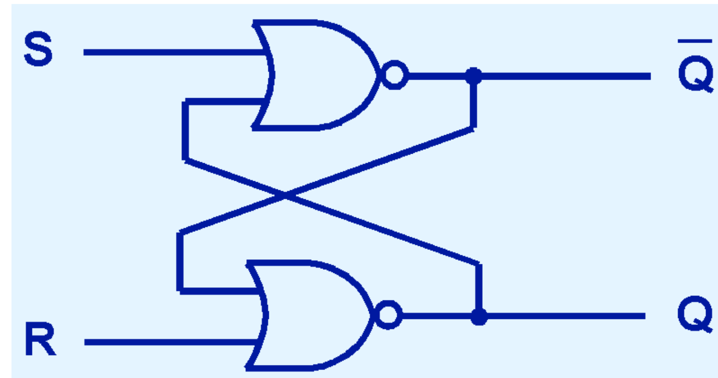


SLC - SR flip flop

- The internals of a Set/Reset flip-flop are shown below

(*NOR gate can be replace by NAND*)





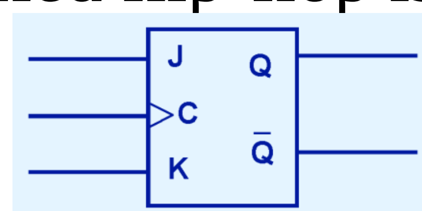
- The behavior of an SR flip-flop is described by a characteristic table.

S	R	Q(t+1)
0	0	Q(t) (no change)
0	1	0 (reset to 0)
1	0	1 (set to 1)
1	1	undefined

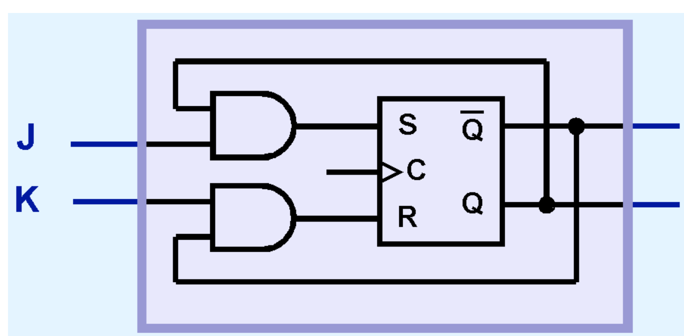
- *Q(t) means the value of the output at time t.*
- *Q(t+1) is the value of Q after the next clock pulse.*

SLC - JK flip flop

- SR flip-flop actually has 3 inputs: S, R, and Q.
- Thus, we can construct a truth table as shown at the left.
- Notice: when S=1 and R=1, the SR flip-flop is unstable.
- If the inputs will never both be 1, we will never have an unstable circuit. This may not always be the case.
- The SR flip-flop can be modified to provide a stable state when both inputs are 1. This modified flip-flop is called a JK flip-flop, shown at the right.



S	R	Present State Q(t)	Next State Q(t+1)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	undefined
1	1	1	undefined

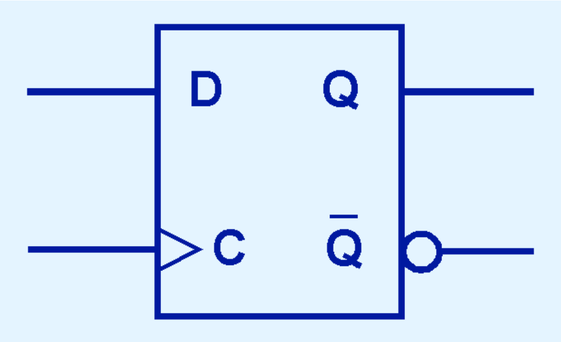


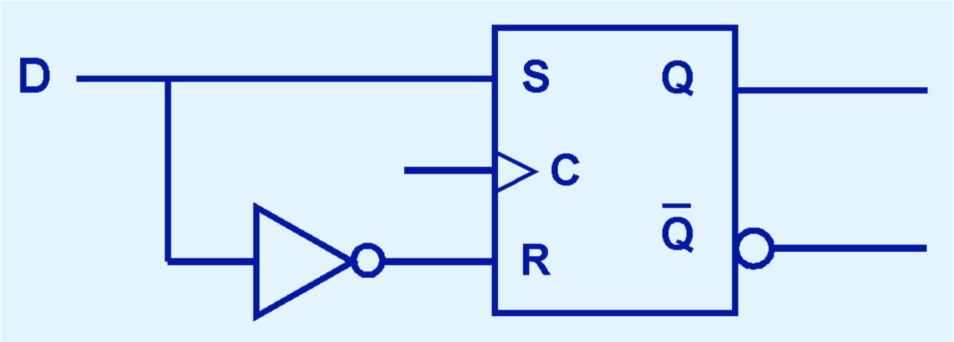
J	K	Q(t+1)
0	0	Q(t) (no change)
0	1	0 (reset to 0)
1	0	1 (set to 1)
1	1	$\bar{Q}(t)$

