EEE 213
Introduction to Logic Design

Assist. Prof. Dr. Şenol GÜLGÖNÜL

2024-2025 Fall Semester

Manager of Control&Electronic Systems
BMC Fower · Full-time
2017 - 2022 · 5 yrs

-Managing Engine Control Unit (ECU) and (Transmission Control Unit) development of ALTAY Main Battle Tank and other military diesel ...see mor

Skills: OpenECU · MIL-STD-461 · MIL-STD-464 · MIL-STD-1275 · MIL-STD-810 · ISO 26262 · ECE R10 · IPC 620 · Simulink · Embedded Systems · Project

Parttime Instructor
Çankaya Üniversitesi · Part-time
2017 · Less than a year

ECE 439: Satellite and Mobile Communication Systems
ECE 246: Fundamentals of Electronics

Skills: Electronics · Satellite Communications (SATCOM)

Technical Manager
Turksat Uydu Haberleşme Kablo TV ve İşletme A.Ş. · Full-time
2004 - 2016 · 12 yrs
Ankara, Turkey

-VP of Satellite Operations: Involved in Turksat 4A&4B, Turksat 5A&5B and Turksat 6A satellite projects projects. Daily operation of teleport ...see mor

Skills: Satellite Systems Engineering · Satellite Ground Systems · Satellite TV Global Navigation Satellite System (GNSS) · Satellite Communications

Network Manager
KoçSistem · Full-time
Mar 2000 - Jul 2000 · 5 mos

Network Manager of Cisco routers

Skills: Cisco Routers

Senior Network Engineer
TurkNet · Full-time
1996 - 1999 · 3 yrs

Network manager of Turnet which was the first commercial internet backbone o SATCOM, Cisco routers

Skills: VSAT · Cisco Routers · Satellite Communications (SATCOM)

Chief Engineer
Turk Telekom · Full-time
1993 - 1995 · 2 yrs

Chief Ground Control Systems engineer of Turksat Satellite Control Center. The first communication sate of Turkey: Turksat-1A, Turksat-1B and Turksat-1C

Bilkent University
Bachelor of Science - BS
1986 - 1992

Gebze Technical University
Master's degree
1996 - 1998

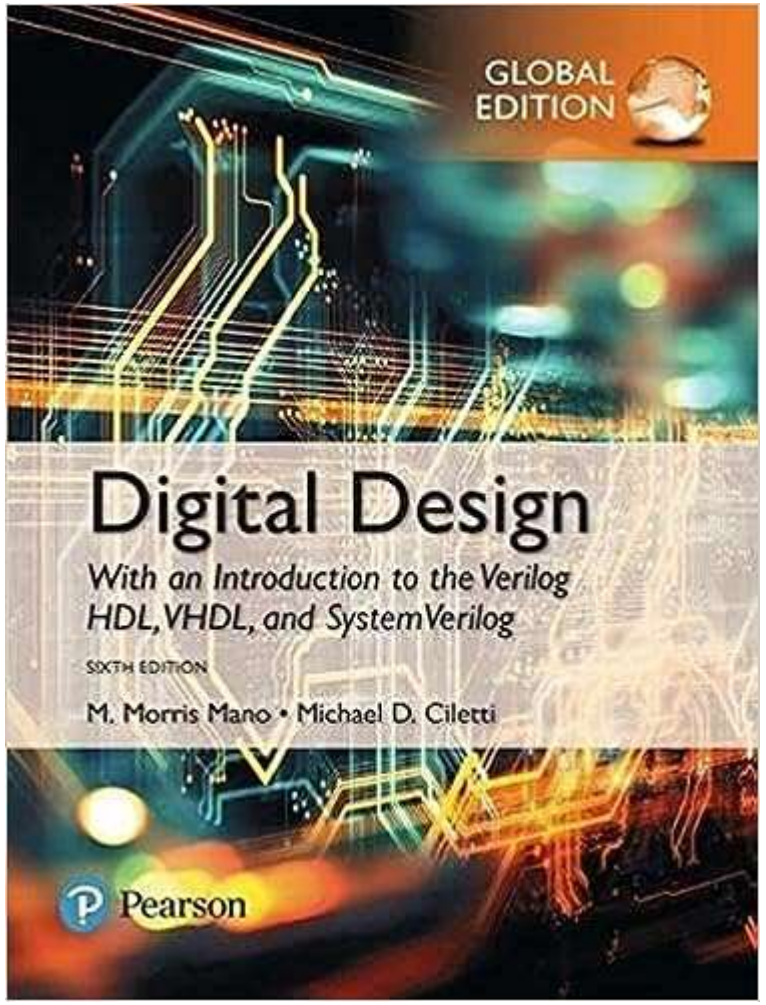
Sakarya University
Doctor of Philosophy - PhD
2009 - 2014

ABOUT

- Text Book: Digital Design, Global Edition
  Mano, M. Morris, Ciletti, Michael

- Grades: 20% Midterm + 20% Project + 60% Final

- Course Objectives: Boolean Algebra, implementation using Logic Gates



Additional Resources:

- Logic Simulator (hneemann/Digital: A digital logic designer and circuit simulator. (github.com))

- senolgulgonul/introductiontologic: Introduction to Logic Design Lecture Files (github.com)

Additional Resources

- Arduino Uno
- Arduino Proto Shield
- **Saleae Logic Analyzer**
- LED (red) x 10
- Resistor (1W/100 ohm) x 10
- 10 cm 40 Pin M-M Jumper Cable
- 10 cm 40 Pin M-F Jumper Cable
- 10 cm 40 Pin F-F Jumper Cable
- 1K Potentiometer x 2

- 74HC04 Inverter
- 74HC08 AND
- 74HC32 OR
- 74HC86 XOR
- 74HC138 Decoder
- 74HC151 Multiplexer
- 74HC74 D-Flip Flop

WHY

- Electronics
  - Analog: RLC, Transistors, Amplifiers, Filters, Electromagnetic, Antenna etc.
  - Digital: Logic Gates, Logic IC’s, Microcontrollers, FPGA etc.

SYLLABUS

Week	Subject
1	Boolean Algebra and Logic Gates
2	Gate Level Minimization
3	Combinational Logic
4	Combinational Logic
5	Combinational Logic
6	Midterm
7	Sequential Logic
8	Sequential Logic
9	Registers Counter
10	Registers Counter
11	Memory and Programmable Logic
12	Microcontroller Architecture
13	Microcontroller Architecture
14	Final
15	Final
16	

HISTORY OF LOGIC

- ‘Logic’ originates from Greek word ‘logos’, means ‘reason’
- Logic is a feature of being alive and decision making
- Most advanced form of logical reasoning is in humans (as far as we know! excluding ChatGPT☺)
- Aristoteles can be named as the father of logic, who set rules of logical thinking

AISTOTELES (384–322 BC)

His followers compiled the logical works of Aristotle into a collection of six books known as the 'Organon' around 40 BC.

1. Categories
2. On Interpretation
3. Prior Analytics
4. Posterior Analytics
5. Topics
6. On Sophistical Refutations

EXAMPLES OF ARISTOTELES LOGIC

Aristotle’s central observation was to check validity of arguments based on their logical structure. Aristotle, in his work "Sophistical Refutations," aimed to expose and demonstrate the invalidity of various fallacies used by sophists.

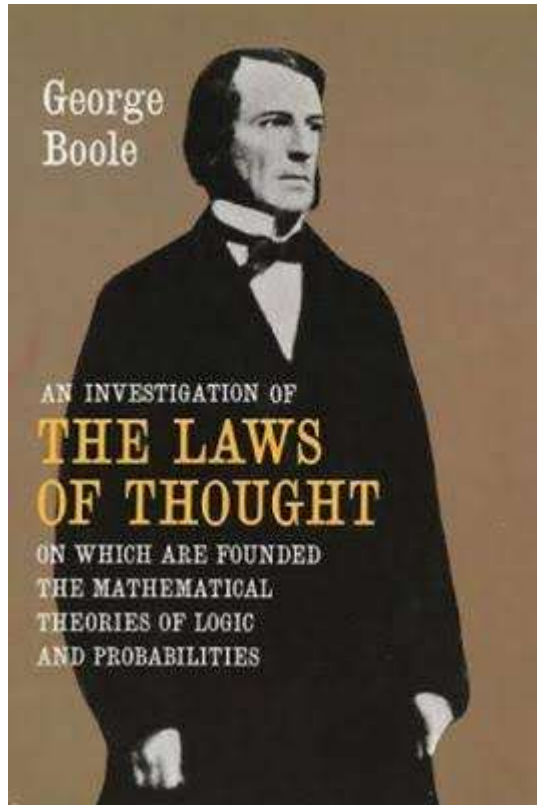
- An object is what it is (Law of Identity)
- No statement can be both true and false (Law of Non-contradiction)
- Every statement is either true or false (Law of the Excluded Middle)
- Categorical Syllogism:

  All humans are mortal. Socrates is a human. Therefore, Socrates is mortal.

GEORGE BOOLE (1815-1864)

The Organon, occupied a central place in the scholarly canon for more than 2,000 years. It was widely believed that Aristotle had written almost all there was to say on the topic. The great philosopher Immanuel Kant commented that, since Aristotle, logic had been “unable to take a single step forward, and therefore seems to all appearance to be finished and complete.”

George Boole’s, "An Investigation of the Laws of Thought," published in 1854, laid the foundation for symbolic logic. Boole’s goal was to do free for Aristotelean logic from the limits of human intuition by giving it a precise algebraic notation.



BOOLEAN ALGEBRA-SHANNON

Boolean algebra in ‘Laws of Thought’ defined for classes (sets), converts words to symbols.

‘All men are mortal’ can be converted as:

$y=y.x$

where y=‘men’, x=‘mortal’

Claude Shannon introduced the use of Boolean algebra in the analysis and design of switching circuits in 1936 (Master Thesis).

SHANON APPLICATION OF BOOLEAN ALGEBRA

Open=1, Close=0 : X.Y

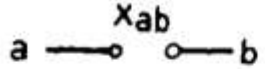


Figure 1 (left). Symbol for hindrance function

Figure 2 (right). Interpretation of addition

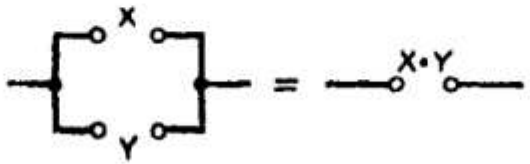


Figure 3 (middle). Interpretation of multiplication

Open=1, Close=0 : X + Y

Any expression formed with the operations of addition, multiplication, and negation represents explicitly a circuit containing only series and parallel connections.

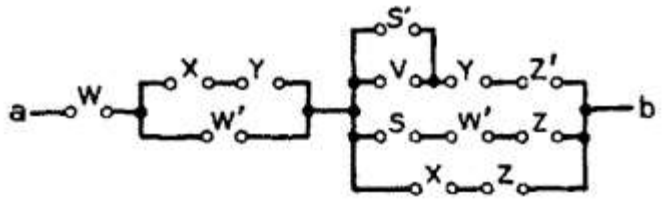


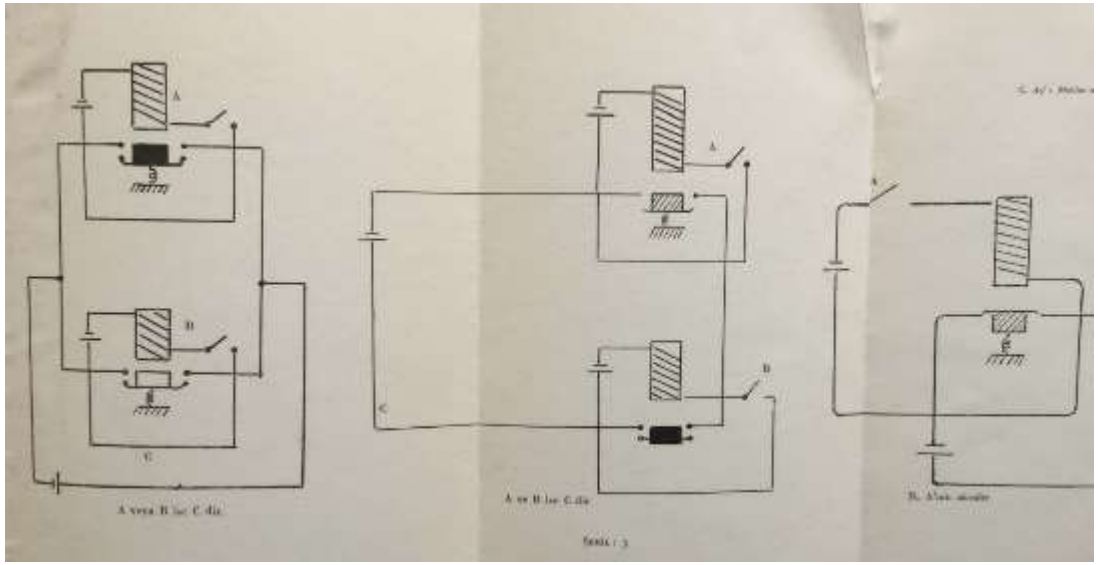
Figure 5. Circuit to be simplified

$$X_{ab} = W + W'(X + Y) + (X + Z)(S + W' + Z)(Z' + Y + S'V)$$

Cahit ARF: Can a machine think?

- Using only relays Boolean algebra can be implemented completely

- Turkish mathematician Cahit Arf discusses if a machine can think at a paper dated 1959 and show AND, OR, NOT gates using relays



LOGIC OPERATORS to LOGIC GATES

Invention of the transistor in 1947 by William Shockley and his colleagues at Bell Labs. dramatically improved versions of Shannon's electrical relays—the best known way to physically encode Boolean operations

Open=1, Close=0

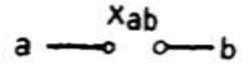


Figure 1 (left). Symbol for hindrance function

Open=1, Close=0

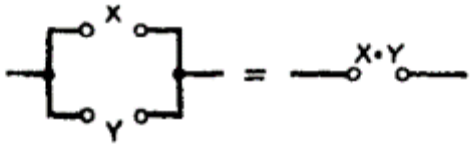


Figure 3 (middle). Interpretation of mult cation

Open=1, Close=0

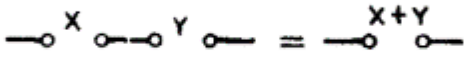
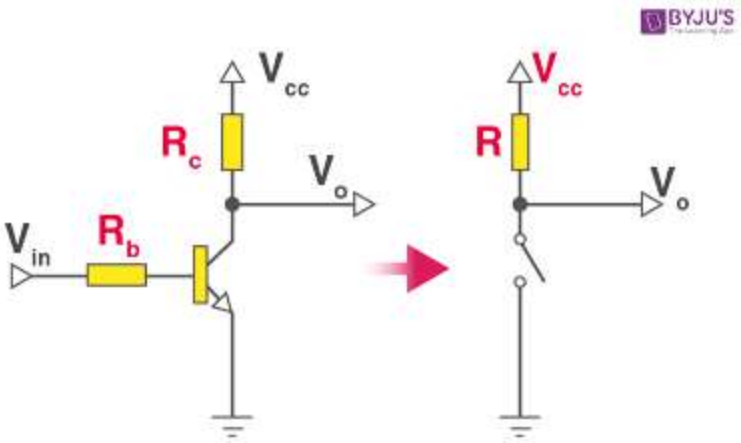
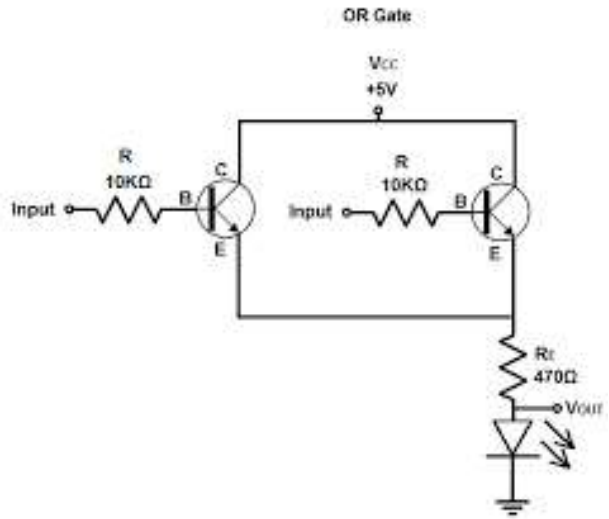
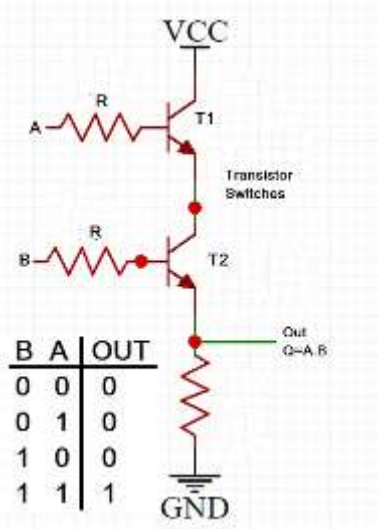


Figure 2 (right). Interpretation of addition



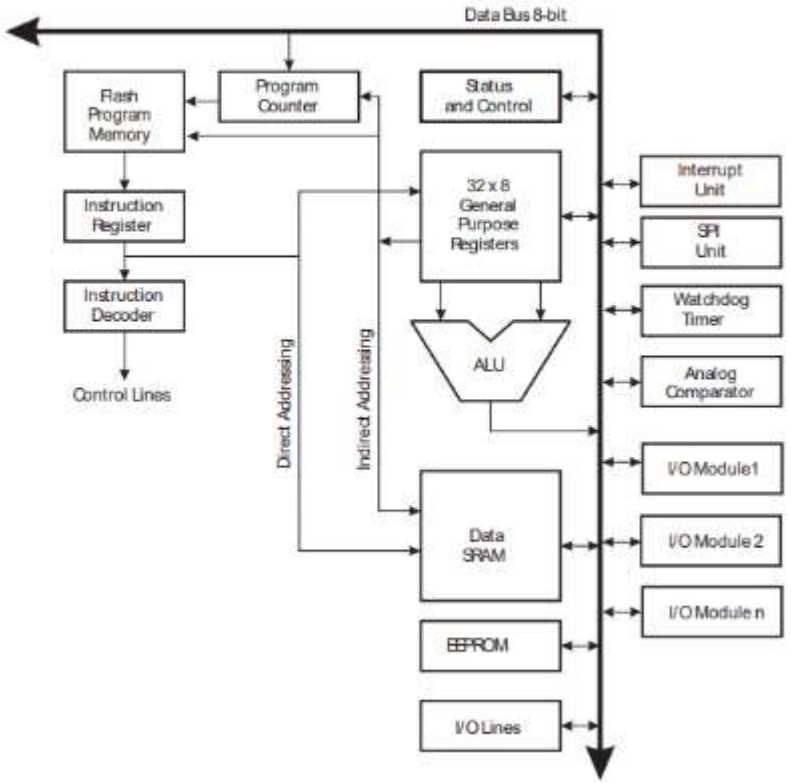




LOGIC GATES to COMPUTERS

While Claude Shannon showed how to map logic onto the physical world, Alan Turing showed how to design computers in the language of mathematical logic. In 1945, he wrote the specification of the EDVAC—the first stored program, logic-based computer—which is generally considered the definitive source guide for modern computer design.

Figure 7-1. Block Diagram of the AVR Architecture



DECIMAL to BINARY

- We can convert numbers to binary 0,1 and perform arithmetic operations using logic gates.
- Converting integer numbers to binary
- LEFTMOST BIT is the MOST SIGNIFICANT BIT of the number

EXAMPLE 1.1

Convert decimal 41 to binary. First, 41 is divided by 2 to give an integer quotient of 20 and a remainder of $\frac{1}{2}$. Then the quotient is again divided by 2 to give a new quotient and remainder. The process is continued until the integer quotient becomes 0. The *coefficients* of the desired binary number are obtained from the *remainders* as follows:

	Integer Quotient		Remainder	Coefficient
$41/2 =$	20	+	$\frac{1}{2}$	$a_0 = 1$
$20/2 =$	10	+	0	$a_1 = 0$
$10/2 =$	5	+	0	$a_2 = 0$
$5/2 =$	2	+	$\frac{1}{2}$	$a_3 = 1$
$2/2 =$	1	+	0	$a_4 = 0$
$1/2 =$	0	+	$\frac{1}{2}$	$a_5 = 1$

Therefore, the answer is $(41)_{10} = (a_5a_4a_3a_2a_1a_0)_2 = (101001)_2$.

DECIMAL to BINARY

- We can convert numbers to binary 0,1 and perform arithmetic operations using logic gates.
- Converting integer numbers to binary
- LEFTMOST BIT is the MOST SIGNIFICANT BIT of the number

EXAMPLE 1.1

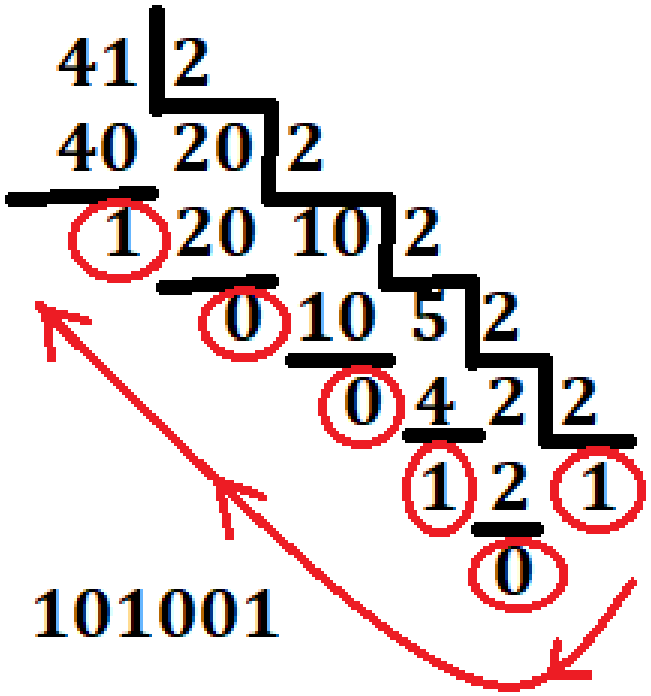
Convert decimal 41 to binary. First, 41 is divided by 2 to give an integer quotient of 20 and a remainder of $\frac{1}{2}$. Then the quotient is again divided by 2 to give a new quotient and remainder. The process is continued until the integer quotient becomes 0. The *coefficients* of the desired binary number are obtained from the *remainders* as follows:

The arithmetic process can be manipulated more conveniently as follows:

Integer		Remainder	
41	/2		
20	/2	1	
10	/2	0	
5	/2	0	
2	/2	1	
1	/2	0	
0		1	101001 = answer

DECIMAL to BINARY

- We can convert numbers to binary 0,1 and perform arithmetic operations using logic gates.
- Converting integer numbers to binary
- divide to 2 till 1 or 0, write remainders in REVERSE order
- LEFTMOST BIT is the MOST SIGNIFICANT BIT of the number



BINARY to DECIMAL

- While converting from binary to decimal: **multiply and add** binary value of bit and decimal value of the bit position

32	16	8	4	2	1	2^x
1	**0**	**1**	**0**	**0**	**1**	**Binary**
32	0	8	0	0	1	=41

Two-Valued Boolean Algebra

One can formulate many Boolean algebras, depending on the choice of elements of B and the rules of operation. In our subsequent work, **we deal only with a two-valued Boolean algebra** (i.e., a Boolean algebra with only two elements).

A two-valued Boolean algebra is defined on a set of two elements, B = {0, 1}

two binary operators + and . as shown in the following operator tables (the rule for the complement operator is for verification of postulate 5)

AND

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

NOT

x	x'
0	1
1	0

Huntington Postulates

Huntington introduced a set of postulates known as "Huntington's Postulates" that describe the fundamental properties of Boolean algebra in 1904. These postulates provide a basis for manipulating logical expressions and establishing the laws of Boolean algebra

Huntington postulates are valid for the set B = {0, 1} and the two binary operators **+ and .**

Huntington Postulates

1. That the structure is **CLOSED** with respect to the two operators is obvious from the tables, since the result of each operation is either 1 or 0 and 1, 0 ∈ B.

2. (a) The element 0 is an **identity element** with respect to +; that is, x + 0 =0 + x = x
   (b) The element 1 is an **identity element** with respect to . ; that is, x . 1 = 1 . x = x

3. (a) The structure is **commutative** with respect to +; that is, x + y = y + x
   (b) The structure is **commutative** with respect to . ; that is, x . y = y . x

4. (a) The operator . is **distributive** over +; that is, x . (y + z) = (x . y) + (x . z)
   (b) The operator + is **distributive** over . ; that is, x + (y . z) = (x + y) . (x + z)

5. For every element x ∈ B, there exists an element x′ ∈ B called the **complement** of x such that
(a) x + x′ = 1
(b) x . x′ = 0

6. There exist at least two elements x, y ∈ B such that x ≠ y, x=1, y=0 and 1 ≠0

Huntington Postulates

- Table 2.1 lists six theorems of Boolean algebra and four of its postulates.
- **The notation is simplified by omitting '.'**
- The theorems and postulates listed are the most basic relationships in Boolean algebra.
- The theorems, like the postulates, are listed in pairs; **each relation is the dual of the one paired with it**.
- The postulates are basic axioms of the algebraic structure and need no proof.
- The theorems must be proven from the postulates.

Table 2.1
Postulates and Theorems of Boolean Algebra

Postulate 2	(a)	$x + 0 = x$	(b)	$x \cdot 1 = x$
Postulate 5	(a)	$x + x' = 1$	(b)	$x \cdot x' = 0$
Theorem 1	(a)	$x + x = x$	(b)	$x \cdot x = x$
Theorem 2	(a)	$x + 1 = 1$	(b)	$x \cdot 0 = 0$
Theorem 3, involution		$(x')' = x$		
Postulate 3, commutative	(a)	$x + y = y + x$	(b)	$xy = yx$
Theorem 4, associative	(a)	$x + (y + z) = (x + y) + z$	(b)	$x(yz) = (xy)z$
Postulate 4, distributive	(a)	$x(y + z) = xy + xz$	(b)	$x + yz = (x + y)(x + z)$
Theorem 5, DeMorgan	(a)	$(x + y)' = x'y'$	(b)	$(xy)' = x' + y'$
Theorem 6, absorption	(a)	$x + xy = x$	(b)	$x(x + y) = x$

Postulates and Theorems of Boolean Algebra

The theorems of Boolean algebra can be proven by means of truth tables

- THEOREM 1(a): x + x = x
- THEOREM 1(b): x . x = x

- THEOREM 2(a): x + 1 = 1
- THEOREM 2(b): x . 0 = 0

- THEOREM 3: (x')' = x **(involution)**

- THEOREM 5(a): (x + y)' = x'y' **(DeMorgan)**
- THEOREM 5(b): (x y)' = x'+y'

- THEOREM 6(a): x + xy = x **(absorption)**
- THEOREM 6(b): x(x + y) = x

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

x	x'
0	1
1	0

x	y	$x + y$	$(x + y)'$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

x'	y	$x'y'$
1	1	1
1	0	0
0	1	0
0	0	0

x	y
0	0
0	1
1	0
1	1

xy	$x + xy$
0	0
0	0
0	1
1	1

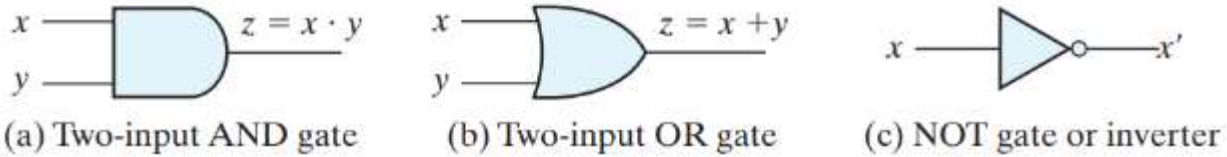
Operator Precedence

The operator precedence for evaluating Boolean expressions is:

(1) Parentheses: **(x+y)**’.x+y
(2) Complement: **(x+y)’**.x+y
(3) AND: **(x+y)’.x**+y
(4) OR: **(x+y)’.x+y**

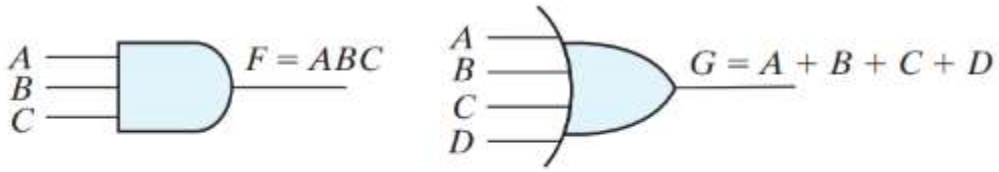
Implementation of Boolean Functions

- Boolean functions can be realized using logic gates
- Logic gates are electronic circuits that operate on one or more physical input signals to produce an output signal.



(a) Two-input AND gate (b) Two-input OR gate (c) NOT gate or inverter

FIGURE 1.4
Symbols for digital logic circuits



(a) Three-input AND gate (b) Four-input OR gate

FIGURE 1.6
Gates with multiple inputs

Table 1.8
Truth Tables of Logical Operations

AND			OR			NOT	
x	y	$x \cdot y$	x	y	$x + y$	x	x'
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

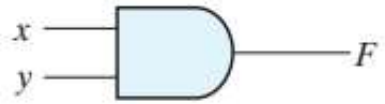
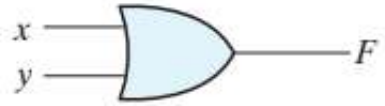
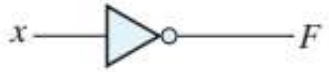
Algebraic Expression to Truth Table

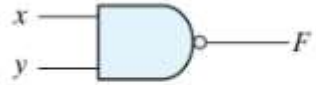
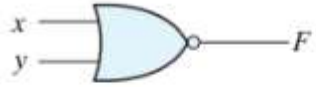
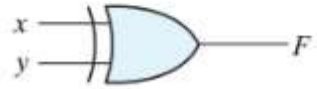
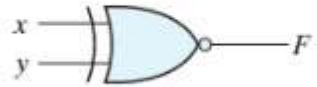
- A truth table is a table of all possible combinations of the variables, showing the relation between the values that the variables may take and the result of the operation

Table 1.8
Truth Tables of Logical Operations

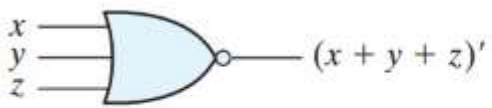
AND			OR			NOT	
x	y	$x \cdot y$	x	y	$x + y$	x	x'
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

2.7. Logic Gates

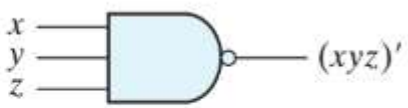
AND		$F = x \cdot y$	x y F 0 0 0 0 1 0 1 0 0 1 1 1
OR		$F = x + y$	x y F 0 0 0 0 1 1 1 0 1 1 1 1
Inverter		$F = x'$	x F 0 1 1 0
Buffer		$F = x$	x F 0 0 1 1

NAND		$F = (xy)'$	x y F 0 0 1 0 1 1 1 0 1 1 1 0
NOR		$F = (x + y)'$	x y F 0 0 1 0 1 0 1 0 0 1 1 0
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	x y F 0 0 0 0 1 1 1 0 1 1 1 0
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= (x \oplus y)'$	x y F 0 0 1 0 1 0 1 0 0 1 1 1

2.7. Logic Gates (multiple input)



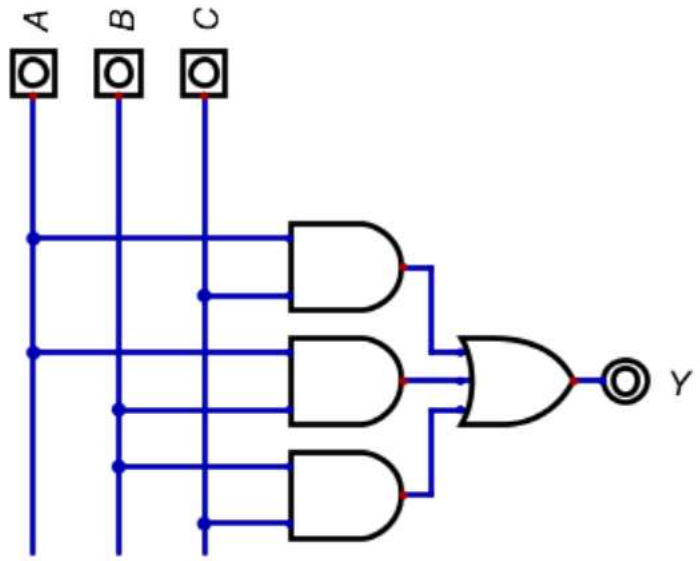
(a) 3-input NOR gate



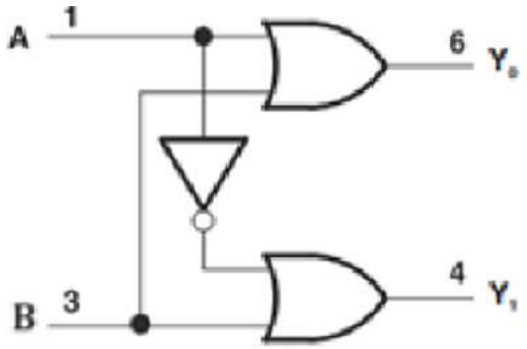
(b) 3-input NAND gate

Logic Circuits Analysis

A=0 B=1 C=1 Y=?
A=1 B=0 C=0 Y=?



Logic Circuits Analysis

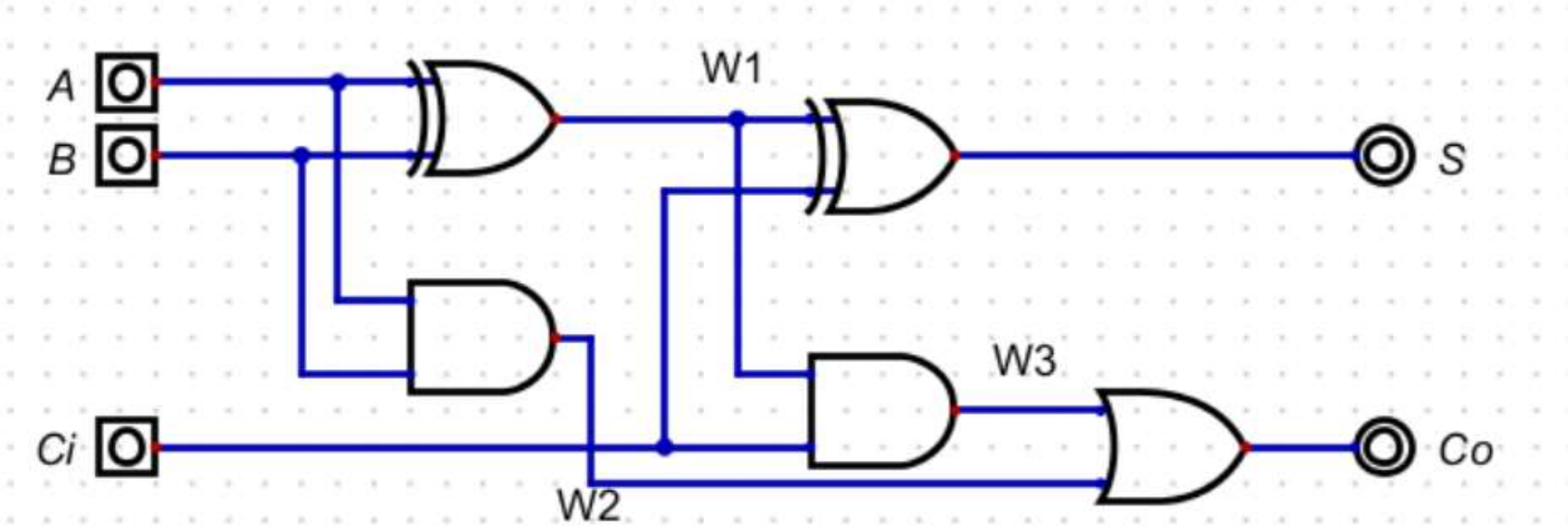


10. (5 points) B=1 what is Y0=_____and Y1=_____

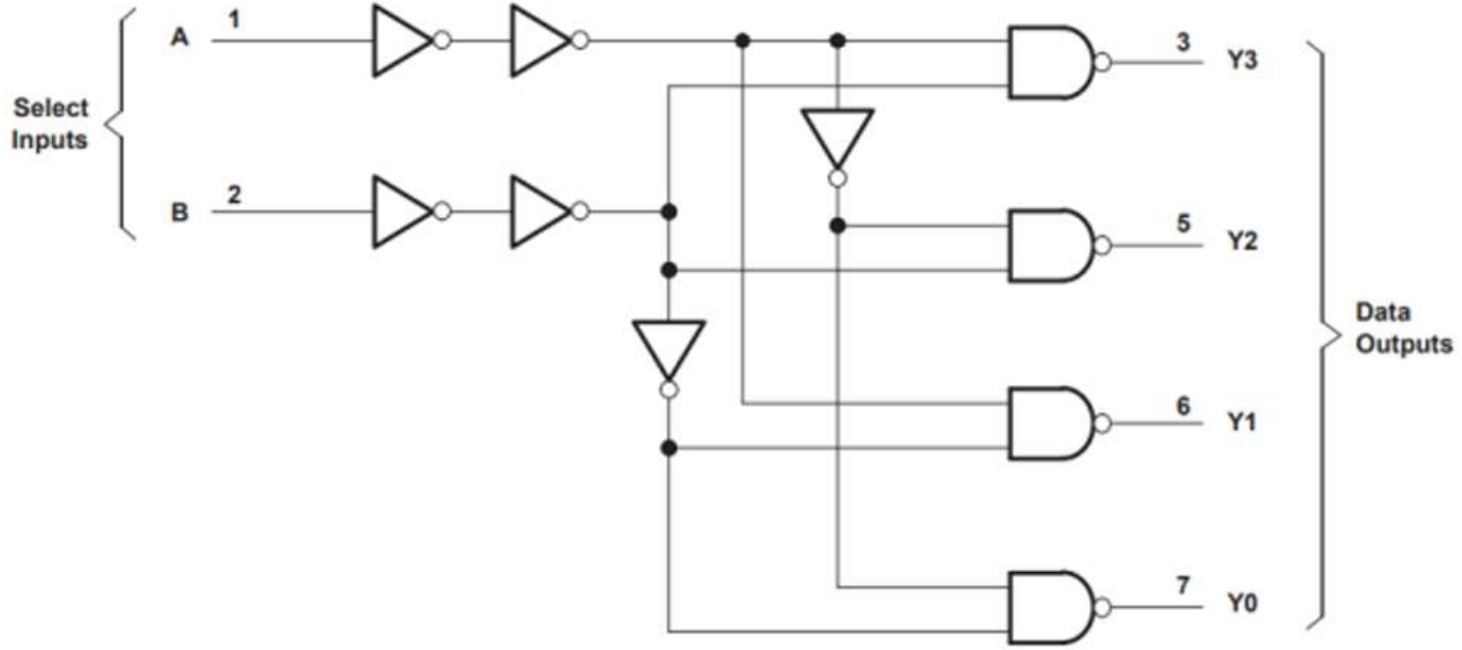
Logic Circuits Analysis

A=0 B=1 Ci=1 S= 0 Co=1

A=1 B=1 C=1 S=1 Co=1



Logic Circuits Analysis

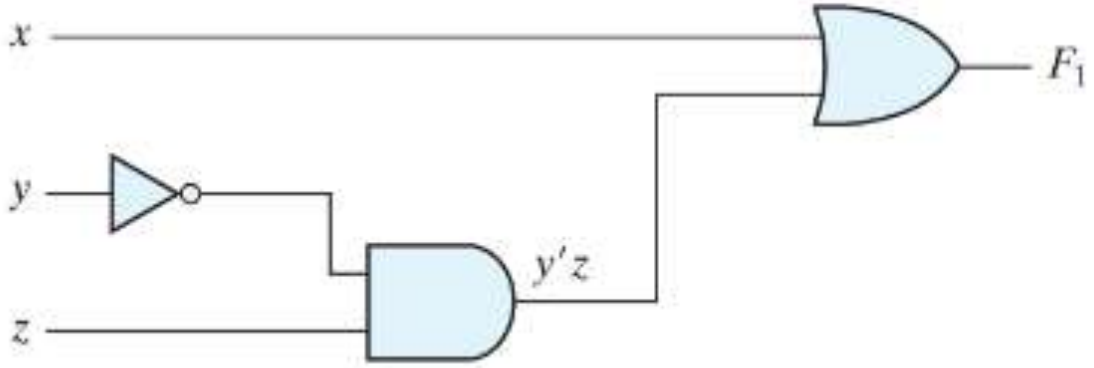


3. (10 points) What is the output Y3 Y2 Y1 Y0 of given logic circuit for inputs A=1, B=0?

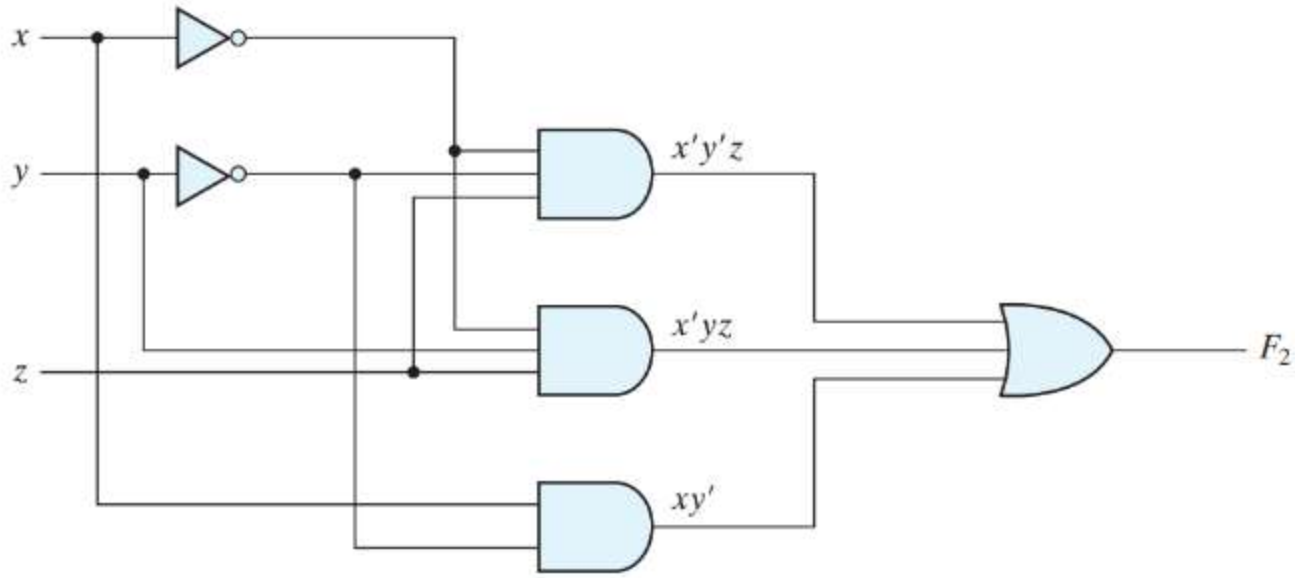
Y3 Y2 Y1 Y0=_____

Expression to Logic Diagrams

Boolean function can be transformed from an algebraic expression into a circuit diagram composed of logic gates connected in a particular structure.



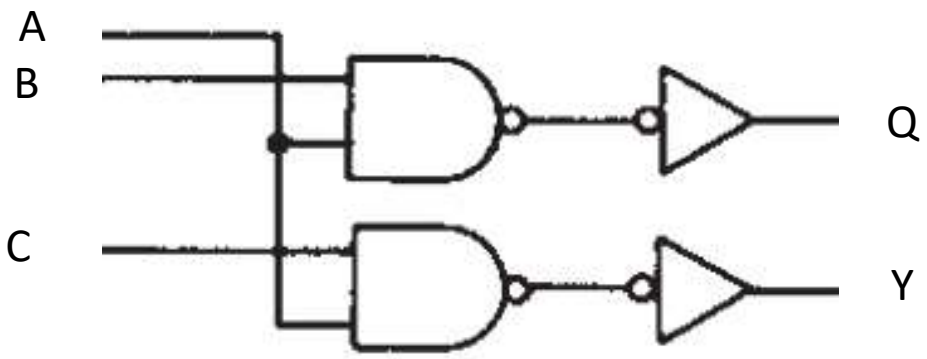
for the Boolean function $F_1 = x + y'z$



(a) $F_2 = x'y'z + x'yz + xy'$

Expression to Logic Diagrams

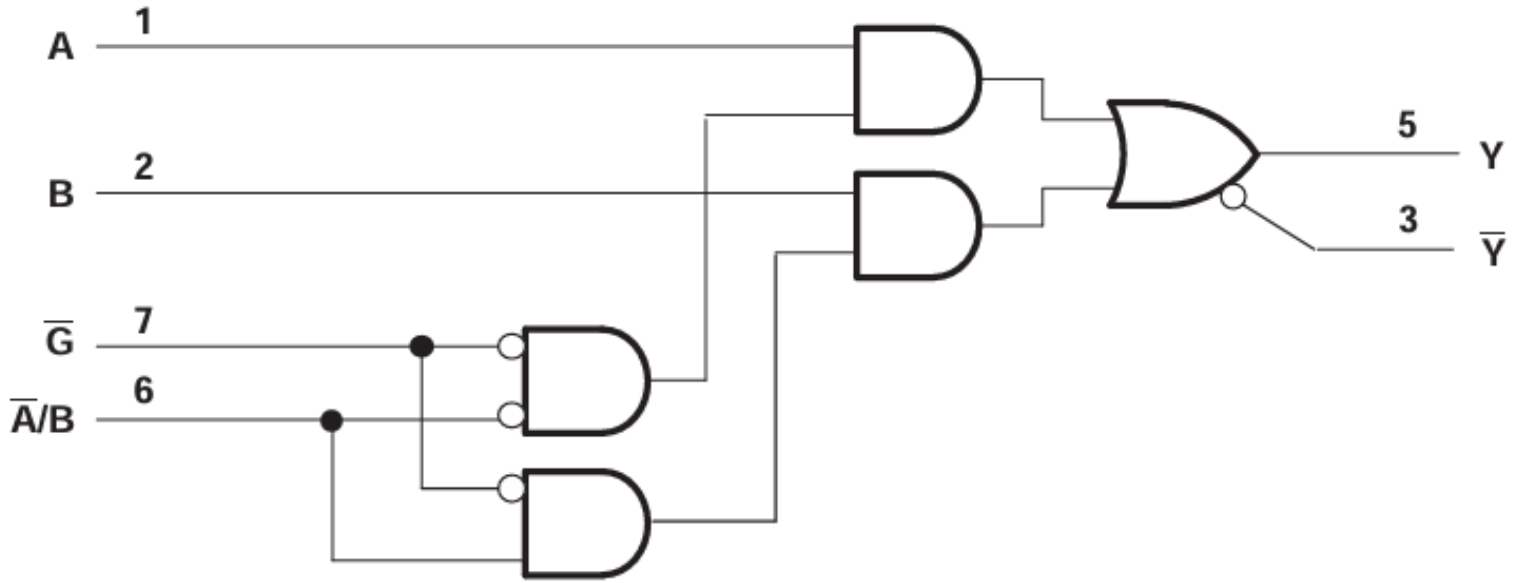
Boolean function can be transformed from an algebraic expression into a circuit diagram composed of logic gates connected in a particular structure.



Q = A . B
Y = A . C

Expression to Logic Diagrams

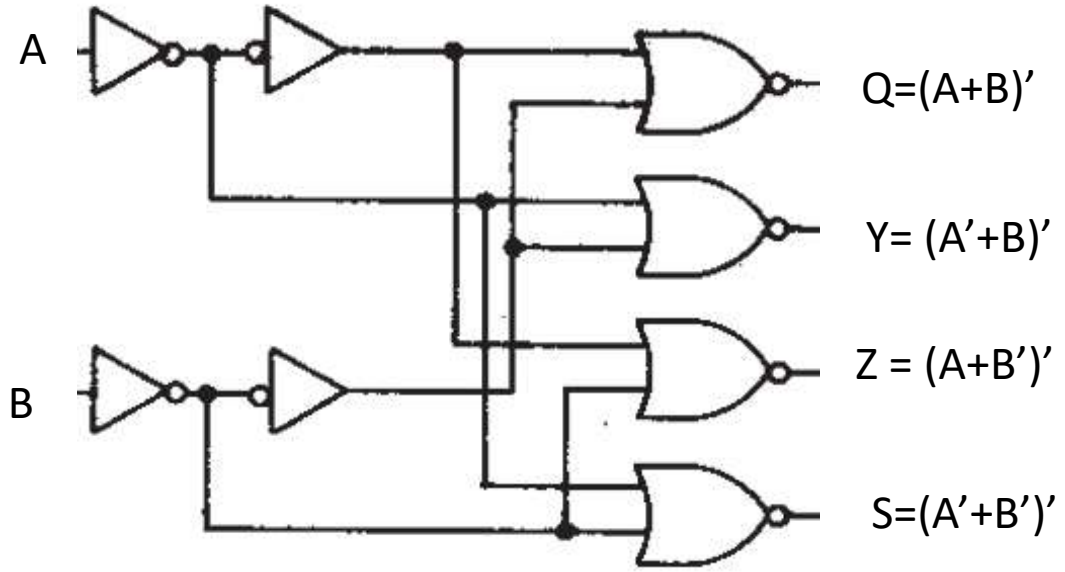
Boolean function can be transformed from an algebraic expression into a circuit diagram composed of logic gates connected in a particular structure.



Y= A.G'.C' + B.G'.C

Expression to Logic Diagrams

Boolean function can be transformed from an algebraic expression into a circuit diagram composed of logic gates connected in a particular structure.

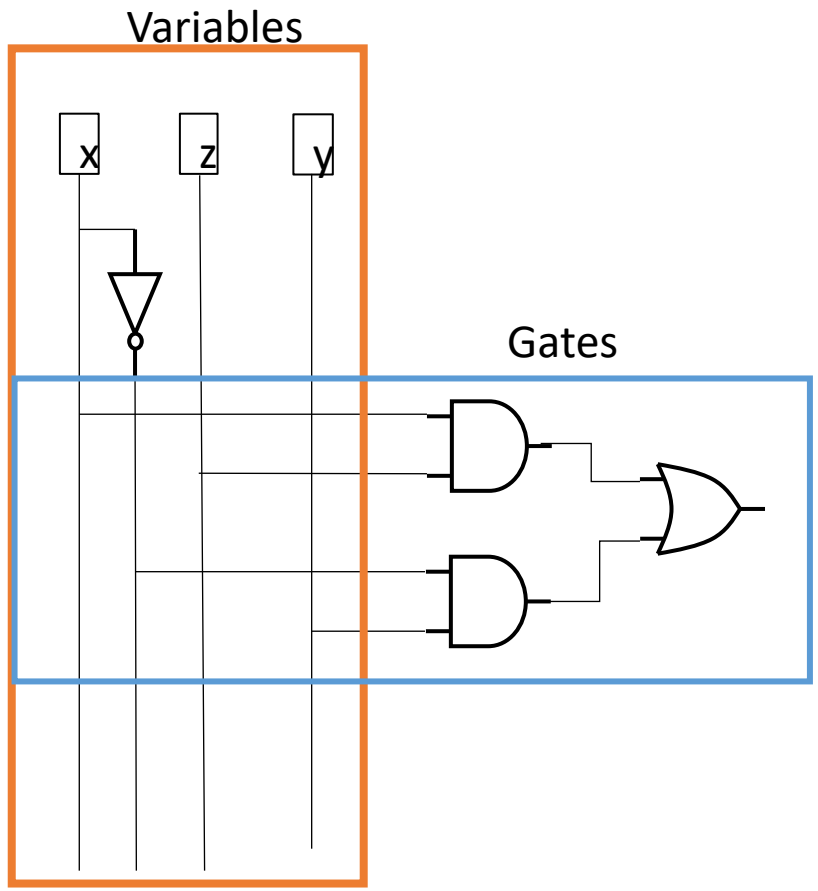


Expression to Logic Circuit

Given F = x.z + x’.y

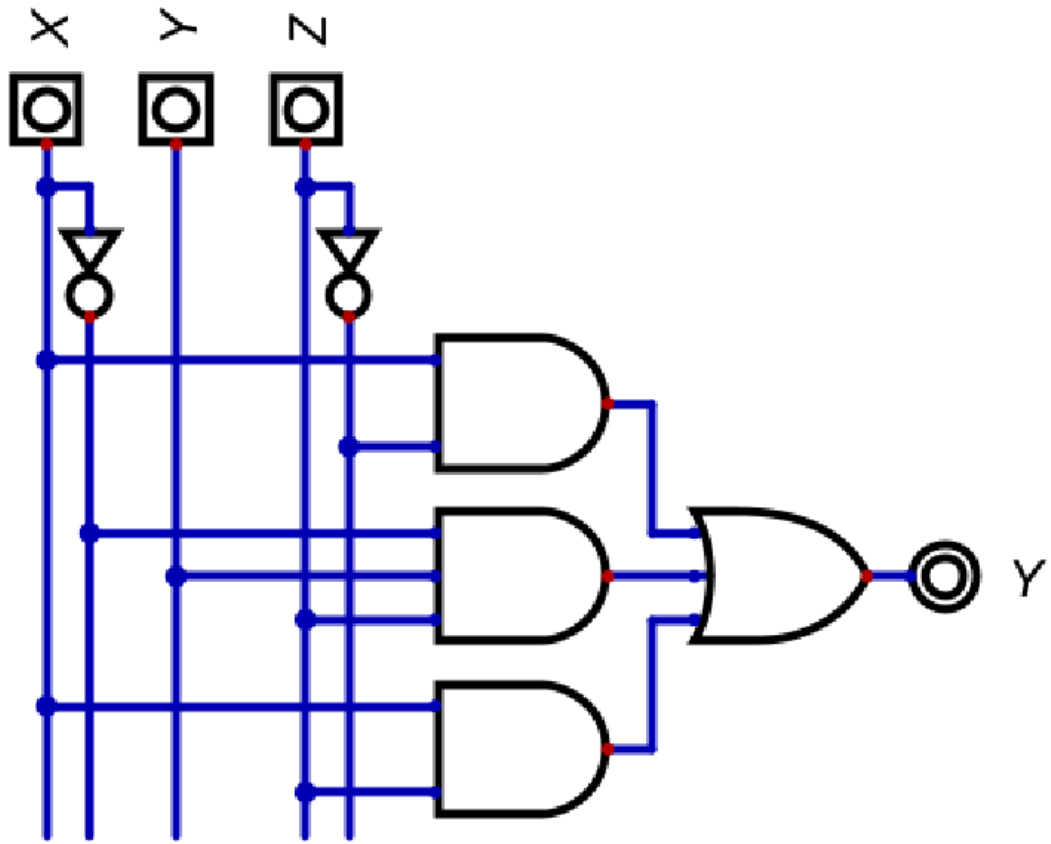
a) implement using, AND, OR, NOT Gates

- Draw variables and their complements VERTICALLY
- Connect variables to Gates according to boolean expression
- First create xz using AND
- second create x’y using AND
- connect to outputs to OR



FUNCTION to LOGIC Gates

Given F = x.z’ + x’.y.z + x.z



Algebraic Expression to Truth Table

- A truth table is a table of all possible combinations of the variables, showing the relation between the values that the variables may take and the result of the operation

Practice Exercise 2.2

Develop a truth table for the Boolean expression $F = x'y'z$.

Answer:

$x\ y\ z$	F
0 0 0	0
0 0 1	1
0 1 0	0
0 1 1	0
1 0 0	0
1 0 1	0
1 1 0	0
1 1 1	0

Algebraic Expression to Boolean Functions

A Boolean function described by an algebraic expression consists of binary variables, the constants 0 and 1, and the logic operation symbols. For a given value of the binary variables, the function can be equal to either 1 or 0.

As an example, consider the Boolean function

F1 = x + y′.z

F2 = x′.y′.z + x′.y.z + x.y′

Table 2.2
Truth Tables for F_1 and F_2

x	y	z	F_1	F_2
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	0
1	1	1	1	0

Truth Table to Algebraic Expression (Minterms)

- **A Boolean function can be expressed algebraically from a given truth table by forming a minterm for each combination of the variables that produces a 1 in the function and then taking the OR of all those terms.**

- **Given the truth table of F1, Find logical expression of F1**
- **Minterms are combination of variables (x,y,z..etc.) and their 'complements' with AND operator**
- **Answer: Find minterms=1 (F1=1) and OR them SO EASY!!**
  F1=x'y'z + xy'z' +xyz

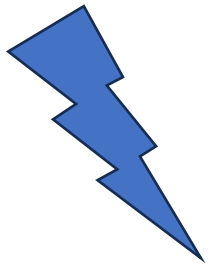


Table 2.4
Functions of Three Variables

x	y	z	Function f_1	Function f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	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

Table 2.3
Minterms and Maxterms for Three Binary Variables

			Minterms	
x	y	z	Term	Designation
0	0	0	$x'y'z'$	m_0
0	0	1	$x'y'z$	m_1
0	1	0	$x'yz'$	m_2
0	1	1	$x'yz$	m_3
1	0	0	$xy'z'$	m_4
1	0	1	$xy'z$	m_5
1	1	0	xyz'	m_6
1	1	1	xyz	m_7

X' i called as :

- **X's complement,**
- **X prime**
- **not X**

Truth Table to Algebraic Expression (Minterms)

- **Given the truth table of F2, Find logical expression of F2**

- **Answer: Find minterms=1 (F2=1) and OR them SO EASY!!**
  F2=x'yz + xy'z +xyz' + xyz

Table 2.4
Functions of Three Variables

x	y	z	Function f_1	Function f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	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

Truth Table to Algebraic Expression (Maxterms)

- **Given the truth table of F2, Find logical expression of F2**
- **Maxterms= combination of variables by OR**
- **Answer: Find maxterms=0 (F2=0), AND them all**
  F2=(x+y+z)(x+y+z')(x+y'+z)(x'+y+z)

Table 2.3
Minterms and Maxterms for Three Binary Variables

			Minterms		Maxterms	
x	y	z	Term	Designation	Term	Designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

Table 2.4
Functions of Three Variables

x	y	z	Function f_1	Function f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	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

Truth Table to Algebraic Expression (summary)

- Boolean functions expressed as a **sum of minterms** or **product of maxterms** are said to be in **canonical form**

- **We will prefer sum of minterms in this lecture**

- **F1=x'y'z + xy'z' + xyz**

Table 2.4
Functions of Three Variables

x	y	z	Function f_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

DESIGN GUIDE

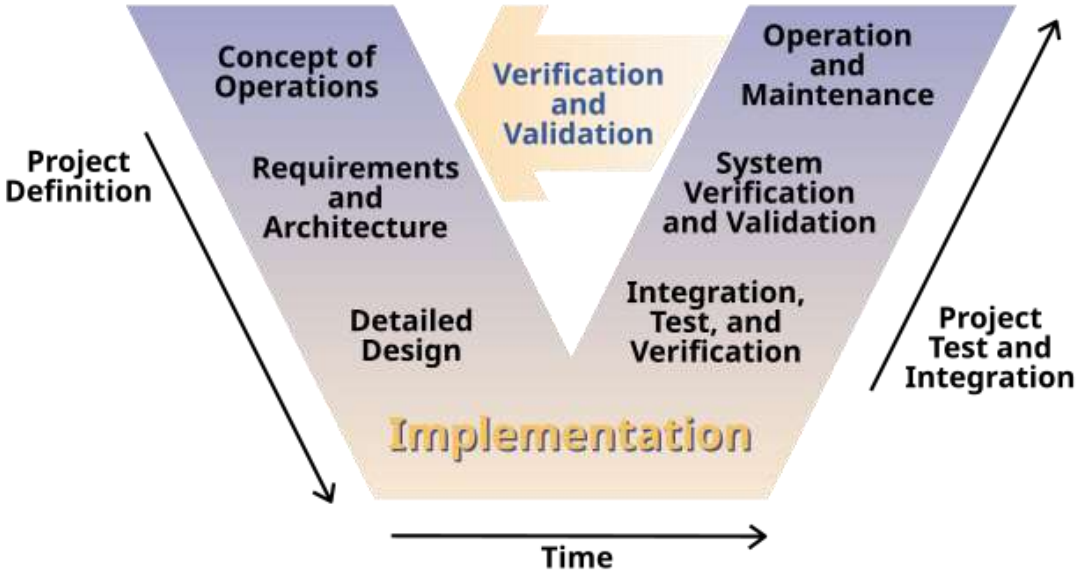
1. **What is the requirement: Description of function**

2. **Create using Truth Table**

3. **Write logical expression using Truth Table**

4. **Implement logical expression using AND, OR, NOT gates**

5. **Minimize number of gates**

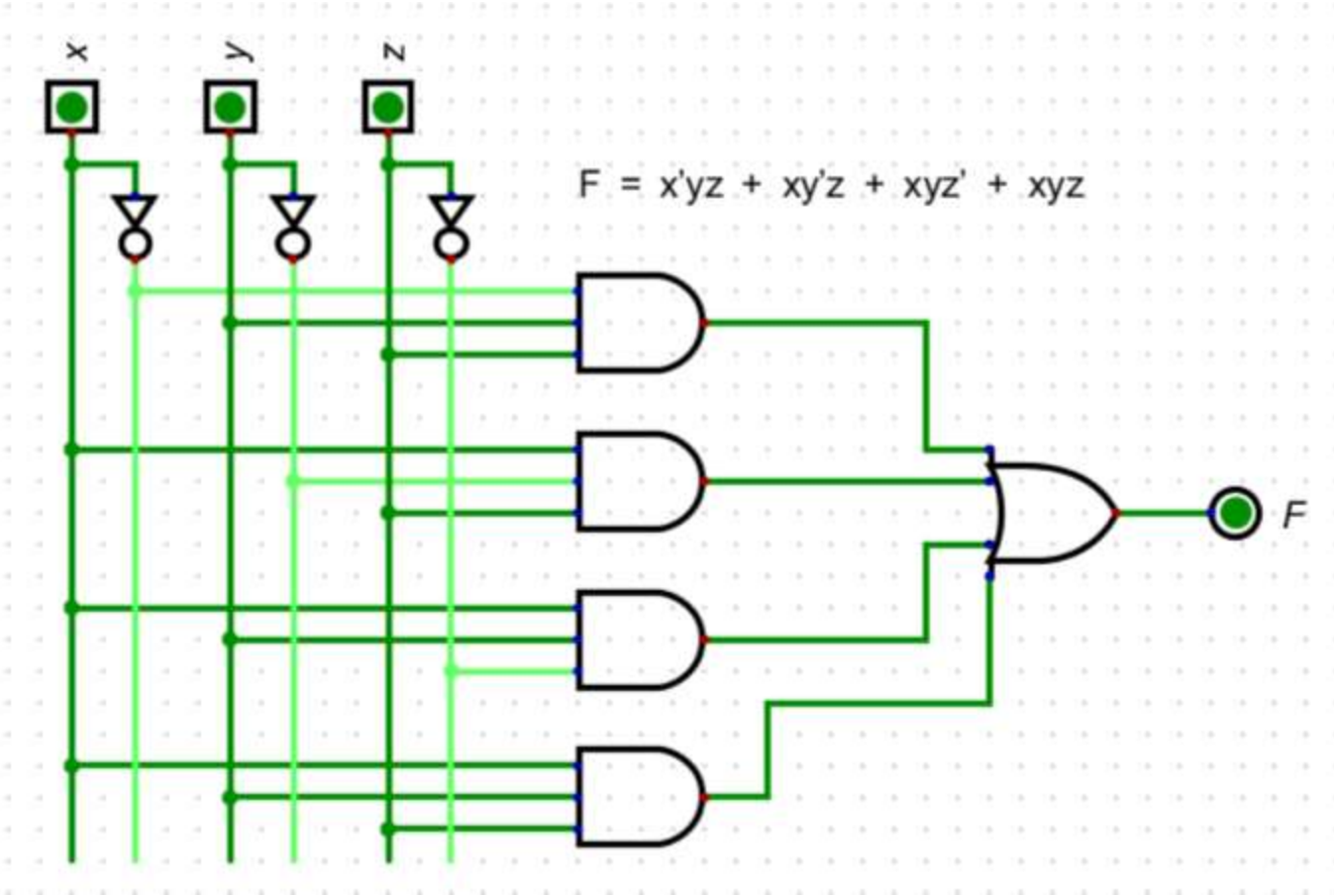


Truth Table to Algebraic Expression: TwoOne

- Create logic circuit with 3 input and gives 1 if two or more 1's

F = x'yz + xy'z + xyz' + xyz

x	y	z	F
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

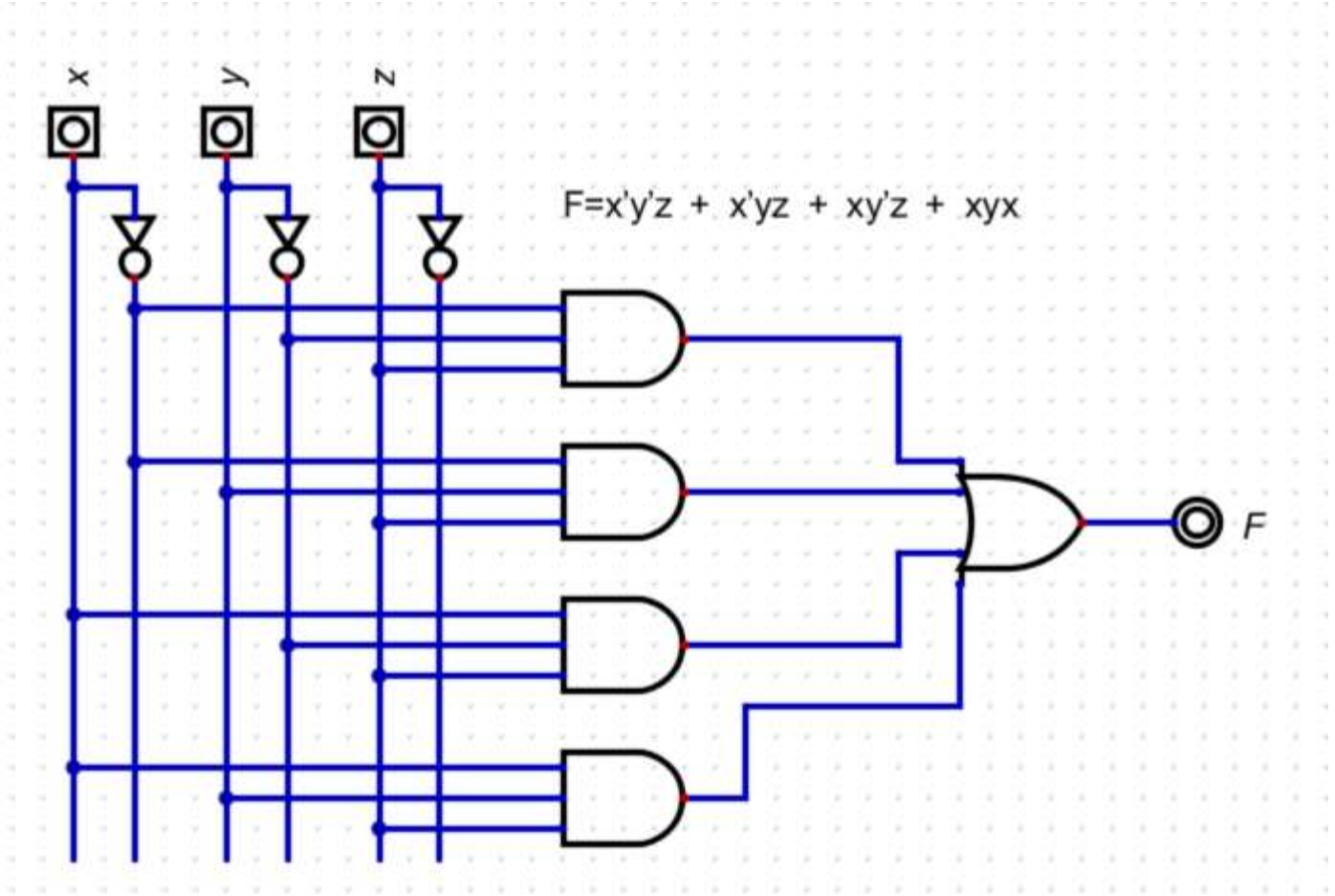


Truth Table to Algebraic Expression (catch odd)

- design a logic circuit which catches odd decimal numbers (1,3,5,7), 3-bit input

F=x'y'z + x'yz + xy'z + xyz

Dec	x	y	z	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1



Design Example

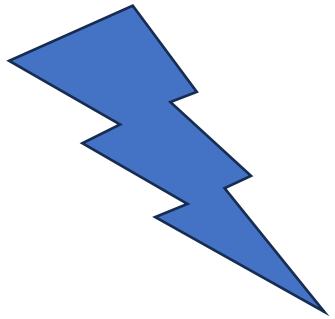
- F=1 if input decimal is above or equal 5

dec	x	y	z	F
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

F=xy’z + xyz’ + xyz

ANALOG to DIGITAL Converters (ADC)

- ADC convert continuous analog signals into discrete digital values
- Example 0-5V range with 3bit (=8) **resolution**
- 5V/7**=0.714V resolution**
- **So 3V is 101, as 2.9V is also 101**



Analog	x	y	z
0	0	0	0
0.714	0	0	1
1.428	0	1	0
2.142	0	1	1
2.857	1	0	0
3.571	1	0	1
4.285	1	1	0
5	1	1	1

ANALOG to DIGITAL Converters (ADC)

- ADC0831 convert continuous analog signals into discrete digital values
- Example 0-5V range with 8bit (=255) resolution
- 5V/255=0.02V resolution
- Vref can be equal to Vcc or different

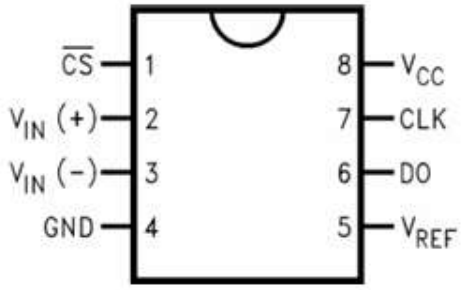
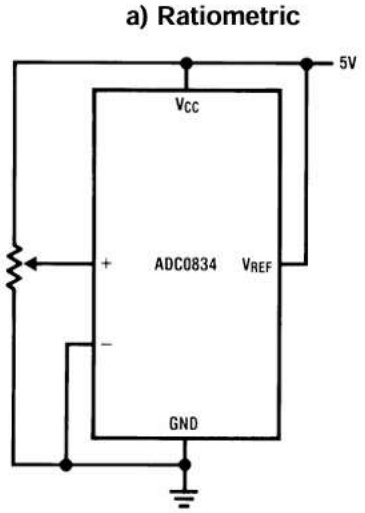


Figure 4. ADC0831-N Single Differential Input PDIP Package (P) Top View



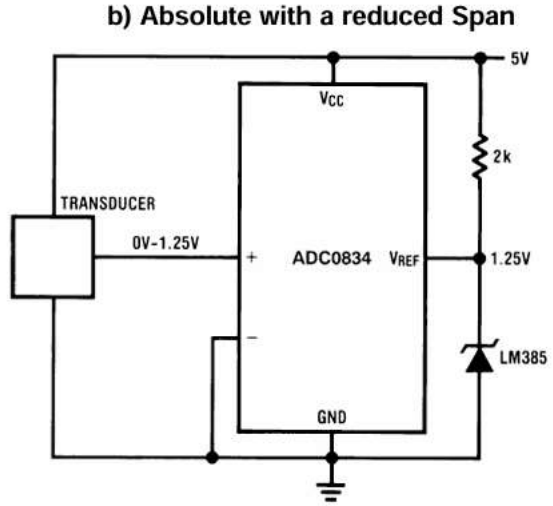


Figure 23. Reference Examples

ANALOG to DIGITAL Converters (ADC)

- ADC convert continuous analog signals into discrete digital values
- Example 0-5V range with 8bit (=255) resolution
- 5V/255=0.02V resolution

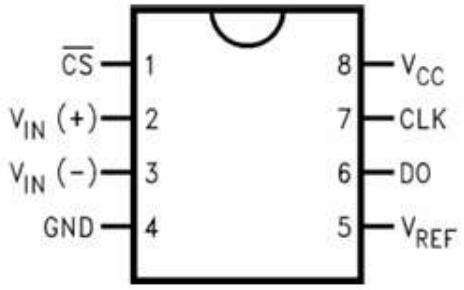
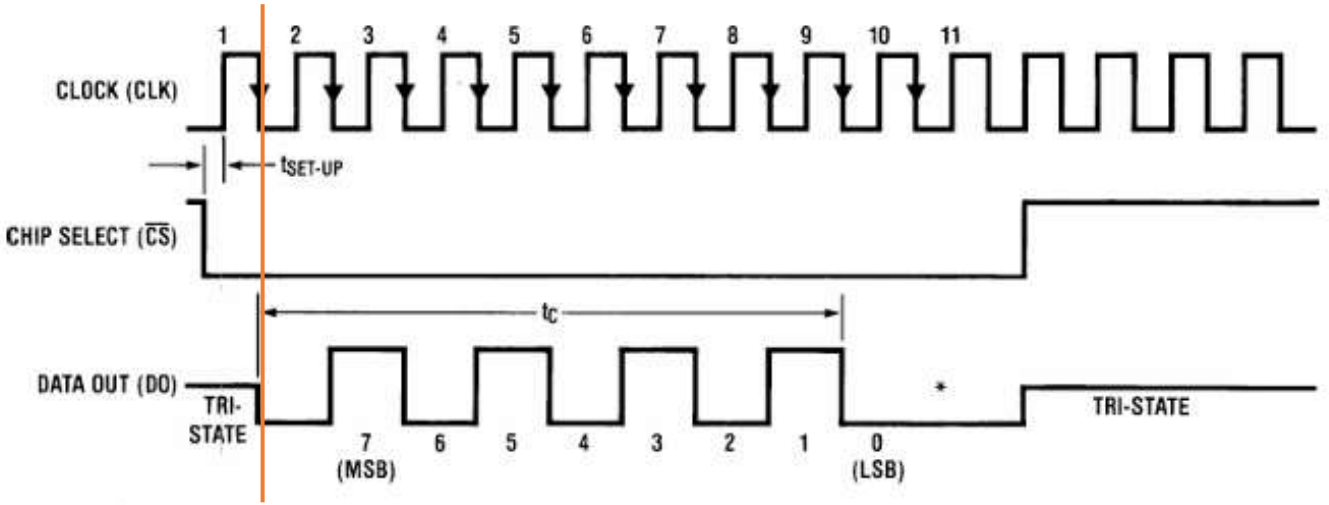


Figure 4. ADC0831-N Single Differential Input PDIP Package (P) Top View



*LSB first output not available on ADC0831-N.

Figure 18. ADC0831-N Timing

Design Example

- 3 bit ADC, 0-5V
- if analog volt is above 3V F=1 otherwise F=0

- resolution 3bit=7, 5V/7=0.7V

- above 3V, 3.5, 4.2, 5.0 volts

- we are loosing some analog information due to resolution

Volt	x	y	z	F
0.0	0	0	0	0
0.7	0	0	1	0
1.4	0	1	0	0
2.1	0	1	1	0
2.8	1	0	0	0
3.5	1	0	1	1
4.2	1	1	0	1
5.0	1	1	1	1

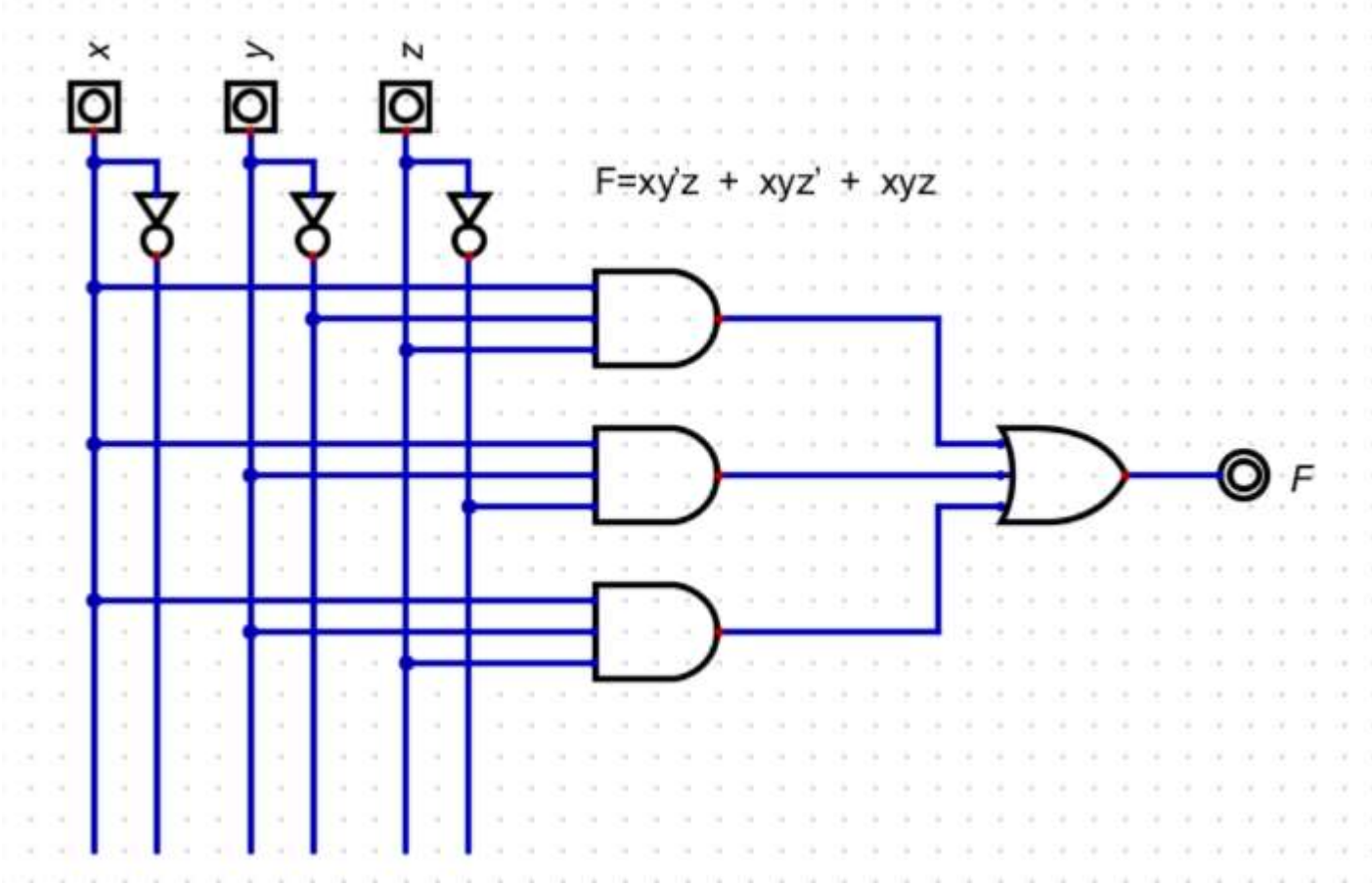
F=xy’z + xyz’ + xyz

Design Example

- 3 bit ADC, 0-5V
- if analog volt is above 3V F=1 otherwise F=0
- resolution 3bit=8, 5V/7=0.7V
- above 3V means 3bit is above 5=101

x	y	z	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

F=xy’z + xyz’ + xyz



2.7. Other Logical Operations

- We are not limited to AND OR NOT Gates, other Gates are also may help to reduce number of gates
- Let us evaluate all possible truth table combinations of two variable functions

Table 2.7
Truth Tables for the 16 Functions of Two Binary Variables

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

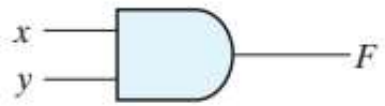
- AND, OR, NOT
- NAND, NOR
- XOR, XNOR

Table 2.8
Boolean Expressions for the 16 Functions of Two Variables

Boolean Functions	Operator Symbol	Name	Comments
$F_0 = 0$		Null	Binary constant 0
$F_1 = xy$	$x \cdot y$	AND	x and y
$F_2 = xy'$	x/y	Inhibition	x, but not y
$F_3 = x$		Transfer	x
$F_4 = x'y$	y/x	Inhibition	y, but not x
$F_5 = y$		Transfer	y
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive-OR	x or y, but not both
$F_7 = x + y$	$x + y$	OR	x or y
$F_8 = (x + y)'$	$x \downarrow y$	NOR	Not-OR
$F_9 = xy + x'y'$	$(x \oplus y)'$	Equivalence	x equals y
$F_{10} = y'$	y'	Complement	Not y
$F_{11} = x + y'$	$x \subset y$	Implication	If y, then x
$F_{12} = x'$	x'	Complement	Not x
$F_{13} = x' + y$	$x \supset y$	Implication	If x, then y
$F_{14} = (xy)'$	$x \uparrow y$	NAND	Not-AND
$F_{15} = 1$		Identity	Binary constant 1

2.7. Logic Gates

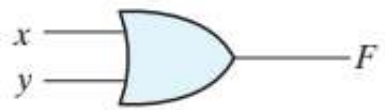
AND



$F = x \cdot y$

x	y	F
0	0	0
0	1	0
1	0	0
1	1	1

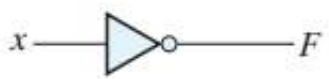
OR



$F = x + y$

x	y	F
0	0	0
0	1	1
1	0	1
1	1	1

Inverter



$F = x'$

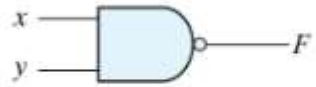
x	F
0	1
1	0

Buffer

$F = x$

x	F
0	0
1	1

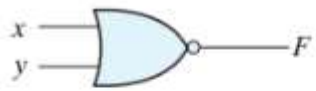
NAND



$F = (xy)'$

x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

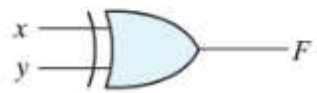
NOR



$F = (x + y)'$

x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

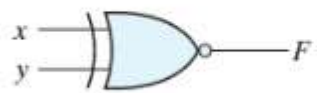
Exclusive-OR (XOR)



$F = xy' + x'y$
$= x \oplus y$

x	y	F
0	0	0
0	1	1
1	0	1
1	1	0

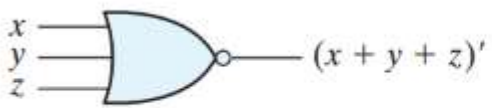
Exclusive-NOR or equivalence



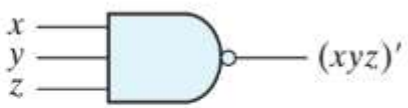
$F = xy + x'y'$
$= (x \oplus y)'$

x	y	F
0	0	1
0	1	0
1	0	0
1	1	1

2.7. Logic Gates (multiple input)



(a) 3-input NOR gate



(b) 3-input NAND gate

3. Gate Level Minimization

- Gate-level minimization is the design task of finding an optimal gate-level implementation of the Boolean functions describing a digital circuit.
- This task is well understood, but is difficult to execute by manual methods when the logic has more than a few inputs.
- Fortunately, this dilemma has been solved by computer-based logic synthesis tools that minimize a large set of Boolean equations efficiently and quickly.
- Nevertheless, it is important that a designer understands the underlying mathematical description and solution of the gatelevel minimization problem.

Huntington Postulates

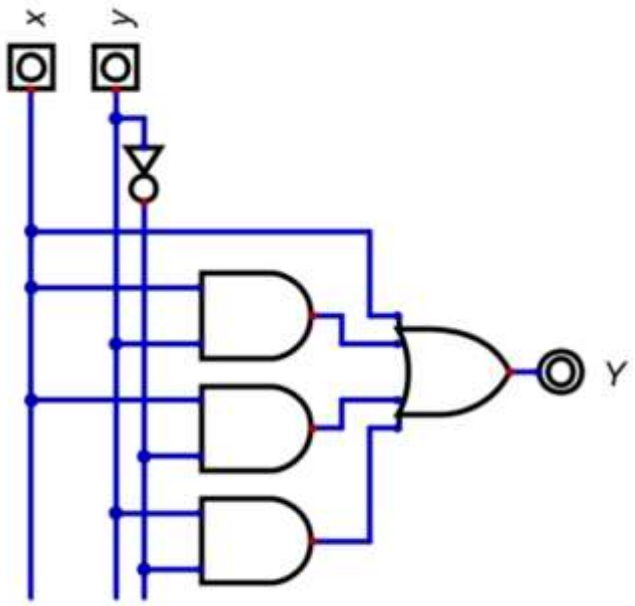
- Table 2.1 lists six theorems of Boolean algebra and four of its postulates.
- **The notation is simplified by omitting '.'**
- The theorems and postulates listed are the most basic relationships in Boolean algebra.
- The theorems, like the postulates, are listed in pairs; **each relation is the dual of the one paired with it**.
- The postulates are basic axioms of the algebraic structure and need no proof.
- The theorems must be proven from the postulates.