SLC - D flip flop

- Another modification of the SR flip-flop is the D flip-flop, shown below with its characteristic table.
- You will notice that the output of the flip-flop remains the same during subsequent clock pulses. The output changes only when the value of D changes.
- The D flip-flop is the fundamental circuit of computer memory.

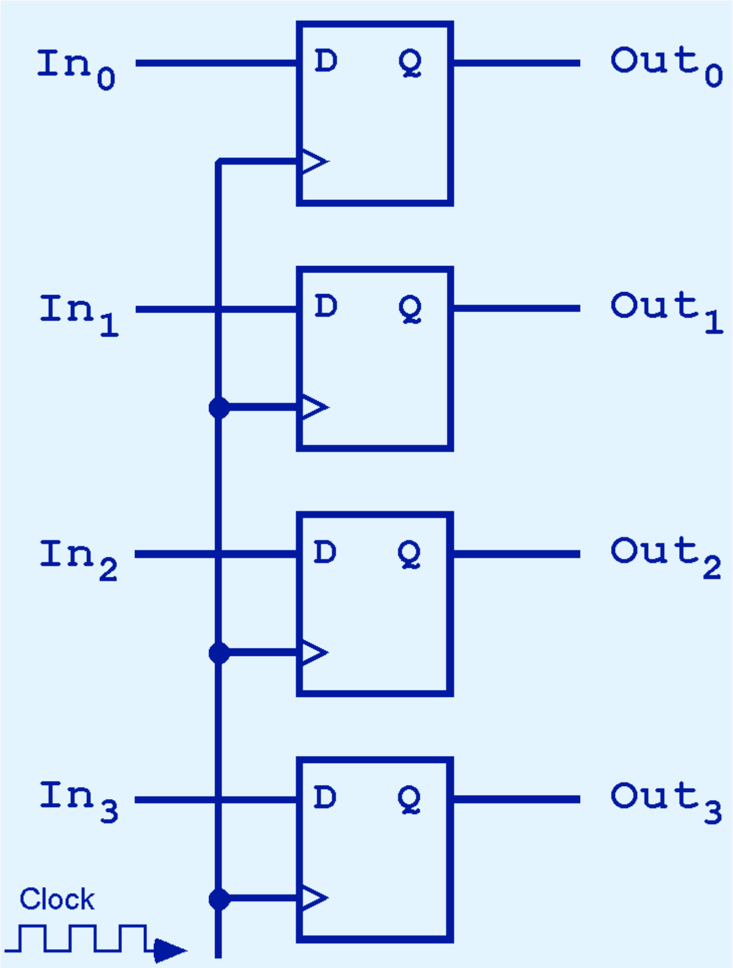
D	Q(t+1)
0	0
1	1





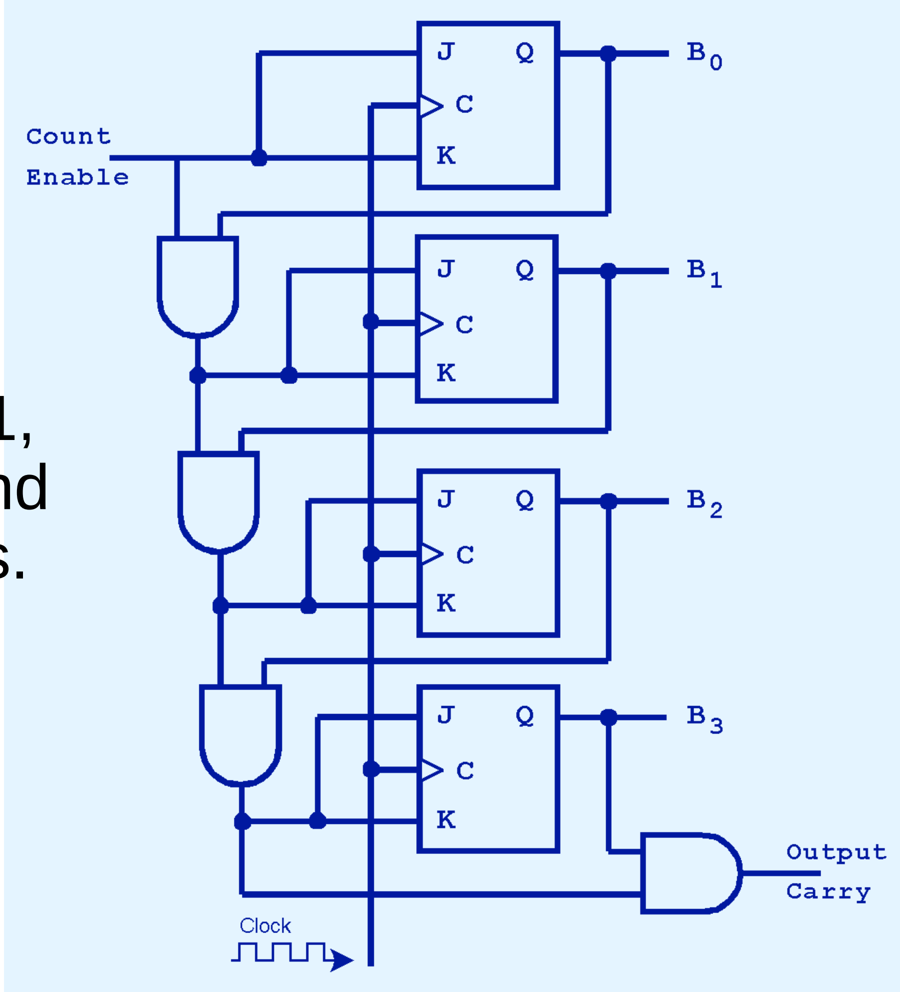
SLC - Register with D flip flop

This illustration shows a 4-bit register consisting of D flip-flops



SLC - Binary Counter

- A binary counter is another example of a sequential circuit
- The low-order bit is complemented at each clock pulse.
- Whenever it changes from 0 to 1, the next bit is complemented, and so on through the other flip-flops.



Designing Circuits from Truth Table

Algorithm

- We can easily create logic diagrams using the values specified in the truth table.
- To create the Boole function for drawing the circuit, we need to perform the following steps:
  1. Identify lines where the output is 1.
  2. In each line, determine the value of each input and the norm (the product of all input)
  3. Set up the Boole function In the sum-of-products form
  4. Reduce this Boole function to a simple expression that has the least number of legal gates to use
  5. Draw a logic circuit from the reduced Boole expression.

Designing Circuits - Example

Design the logic circuit from the truth table (in the best possible way):

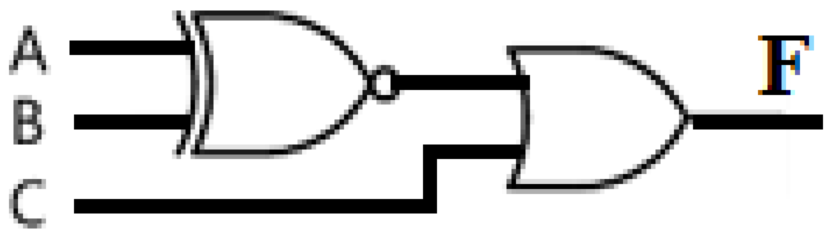