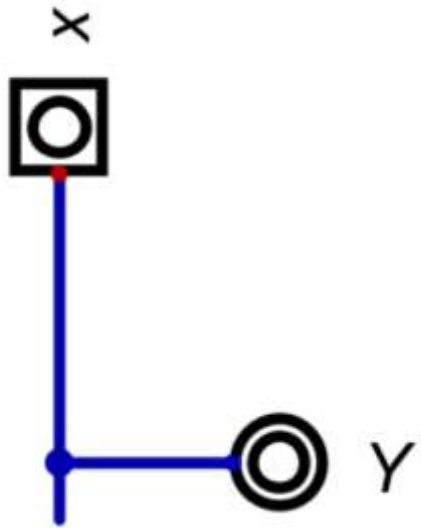
Table 2.1
Postulates and Theorems of Boolean Algebra

Postulate 2	(a)	$x + 0 = x$	(b)	$x \cdot 1 = x$
Postulate 5	(a)	$x + x' = 1$	(b)	$x \cdot x' = 0$
Theorem 1	(a)	$x + x = x$	(b)	$x \cdot x = x$
Theorem 2	(a)	$x + 1 = 1$	(b)	$x \cdot 0 = 0$
Theorem 3, involution		$(x')' = x$		
Postulate 3, commutative	(a)	$x + y = y + x$	(b)	$xy = yx$
Theorem 4, associative	(a)	$x + (y + z) = (x + y) + z$	(b)	$x(yz) = (xy)z$
Postulate 4, distributive	(a)	$x(y + z) = xy + xz$	(b)	$x + yz = (x + y)(x + z)$
Theorem 5, DeMorgan	(a)	$(x + y)' = x'y'$	(b)	$(xy)' = x' + y'$
Theorem 6, absorption	(a)	$x + xy = x$	(b)	$x(x + y) = x$

Algebraic Manipulation

3. $(x + y)(x + y') = x + xy + xy' + yy' = x(1 + y + y') = x.$





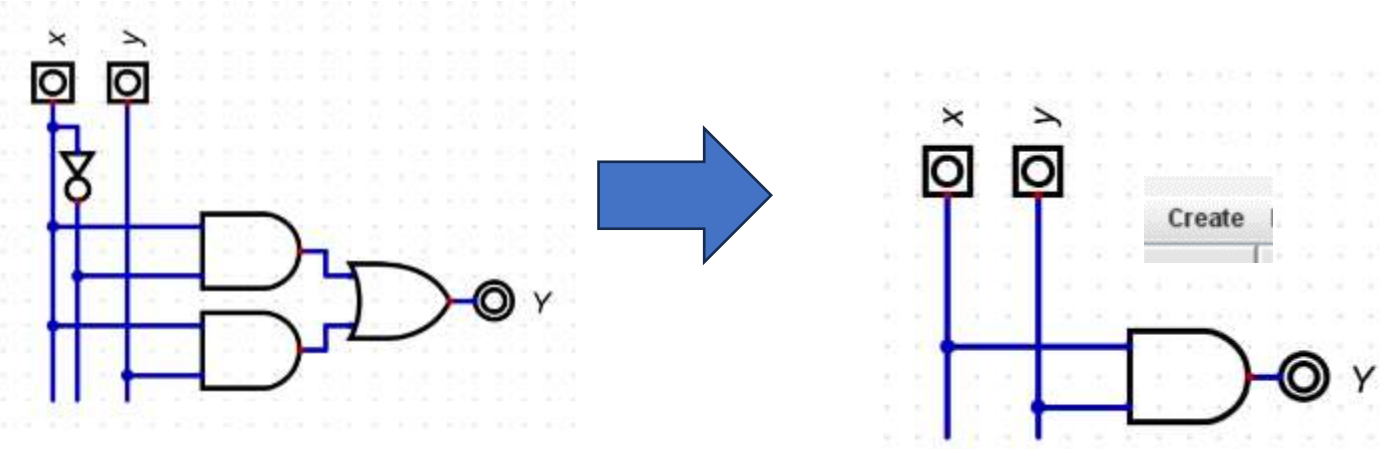
Minimization of Number of Gates

Finding the most economic representation of the logic is an important design task. By reducing the number of terms, the number of literals, or both in a Boolean expression, it is often possible to obtain a simpler circuit. The manipulation of Boolean algebra consists mostly of reducing an expression for the purpose of obtaining a simpler circuit

EXAMPLE 2.1

Simplify the following Boolean expressions to a minimum number of literals.

1. $x(x' + y) = xx' + xy = 0 + xy = xy.$



Algebraic Manipulation (Claude Shannon Paper)

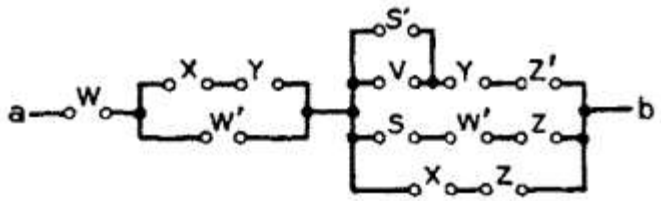


Figure 5. Circuit to be simplified

As an example of the simplification of expressions consider the circuit shown in Figure 5. The hindrance function X_{ab} for this circuit will be:

$$X_{ab} = W + W'(X + Y) + (X + Z)(S + W' + Z)(Z' + Y + S'V)$$
$$= W + X + Y + (X + Z)(S + 1 + Z)(Z' + Y + S'V)$$
$$= W + X + Y + Z(Z' + S'V) .$$

These reductions were made with 17b using first W, then X and Y as the "X" of 17b. Now multiplying out:

$$X + f(X,Y,Z,\ldots) = X + f(0,Y,Z,\ldots) , \qquad (17b)$$

$$X_{ab} = W + X + Y + ZZ' + ZS'V$$
$$= W + X + Y + ZS'V .$$

The circuit corresponding to this expression is shown in Figure 6. Note the large reduction in the number of elements.

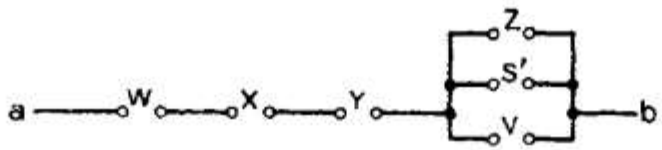


Figure 6. Simplification of figure 5

Algebraic Manipulation (Shannon Paper)

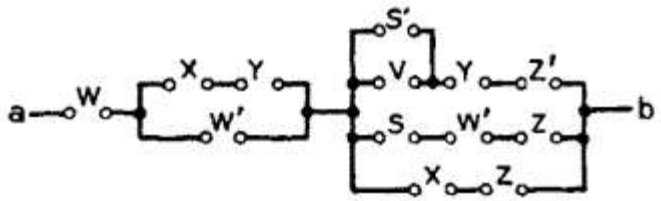


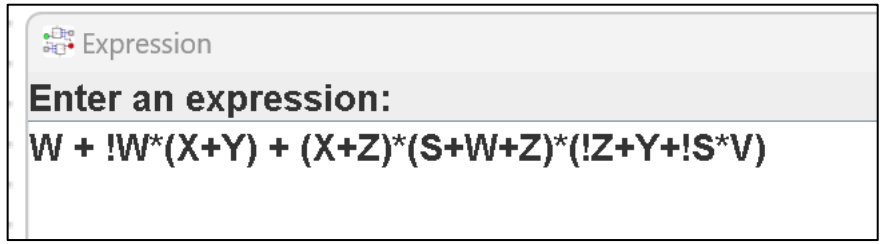
Figure 5. Circuit to be simplified

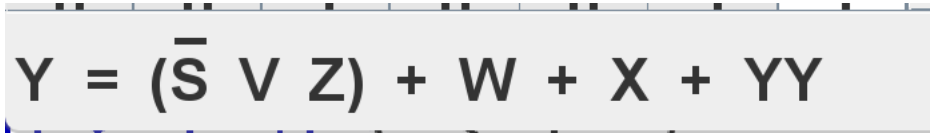
As an example of the simplification of expressions consider the circuit shown in Figure 5. The hindrance function X_{ab} for this circuit will be:

$$X_{ab} = W + W'(X + Y) + (X + Z)(S + W' + Z)(Z' + Y + S'V)$$

We can use Digital Logic Simulation software to simplify the expression in Shannon Paper

we used YY because output of the expression is default Y





The circuit corresponding to this expression is shown in Figure 6. Note the large reduction in the number of elements.

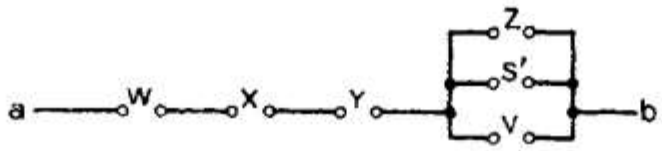


Figure 6. Simplification of figure 5

$$X_{ab} = W + X + Y + ZZ' + ZS'V$$
$$= W + X + Y + ZS'V .$$

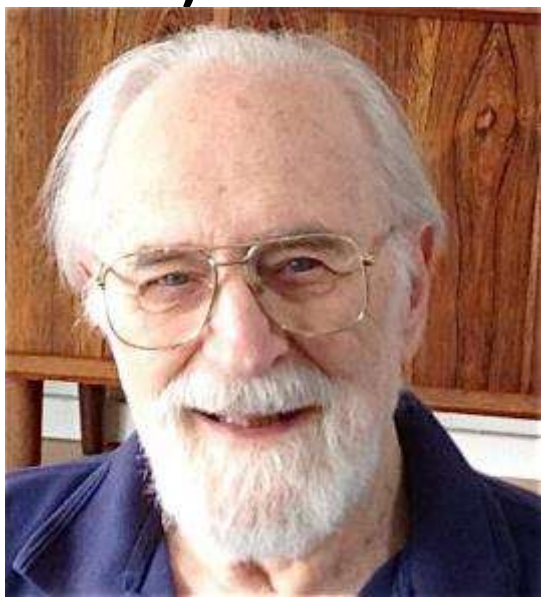
Algebraic Manipulation

- To reach minimum number of Gates using algebraic manipulation using analytical methods is not an easy way
- Thus we need to find a better way (Karnaugh Map)

Karnaugh Map

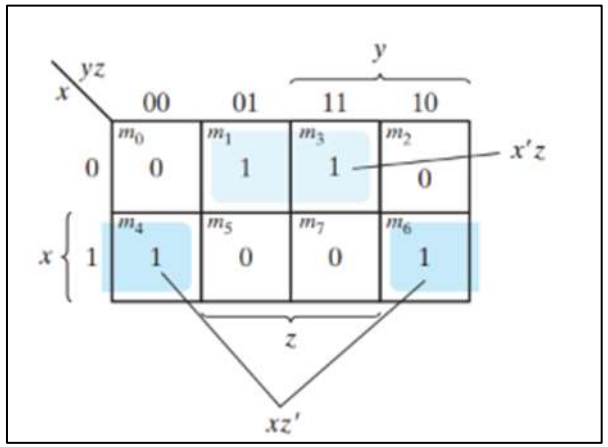
- Boolean expressions may be simplified by algebraic means as discussed in Section 2.4. However, this procedure of minimization is awkward, because it lacks specific rules to predict each succeeding step in the manipulative process.
- The map method presented in this section provides a simple, straightforward procedure for minimizing Boolean functions. This method may be regarded as a pictorial form of a truth table. The map method is also known as the Karnaugh map or K-map method presented in 1954 by

  Maurice Karnaugh (1924-2022)



Truth Table to Karnaugh Map

- Karnaugh Map is more compact form of representation of truth table
- F=xy'z' + xy'z + xyz' + xyz has four terms
- find the location of terms on Karnaugh map and set to 1
- adjacent cells can be combined to a shorter common term
- F=x
- FIND THE INTERSECTION FOR EACH EXPRESSION TERMS



			Minterms	
x	y	z	Term	Designation
0	0	0	$x'y'z'$	m_0
0	0	1	$x'y'z$	m_1
0	1	0	$x'yz'$	m_2
0	1	1	$x'yz$	m_3
1	0	0	$xy'z'$	m_4
1	0	1	$xy'z$	m_5
1	1	0	xyz'	m_6
1	1	1	xyz	m_7

x	y	z	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

F = xy'z' + xy'z + xyz' + xyz

	y'		y	
x'				
x	1	1	1	1
	z'	z		z'

Three Variable Karnaugh Map

- FIND THE COMMON VARIABLE(S) OF ADJACENT SQUARES

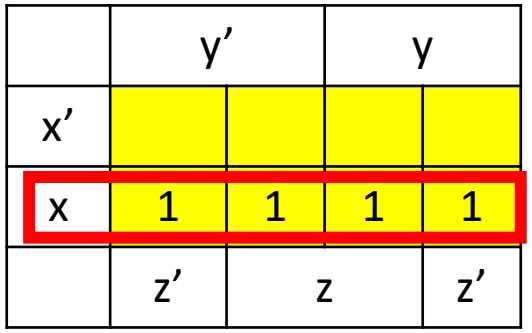
x	y	z	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

F = xy'z' + xy'z

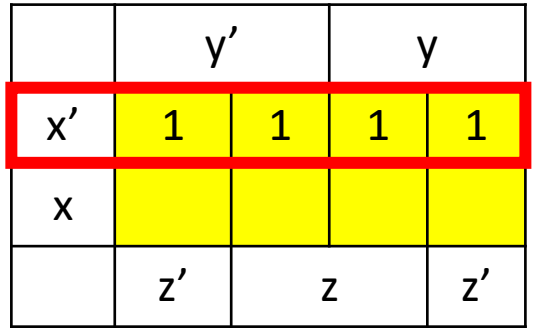
	y'		y	
x'				
x	1	1		
	z'	z		z'

Three Variable Karnaugh Map

- FIND THE COMMON VARIABLE(S) OF ADJACENT SQUARES



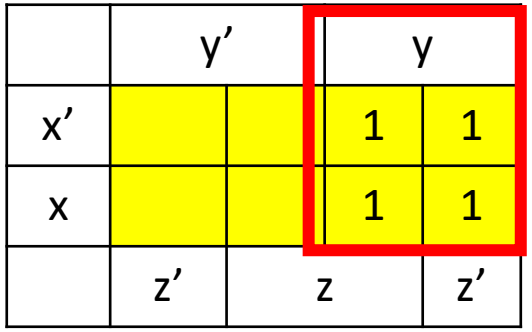
- **F1 = xy'z' + xy'z + xyz' + xyz**
- **F1 = x**



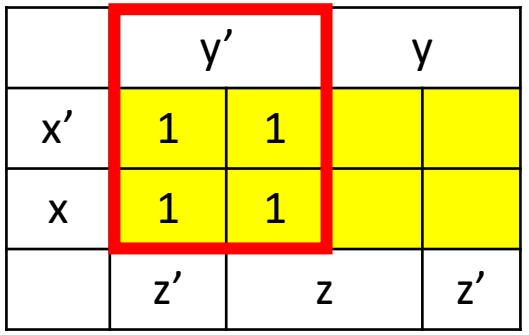
- F2=x'

Three Variable Karnaugh Map

- FIND THE COMMON VARIABLE(S) OF ADJACENT SQUARES



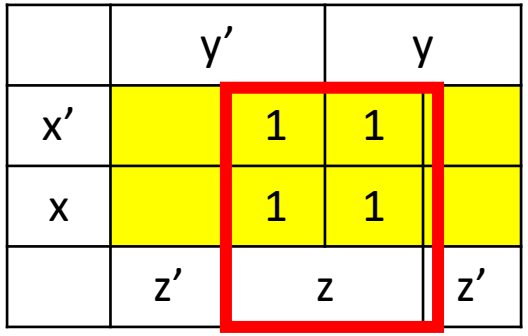
- F1=y



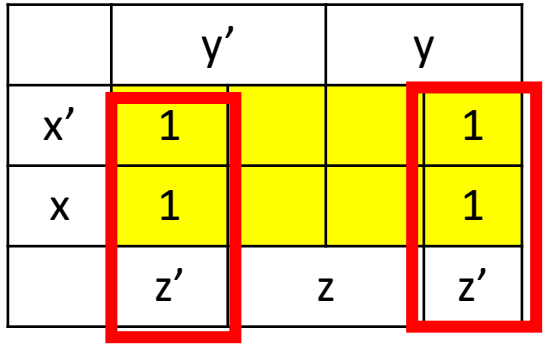
- F2=y'

Three Variable Karnaugh Map

- FIND THE COMMON VARIABLE(S) OF ADJACENT SQUARES



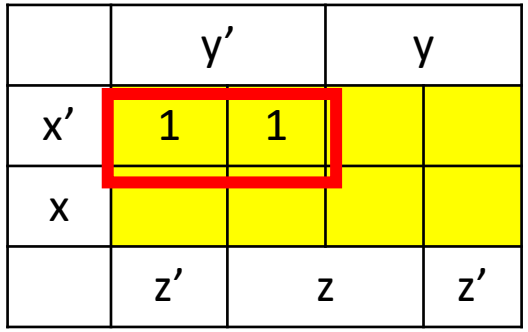
- F1=z



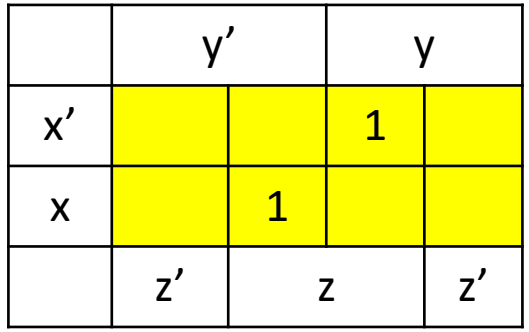
- F2=z'

Three Variable Karnaugh Map

- FIND THE COMMON VARIABLE(S) OF ADJACENT SQUARES



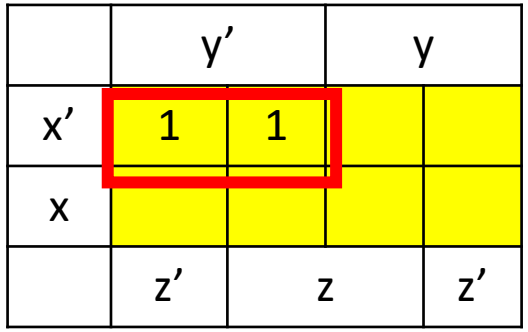
- F1=x'y'



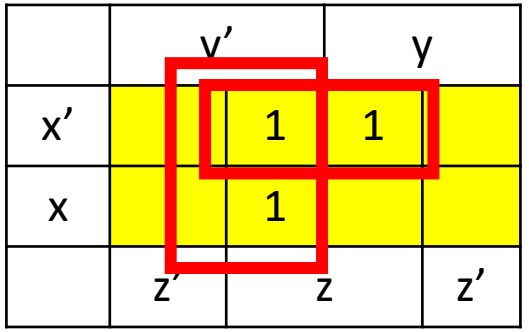
- F2= XY'Z + X'YZ

Three Variable Karnaugh Map

- FIND THE COMMON VARIABLE(S) OF ADJACENT SQUARES



- F1=x'y'



- F2 = X'Z + Y'Z

Three Variable Karnaugh Map

- FIND THE COMMON VARIABLE(S) OF ADJACENT SQUARES

	y’		y	
x’	1			
x	1			
	z’	z		z’

- F1=y’z’

	y’		y	
x’		1		
x		1		
	z’	z		z’

- F2=y’z

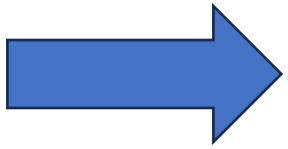
Three Variable Karnaugh Map

- Simplify the Boolean function F =x'yz' + x'yz + xy'z' + xy'z
- Focus on only 1's

x	y	z	F
0	0	0	
0	0	1	
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	
1	1	1	

- FIND ADJACENT SQUARES
- Any two adjacent squares in the map differ by only one variable
- Find the common variable of adjacent cell
- intersect with x

F= x'yz'+x'yz+xy'z'+xy'z



	y'		y	
x'			1	1
x	1	1		
	z'	z		z'

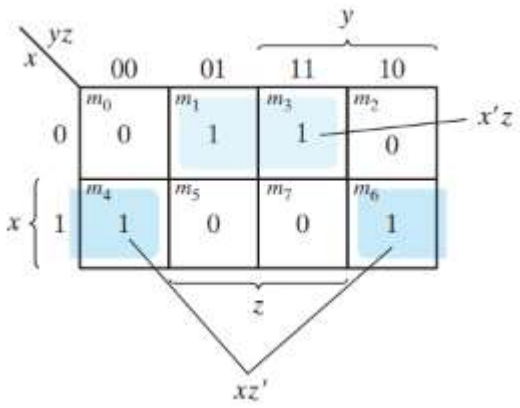
F=xy'+x'y

Three Variable Karnaugh Map

- Consider, for example, the truth table that defines the function F in Table 3.1. In sum-of-minterms form, this function is expressed as F = x'z + xz'

Table 3.1
Truth Table of Function F

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0



Three Variable Karnaugh Map

EXAMPLE 3.3

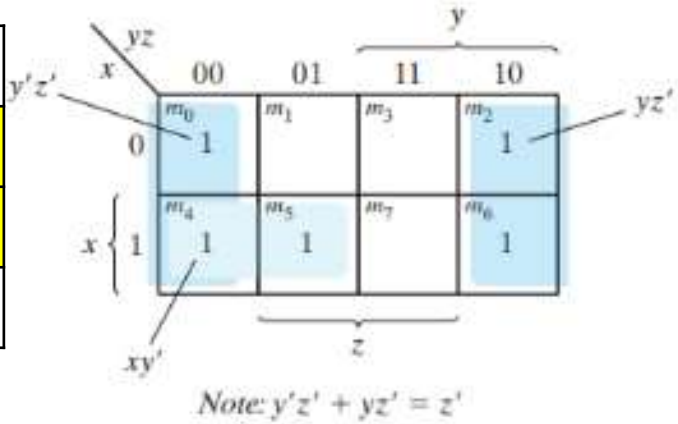
Simplify the Boolean function

$$F(x, y, z) = \Sigma(0, 2, 4, 5, 6)$$

F = x'y'z' + x'yz' + xy'z' + xy'z +xyz'

x	y	z	F
0	0	0	1
0	0	1	
0	1	0	1
0	1	1	
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	

	y'		y	
x'	1			1
x	1	1		1
	z'	z		z'



F = z' + xy'

Three Variable Karnaugh Map

EXAMPLE 3.4

For the Boolean function

$$F = A'C + A'B + AB'C + BC$$

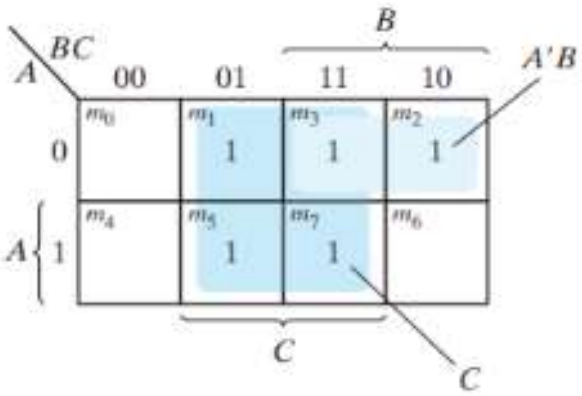
if expression is given we dont need to create to the truth table we can fill Karnaugh Map

F = x’z + x’y + xy’z +yz

	y’		y	
x’		1	1	1
x		1	1	
	z’	z		z’

F = z + x’y

$F = C + A'B$



Karnaugh Map (above 3V)

x	y	z	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

F = xy’z + xyz’ + xyz

	y’		y	
x’				
x		1	1	1
	z’	z		z’

F = xz + xy

Karnaugh Map (catch odd)

- design a logic which catches odd decimal numbers (1,3,5,7), 3-bit input

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

F=x'y'z + x'yz + xy'z + xyz

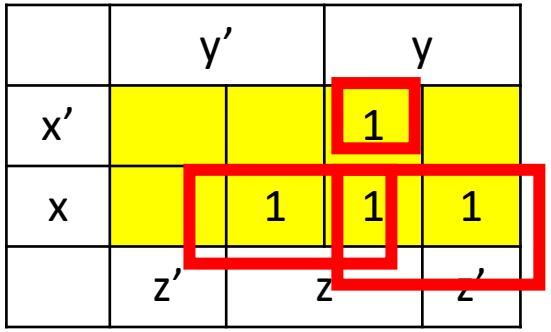
	y'		y	
x'		1	1	
x		1	1	
	z'	z		z'

F=z

Karnaugh Map: TwoOne

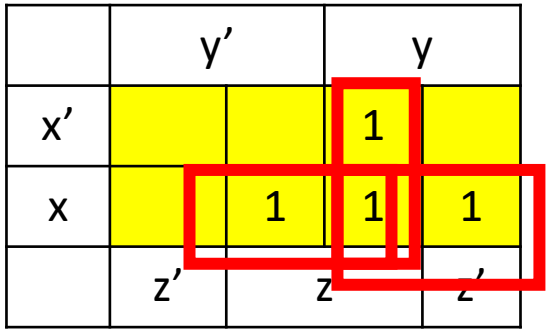
- Create logic circuit with 3 input and gives 1 if two or more 1's

x	y	z	F
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



$F = x'yz + xy'z + xyz' + xyz$

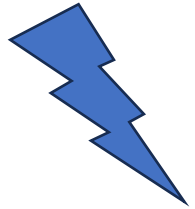
$F = x'yz + xz + xy$



F= yz + xz + xy minimum

Karnaugh Map: TwoOne

- Create logic circuit with 3 input and gives 1 if two or more 1's



x	y	z	F
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

$F = x'yz + xy'z + xyz' + xyz$

	y'		y	
x'			1	
x		1	1	1
	z'	z		z'

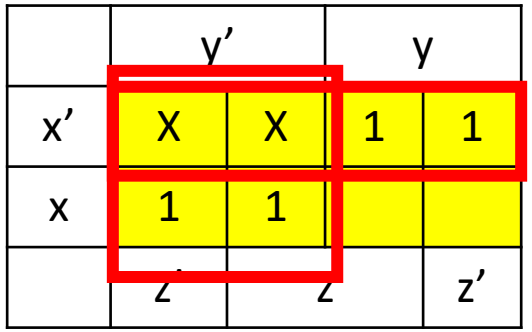
$F = yz + xz + xy$

Dont Care Conditions

- In practice, in some applications the function is not specified for certain combinations of the variables
- These don't-care conditions can be used on a map to provide further simplification of the Boolean expression. So you can assume 1 or 0 as you like for further simplification
- Consider following example

x	y	z	F
0	0	0	X=1
0	0	1	X=1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

$$F = x'y'z' + x'y'z + x'yz' + x'yz + xy'z' + xy'z$$



if X=0 F=xy'+x'y but if Xdontcare=1 we can further sımplify to

F = y' + x'

Dont Care Conditions

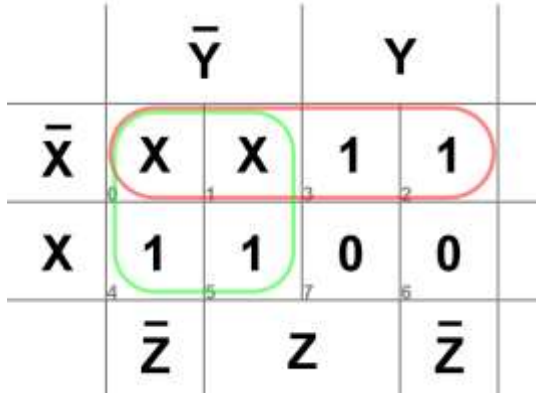
x	y	z	F
0	0	0	X=1
0	0	1	X=1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

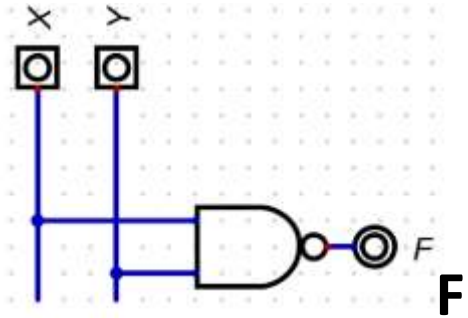
*Table

File New Edit Create K-Map

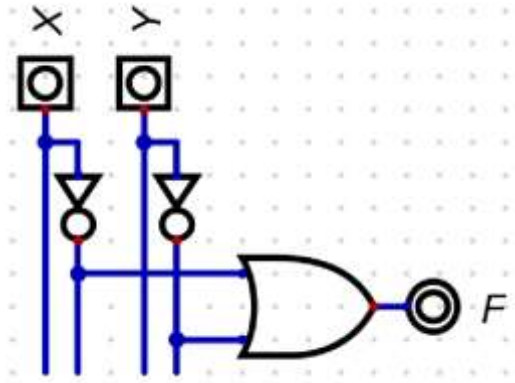
X	Y	Z	F
0	0	0	x
0	0	1	x
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

$F = \bar{X} + \bar{Y}$





F = (XY)'



F = X' + Y'

Four Variable Karnaugh Map

- **Maps for more than three variables are difficult to use and will not be considered in the lecture.**

NAND Gate

- The NAND gate is said to be a **universal gate because** any logic circuit can be implemented with it. To show that any Boolean function can be implemented with NAND gates, we need only to show that the logical operations of AND, OR, and complement can be obtained with NAND gates alone

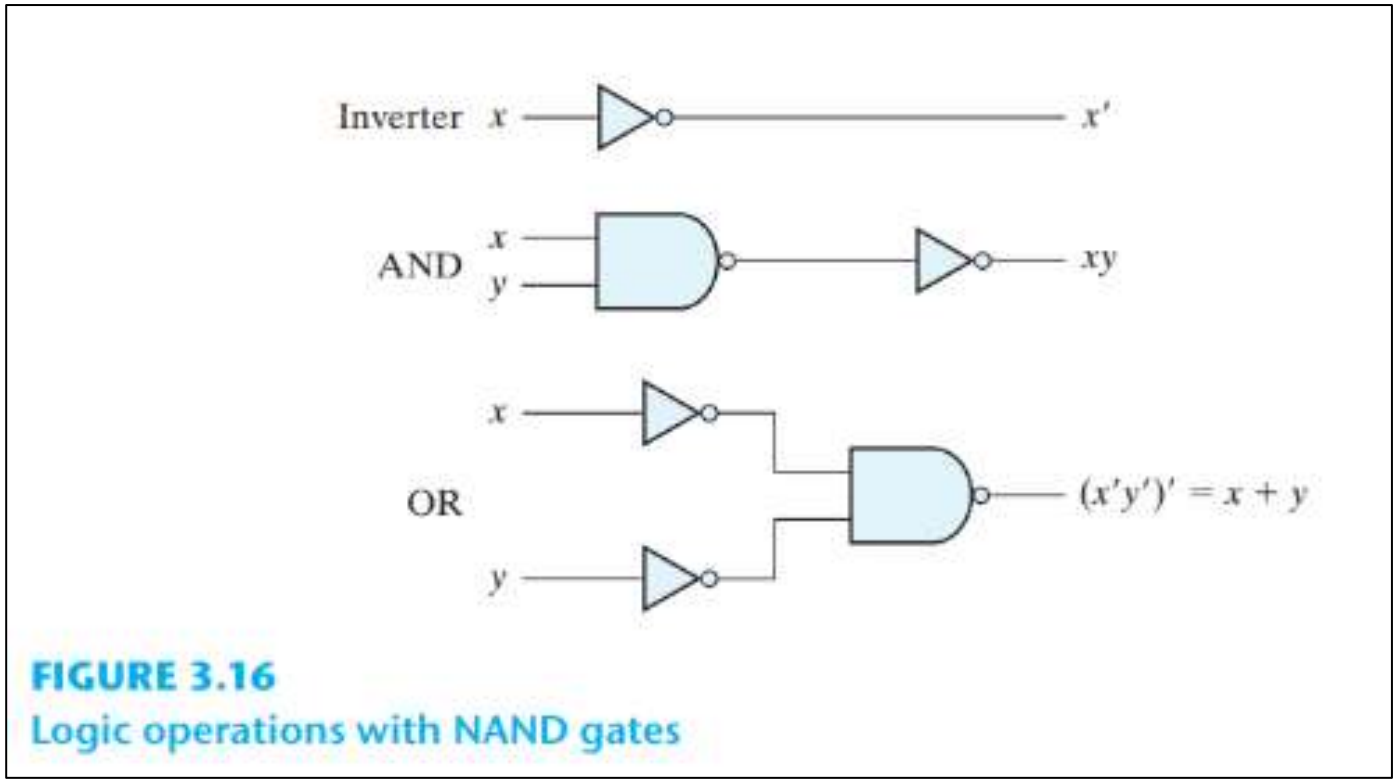


FIGURE 3.16
Logic operations with NAND gates

NAND Gate: Implementation

- The implementation of two-level Boolean functions with NAND gates requires that the functions be in sum-of-products form
- F=AB + CD

- The function it implements can easily be converted to sum-of-products form by ***DeMorgan's theorem***:
- F = ((AB)'(CD)')' = AB + CD

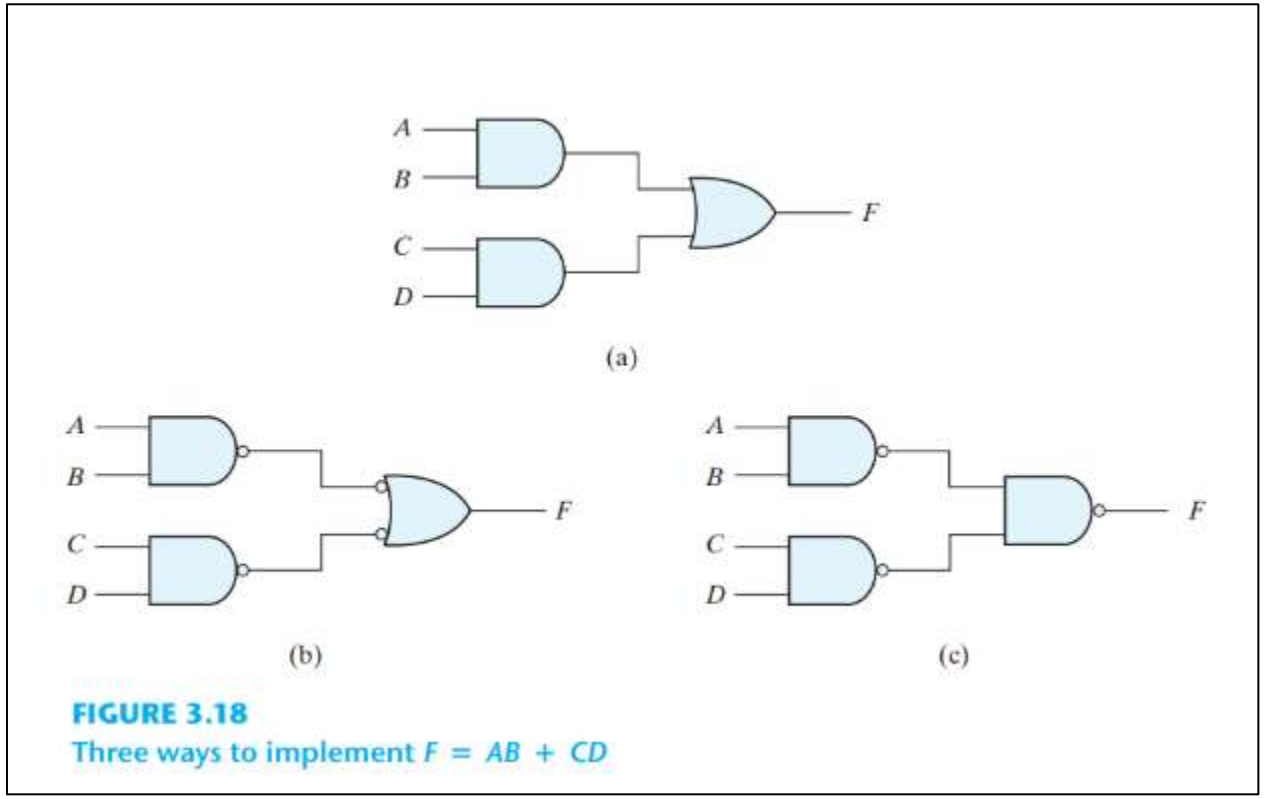


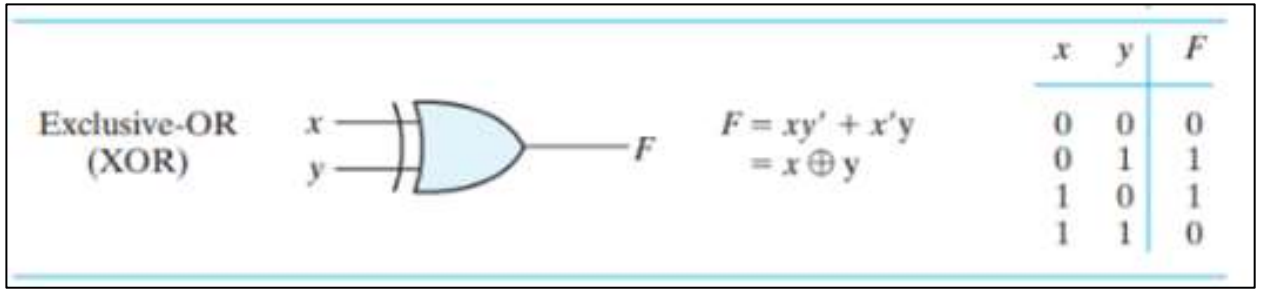
FIGURE 3.18
Three ways to implement $F = AB + CD$

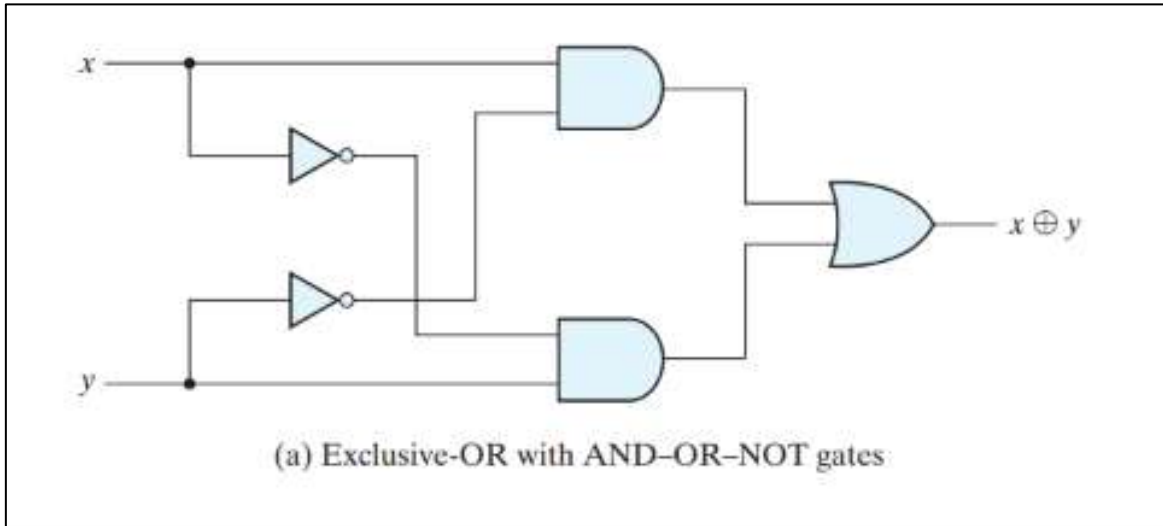
XOR Function

The exclusive-OR (XOR), denoted by the symbol $\oplus$, is a logical operation that performs the following Boolean operation:

$$x \oplus y = xy' + x'y$$

Exclusive-OR functions are very useful in systems requiring error detection and correction codes. As discussed in Section 1.7, a parity bit is used for the purpose of detecting errors during the transmission of binary information. A parity bit is an extra bit included with a binary message to make the number of 1's either odd or even.





(a) Exclusive-OR with AND–OR–NOT gates

XOR Function: Parity Generation and Checking

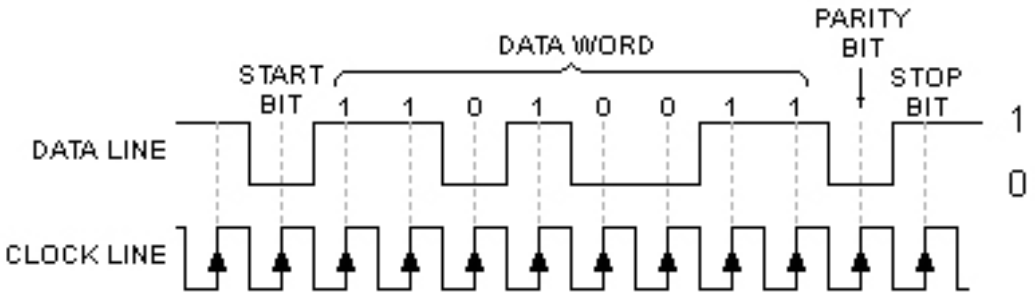
- Exclusive-OR functions are very useful in systems requiring error detection and correction codes.
- A parity bit is an extra bit included with a binary message to make the **number of 1's either odd or even**.
- The message, including the parity bit, is transmitted and then checked at the receiving end for errors.

- **make P=0 or 1 to have EVEN number of 1's**

RS-232 settings 8N1= 8 data, No parity, 1 stop
8E1: 8bit, Even Parity, 1 stop

Table 3.3
Even-Parity-Generator Truth Table

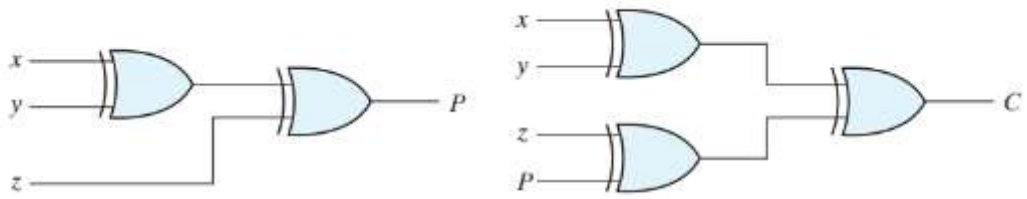
Three-Bit Message			Parity Bit
x	*y*	*z*	*P*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



XOR Function: Parity Generation and Checking

The three bits in the message, together with the parity bit, are transmitted to their destination, where they are applied to a parity-checker circuit to check for possible errors in the transmission. Since the information was transmitted with even parity, the four bits received must have an even number of 1's. An error occurs during the transmission if the four bits received have an odd number of 1's, indicating that one bit has changed in value during transmission

C=1 means there is an ERROR in XYZP



(a) 3-bit even-parity generator (b) 4-bit even-parity checker

FIGURE 3.34
Logic diagram of a parity generator and checker

Table 3.4
Even-Parity-Checker Truth Table

Four Bits Received				Parity Error Check
x	*y*	*z*	*P*	*C*
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Logic Gates Applications (XOR gate)

9.1 Application Information

In this application, a 2-input XOR gate is used as a phase difference detector as shown in *Figure 7*. The remaining three gates can be used for other applications in the system, or the inputs can be grounded and the channels left unused.

The SN74HC86-Q1 is used to identify phase difference between a reference clock and another input clock. Whenever the clock states are different, the XOR output will pulse HIGH until the clocks return to the same state. The output is fed into a low-pass filter to obtain a DC representation of the phase difference.

9.2 Typical Application

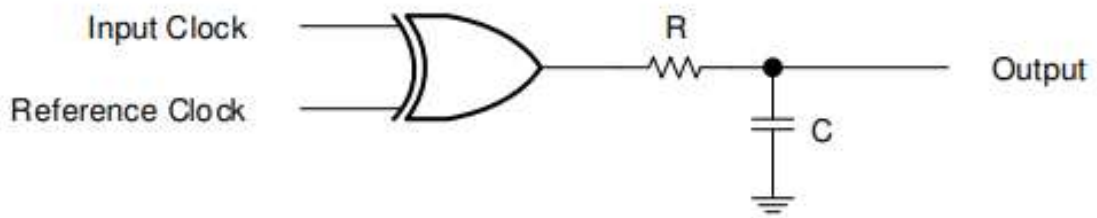
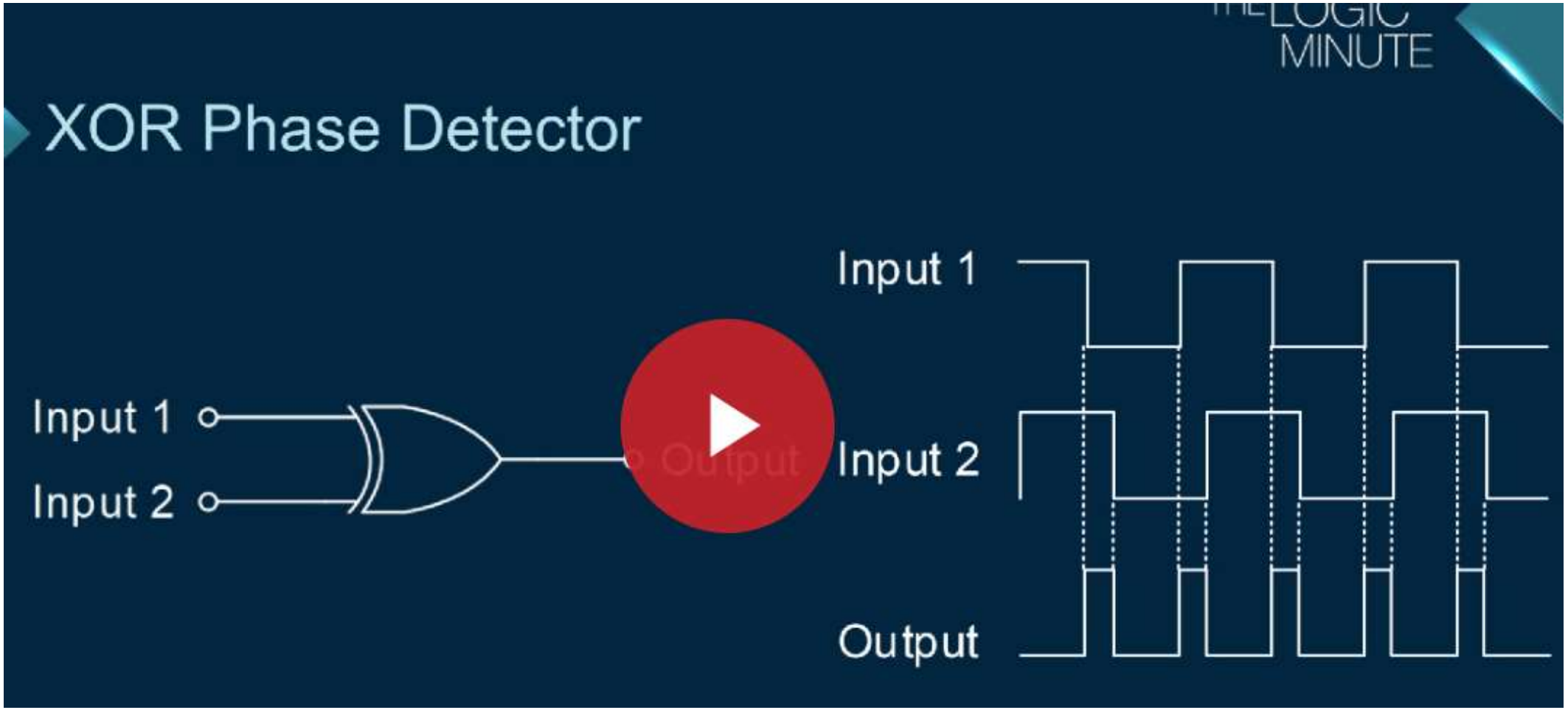


Figure 7. Typical application block diagram

Logic Gates Applications (XOR gate)



Logic Gates Applications (XOR gate)

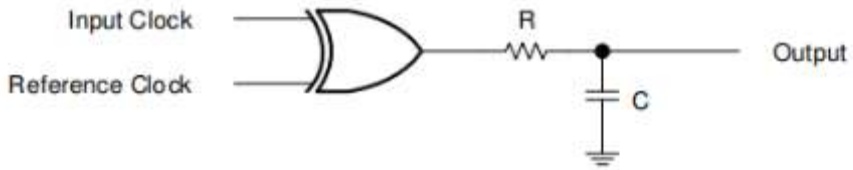
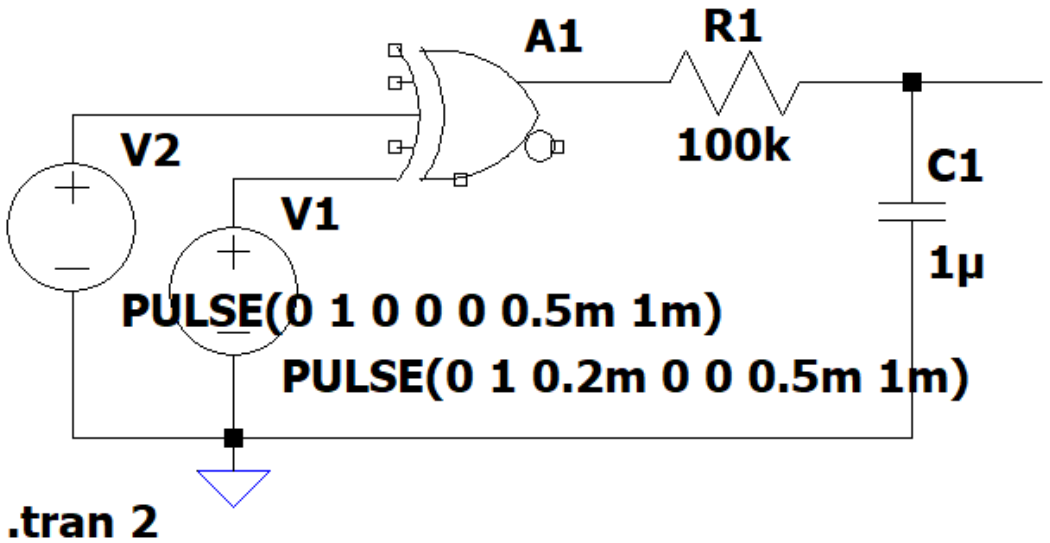
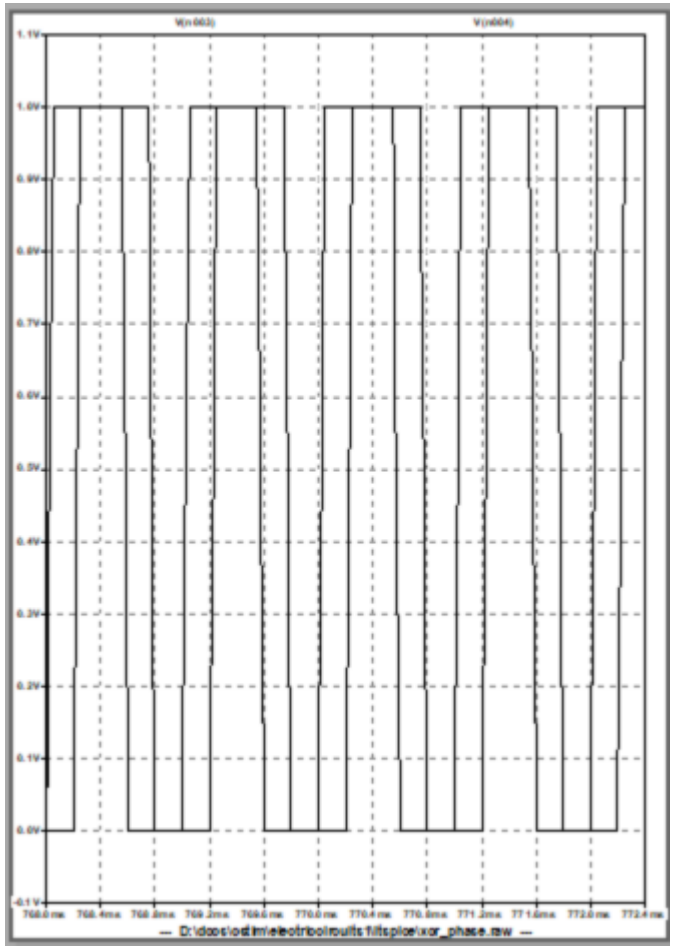


Figure 7. Typical application block diagram



Logic Gates Applications (XOR gate)



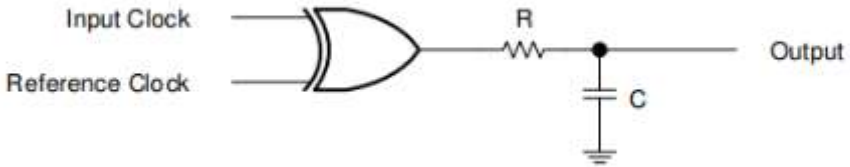
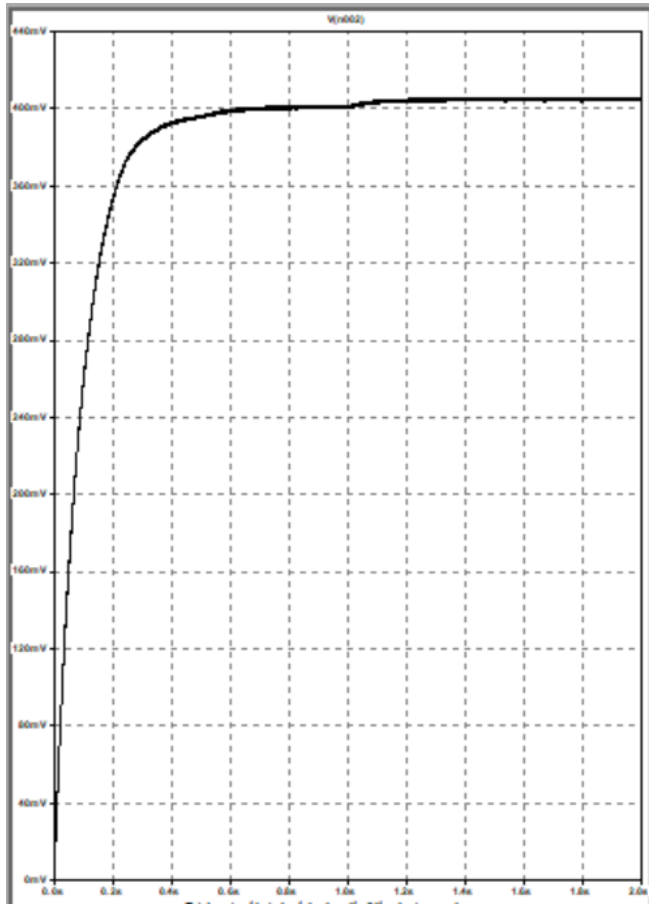
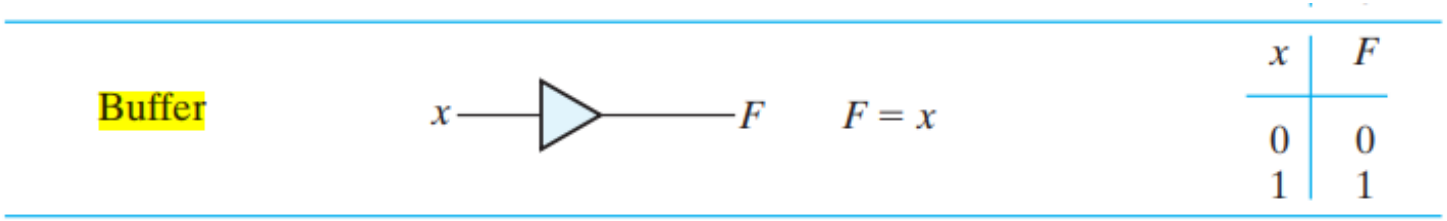


Figure 7. Typical application block diagram



Buffer / Three State Buffer

- buffer produces the transfer function, but does not produce a logic operation, since the binary value of the output is equal to the binary value of the input. This circuit is **used for power amplification** of the signal and is equivalent to two inverters connected in cascade.
- three-state buffer gate is shown in Fig. 4.29. It is distinguished from a normal buffer by an input control line entering the bottom of the symbol. The buffer has a normal input, an output, and a control input that determines the state of the output. When the control input is equal to 1, the output is enabled and the gate behaves like a conventional buffer, with the output equal to the normal input. When **the control input is 0, the output is disabled and the gate goes to a high-impedance state, regardless of the value in the normal input. this is important for BUS connections. listener has to go high impedance not to disturb others**



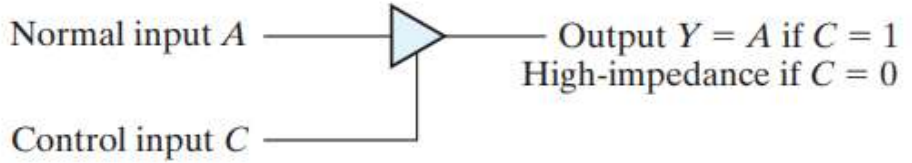
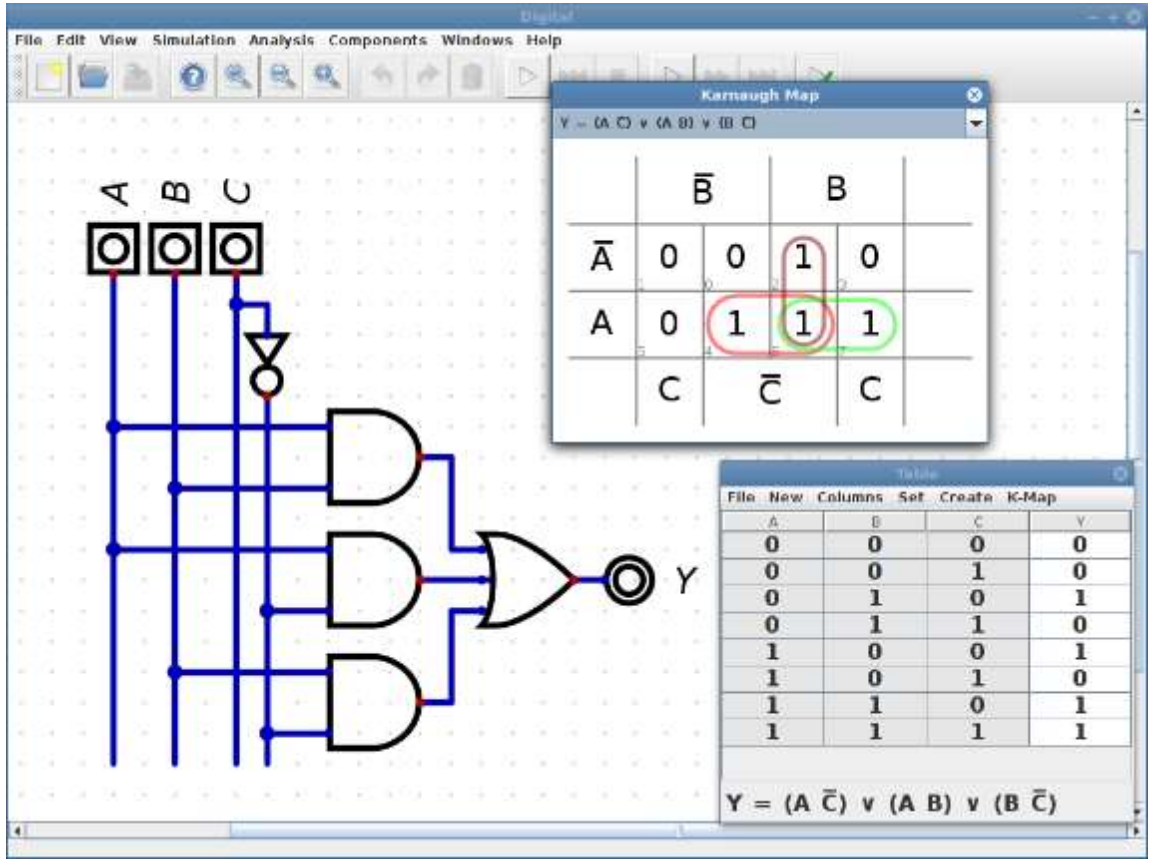


FIGURE 4.29
Graphic symbol for a three-state buffer

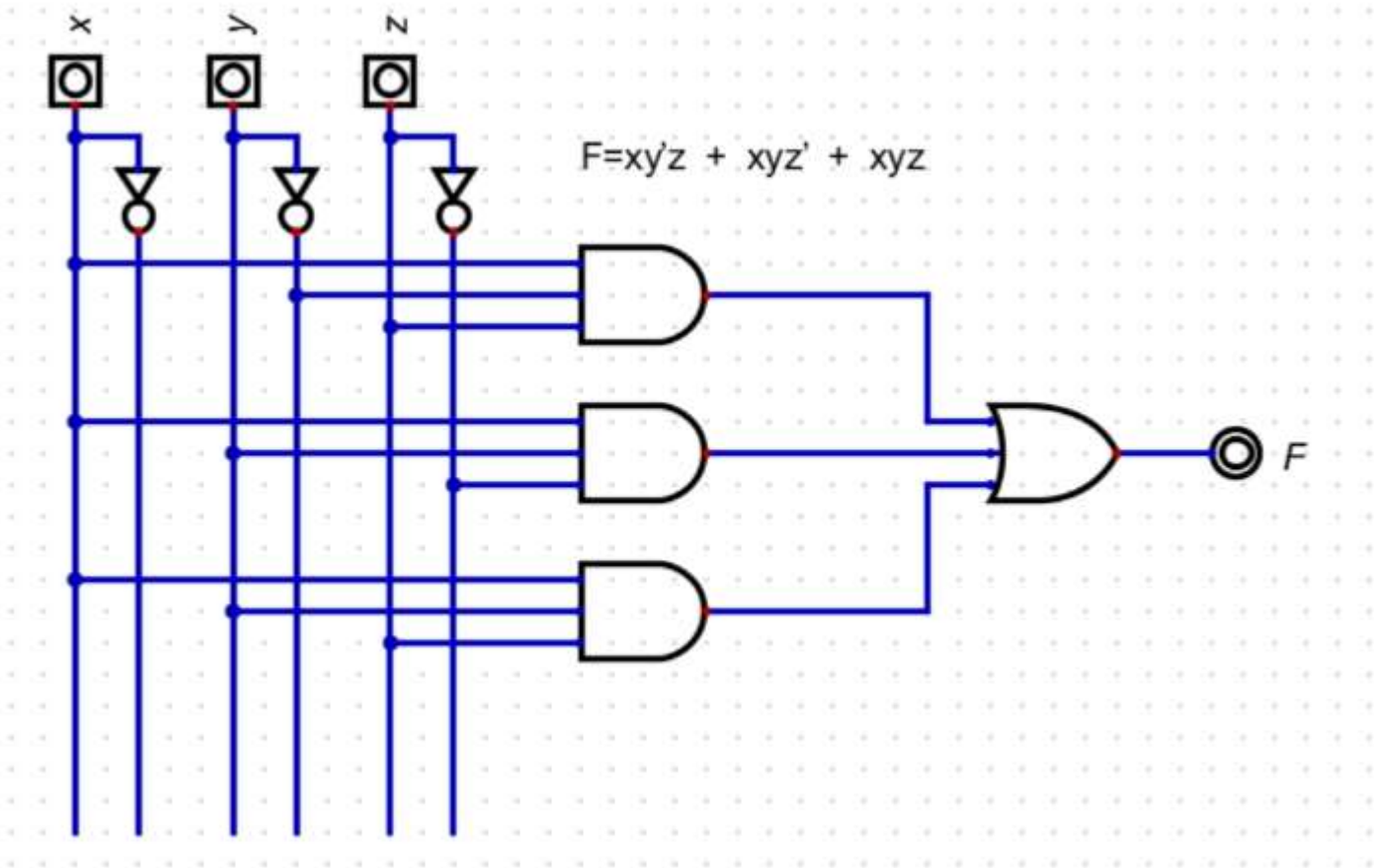
Logic Simulator (GitHub - hneemann/Digital: A digital logic designer and circuit simulator.)

Digital is an easy-to-use digital logic designer and circuit simulator designed for educational purposes.



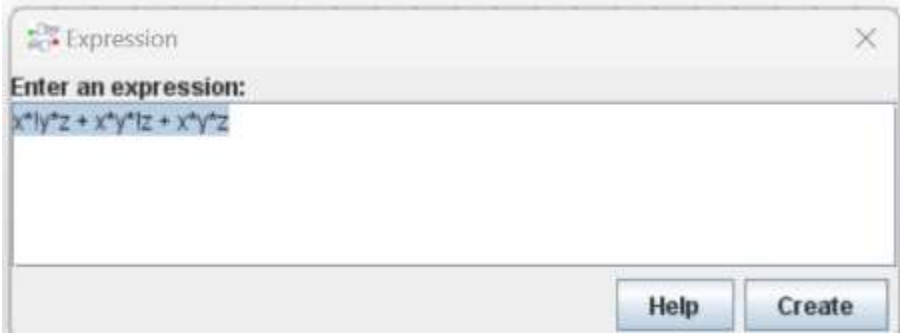
Logic Simulator-Manual

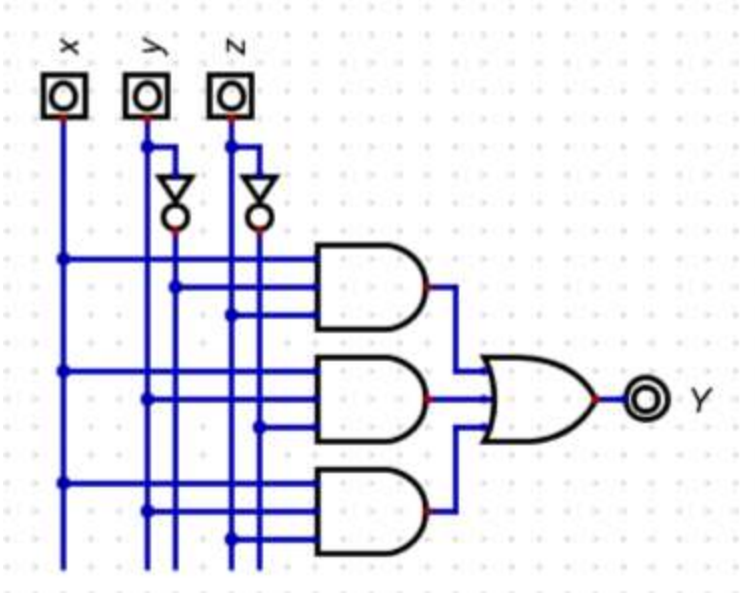
if Boolean expression is given circuit can be implemented using components and connections



Logic Simulator-Synthesis

We can create the circuit automatically using logical **Expression**
Careful about the syntax : *=AND, +=OR , !=NOT
Remember the circuit for above 3V: F=xy'z + xyz' + xyz
x*!y*z + x*y*!z + x*y*z





Logic Simulator-Truth Table

We can create the circuit automatically using truth table
Minimization done automatically using Karnaugh Map

*Table

File New Edit Create K-Map

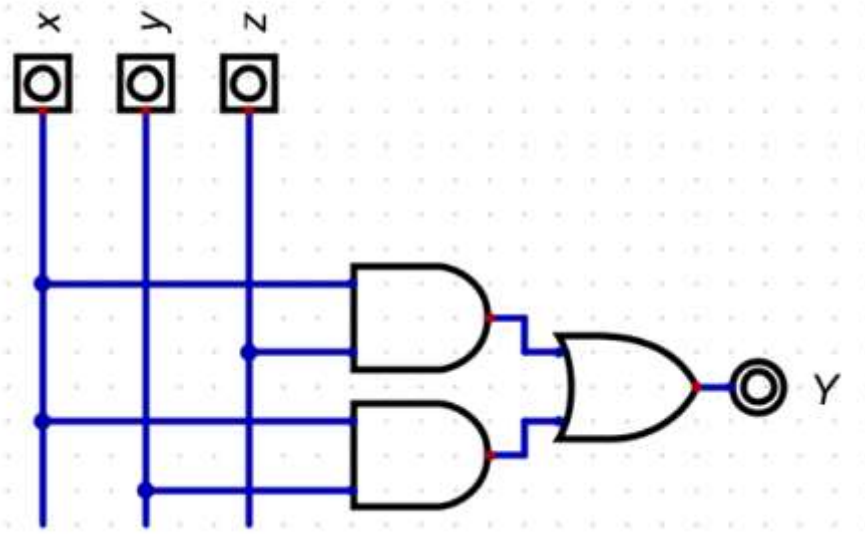
x	y	z	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Y = (x z) + (x y)

Karnaugh Map

Y = (x z) + (x y)

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

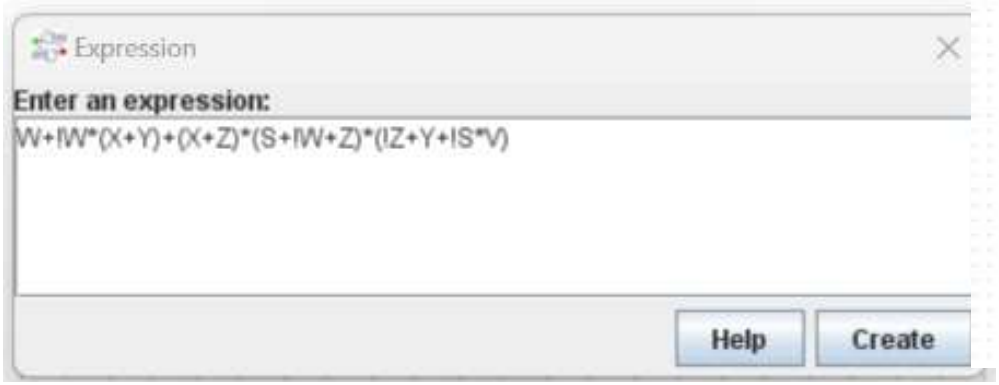


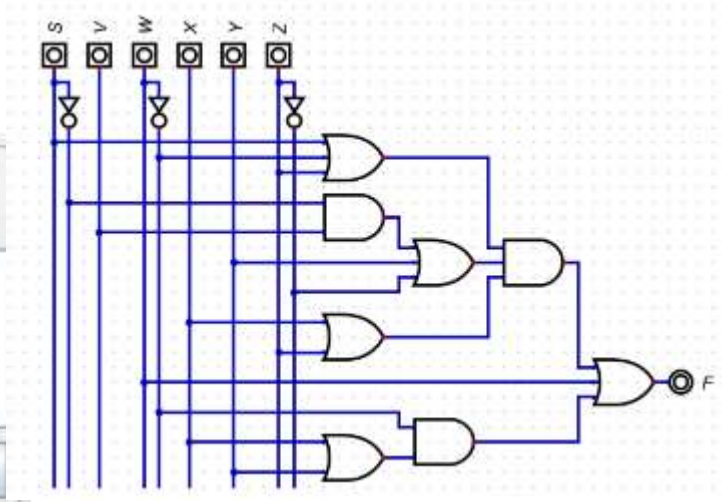
Logic Simulator-Claude Shannon Paper

- Lets have a look to the expression in Shannon paper
- write the expression in syntax
- create the circuit
- look at truth table and minimized expression
- We have the same results as in Shannon Paper

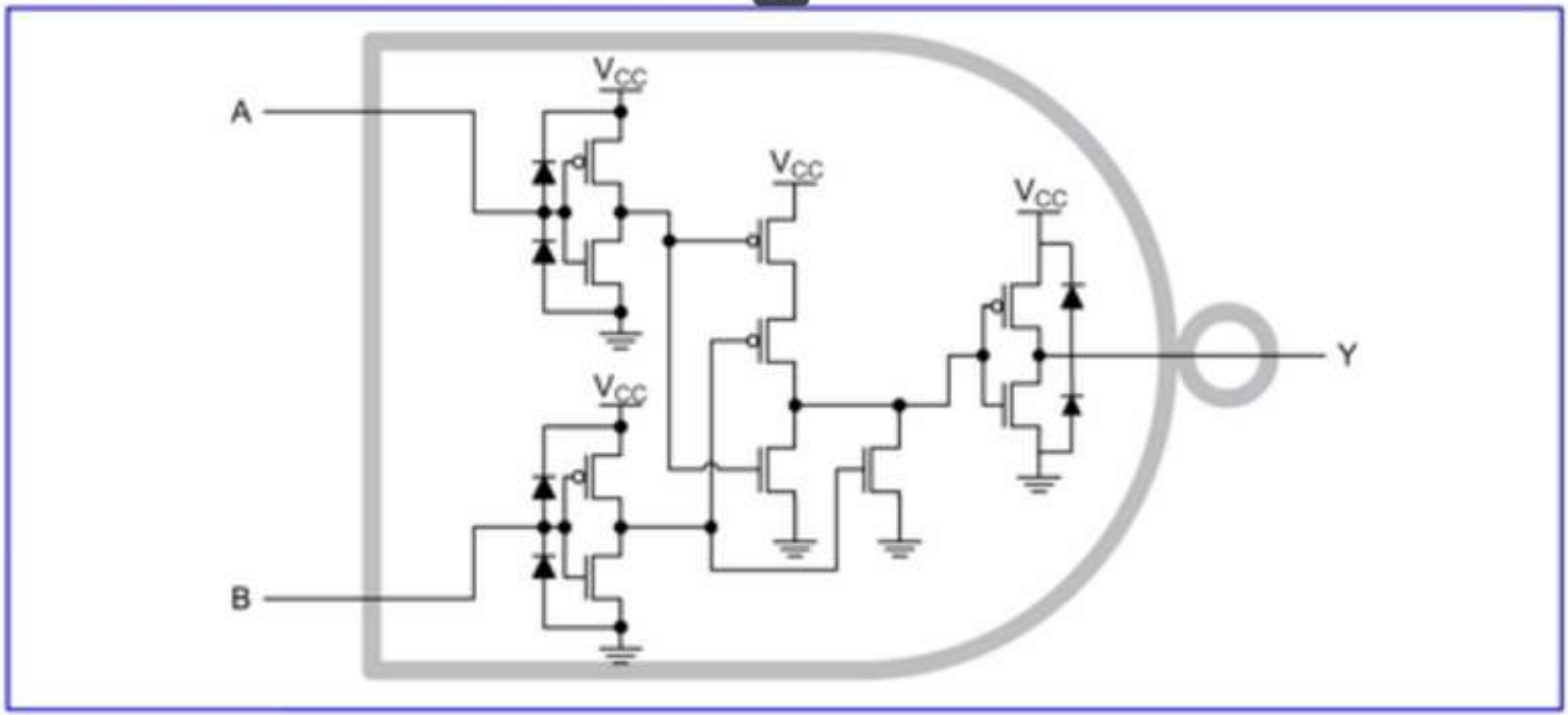
As an example of the simplification of expressions consider the circuit shown in Figure 5. The hindrance function X_{ab} for this circuit will be:

$$X_{ab} = W + W'(X + Y) + (X + Z)(S + W' + Z)(Z' + Y + S'V)$$
$$= W + X + Y + ZS'V .$$





What is inside a typical logic gate?



Logic Gates Applications (AND gate)

9.1 Application Information

In this application, three 2-input AND gates are combined to produce a 4-input AND gate function as shown in Figure 9-1. Multiple SN74AHCT1G08-Q1 are used to directly control the $\overline{\text{RESET}}$ pin of a motor controller. The controller requires four input signals to all be HIGH before being enabled, and should be disabled in the event that any one signal goes LOW. The 4-input AND gate function combines the four individual reset signals into a single active-low reset signal.

9.2 Typical Application

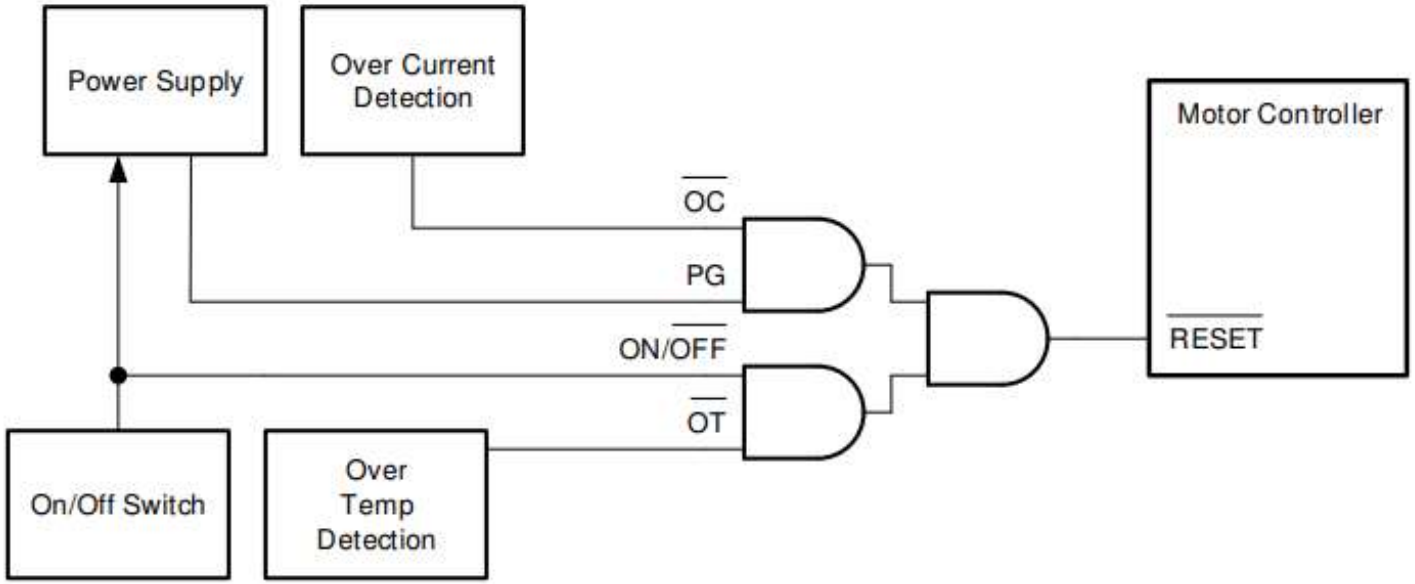


Figure 9-1. Typical Application Block Diagram

Logic Gates Applications (AND gate)

9.1 Application Information

The SN74LVC1G11 device offers logical AND configuration for many design applications. This example describes basic power sequencing using the AND gate configuration. Power sequencing is often used in applications that require a processor or other delicate device with specific voltage timing requirements in order to protect the device from malfunctioning. In the application below, the power-good signals from the supplies tell the MCU to continue an operation.

9.2 Typical Application

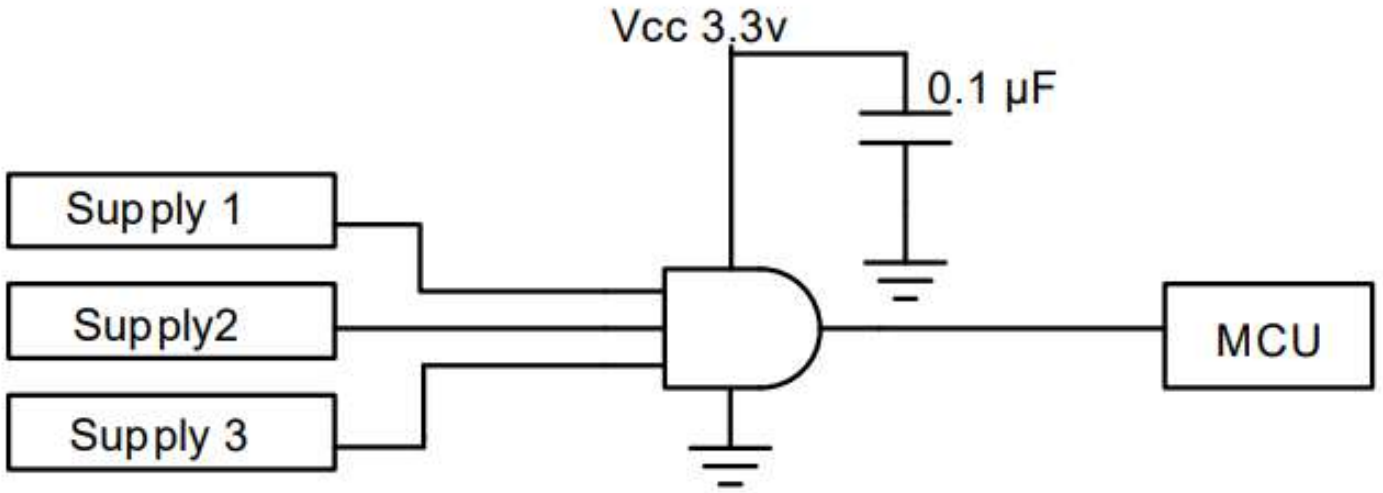


Figure 5. Typical Application Diagram

Logic Gates Applications (OR gate)

9.1 Application Information

In this application, three 2-input OR gates are combined to produce a 4-input OR gate function as shown in Figure 9-1. The fourth gate can be used for another application in the system, or the inputs can be grounded and the channel left unused.

The SNx4AHC32 is used to directly control the Enable pin of a fan driver. The fan driver requires only one input signal to be HIGH before being enabled, and should be disabled in the event that all signals go LOW. The 4-input OR gate function combines the four individual overheat signals into a single active-high enable signal.

Temperature sensors can often be spread throughout a system rather than being in a centralized location. This would mean longer length traces or wires to pass signals through leading to slower edge transitions. This makes the SNx4AHC32 useful for combining the incoming signals.