F=1 when: {A=0, B=0, C=0} or {A=0, B=0, C=1} or {A=0, B=1, C=1} or {A=1, B=0, C=1} or ...

$$F = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$
$$= (\bar{A}\bar{B} + AB)\bar{C} + (\bar{A}\bar{B} + \bar{A}B + A\bar{B} + AB)C$$
$$= (\bar{A}\bar{B} + AB)\bar{C} + C = (\bar{A}\bar{B} + AB) + C$$

= (A ***xnor*** B) ***or*** C

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

$\Rightarrow$ Diagram of Circuit:



(In this case, we can work with ~F and have a simpler BF)

Designing Complex Circuits

- Digital designers rely on specialized software to create efficient circuits. Software is an enabler for the construction of better hardware.
- When we need to implement a specialized algorithm and its execution speed must be as fast as possible, hardware solution is often preferred.
- This is the idea behind embedded systems, which are small special-purpose computers in everywhere.
- Embedded systems require special programming that demands an understanding of the operation of logic circuits - the basics of which you have learned in this chapter.

Summary

Summary

- Computers are implementations of Boolean logic.
- Boolean functions are completely described by truth tables.
- Logic gate is a small circuit that implement Bool operators.
- The basic gates are AND, OR, and NOT. The “universal gates” are NOR, and NAND.
- Computer circuits consist of CLCs and SLCs.
- CLCs produce outputs immediately when their inputs change.
- SLCs require clocks to control their changes of state.
- The basic CLC unit is the flip-flop: The behaviors of the SR, JK, and D flip-flops are the most important to know.