9.2 Typical Application

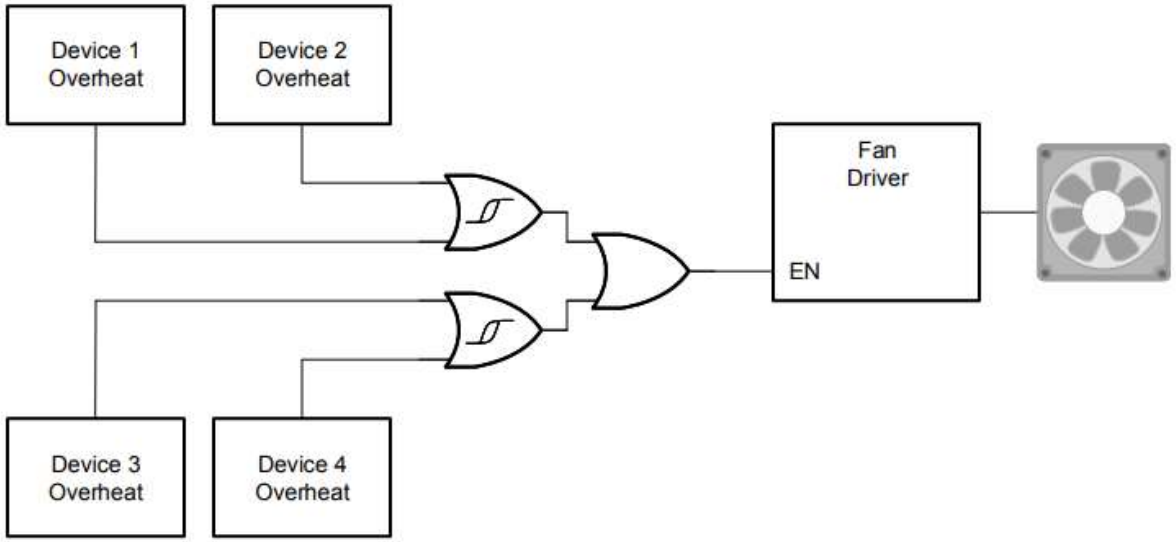
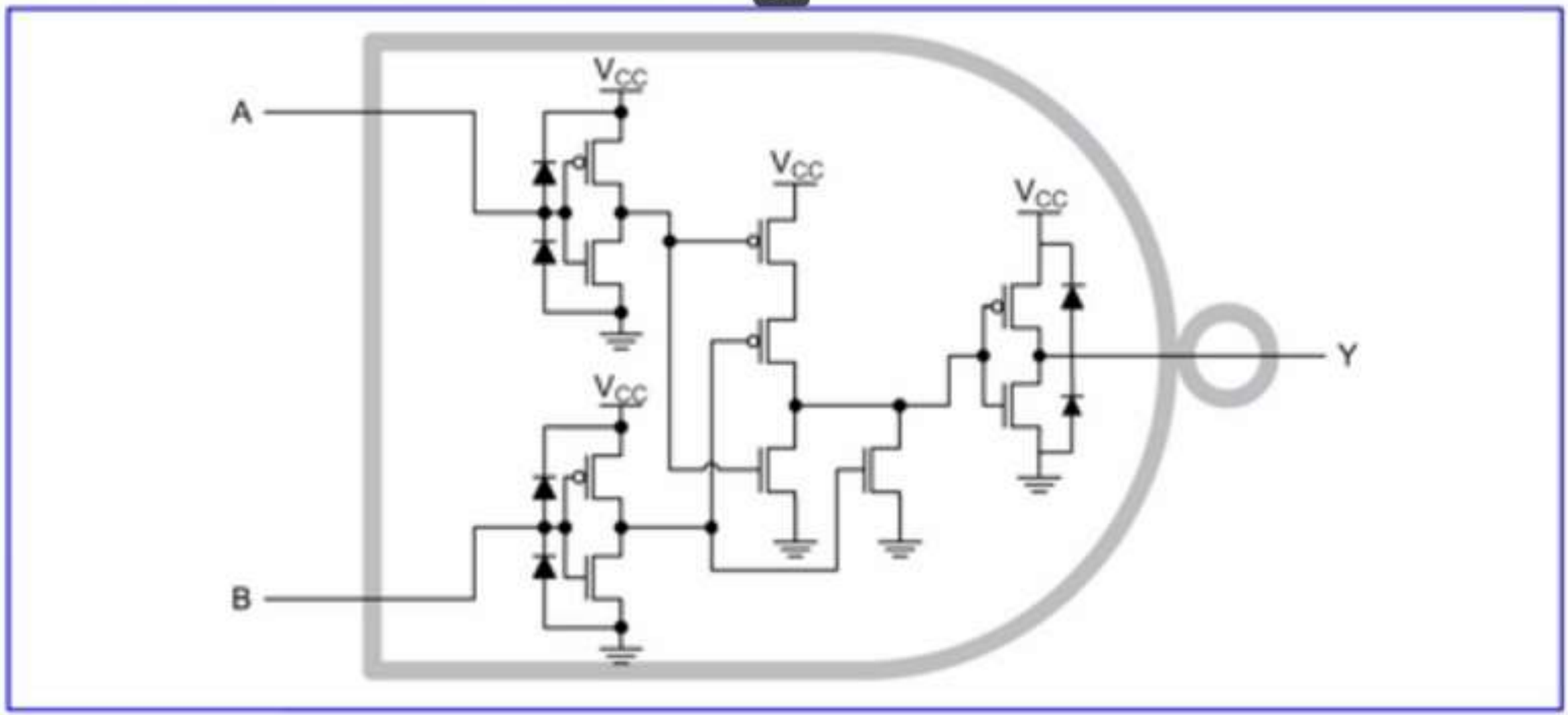
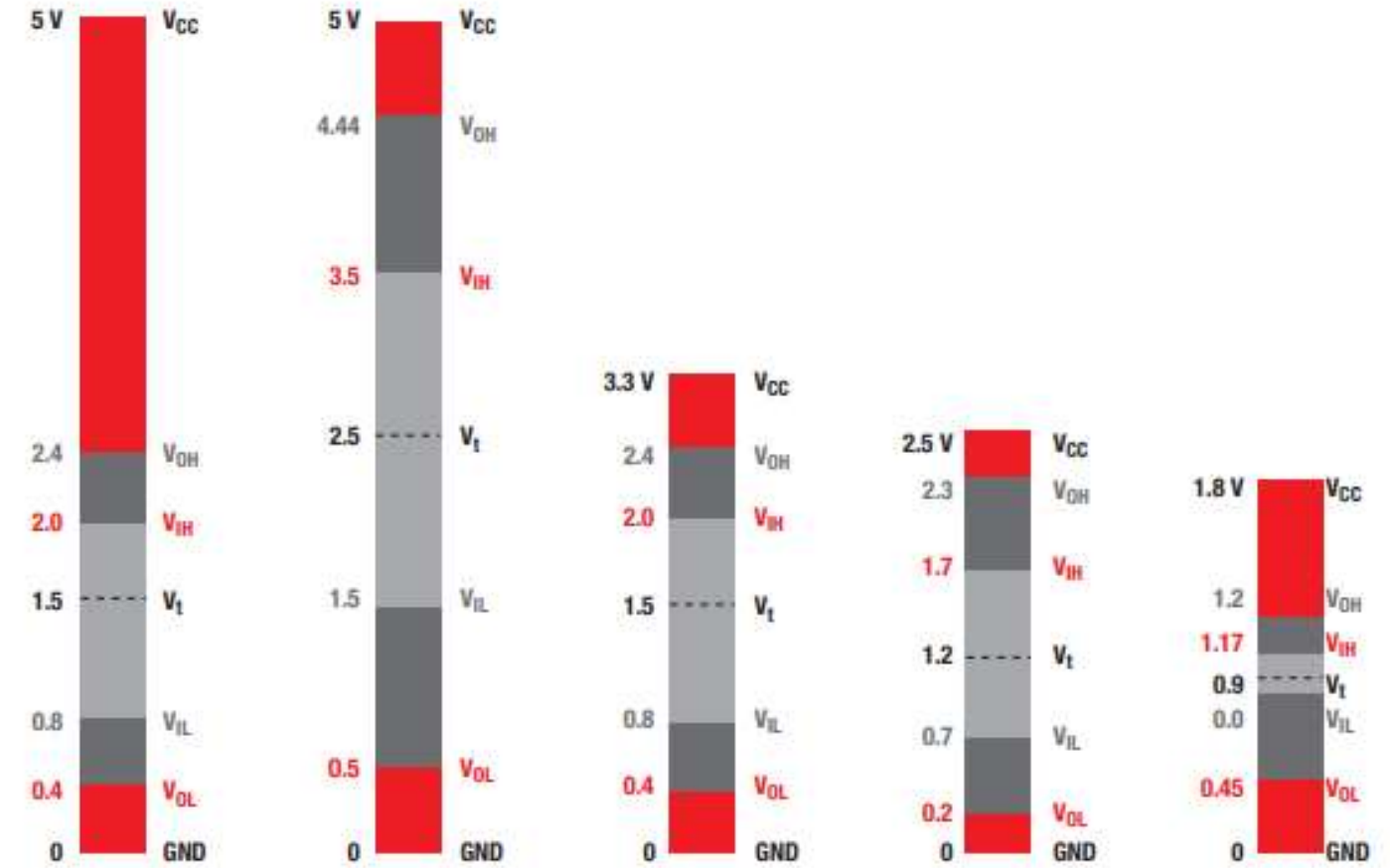


Figure 9-1. Typical Application Block Diagram

What is inside a typical logic gate?



Voltage Levels



5-V TTL
Standard TTL: ABT, AHCT, HCT, ACT, bipolar, LV1T, LV4T

5-V CMOS
Rail-to-Rail 5 V HC, AHC, AC, LV-A, LV1T, LV4T

3.3-V LVTTL
LVT, LV1T, LV4T, LVC, ALVC, AUP, LV-A, ALVT

2.5-V CMOS
AUC, AUP, AVC, ALVC, LVC, ALVT, LV1T, LV4T

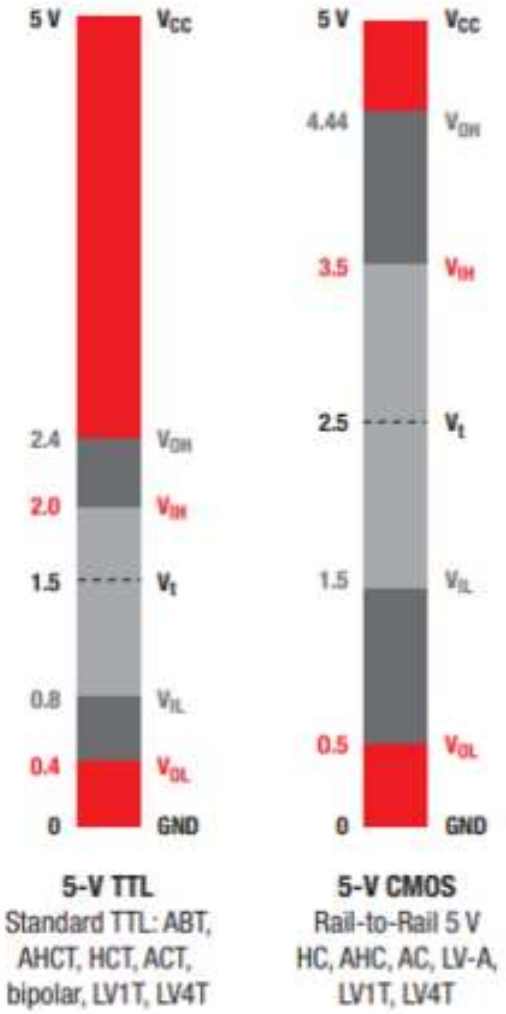
1.8-V CMOS
AUC, AUP, AVC, ALVC, LVC, LV1T, LV4T

Is V_{OH} higher than V_{IH}?
Is V_{OL} less than V_{IL}?

D \ R	5 TTL	5 CMOS	3 LVTTL	2.5 CMOS	1.8 CMOS
5 TTL	Yes	No	Yes*	Yes*	Yes*
5 CMOS	Yes	Yes	Yes*	Yes*	Yes*
3 LVTTL	Yes	No	Yes	Yes*	Yes*
2.5 CMOS	Yes	No	Yes	Yes	Yes*
1.8 CMOS	No	No	No	No	Yes*

* Requires V_{IH} Tolerance

Voltage Levels



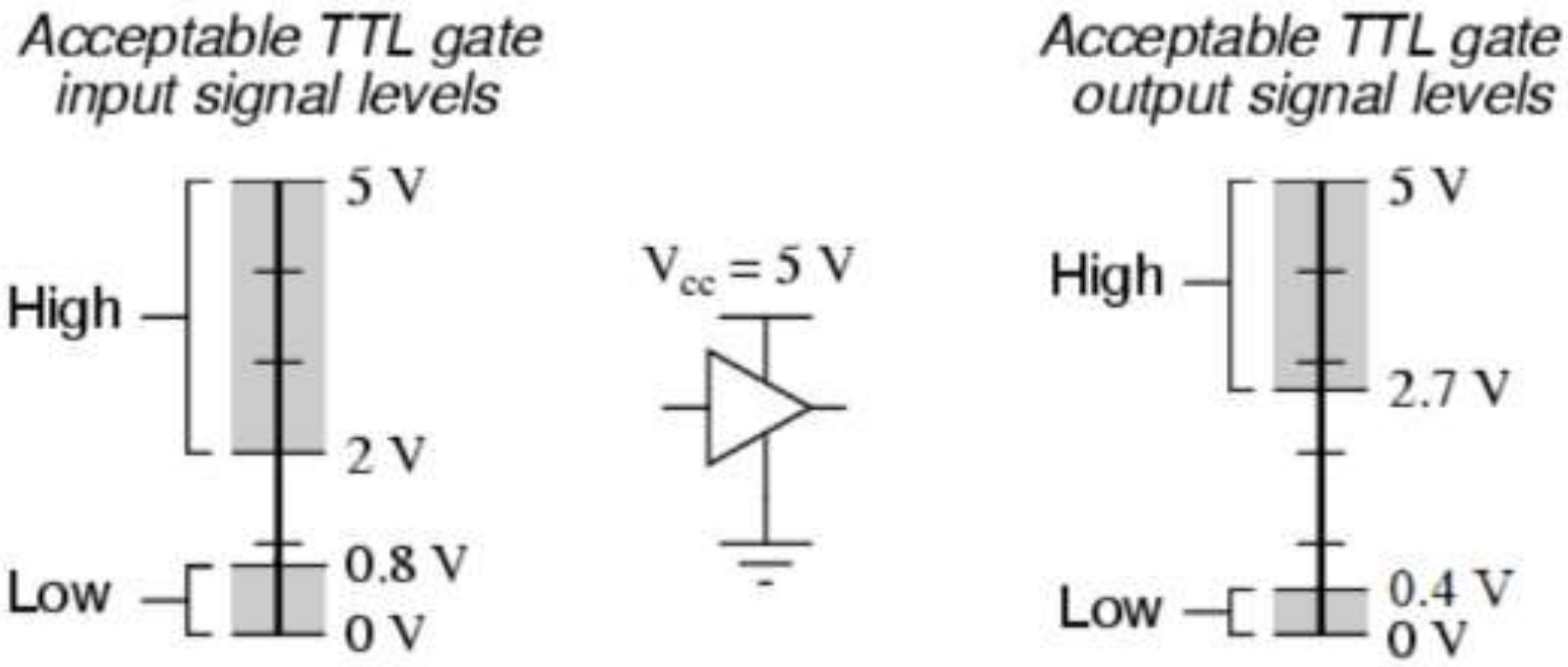
What happens if input is in middle gray region Vt?

Shown below are the switching input/ output comparison table and graphic that illustrate V_{IH} and V_{IL}, which are the minimum switching levels for guaranteed operation. V_t is the approximate switching level and the V_{OH} and V_{OL} levels are the guaranteed outputs for the V_{CC} specified.

Voltage Levels

You will also notice that there is cushion of 0.7 V between the output of one device and the input of another. This is sometimes referred to as **noise margin**.

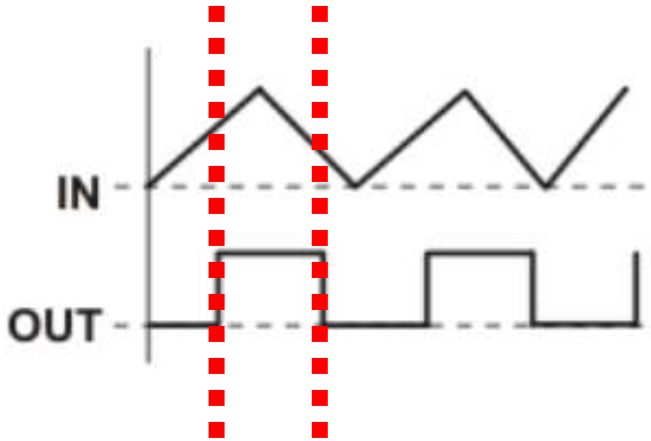
What happens if you have a voltage that is in between 0.8 V and 2 V? Well, your guess is as good as mine. Honestly, this range of voltages is undefined and results in an **invalid state**, often referred to as **floating**. If an output pin on your device is "**floating**" in this range, there is no certainty with what the signal will result in. It may bounce arbitrarily between HIGH and LOW.

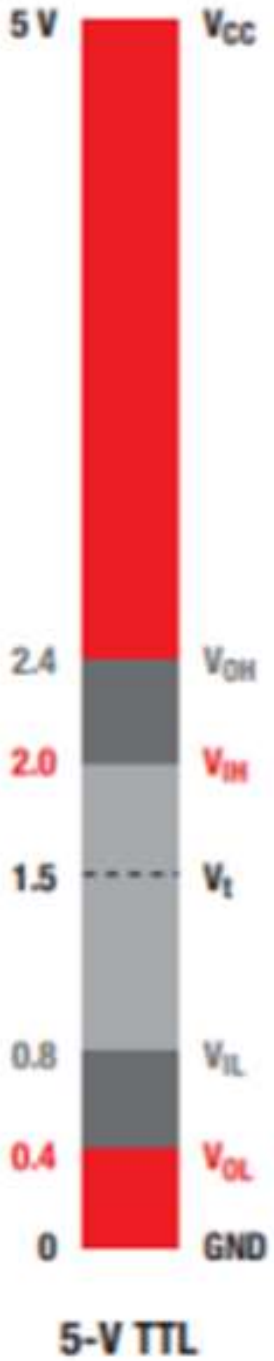


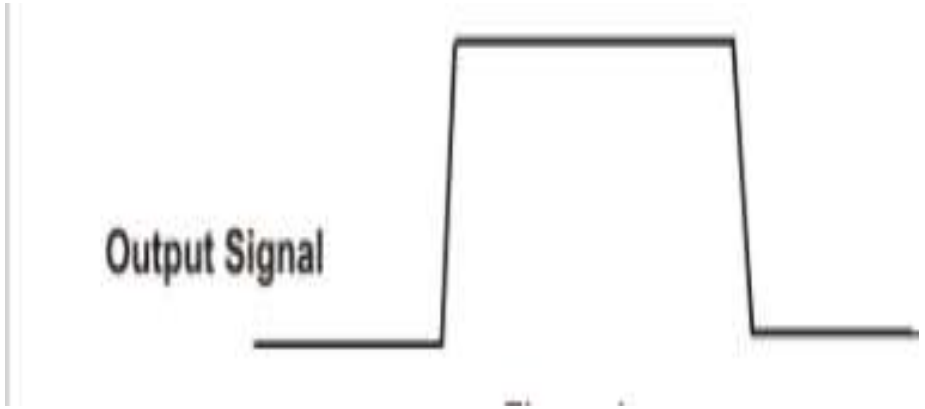
[Logic Levels - SparkFun Learn]

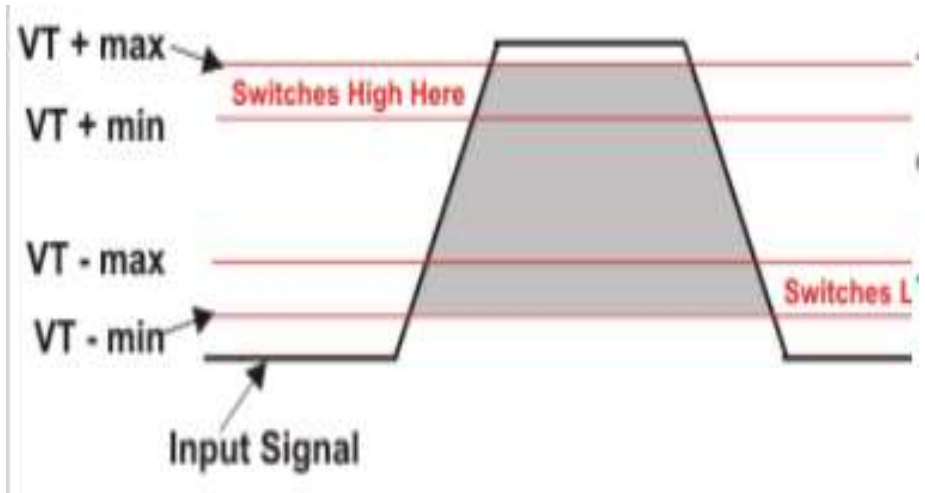
Schmitt Triggers

- It is important to remember (Vt+ max) = Vih and (VT– min) = Vil. In the specs, multiple limits are related to the Schmitt trigger inputs. All of the limits are important for different reasons.
- On the input rising edge, the part will switch between (Vt+ min) and (Vt+ max).
- On the falling edge, the part will switch between (Vt– max) and (Vt– min).
- **The part will not switch between (Vt– max) and (Vt+ min). This is important for noise rejection.**









TTL vs CMOS Logic

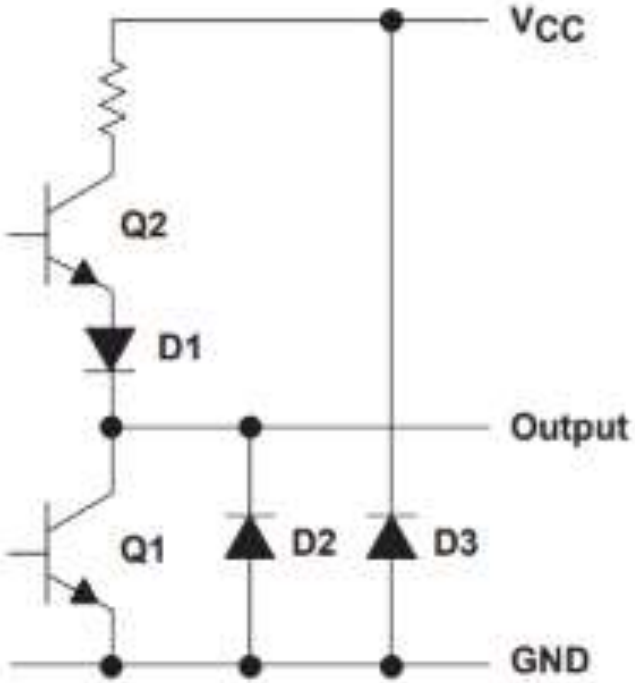


Figure 2. Device Output From SN74 Family (Standard TTL)

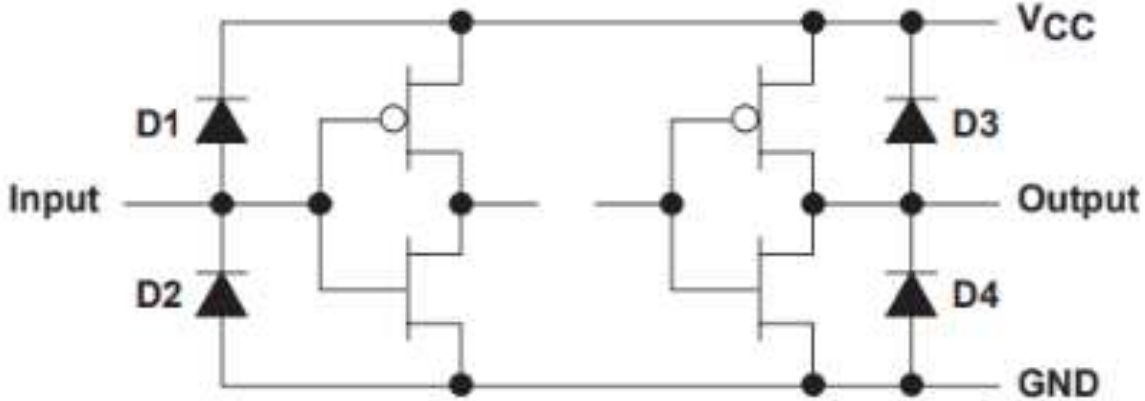


Figure 5. Diode Paths in CMOS Devices

Positive and Negative Logic

- Choosing the high-level H to represent logic 1 defines a positive logic system.
- Choosing the low-level L to represent logic 1 defines a negative logic system

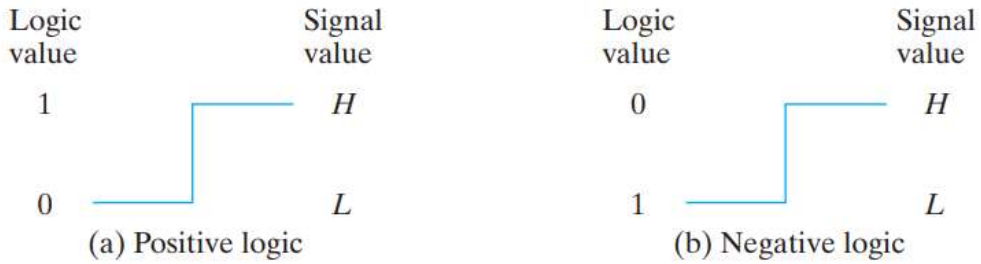
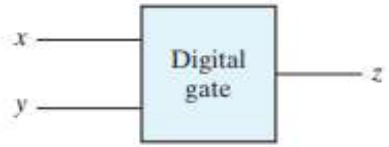


FIGURE 2.9
Signal assignment and logic polarity

x	y	z
L	L	L
L	H	L
H	L	L
H	H	H

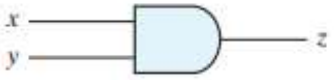
(a) Truth table with *H* and *L*



(b) Gate block diagram

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

(c) Truth table for positive logic



(d) Positive logic AND gate

x	y	z
1	1	1
1	0	1
0	1	1
0	0	0

(e) Truth table for negative logic

(f) Negative logic OR gate

Active-Low and Active-High

When working with ICs and microcontrollers, you'll likely encounter pins that are active-low and pins that are active-high. Simply put, this just describes how the pin is activated. If it's an active-low pin, you must "pull" that pin LOW by connecting it to ground. For an active high pin, you connect it to your HIGH voltage (usually 3.3V/5V).

For example, let's say you have a shift register that has a chip enable pin, CE. If you see the CE pin anywhere in the datasheet with a line over it like this, $\overline{CE}$, then that pin is active-low. The CE pin would need to be pulled to GND in order for the chip to become enabled. If, however, the CE pin doesn't have a line over it, then it is active high, and it needs to be pulled HIGH in order to enable the pin.

Many ICs will have both active-low and active-high pins intermingled. Just be sure to double check for pin names that have a line over them. The line is used to represent NOT (also known as bar). When something is NOTTED, it changes to the opposite state. So if an active-high input is NOTTED, then it is now active-low. Simple as that!

Active low can be considered as if the signal is inverted inside the IC

4.2. Combinational Logic Circuits

- A combinational circuit consists of an **interconnection of logic gates**. Combinational logic gates react to the values of the signals at their inputs and produce the value of the output signal, transforming binary information from the given input data to a required output data

- The logic diagram of a combinational circuit has logic gates with **no feedback** paths or **memory** elements.

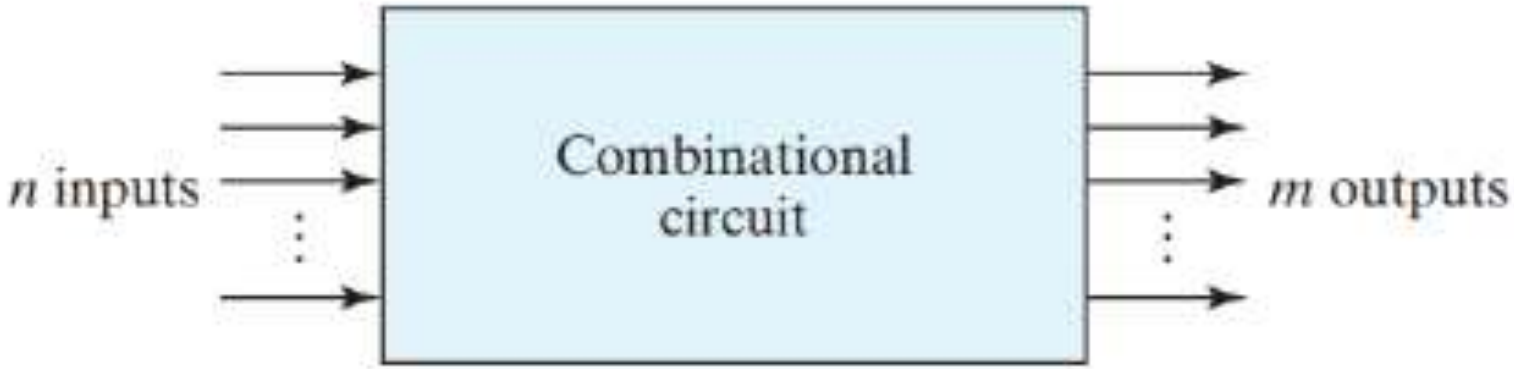


FIGURE 4.1
Block diagram of combinational circuit

Logic Diagram to Expression

Starting from input to output, construct the expression

F = ABC + (A+B+C)(AB+AC+BC)’

$$
\begin{aligned}
&= (A' + B')(A' + C')(B' + C')(A + B + C) + ABC \\
&= (A' + B'C')(AB' + AC' + BC' + B'C) + ABC \\
&= A'BC' + A'B'C + AB'C' + ABC
\end{aligned}
$$

Table 4.1
Truth Table for the Logic Diagram of Fig. 4.2

A	B	C	F_2	F_2'	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

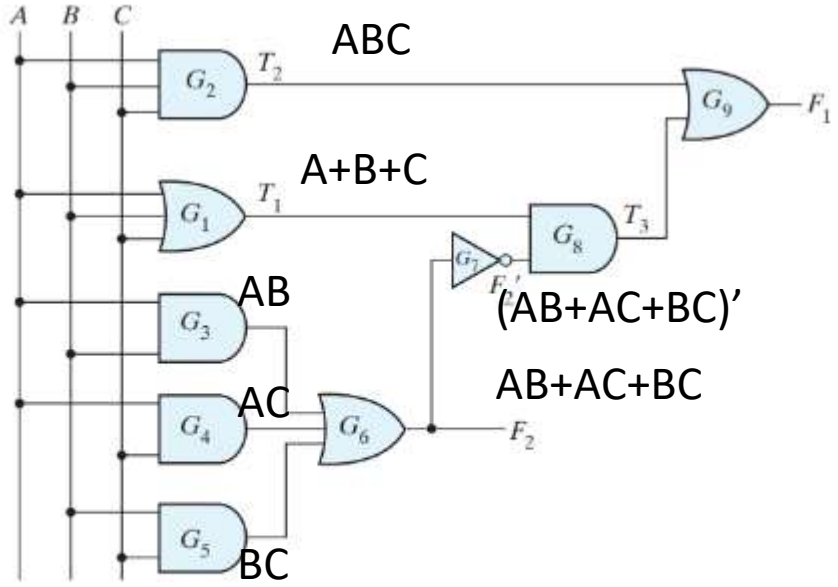


FIGURE 4.2
Logic diagram for analysis example

Logic Diagram to Expression

Practice Exercise 4.1

Analyze the logic diagram in Fig. PE4.1 and find the Boolean expressions for F_1 and F_2.

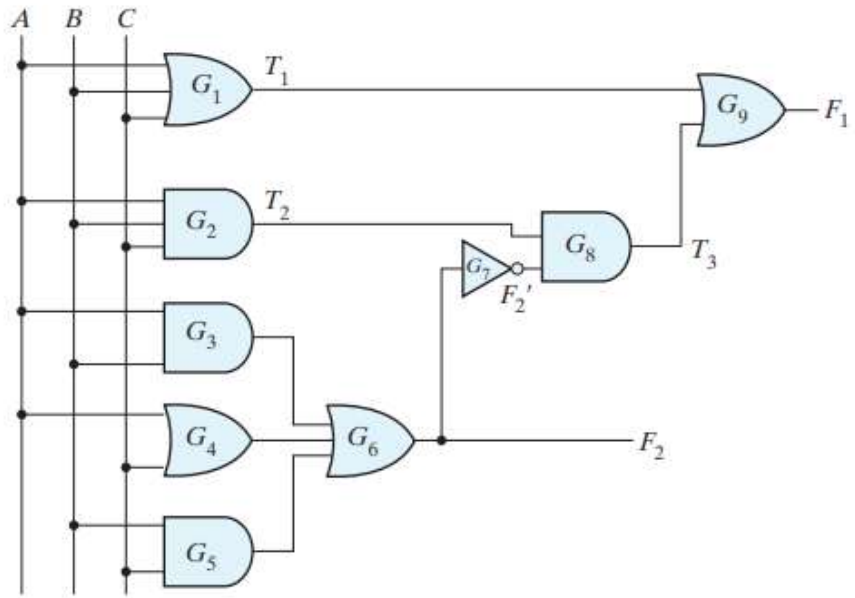


FIGURE PE4.1

Answer: $T_1 = A + B + C$

$T_2 = ABC$

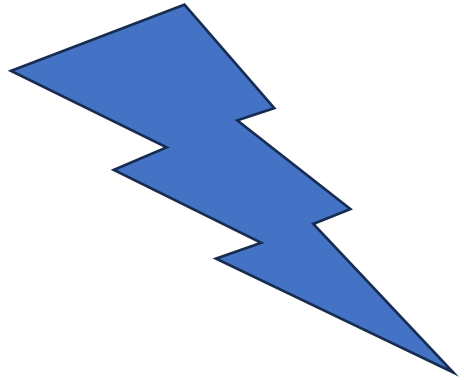
$F_2 = AB + A + B + BC = A + B + BC = A + B$

$F_2' = A'B'$

$T_3 = (ABC)(A'B') = 0$

$F_1 = T1 = A + B + C$

DESIGN GUIDE



1. **Description of function**
2. **Create using Truth Table**
3. **Write logical expression using Truth Table**
4. **Implement logical expression using AND, OR, NOT gates**
5. **Minimize number of gates**

DECIMAL to BINARY

- We can convert numbers to binary 0,1 and perform arithmetic operations using logic gates.
- Coverting integer numbers to binary
- LEFTMOST BIT is the MOST SIGNIFICANT BIT of the number

EXAMPLE 1.1

Convert decimal 41 to binary. First, 41 is divided by 2 to give an integer quotient of 20 and a remainder of $\frac{1}{2}$. Then the quotient is again divided by 2 to give a new quotient and remainder. The process is continued until the integer quotient becomes 0. The *coefficients* of the desired binary number are obtained from the *remainders* as follows:

	Integer Quotient		Remainder	Coefficient
$41/2 =$	20	+	$\frac{1}{2}$	$a_0 = 1$
$20/2 =$	10	+	0	$a_1 = 0$
$10/2 =$	5	+	0	$a_2 = 0$
$5/2 =$	2	+	$\frac{1}{2}$	$a_3 = 1$
$2/2 =$	1	+	0	$a_4 = 0$
$1/2 =$	0	+	$\frac{1}{2}$	$a_5 = 1$

Therefore, the answer is $(41)_{10} = (a_5a_4a_3a_2a_1a_0)_2 = (101001)_2$.

DECIMAL to BINARY

- We can convert numbers to binary 0,1 and perform arithmetic operations using logic gates.
- Coverting integer numbers to binary
- LEFTMOST BIT is the MOST SIGNIFICANT BIT of the number

EXAMPLE 1.1

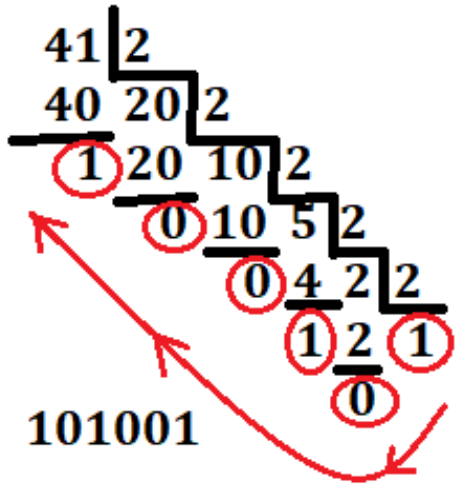
Convert decimal 41 to binary. First, 41 is divided by 2 to give an integer quotient of 20 and a remainder of $\frac{1}{2}$. Then the quotient is again divided by 2 to give a new quotient and remainder. The process is continued until the integer quotient becomes 0. The *coefficients* of the desired binary number are obtained from the *remainders* as follows:

The arithmetic process can be manipulated more conveniently as follows:

Integer		Remainder	
41	/2		
20	/2	1	
10	/2	0	
5	/2	0	
2	/2	1	
1	/2	0	
0		1	101001 = answer

DECIMAL to BINARY

- We can convert numbers to binary 0,1 and perform arithmetic operations using logic gates.
- Converting integer numbers to binary
- divide to 2 till 1 or 0, write remainders in REVERSE order
- LEFTMOST BIT is the MOST SIGNIFICANT BIT of the number



BINARY ARITHMETIC

- We can convert numbers to binary 0,1 and perform arithmetic operations using logic gates.
- Converting integer numbers to binary
- Arithmetic operations with numbers in base 2 follow the same rules as for decimal numbers.
- full adder with carry, augend and addend

	8	4	2	1
carry				
augend		1	0	1
addend		0	1	0
sum		1	1	1

5
2
+___
7

$$\begin{array}{r} c_{k+1}\ c_k \ldots c_3\ c_2\ c_1 \\ a_k \ldots a_3\ a_2\ a_1\ a_0 \\ b_k \ldots b_3\ b_2\ b_1\ b_0 \\ + \_\_\_\_\_ \\ c_{k+1}\ s_k \ldots s_3\ s_2\ s_1\ s_0 \end{array}$$

BINARY ARITHMETIC

	8	4	2	1
carry	1			
augend		1	0	0
addend		1	0	1
sum	1	0	0	1

$$\begin{array}{r} 4 \\ 5 \\ + \\ \hline 9 \end{array}$$

	8	4	2	1
carry	1	1	1	
augend		1	1	1
addend		1	0	1
sum	1	1	0	0

$$\begin{array}{r} 7 \\ 5 \\ + \\ \hline 12 \end{array}$$

$$\begin{array}{r} c_{k+1}\ c_k \ldots c_3\ c_2\ c_1 \\ a_k \ldots a_3\ a_2\ a_1\ a_0 \\ b_k \ldots b_3\ b_2\ b_1\ b_0 \\ + \\ \hline c_{k+1}\ s_k \ldots s_3\ s_2\ s_1\ s_0 \end{array}$$

BINARY ARITHMETIC

	8	4	2	1
carry	1	1	1	
augend		1	0	1
addend		1	1	1
sum	1	1	0	0

$$\begin{array}{r} 5 \\ 7 \\ + \\ \hline 12 \end{array}$$

$$\begin{array}{r} c_{k+1}\ c_k \ldots c_3\ c_2\ c_1 \\ a_k \ldots a_3\ a_2\ a_1\ a_0 \\ b_k \ldots b_3\ b_2\ b_1\ b_0 \\ + \\ \hline c_{k+1}\ s_k \ldots s_3\ s_2\ s_1\ s_0 \end{array}$$

	8	4	2	1
carry	1	1	1	
augend		1	1	1
addend		1	1	1
sum	1	1	1	0

$$\begin{array}{r} 7 \\ 7 \\ + \\ \hline 14 \end{array}$$

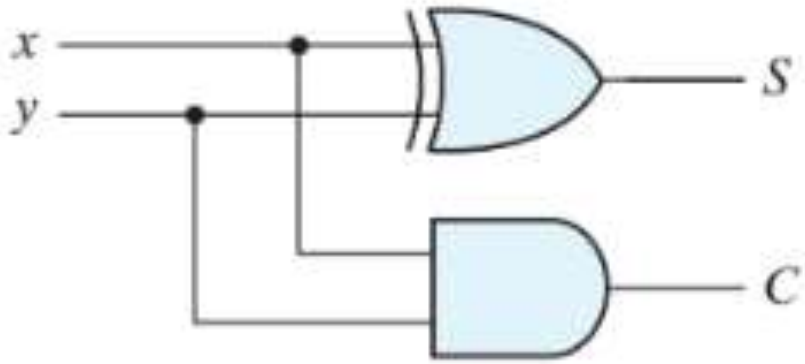
HALF ADDER = 2bit adder

- A combinational circuit that performs the addition of two bits is called a half adder

Table 4.3
Half Adder

x	Y	C	S	DEC
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	0	2

$$S = x'y + xy'$$
$$C = xy$$



(b) $S = x \oplus y$
$C = xy$

Full Adder = 3bit adder

- A full adder is a combinational circuit that forms the arithmetic sum of three bits.
- It consists of three inputs and two outputs.
- Two of the input variables, denoted by x and y, represent the two significant bits to be added.
- The third input, z, represents the carry from the previous lower significant position.
- The two outputs are designated by the symbols S for sum and C for carry.

Table 4.4
Full Adder

x	y	z	C	S	DEC
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	1
0	1	1	1	0	2
1	0	0	0	1	1
1	0	1	1	0	2
1	1	0	1	0	2
1	1	1	1	1	3

c_{k+1}	$c_k \quad c_{j+1} c_j \quad c_2 c_1$	Carried numbers
	$a_k \text{---} a_{j+1} a_j \text{---} a_2 a_1 a_0$	First number
	$b_k \quad b_{j+1} b_j \quad b_2 b_1 b_0$	Second number
c_{k+1}	$s_k \text{---} s_{j+1} s_j \text{---} s_2, s_1 s_0$	Sum

Electric Adder to the Base Two

- We can convert numbers to binary 0,1 and perform arithmetic operations using logic gates.
- Idea has been presented in Shannon 1938 thesis

Electric Adder to the Base Two

A circuit is to be designed that will automatically add two numbers, using only relays and switches. Although any numbering base could be used the circuit is greatly simplified by using the scale of two. Each digit is thus either 0 or 1; the number whose digits in order are

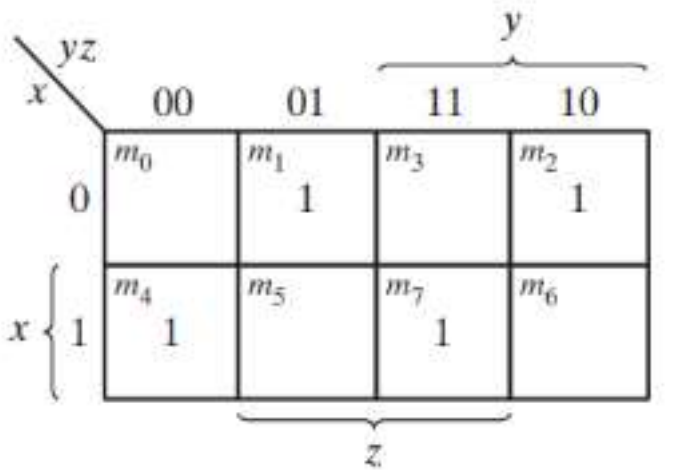
c_{k+1}	c_k $c_{j+1}c_j$ c_2c_1		Carried numbers
	a_k---$a_{j+1}a_j$---$a_2a_1a_0$		First number
	b_k $b_{j+1}b_j$ $b_2b_1b_0$		Second number
c_{k+1}	s_k---$s_{j+1}s_j$---s_2,s_1s_0		Sum

Full Adder (three bits adder)

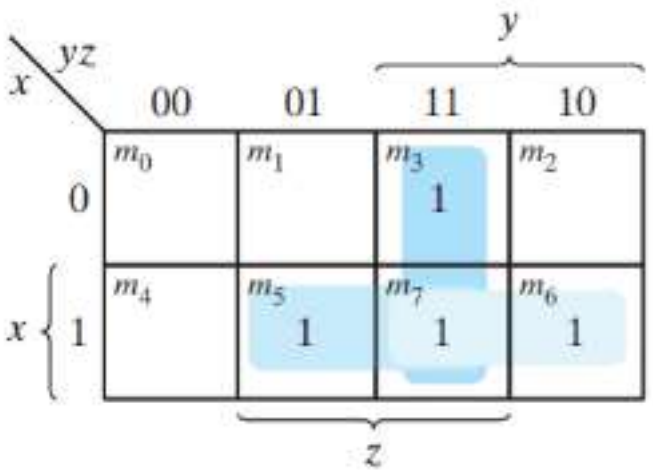
- A full adder is a combinational circuit that forms the arithmetic sum of three bits.
- It consists of three inputs and two outputs.
- Two of the input variables, denoted by x and y, represent the two significant bits to be added.
- The third input, z, represents the carry from the previous lower significant position.
- The two outputs are designated by the symbols S for sum and C for carry.

Table 4.4
Full Adder

x	y	z	C	S	DEC
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	1
0	1	1	1	0	2
1	0	0	0	1	1
1	0	1	1	0	2
1	1	0	1	0	2
1	1	1	1	1	3



(a) $S = x'y'z + x'yz' + xy'z' + xyz$



(b) $C = xy + xz + yz$

FULL ADDER

- One that performs the addition of three bits (two significant bits and a previous carry) is a full-adder.

Table 4.4
Full Adder

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = x'y'z + x'yz' + xy'z' + xyz$$
$$C = xy + xz + yz$$

$$\begin{aligned} S &= z \oplus (x \oplus y) \\ &= z'(xy' + x'y) + z(xy' + x'y)' \\ &= z'(xy' + x'y) + z(xy + x'y') \\ &= xy'z' + x'yz' + xyz + x'y'z \end{aligned}$$

$$C = z(xy' + x'y) + xy = xy'z + x'yz + xy$$

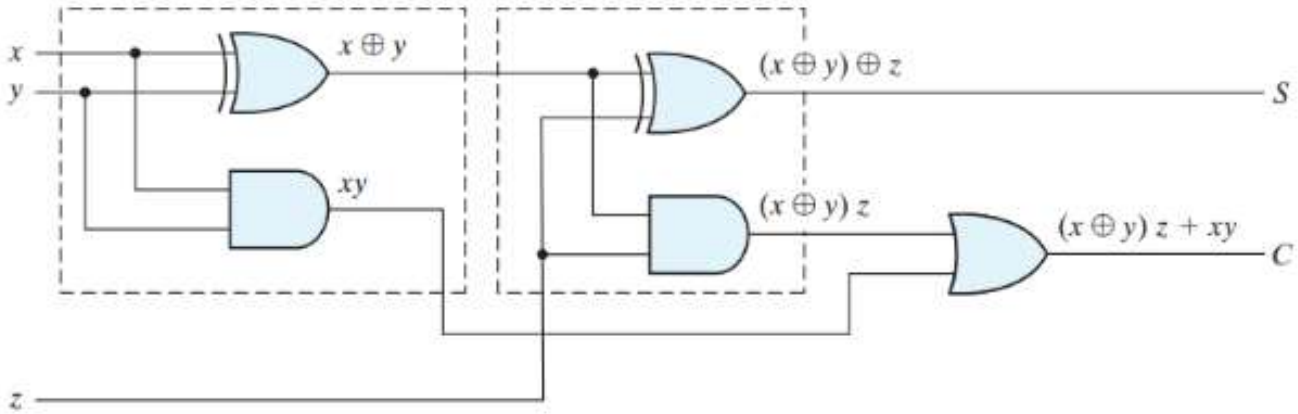
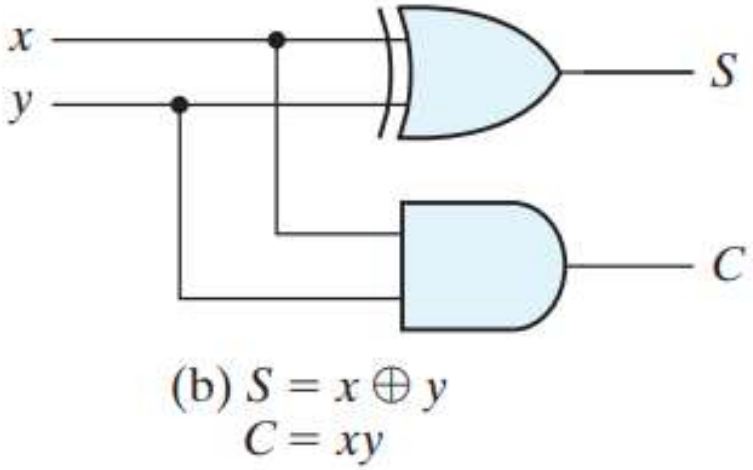


FIGURE 4.8
Implementation of full adder with two half adders and an OR gate

Half Adder → Full Adder

- It can also be implemented with **two half adders** and one OR gate, as shown in Fig. 4.8.



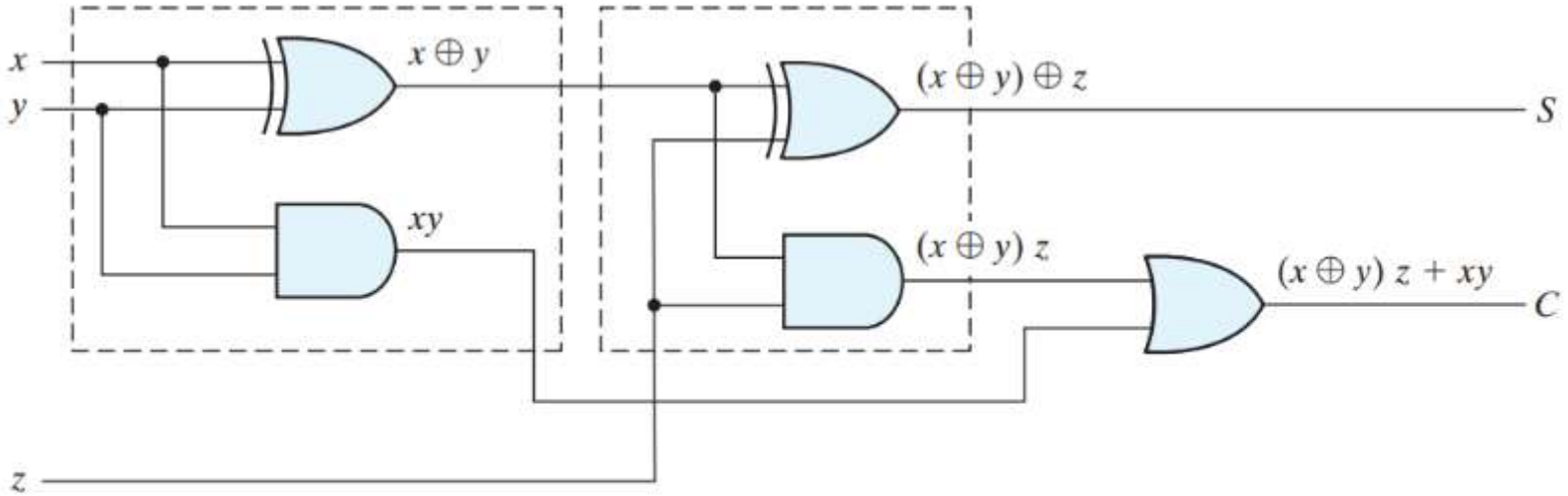


FIGURE 4.8
Implementation of full adder with two half adders and an OR gate

4-bit Binary Adder

- 4-bit binary adder is a digital circuit that produces the arithmetic sum of two binary numbers.
- It can be constructed with full adders connected in cascade, with the **output carry from each full adder connected to the input carry** of the next full adder in the chain

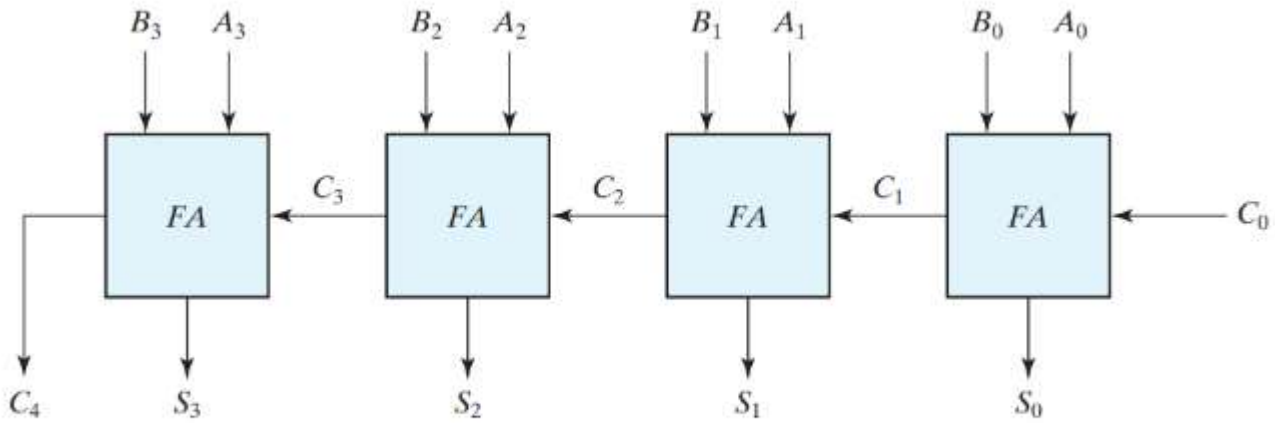


FIGURE 4.9
Four-bit adder

	8	4	2	1
carry	1	1	1	
augend		1	1	1
addend		1	1	1
sum	1	1	1	0

Full Adder

- Full adder implementation from truth table

Table 4.4
Full Adder

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

*Table

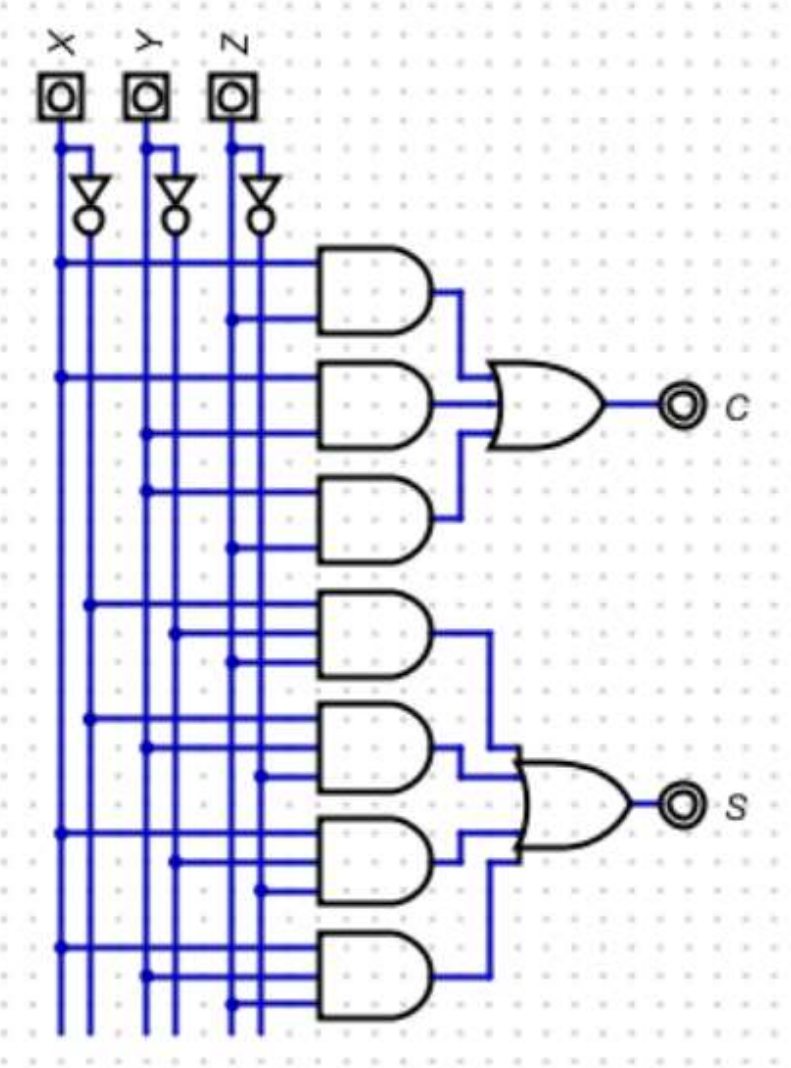
File New Edit Create K-Map

X	Y	Z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

All possible solutions

$C = (X\ Z) + (X\ Y) + (Y\ Z)$

$S = (\bar{X}\ \bar{Y}\ Z) + (\bar{X}\ Y\ \bar{Z}) + (X\ \bar{Y}\ \bar{Z}) + (X\ Y\ Z)$

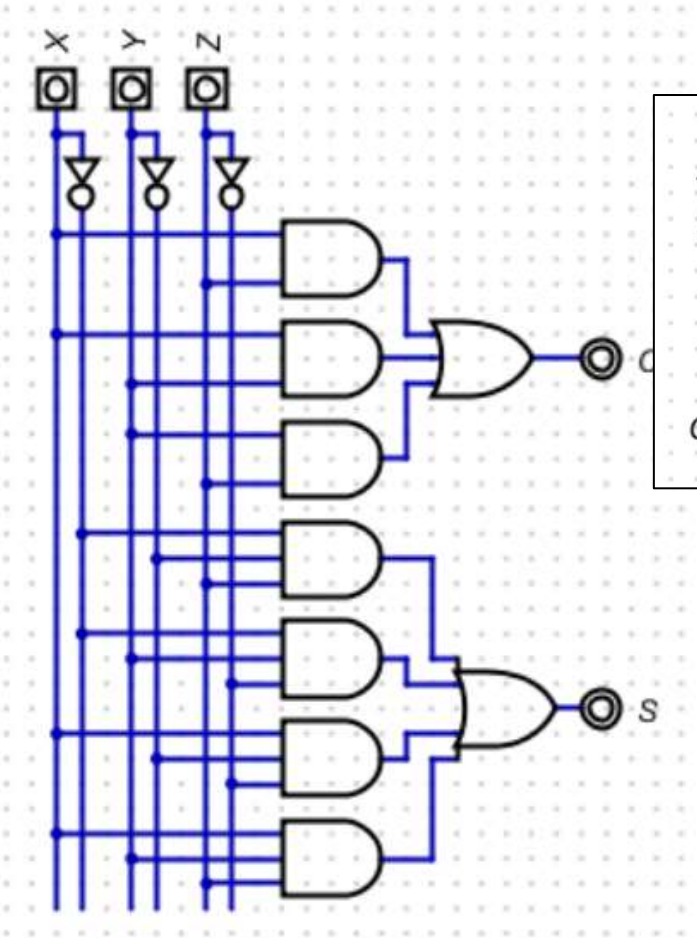


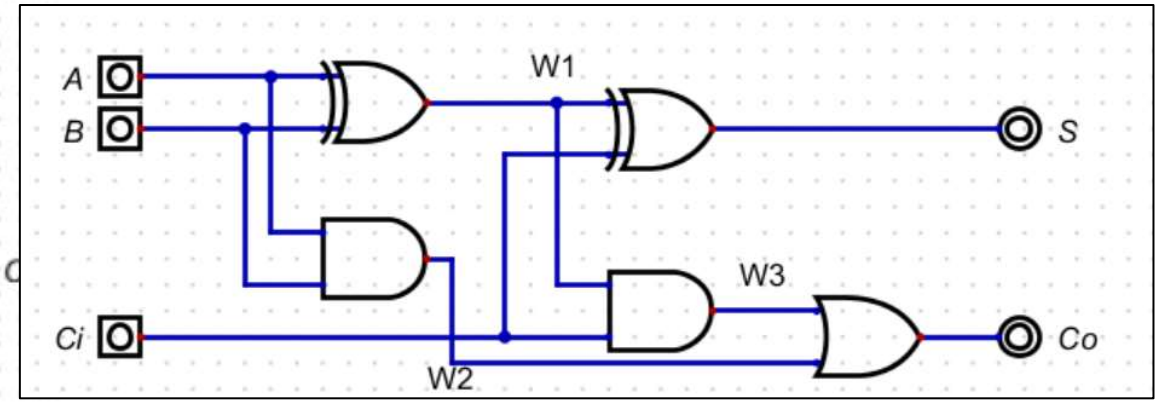
Comparison of Different Implementations

- Full adder implementation from truth table

Table 4.4
Full Adder

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1





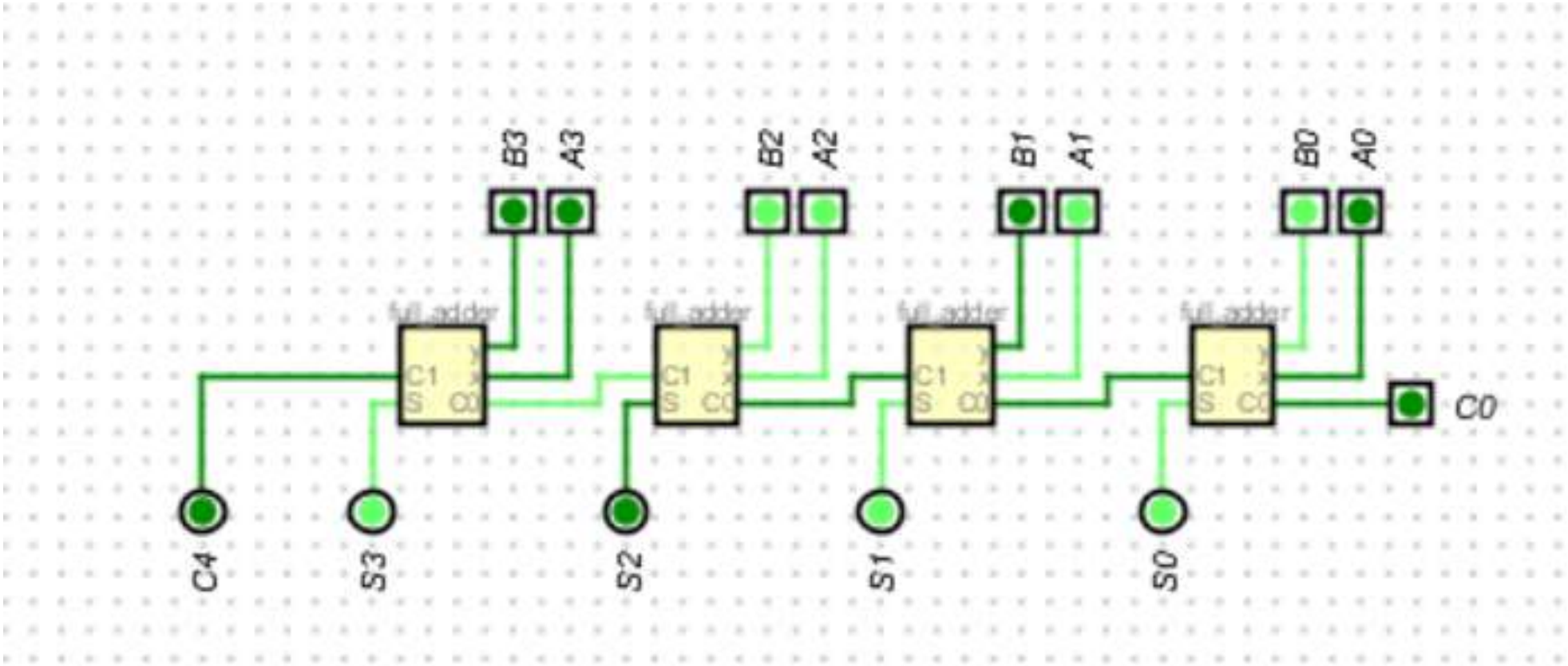
All possible solutions

$$C = (X\ Z) + (X\ Y) + (Y\ Z)$$

$$S = (\bar{X}\ \bar{Y}\ Z) + (\bar{X}\ Y\ \bar{Z}) + (X\ \bar{Y}\ \bar{Z}) + (X\ Y\ Z)$$

4-bit Binary Adder

- 4-bit binary adder is a digital circuit that produces the arithmetic sum of two binary numbers.
- It can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of the next full adder in the chain



4-bit Binary Adder

- To demonstrate with a specific example, consider the two binary numbers A = 1011 and B = 0011. Their sum S = 1110 is formed with the four-bit adder as follows:
- Co=0

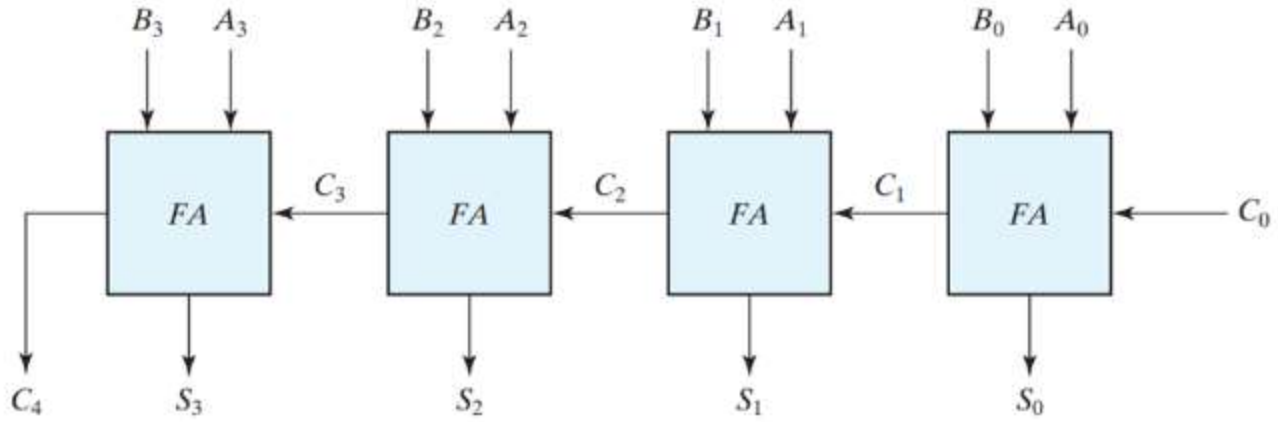
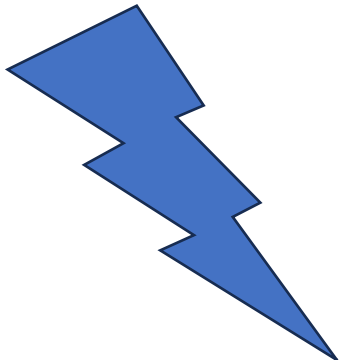


FIGURE 4.9
Four-bit adder

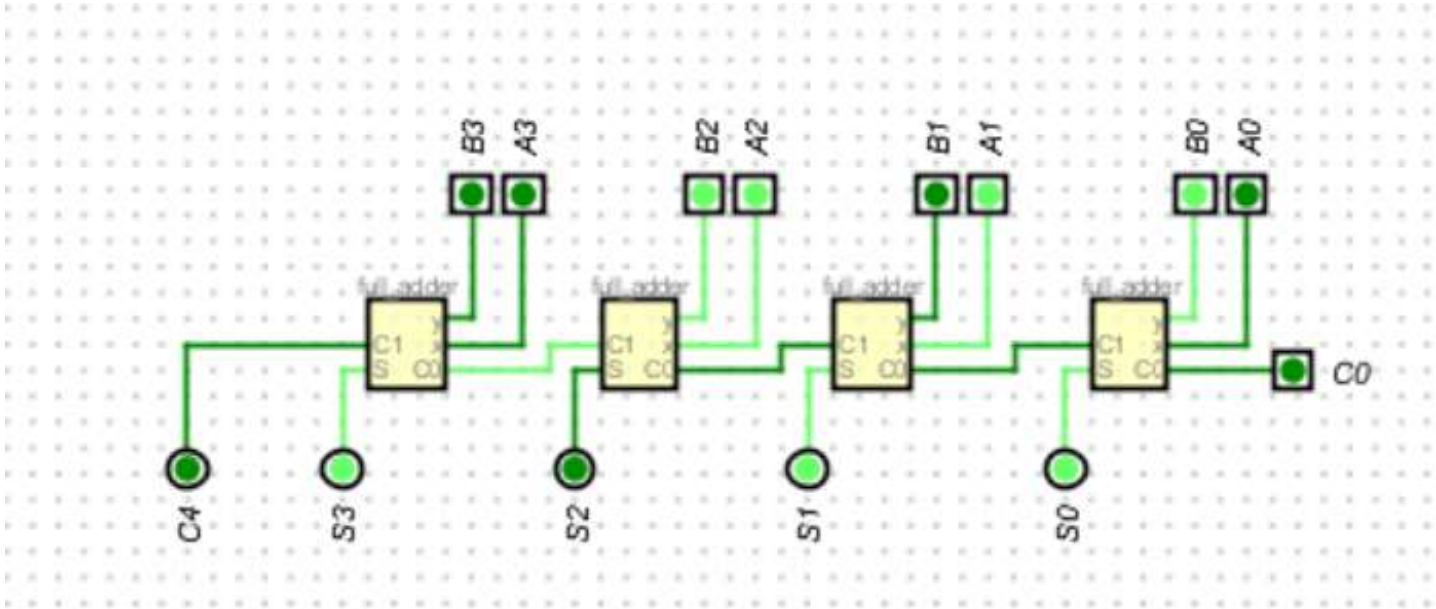
carry			1	1	
augend		1	0	1	1
addend		0	0	1	1
sum		1	1	1	0

```
  11
   3
+____
  14
```



4-bit Binary Adder

- To demonstrate with a specific example, consider the two binary numbers A = 1011 (11) and B = 0011 (3). Their sum S = 1110 (14) is formed with the four-bit adder as follows:
- Co=0, C4=0



carry			1	1	
augend		1	0	1	1
addend		0	0	1	1
sum		1	1	1	0

```
  11
   3
+____
  14
```

4-bit Binary Adder

- if sum exceeding 4bits=15, overflow will occur **C4=1**, and 4 bit sum will be wrapped around
- so we need to stay within number of bits, otherwise unexpected results may seen

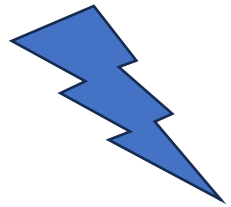
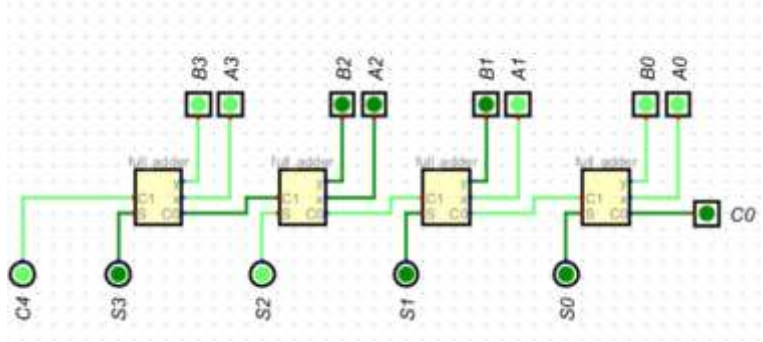


Table 4.4
Full Adder

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

carry					
augend		0	0	1	0
addend		0	1	0	0
sum		0	1	1	0

$$\begin{array}{r} 2 \\ 4 \\ +\_\_\_\_ \\ 6 \end{array}$$



Numbers in Arduino

Arduino Data Types	Value Assigned	Value Ranges
boolean	8 Bit	True or False
byte	8 Bit	0 to 255
char	8 Bit	-127 to 128
unsigned char	8 Bit	0 to 255
word	16 Bit	0 to 65535
unsigned int	16 Bit	0 to 65535
int	16 Bit	-32768 to 32767
long	32 Bit	-2,147,483,648 to 2,147,483,647
float	32 Bit	-3.4028235E38 to 3.4028235E38
www.TheEngineeringProjects.com		

8-bit Binary Adder in Arduino with overflow

- if sum exceeding 4bits=15, overflow will occur C4=1, and 4 bit sum will be wrapped around
- so we need to stay within number of bits, otherwise unexpected results may seen

```
1
2  // Arduino overflow example
3  // Senol Gulgonul
4  // 11.10.2023
5  uint8_t a = 255;
6  uint8_t b = 2;
7  uint8_t c = 0;
8  void setup() {
9    Serial.begin(9600);
10 }
11 void loop() {
12   c=a+b;
13   Serial.print("255+2: ");
14   Serial.println(c);
15   delay(1000);
16 }
17
18
19
```

Serial Monitor

```
Value: 1
255+2: 1
255+2: 1
255+2: 1
```

Carry Propagation

- C2 waits for C1, C3 waits for C2, C4 waits for C3, this causes a propagation delay which directly defines performance of the adder.
- The most widely used technique employs the principle of **carry lookahead** logic.
- C3 does not have to wait for C2 and C1 to propagate;

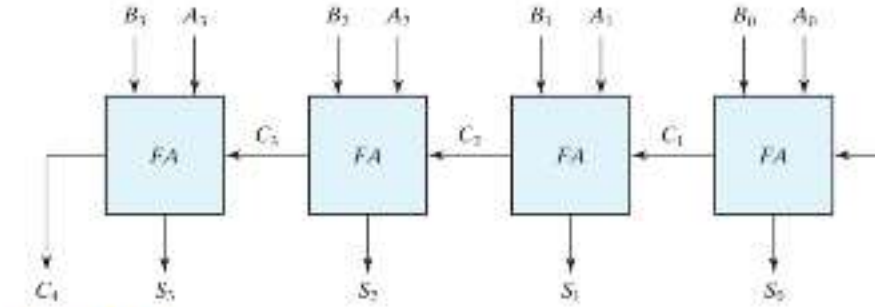


FIGURE 4.9
Four-bit adder

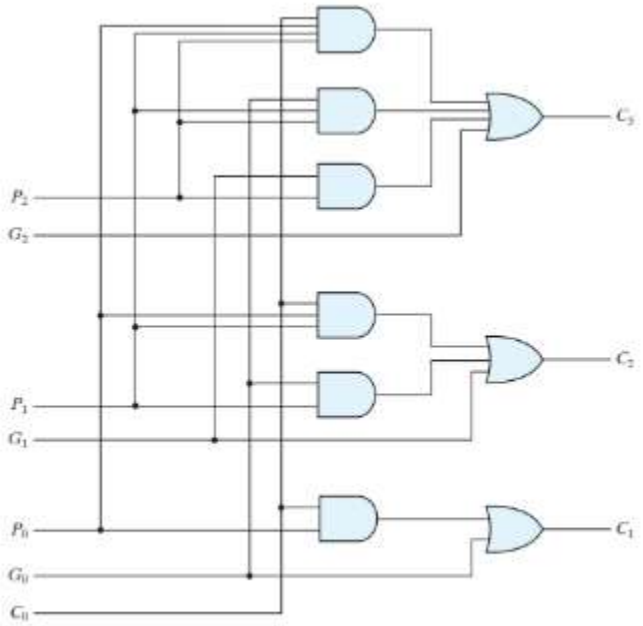


FIGURE 4.11
Logic diagram of carry lookahead generator

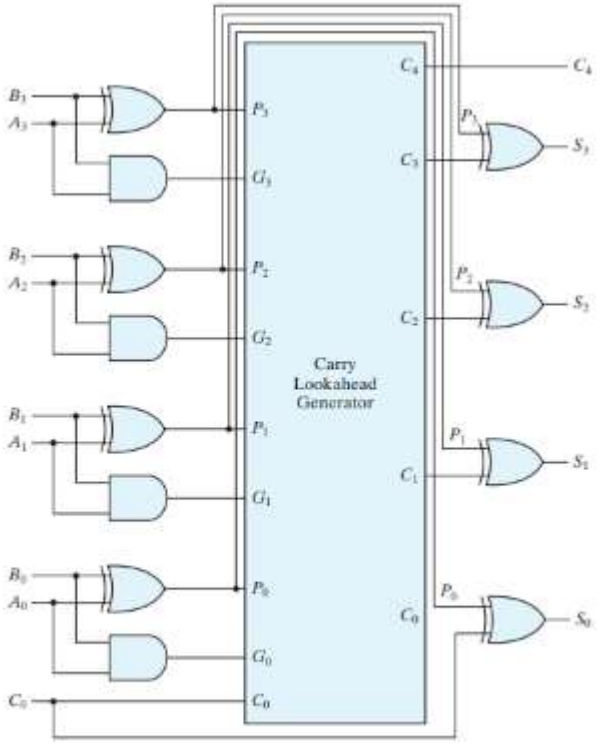


FIGURE 4.12
Four-bit adder with carry lookahead

Carry Propagation

Gi is called a carry generate.
Pi is called a carry propagate

$$P_i = A_i \oplus B_i$$
$$G_i = A_i B_i$$
$$C_0 = \text{input carry}$$
$$C_1 = G_0 + P_0 C_0$$
$$C_2 = G_1 + P_1 C_1 = G_1 + P_1(G_0 + P_0 C_0) = G_1 + P_1 G_0 + P_1 P_0 C_0$$
$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$$

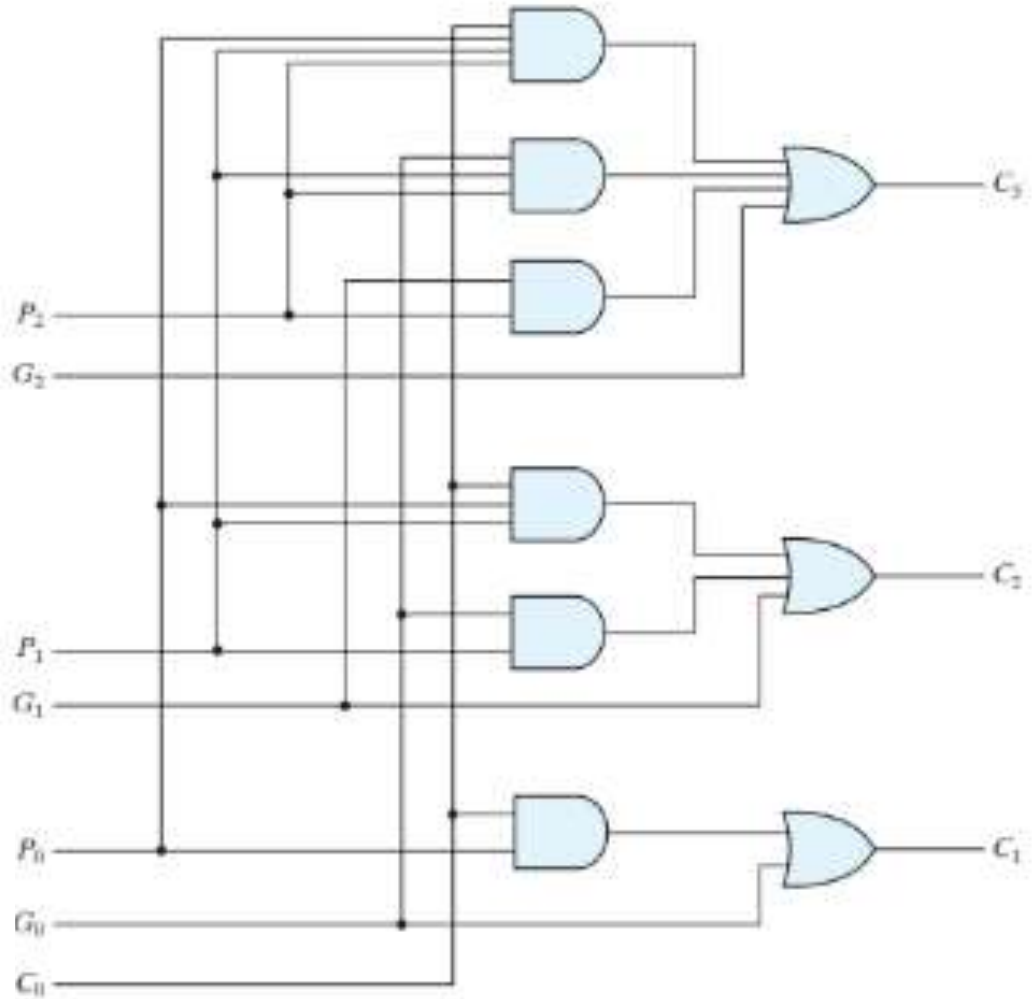


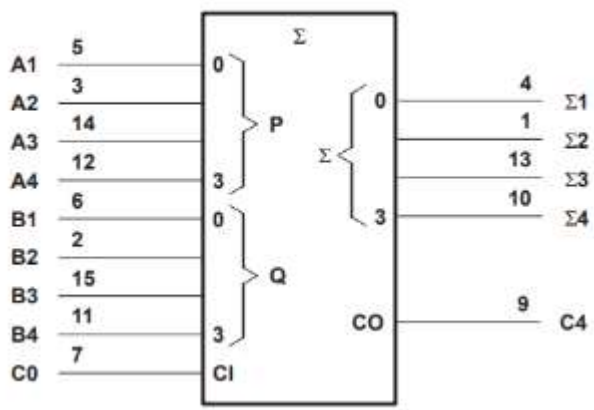
FIGURE 4.11
Logic diagram of carry lookahead generator

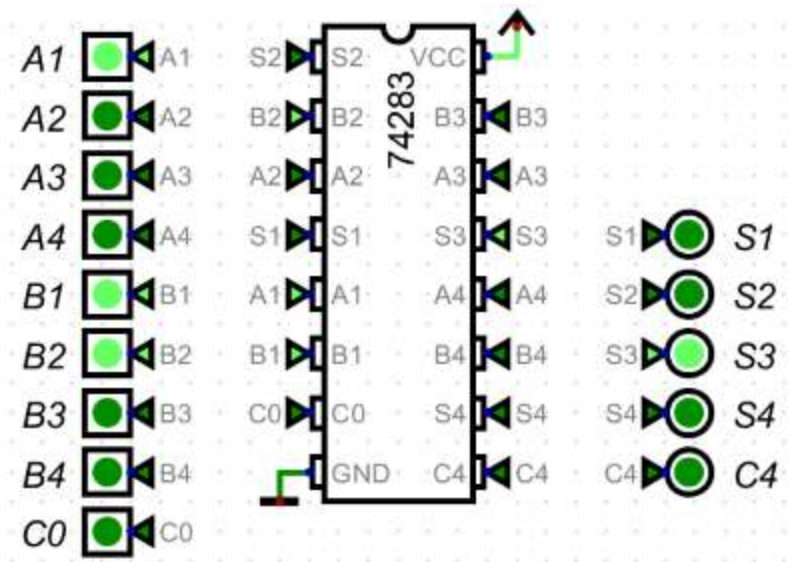
SN74F283 ACTIVE

4-Bit Binary Full Adders With Fast Carry

TEXAS INSTRUMENTS

- first number input A1,A2,A3,A4
- second number input B1,B2,B3,B4
- C0 is carry input
- S1,S2,S3,S4 are output
- C4 is resultant carry output

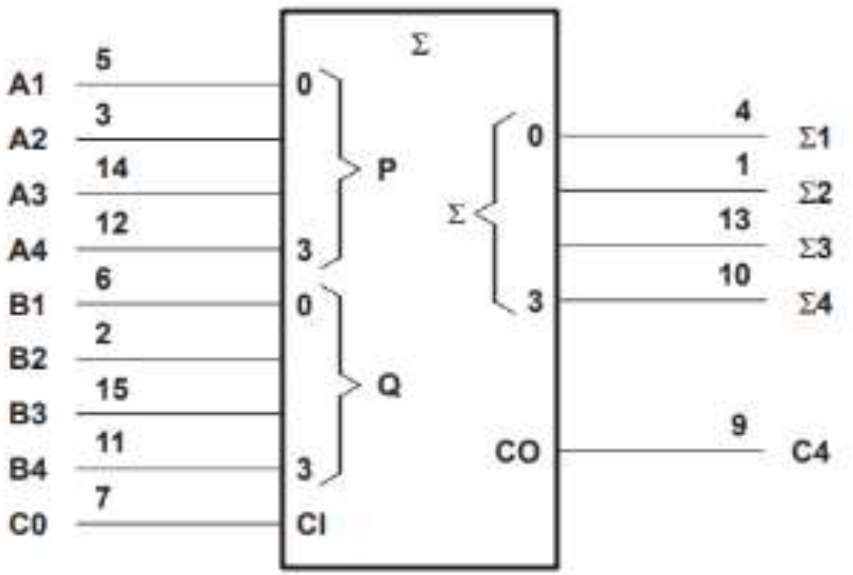




SN74F283 ACTIVE

4-Bit Binary Full Adders With Fast Carry

- NOTE: Input conditions at A1, B1, A2, B2, and C0 are used to determine outputs Σ1and Σ2 and the value of the internal carry C2.
- The values at C2, A3, B3, A4, and B4 are then used to determine outputs Σ3, Σ4, and C4
- Can be connected cascade to create higher bit adders (8,16,32)



TEXAS INSTRUMENTS

FUNCTION TABLE

INPUTS				OUTPUTS					
				WHEN C0 = L			WHEN C0 = H		
				WHEN C2 = L			WHEN C2 = H		
A1	B1	A2	B2	Σ1	Σ2	C2	Σ1	Σ2	C2
A3	B3	A4	B4	Σ3	Σ4	C4	Σ3	Σ4	C4
L	L	L	L	L	L	L	H	L	L
H	L	L	L	H	L	L	L	H	L
L	H	L	L	H	L	L	L	H	L
H	H	L	L	L	H	L	H	H	L
L	L	H	L	L	H	L	H	H	L
H	L	H	L	H	H	L	L	L	H
L	H	H	L	H	H	L	L	L	H
H	H	H	L	L	L	H	H	L	H
L	L	L	H	L	H	L	H	H	L
H	L	L	H	H	H	L	L	L	H
L	H	L	H	H	H	L	L	L	H
H	H	L	H	L	L	H	H	L	H
L	L	H	H	L	L	H	H	L	H
H	L	H	H	H	L	H	L	H	H
L	H	H	H	H	L	H	L	H	H
H	H	H	H	L	H	H	H	H	H

NOTE: Input conditions at A1, B1, A2, B2, and C0 are used to determine outputs Σ1 and Σ2 and the value of the internal carry C2. The values at C2, A3, B3, A4, and B4 are then used to determine outputs Σ3, Σ4, and C4.

SN74F283 ACTIVE

4-Bit Binary Full Adders With Fast Carry

TEXAS INSTRUMENTS

- Fast Carry means carry lookahead. We can observe ‘carry lookahead’ part in the datasheet
- input parts are different

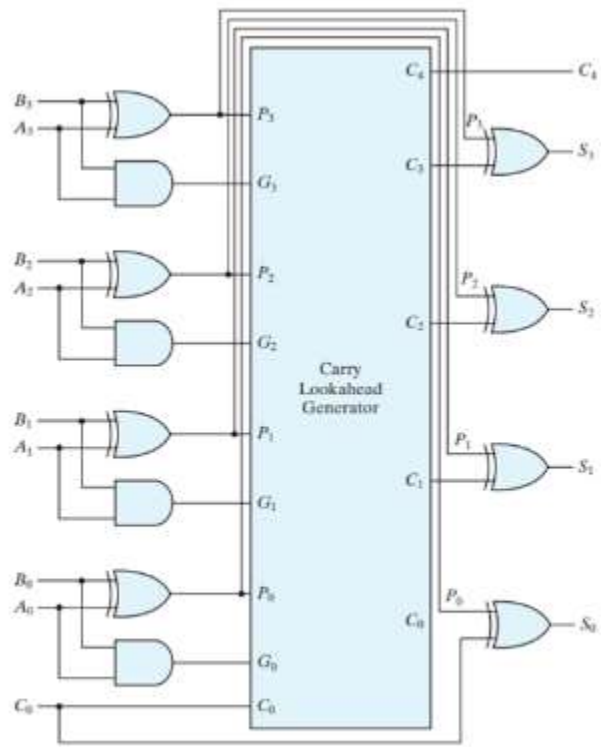
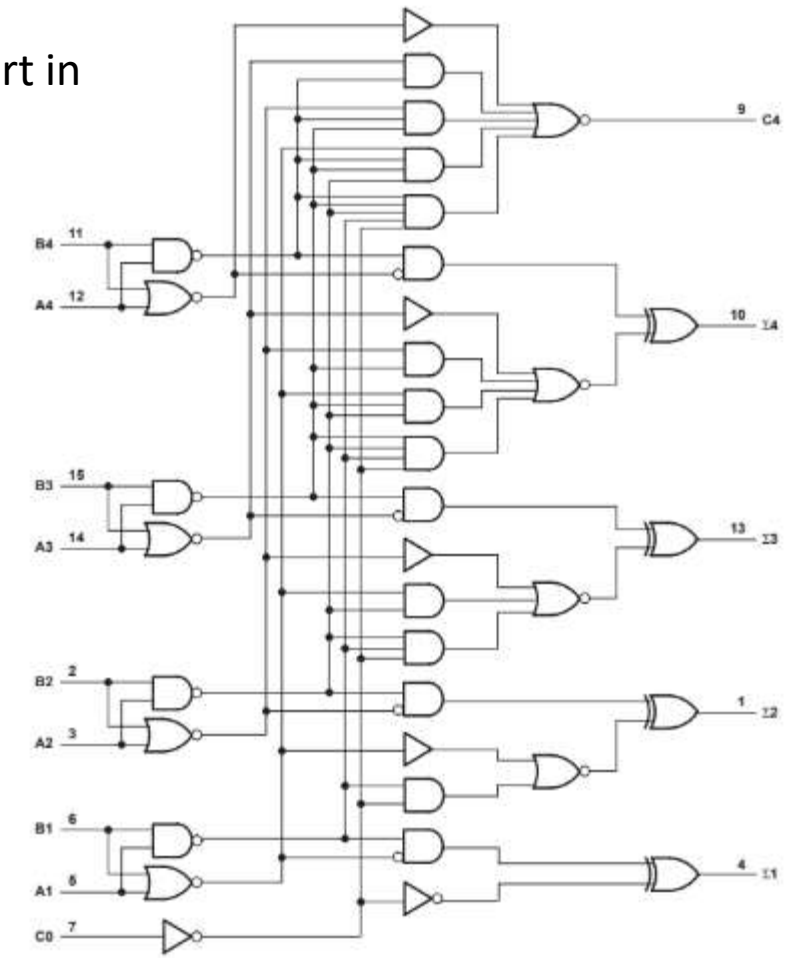
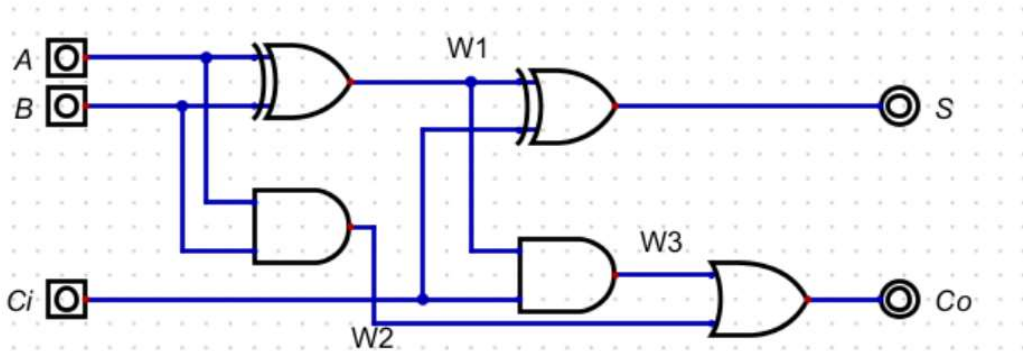


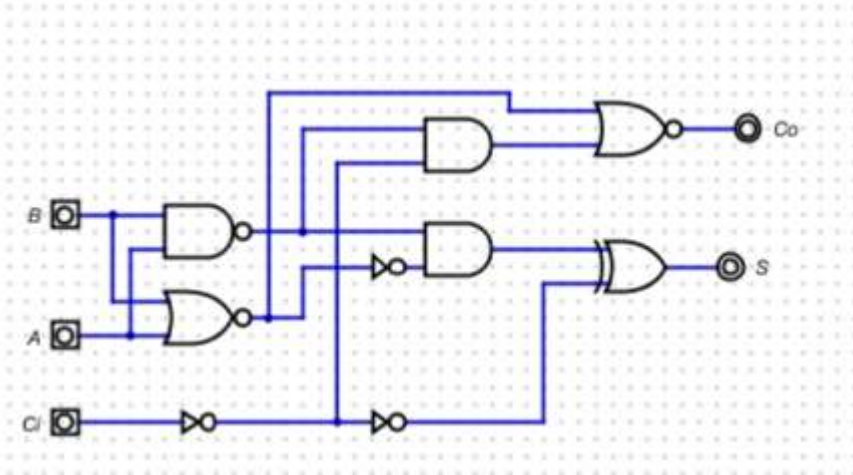
FIGURE 4.12
Four-bit adder with carry lookahead

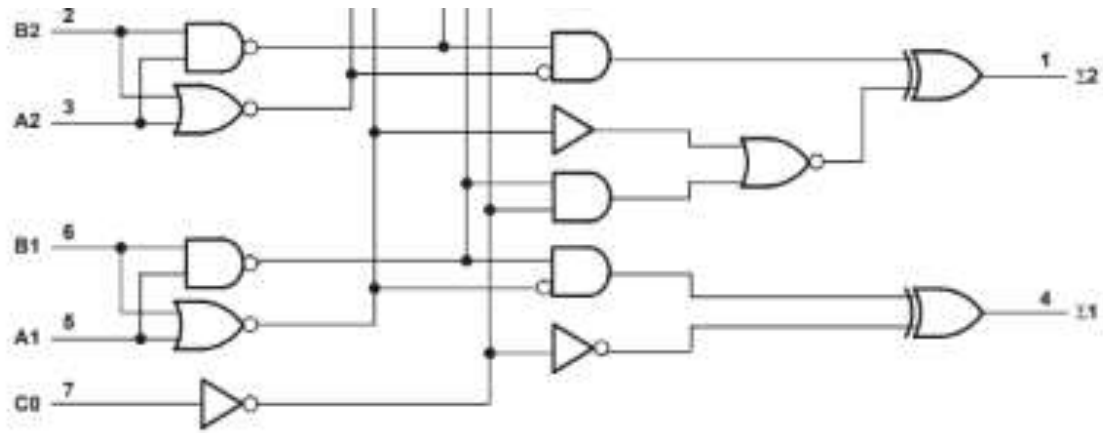


SN74F283 ACTIVE

4-Bit Binary Full Adders With Fast Carry



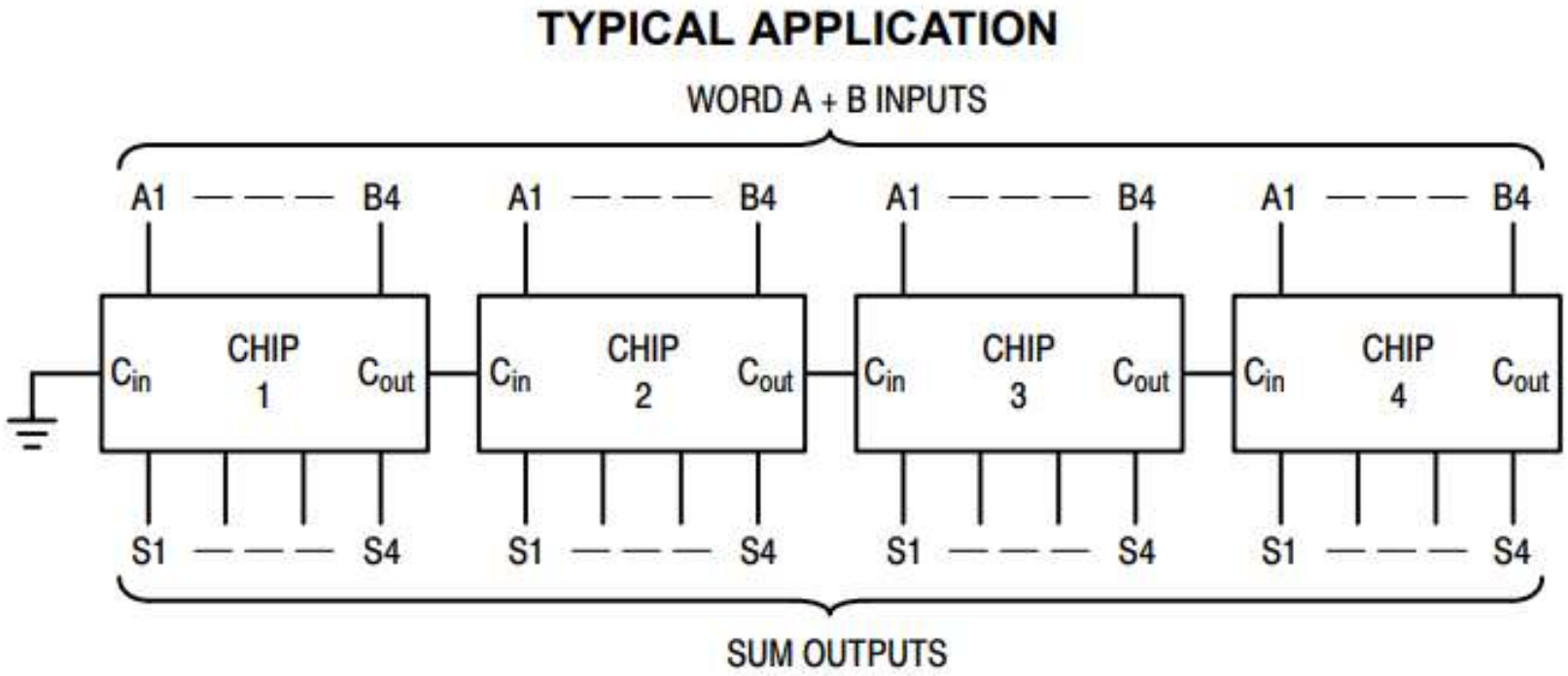




Experiment IC's

IC Number	Description
	Various gates
7447	BCD-to-seven-segment decoder
7474	Dual *D*-type flip-flops
7476	Dual *JK*-type flip-flops
7483	Four-bit binary adder
7493	Four-bit ripple counter
74151	8 × 1 multiplexer
74155	3 × 8 decoder
74157	Quadruple 2 × 1 multiplexers
74161	Four-bit synchronous counter
74189	16 × 4 random-access memory
74194	Bidirectional shift register
74195	Four-bit shift register
7730	Seven-segment LED display
72555	Timer (same as 555)

Cascade 4-Bit Adders



Calculation of 16−bit adder speed:

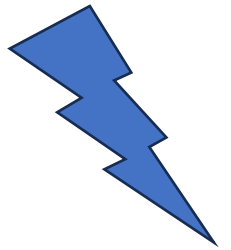
t_P total = t_P (Sum to Carry) + t_P (Carry to Sum) + 2 t_P (Carry to Carry)

The guaranteed 16−bit adder speed at 10 V, 25°C, C_L = 50 pF is:

t_p total = 290 + 310 + 300 = 900 ns

Figure 6. Using the MC14008B in a 16−Bit Adder Configuration

Binary Subtractor (2's Complement)



- The subtraction of **unsigned** binary numbers can be done most conveniently by means of **complements**
- subtraction **A - B** can be done by taking the 2's complement of B and adding it to A
- The 2's complement can be obtained by taking the **1's complement and adding 1**.

- Second way of 2's complement: start from left to the first 1. copy/paste. invert other numbers to the right

carry			1	
dec=6	0	1	1	0
1's comp	1	0	0	1
add 1	0	0	0	1
2's comp=-6	1	0	1	0

C	1		1		
10		1	0	1	0
-6		1	0	1	0
4	1	**0**	**1**	**0**	**0**

```
  10
  -6
+____
   4
```

Binary Subtractor (2's Complement)

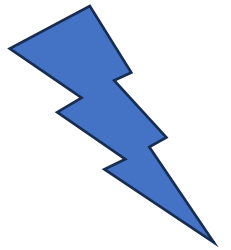
- if A>B, A-B is positive and no overflow can occur

Carry				
9	1	0	0	1
1's comp	0	1	1	0
add 1	0	0	0	1
2's comp=**-9**	0	1	1	1

Carry	1	1			
12		1	1	0	0
-9		0	1	1	1
3	1	**0**	**0**	**1**	**1**

```
   12
   -9
+_____
    3
```

Binary Subtractor (2's Complement)



- The subtraction of **unsigned** binary numbers can be done most conveniently by means of **complements**
- subtraction **A - B** can be done by taking the 2's complement of B and adding it to A
- The 2's complement can be obtained by taking the **1's complement and adding 1**.

- Second way of 2's complement: start from left to the first 1. copy/paste. invert other numbers to the right

carry			1	
dec=6	0	1	1	0
1's comp	1	0	0	1
add 1	0	0	0	1
2's comp=-6	1	0	1	0

C	1		1		
10		1	0	1	0
-6		1	0	1	0
4	1	**0**	**1**	**0**	**0**

```
  10
  -6
+____
   4
```

4-bit Binary Subtractor

- The circuit for subtracting A - B consists of an adder with inverters placed between each data input B and the corresponding input of the full adder.
- The **input carry Co** must be equal to 1 when subtraction is performed. (add 1)
- The addition and subtraction operations can be combined into one circuit with one common binary adder by including an exclusive-OR gate with each full adder
- When M = 0, the circuit is an adder, and when M = 1, the circuit becomes a subtractor

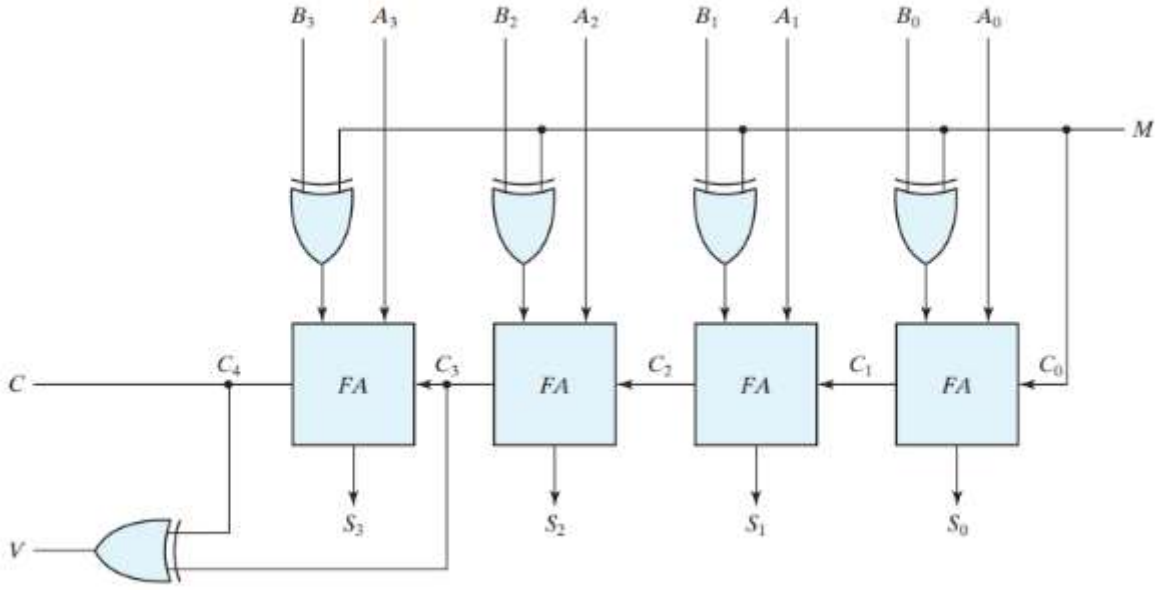
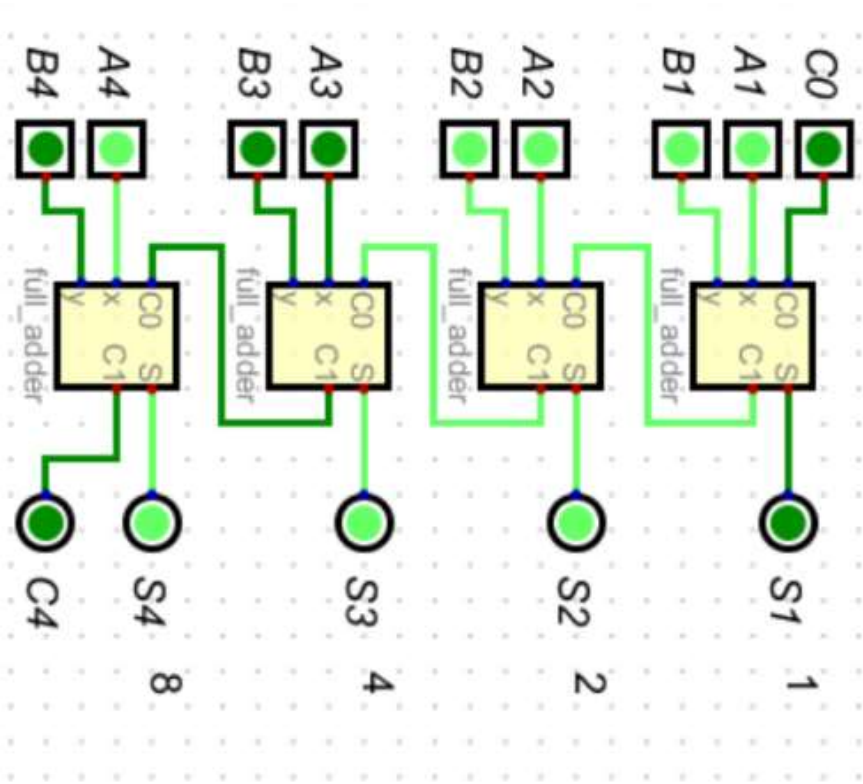


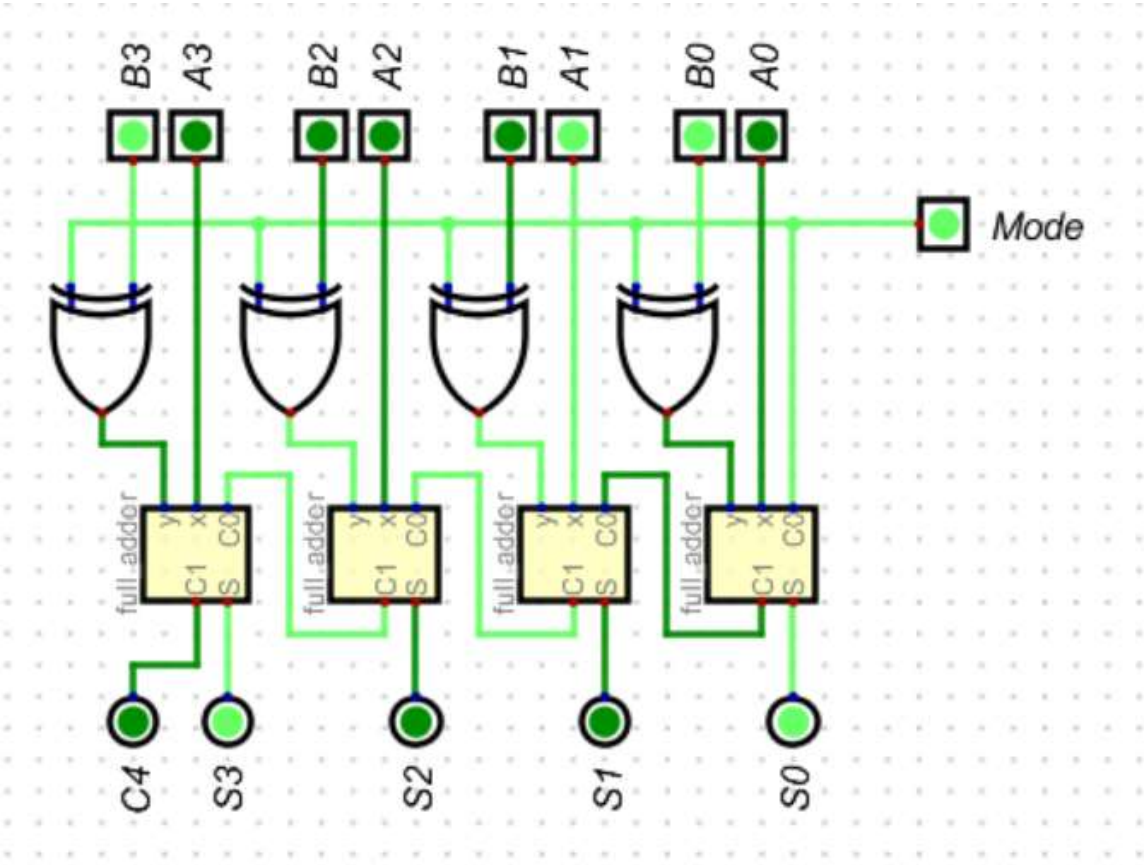
FIGURE 4.13
Four-bit adder–subtractor (with overflow detection)

- notice that:
- M=0; B XOR 0=B
- M=1; B XOR 1=B’ (1’s complement, add 1=C0=M)
- A-B=A + (B XOR 1) +1

4-bit Adder vs 4-Bit Subtractor

- Comparison of adder and subtractor





Binary Subtractor (2's Complement)

- if B>A, A-B will result a negative number, which is 2's complement of a positive number
- unsigned integer, thus result is always a positive number

carry				
9	1	0	0	1
1's comp	0	1	1	0
add 1	0	0	0	1
2's comp=-9	0	1	1	1

2+(-9)=-7 (=9)

C		1	1		
2		0	0	1	0
-9		0	1	1	1
-7 or 9		**1**	**0**	**0**	**1**

7	0	1	1	1
1's comp	1	0	0	0
add 1	0	0	0	1
2's comp=-7	1	0	0	1

Overflow

- When two numbers with n digits each are added and the sum is a number occupying n + 1 digits, we say that an overflow occurred.

- When two unsigned numbers are added, an overflow is detected from the end **carry out of the most significant position**

Overflow Examples in Arduino

```
1
2  // Arduino overflow example
3  // Senol Gulgonul
4  // 11.10.2023
5  uint8_t a = 254;
6  uint8_t b = 10;
7  uint8_t c = 0;
8  void setup() {
9    Serial.begin(9600);
10 }
11 void loop() {
12   c=a+b;
13   Serial.print("254+10: ");
14   Serial.println(c);
15   delay(1000);
16 }
17
18
19
```

Serial Monitor

```
254+10: 8
254+10: 8
254+10: 8
254+10: 8
```

```
1
2  // Arduino overflow example
3  // Senol Gulgonul
4  // 11.10.2023
5  uint8_t a = 3;
6  uint8_t b = 5;
7  uint8_t c = 0;
8  void setup() {
9    Serial.begin(9600);
10 }
11 void loop() {
12   c=a-b;
13   Serial.print("3-5: ");
14   Serial.println(c);
15   delay(1000);
16 }
17
18
19
```

Serial Monitor

```
3-5: 254
3-5: 254
```

Overflow Examples in Arduino

```
1
2  // Arduino overflow example
3  // Senol Gulgonul
4  // 11.10.2023
5  uint8_t a = 4;
6  uint8_t b = 10;
7  uint8_t c = 0;
8  void setup() {
9    Serial.begin(9600);
10 }
11 void loop() {
12   c=a+b;
13   Serial.print("result: ");
14   Serial.println(c);
15   delay(1000);
16 }
17
18
19
```

Serial Monitor

```
result: 14
result: 14
result: 14
```

Numbers in Arduino

Arduino Data Types	Value Assigned	Value Ranges
boolean	8 Bit	True or False
byte	8 Bit	0 to 255
char	8 Bit	-127 to 128
unsigned char	8 Bit	0 to 255
word	16 Bit	0 to 65535
unsigned int	16 Bit	0 to 65535
int	16 Bit	-32768 to 32767
long	32 Bit	-2,147,483,648 to 2,147,483,647
float	32 Bit	-3.4028235E38 to 3.4028235E38
www.TheEngineeringProjects.com		

4.7. Binary Multiplier

- Multiplication of binary numbers is performed in the same way as multiplication of decimal numbers.
- The multiplicand is multiplied by each bit of the multiplier, starting from the least significant bit.
- Each such multiplication forms a partial product.
- Successive partial products are shifted one position to the left.
- The final product is obtained from the sum of the partial products.
- The multiplication of two bits such as A0 and B0 produces a 1 if both bits are 1; otherwise, it produces a 0. This is identical to an AND operation.

X				1	1	0	1
Y					1	1	0
multiply				0	0	0	0
			1	1	0	1	
		1	1	0	1		
sum	1	0	0	1	1	1	0

Multiply 1101 and 110:

$$
\begin{array}{r}
1101 \\
\times \quad 110 \\
\hline
0000 \\
1101 \\
1101 \\
\hline
1001110
\end{array}
\quad \rightarrow 1101_2 \times 110_2 = 1001110_2
$$

4.7. Binary Multiplier

- The multiplicand is multiplied by each bit of the multiplier, starting from the least significant bit.
- Each such multiplication forms a partial product.
- Successive partial products are shifted one position to the left.
- The final product is obtained from the sum of the partial products.

B3B2B1B0				1	1	0	1
A2A1A0					1	1	0
A0x				0	0	0	0
A1x			1	1	0	1	
A2x		1	1	0	1		
sum	1	0	0	1	1	1	0

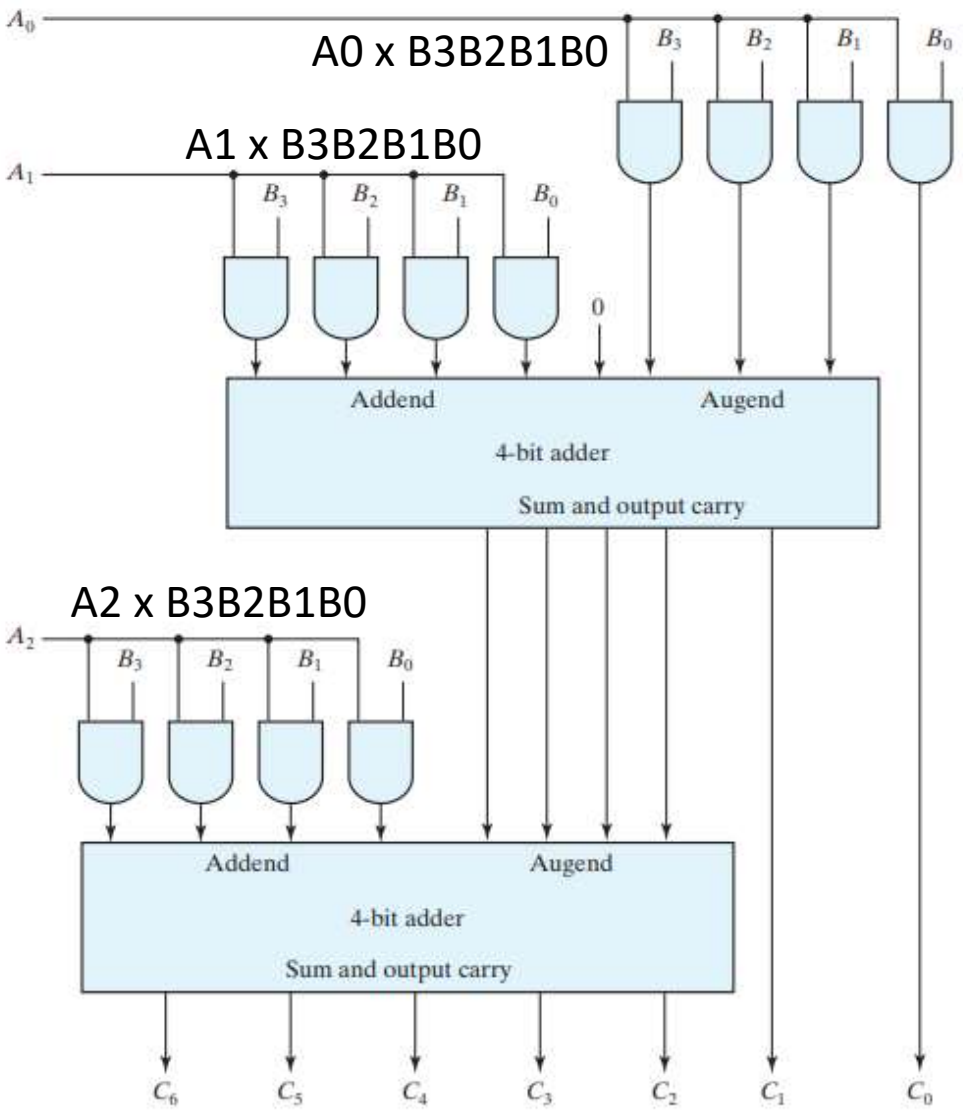


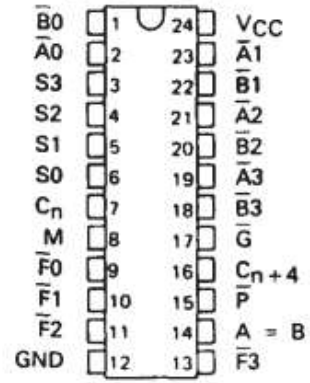
FIGURE 4.16
Four-bit by three-bit binary multiplier

SN54LS181, SN54S181
SN74LS181, SN74S181
ARITHMETIC LOGIC UNITS/FUNCTION GENERATORS

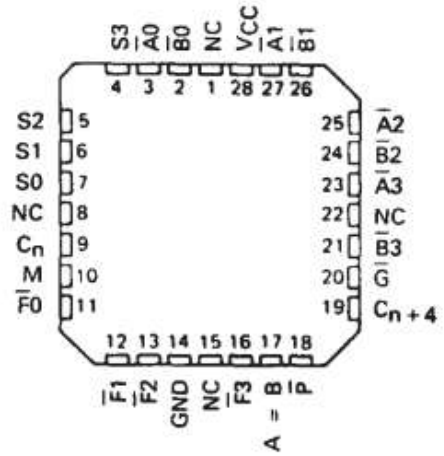
SDLS136 – DECEMBER 1972 – REVISED MARCH 1988

- Full Look-Ahead for High-Speed Operations on Long Words
- Input Clamping Diodes Minimize Transmission-Line Effects
- Darlington Outputs Reduce Turn-Off Time
- Arithmetic Operating Modes:
  - Addition
  - Subtraction
  - Shift Operand A One Position
  - Magnitude Comparison
  - Plus Twelve Other Arithmetic Operations
- Logic Function Modes:
  - Exclusive-OR
  - Comparator
  - AND, NAND, OR, NOR
  - Plus Ten Other Logic Operations

SN54LS181, SN54S181 . . . J OR W PACKAGE
SN74LS181, SN74S181 . . . DW OR N PACKAGE
(TOP VIEW)



SN54LS181, SN54S181 . . . FK PACKAGE
(TOP VIEW)

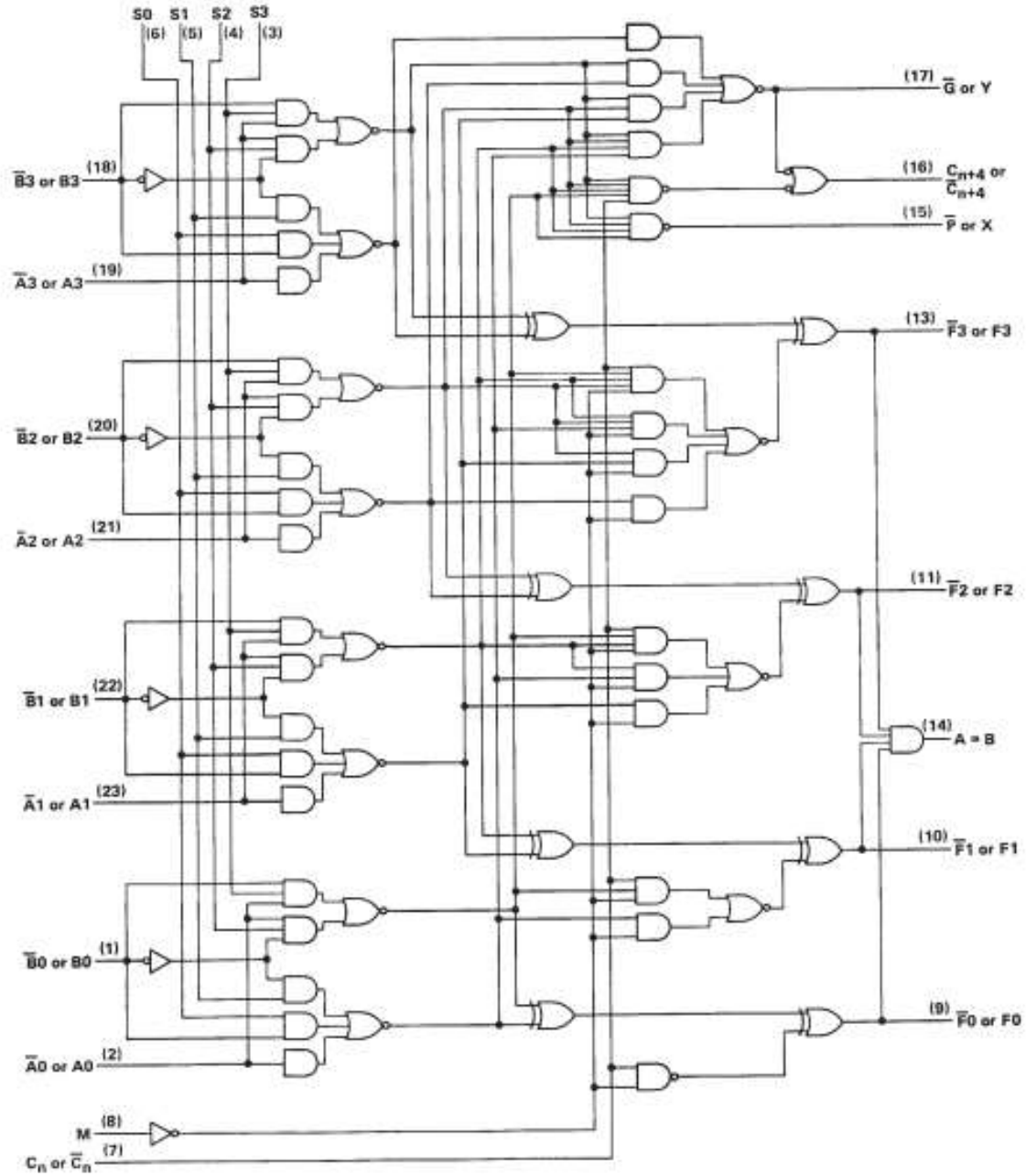


NC - No internal connection

ARITHMETIC LOGIC UNIT

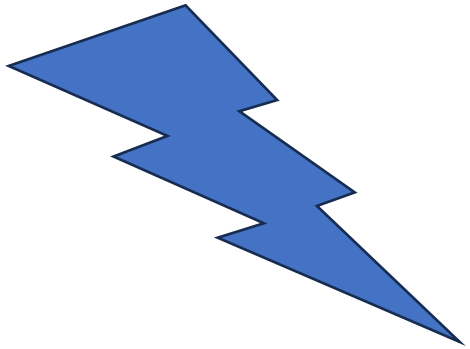
SELECTION				ACTIVE-HIGH DATA		
				M = H	M = L; ARITHMETIC OPERATIONS	
S3	S2	S1	S0	LOGIC FUNCTIONS	C_n = H (no carry)	C_n = L (with carry)
L	L	L	L	$F = \overline{A}$	F = A	F = A PLUS 1
L	L	L	H	$F = \overline{A + B}$	F = A + B	F = (A + B) PLUS 1
L	L	H	L	$F = \overline{A}B$	$F = A + \overline{B}$	$F = (A + \overline{B})$ PLUS 1
L	L	H	H	F = 0	F = MINUS 1 (2's COMPL)	F = ZERO
L	H	L	L	$F = \overline{AB}$	F = A PLUS $A\overline{B}$	F = A PLUS $A\overline{B}$ PLUS 1
L	H	L	H	$F = \overline{B}$	F = (A + B) PLUS $A\overline{B}$	F = (A + B) PLUS $A\overline{B}$ PLUS 1
L	H	H	L	$F = A \oplus B$	F = A MINUS B MINUS 1	F = A MINUS B
L	H	H	H	$F = A\overline{B}$	$F = A\overline{B}$ MINUS 1	F = AB
H	L	L	L	$F = \overline{A} + B$	F = A PLUS AB	F = A PLUS AB PLUS 1
H	L	L	H	$F = \overline{A \oplus B}$	F = A PLUS B	F = A PLUS B PLUS 1
H	L	H	L	F = B	F = (A + $\overline{B}$) PLUS AB	F = (A + $\overline{B}$) PLUS AB PLUS 1
H	L	H	H	F = AB	F = AB MINUS 1	F = AB
H	H	L	L	F = 1	F = A PLUS A†	F = A PLUS A PLUS 1
H	H	L	H	$F = A + \overline{B}$	F = (A + B) PLUS A	F = (A + B) PLUS A PLUS 1
H	H	H	L	F = A + B	F = (A + $\overline{B}$) PLUS A	F = (A + $\overline{B}$) PLUS A PLUS 1
H	H	H	H	F = A	F = A MINUS 1	F = A

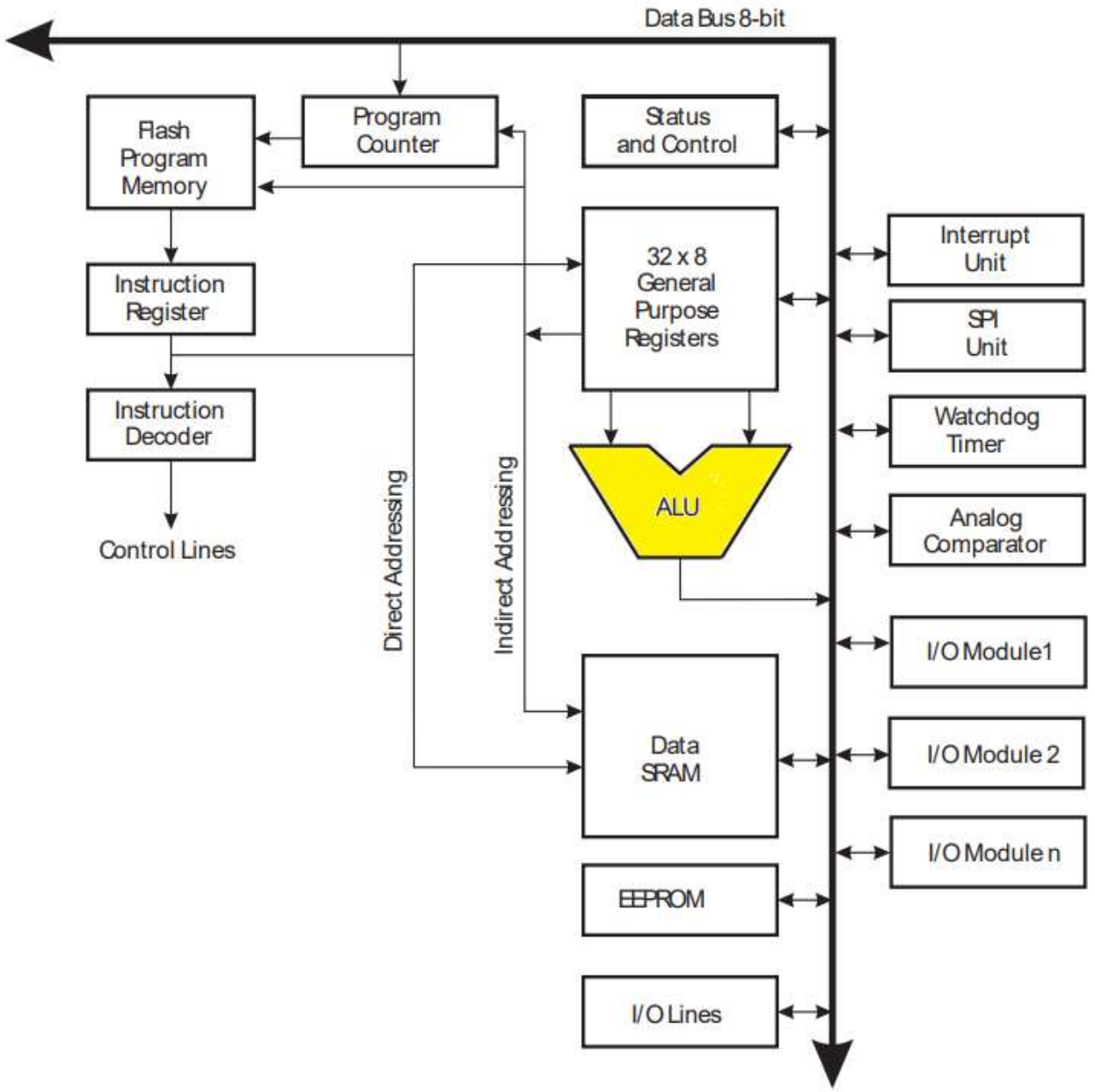
ARITHMETIC LOGIC UNIT



Pin numbers shown are for DW, J, N, and W packages.

ARITHMETIC LOGIC UNIT





MICROCHIP

ARITHMETIC LOGIC UNIT

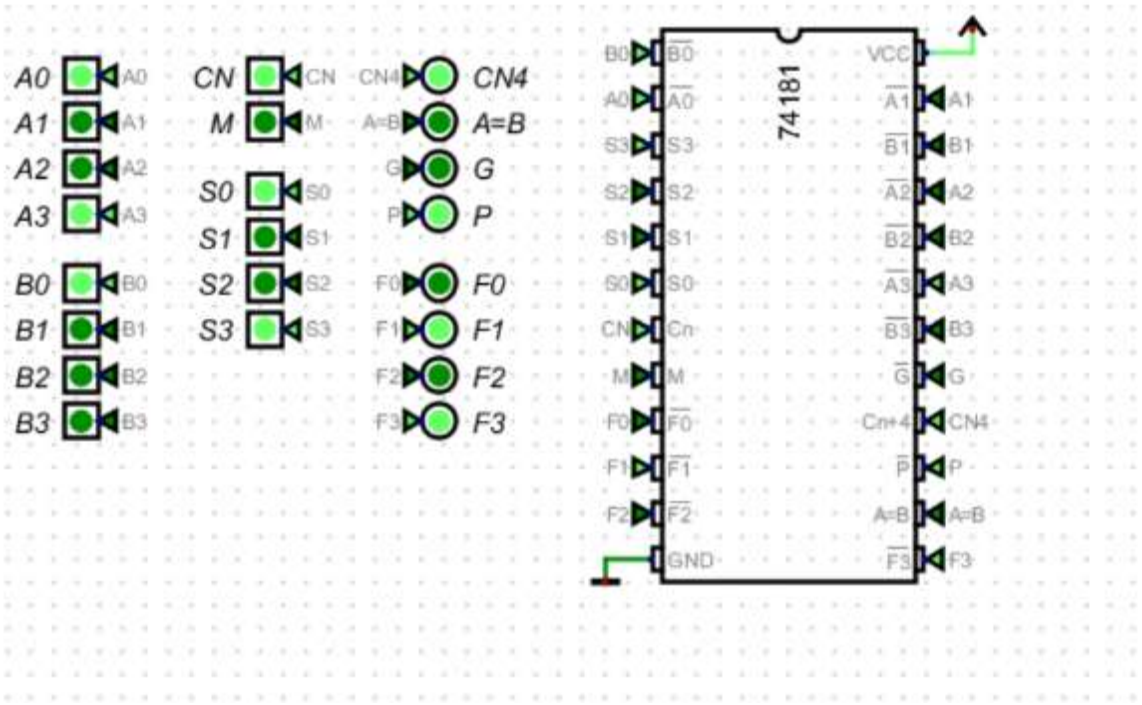
- M=L for Arithmetic, Cn=1 no carry
- S0=1, S3=1 for addition: A plus B

FAIRCHILD SEMICONDUCTOR™

DM74LS181
4-Bit Arithmetic Logic Unit

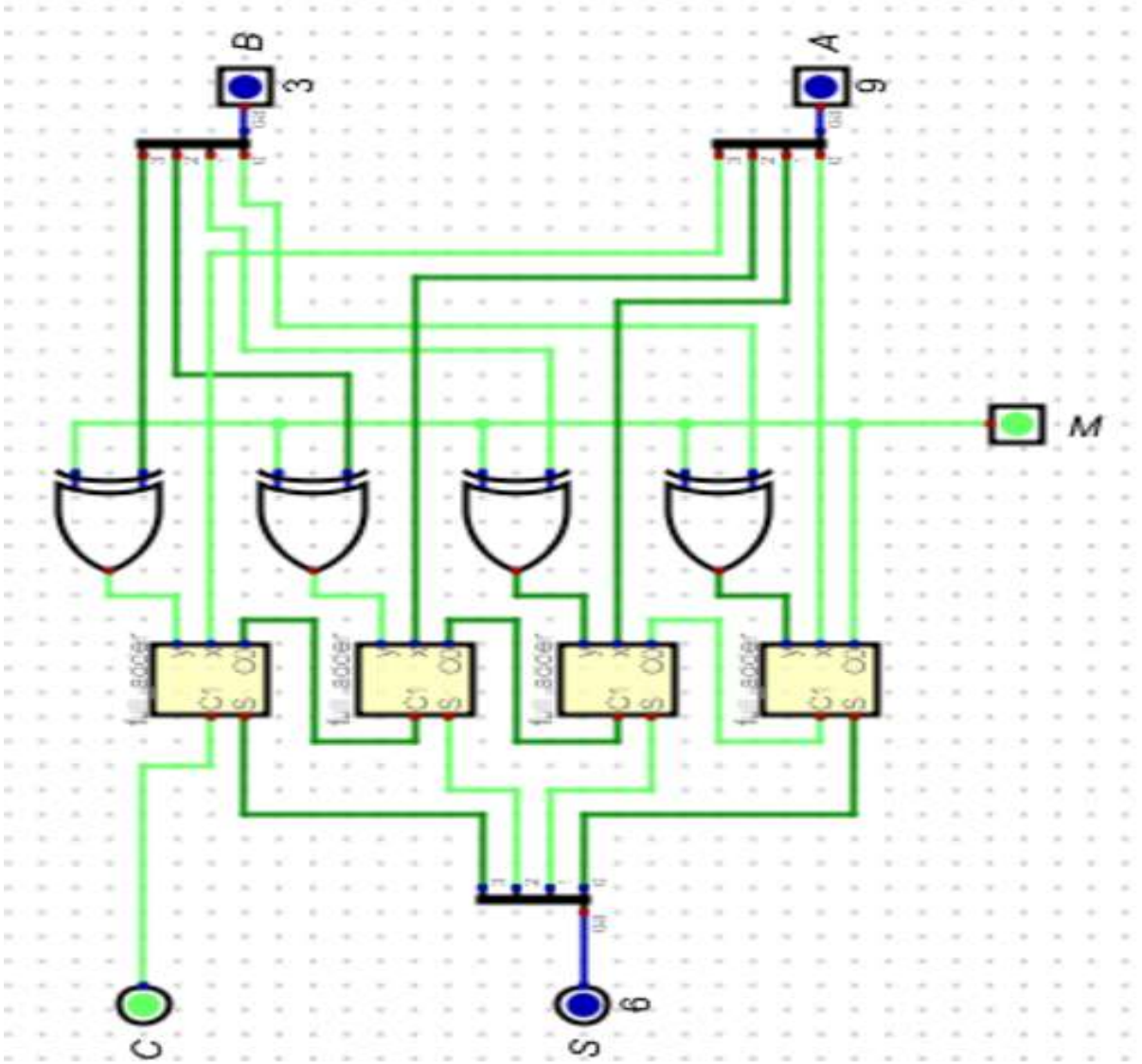
Function Table

Mode Select Inputs				Active LOW Operands & F_n Outputs		Active HIGH Operands & F_n Outputs	
				Logic	Arithmetic (Note 2)	Logic	Arithmetic (Note 2)
S3	S2	S1	S0	(M = H)	(M = L) (C_n = L)	(M = H)	(M = L) (C_n = H)
L	L	L	L	$\overline{A}$	A minus 1	$\overline{A}$	A
L	L	L	H	$\overline{AB}$	AB minus 1	$\overline{A}+\overline{B}$	A + B
L	L	H	L	$\overline{A}+B$	$A\overline{B}$ minus 1	$\overline{A}B$	$A+\overline{B}$
L	L	H	H	Logic 1	minus 1	Logic 0	minus 1
L	H	L	L	$\overline{A}+\overline{B}$	A plus (A + $\overline{B}$)	$\overline{AB}$	A plus $A\overline{B}$
L	H	L	H	$\overline{B}$	AB plus (A + $\overline{B}$)	$\overline{B}$	(A + B) plus $A\overline{B}$
L	H	H	L	$\overline{A}\oplus\overline{B}$	A minus B minus 1	$A\oplus B$	A minus B minus 1
L	H	H	H	$A+\overline{B}$	$A+\overline{B}$	$A\overline{B}$	AB minus 1
H	L	L	L	$\overline{A}B$	A plus (A + B)	$\overline{A}+B$	A plus AB
H	L	L	H	$A\oplus B$	A plus B	$\overline{A\oplus B}$	A plus B
H	L	H	L	B	$A\overline{B}$ plus (A + B)	B	(A + $\overline{B}$) plus AB
H	L	H	H	A + B	A + B	AB	AB minus 1
H	H	L	L	Logic 0	A plus A (Note 1)	Logic 1	A plus A (Note 1)
H	H	L	H	$A\overline{B}$	AB plus A	$A+\overline{B}$	(A + B) plus A
H	H	H	L	AB	$A\overline{B}$ minus A	A + B	(A + $\overline{B}$) plus A
H	H	H	H	A	A	A	A minus 1



ALU_4bit

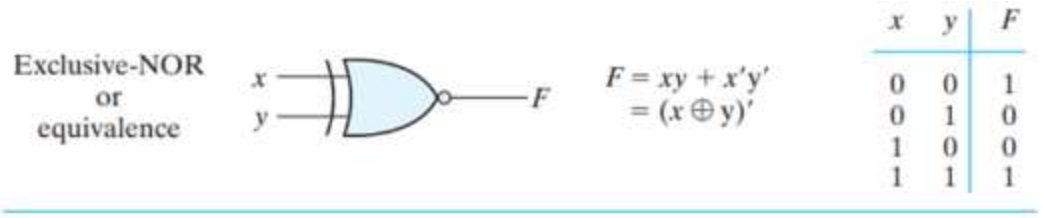
- **4bit ALU performs to operation ADD or SUB according to Mode selection, M=0 ADD, M=1 SUB**

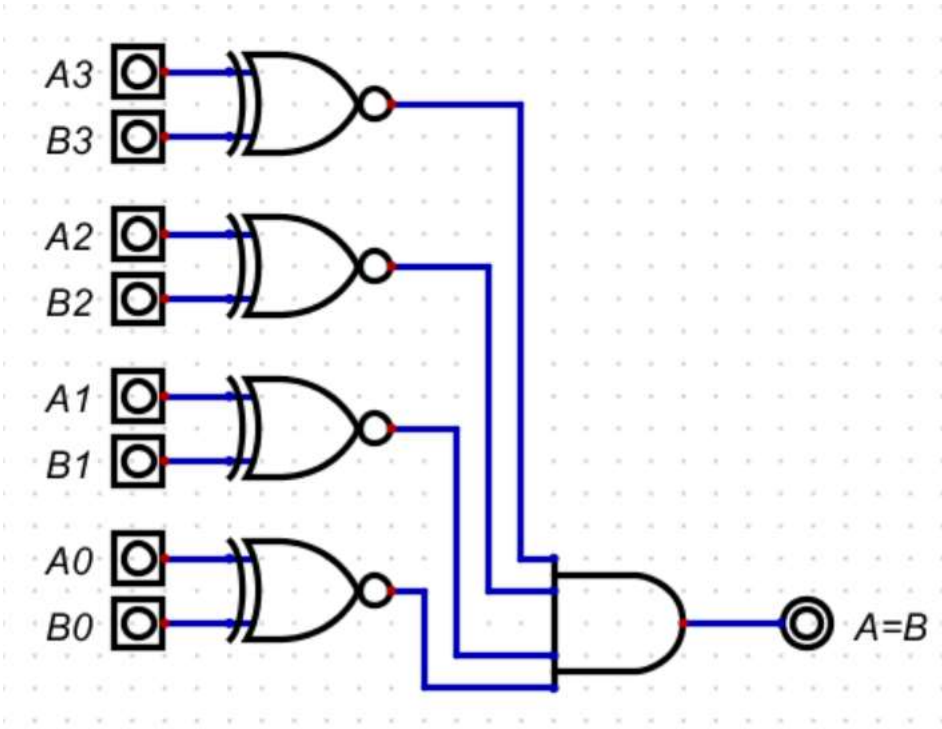


Comparator

- A magnitude comparator is a combinational circuit that compares two numbers A and B and determines their relative magnitudes.
- The outcome of the comparison is specified by three binary variables that indicate whether A > B, A = B, or A < B.

- XNOR logic gate gives 1 if two inputs are the same
- so equivalence check is easy by AND and XNOR gates
- E=(A0 ⊙ B0)(A1 ⊙ B1)(A2 ⊙ B2)(A3 ⊙ B3)





Comparator

- To determine whether A is greater or less than B, we inspect the relative magnitudes of pairs of significant digits, starting from the most significant position.
- to check if A3>B3, only case is A3=1 > B3=0, A3B3'=1 must be 1
- If the two digits of a pair are equal (x3, A3=B3 condition), we compare the next lower significant pair of digits.
- by XNOR gate above conditions can be evaluated

- Xi=Ai ⊙ Bi

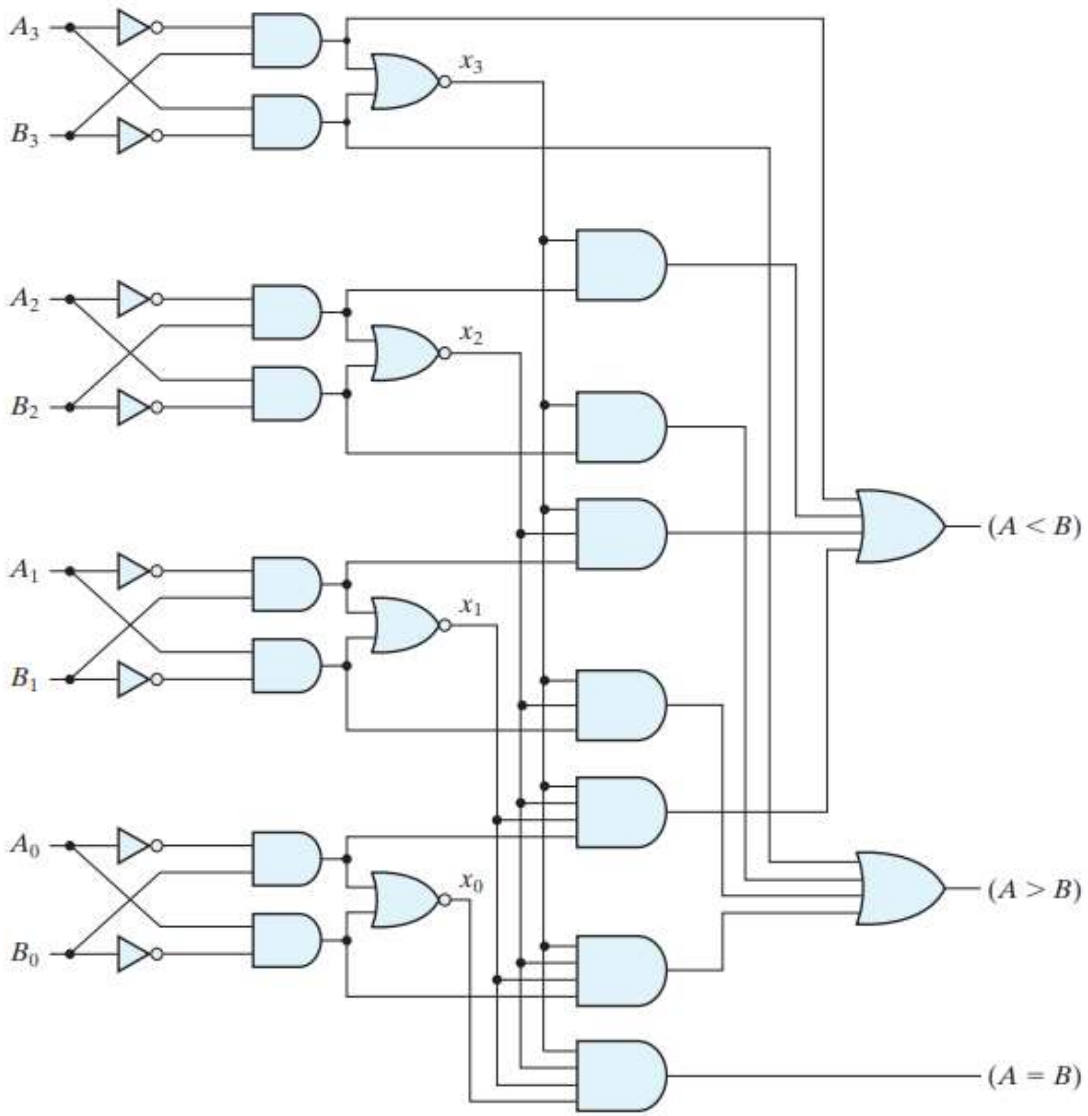
$$(A > B) = A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$$
$$(A < B) = A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1 + x_3x_2x_1A_0'B_0$$

Comparator

- (A=B) = (A0 ⊙ B0)(A1 ⊙ B1)(A2 ⊙ B2)(A3

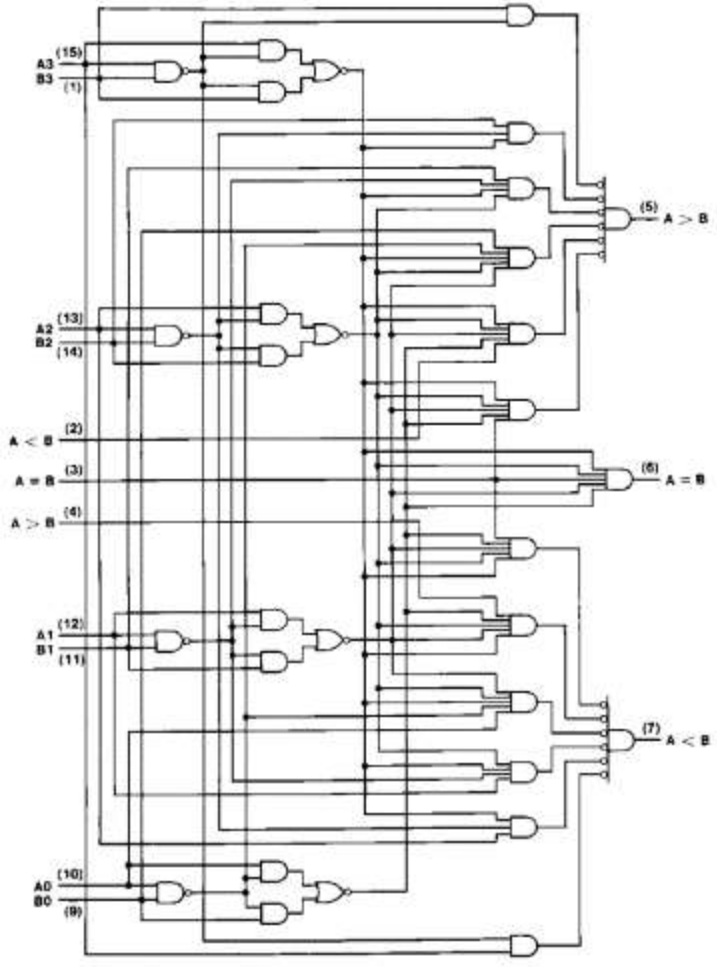
$$(A > B) = A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3$$

$$(A < B) = A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1 + x_3$$



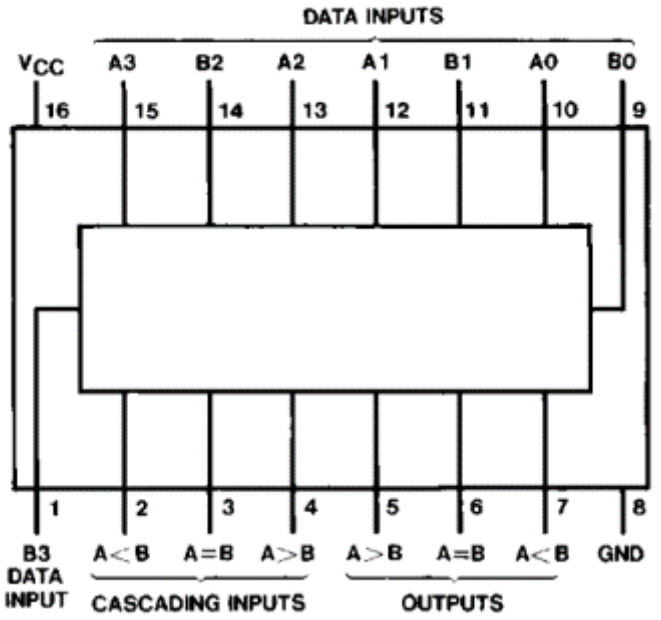
Comparator

- The outcome of the comparison is specified by three binary variables that indicate whether A > B, A = B, or A < B.



FAIRCHILD
SEMICONDUCTOR™

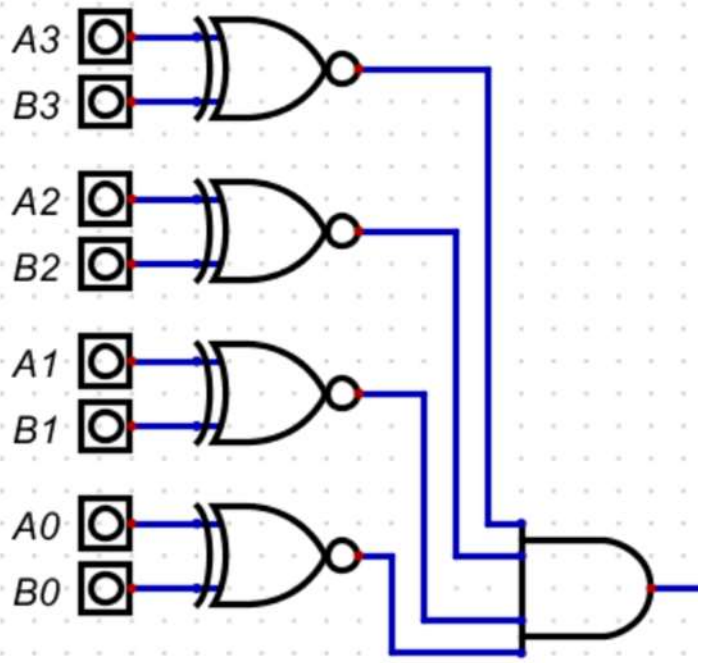
DM74LS85
4-Bit Magnitude Comparator

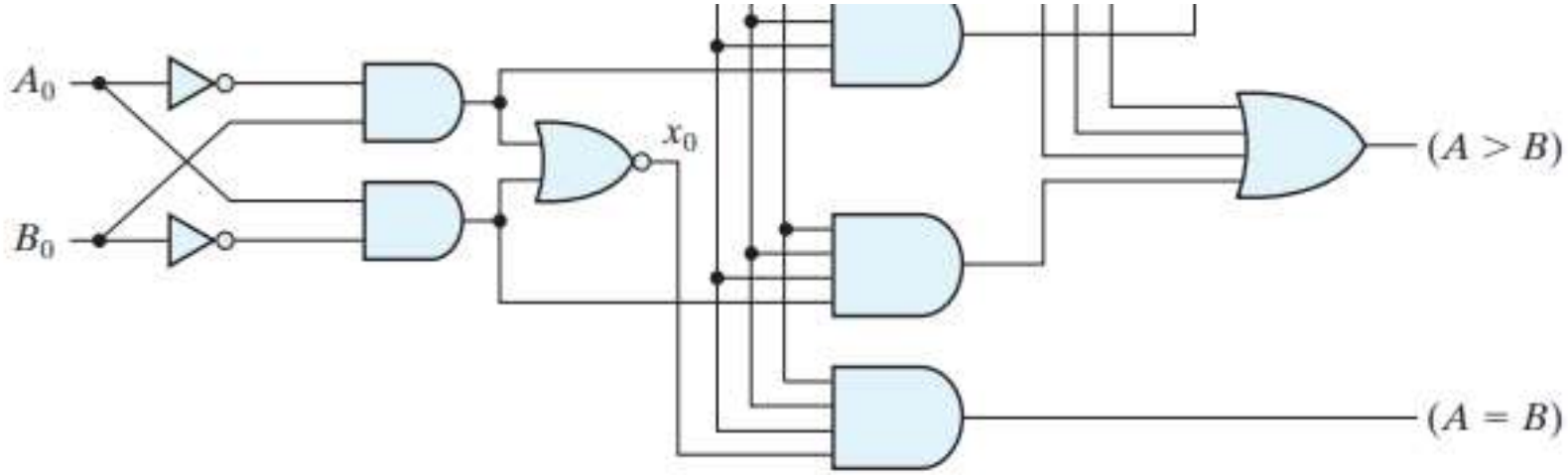


cascade inputs are for cascade connections of higher bits

Comparator

- is this XNOR in the logic diagram?
- x2=(A2’B2+A2B2’)’=(A2+B2’)(A2’+B2)=A2A2’+A2B2+B2’A2’+B2’B2=A2B2+A2’B2’
- YES

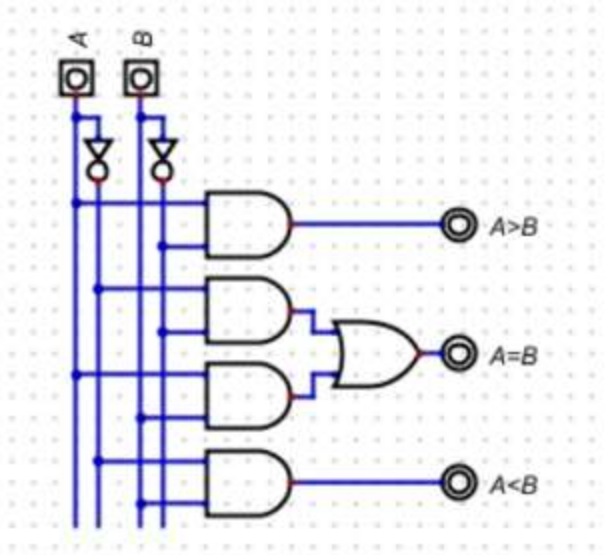


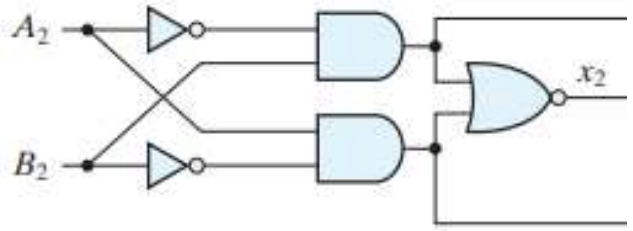


Two-bit Comparator using digital sw

- design using Digital sw, starting from truth map

A	B	A>B	A=B	A<B
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0





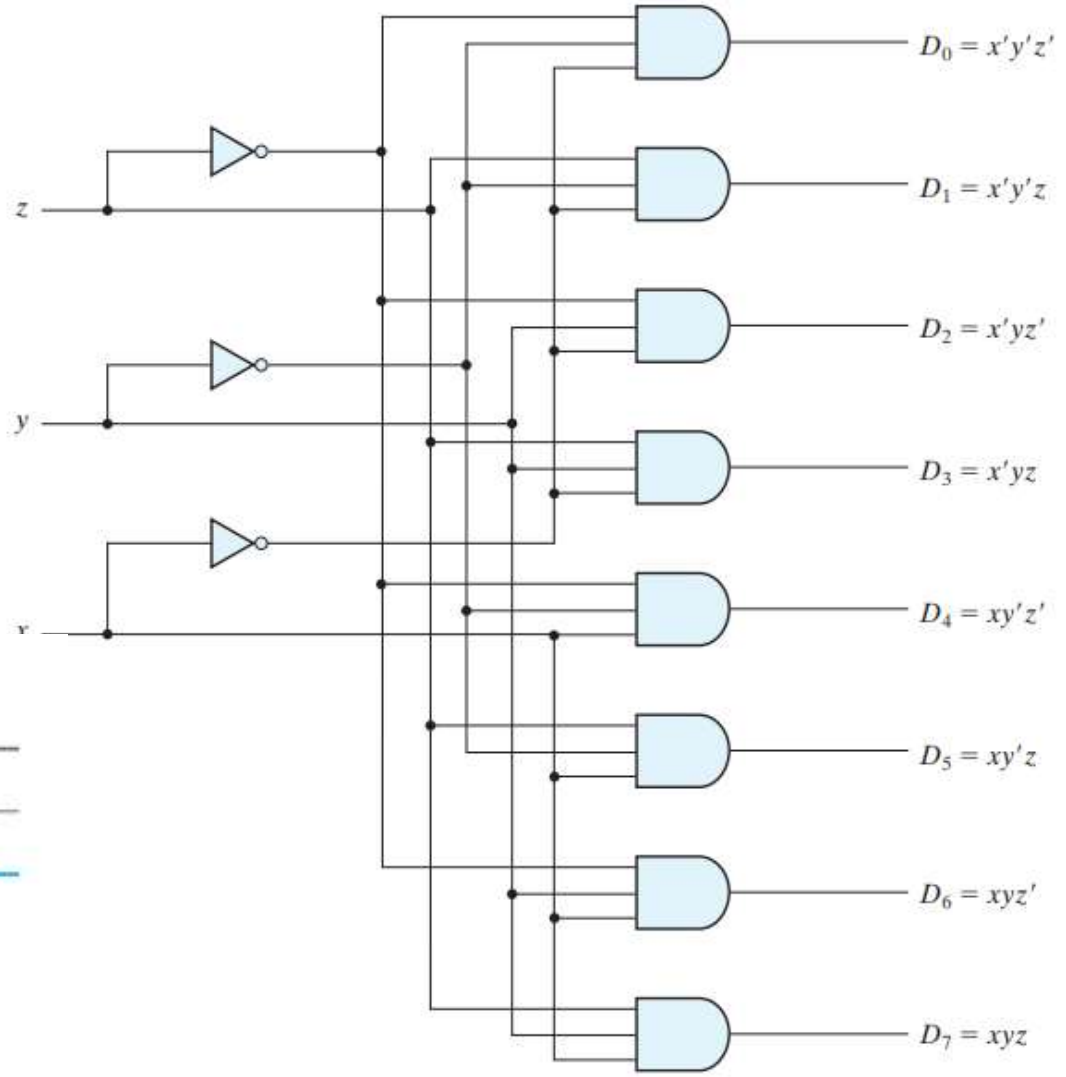
DECODER

- **Decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines.**
- **As an example, consider the three-to-eight-line decoder circuit of Fig. 4.18**

- **Do=x'y'z', D1=x'y'z …**

Table 4.6
Truth Table of a Three-to-Eight-Line Decoder

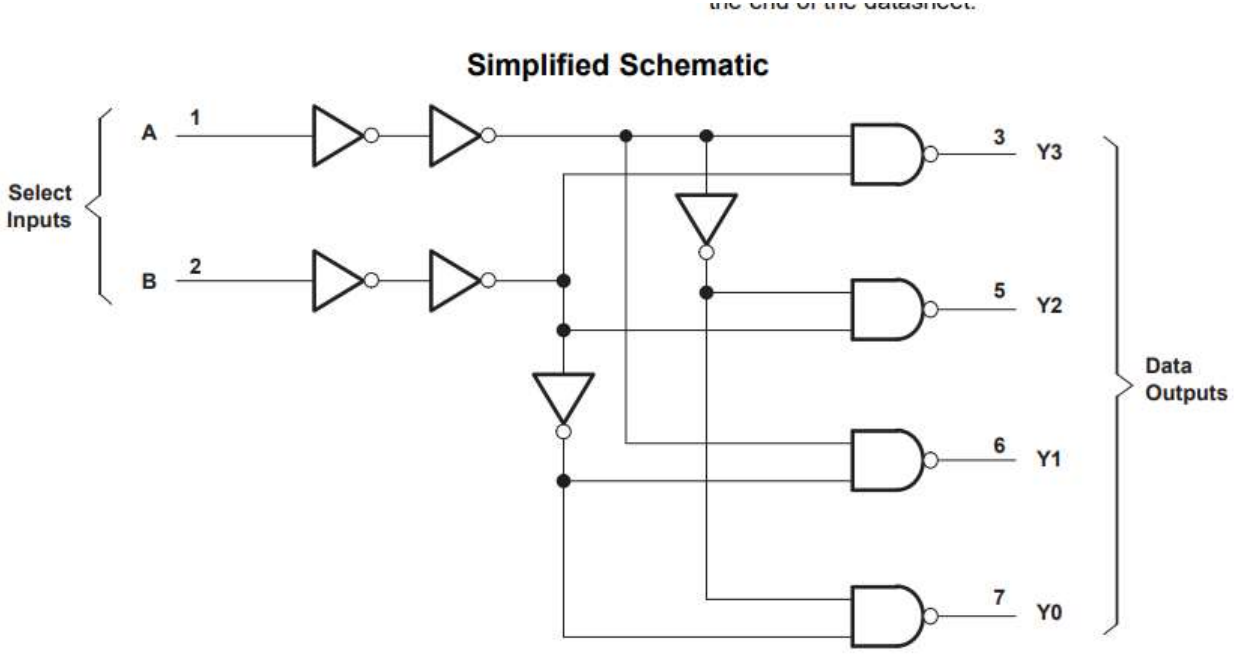
Inputs			Outputs							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



.18
eight-line decoder

DECODER-APPLICATIONS

- The SN74LVC1G139 2-line to 4-line decoder is designed to be used in high-performance memory decoding or data-routing applications



10.2 Typical Application

This is an address line decoder using a 16-bit bus example; address bus lines 14 and 15 are decoded and drive four active low chip selects. Each output covers 16K address space mapped by the address bus lines 0 through 13.

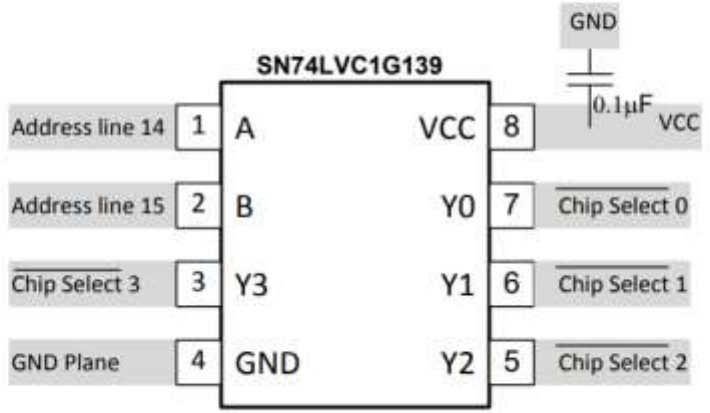
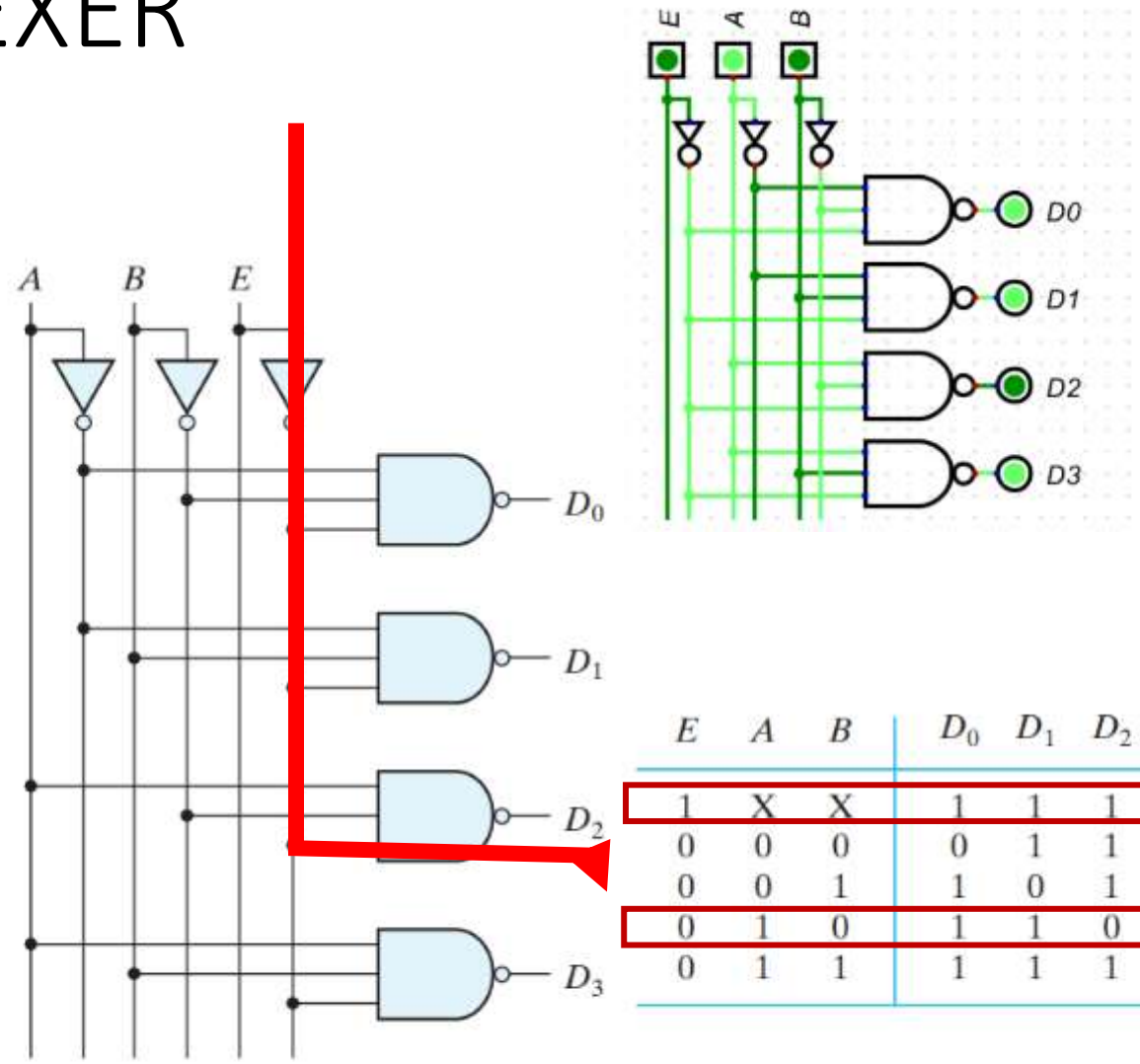


Figure 7. Typical Application Diagram

DECODER/DEMULTIPLEXER

- A decoder with enable input can function as a **demultiplexer**—a circuit that receives information from a single line and directs it to one of 2^n possible output lines.
- The selection of a specific output is controlled by the bit combination of n selection lines.
- The decoder of Fig. 4.19 can function as a one-to-four-line demultiplexer when **E is taken as a data** input line and **A and B are taken as the selection** inputs.
- For example, if the selection lines AB = 10, output D2 will FOLLOW the input value E, while all other outputs are maintained at 1



E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

FIGURE 4.19
Two-to-four-line decoder with enable input

DEMULTIPLEXER Applications

2 Applications

- AV Receivers
- Audio Docks: Portable
- Blu-ray® Players and Home Theater
- MP3 Players/Recorders
- Personal Digital Assistants (PDAs)
- Power: Telecom/Server AC/DC Supply: Single Controller: Analog and Digital
- Solid State Drives (SSDs): Client and Enterprise
- TVs: LCD/Digital and High-Definition (HDTVs)
- Tablets: Enterprise
- Video Analytics: Server
- Wireless Headsets, Keyboards, and Mice

9.2 Typical Application

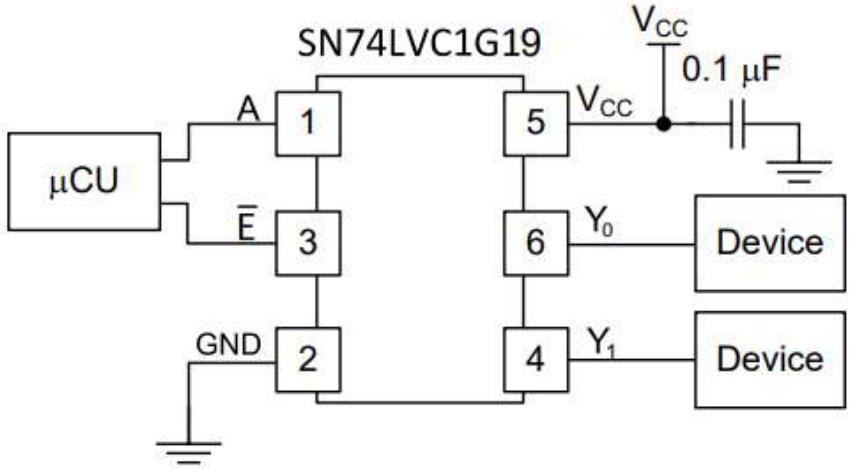
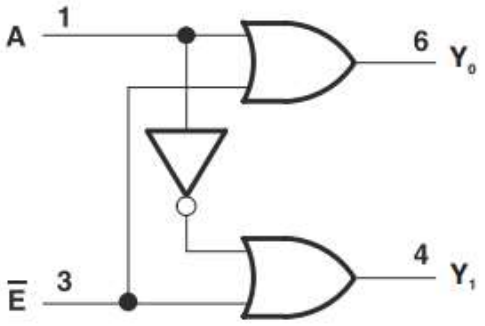


Figure 5. Typical Application Diagram

Simplified Schematic



The SN74LVC1G19 device is a 1-of-2 decoder / demultiplexer. When $\overline{E}$ input is high, the decoder will be disabled and both outputs will be high. When $\overline{E}$ input is low, the A input selects which output will be low.

DECODER APPLICATIONS

9.1 Application Information

The SN74HCS238 is used to control multiple devices that operate on a shared data bus. A decoder provides the capability to have a binary encoded input activate only one of the device's outputs. This is ideal for solid state memory applications where multiple devices have to be read or written to with a limited number of GPIO pins used on the system controller. The decoder is used to activate the chip select (CS) input to the selected memory device, and the controller can then read or write from that device alone when using a shared bus.

9.2 Typical Application

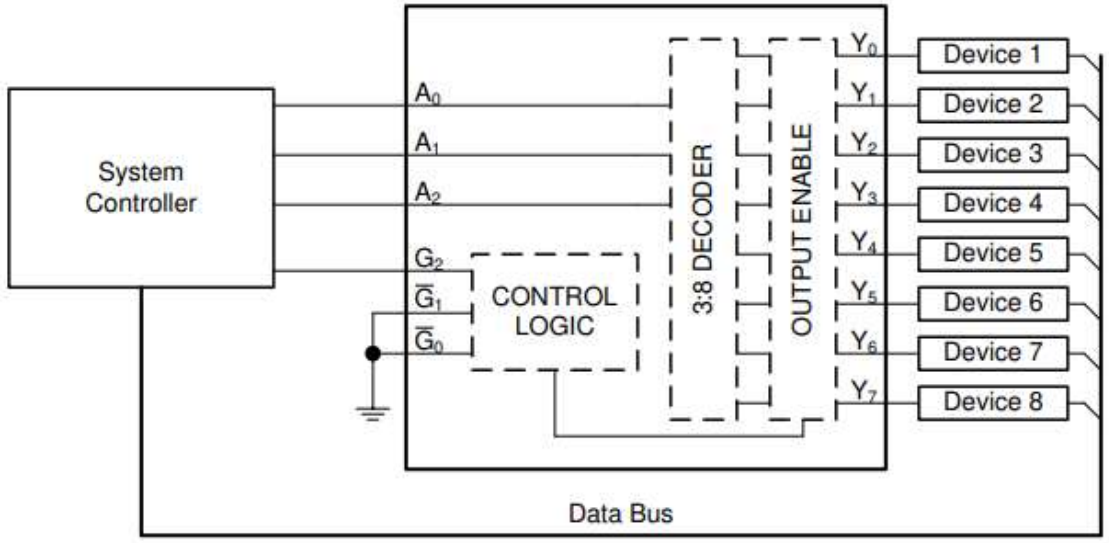
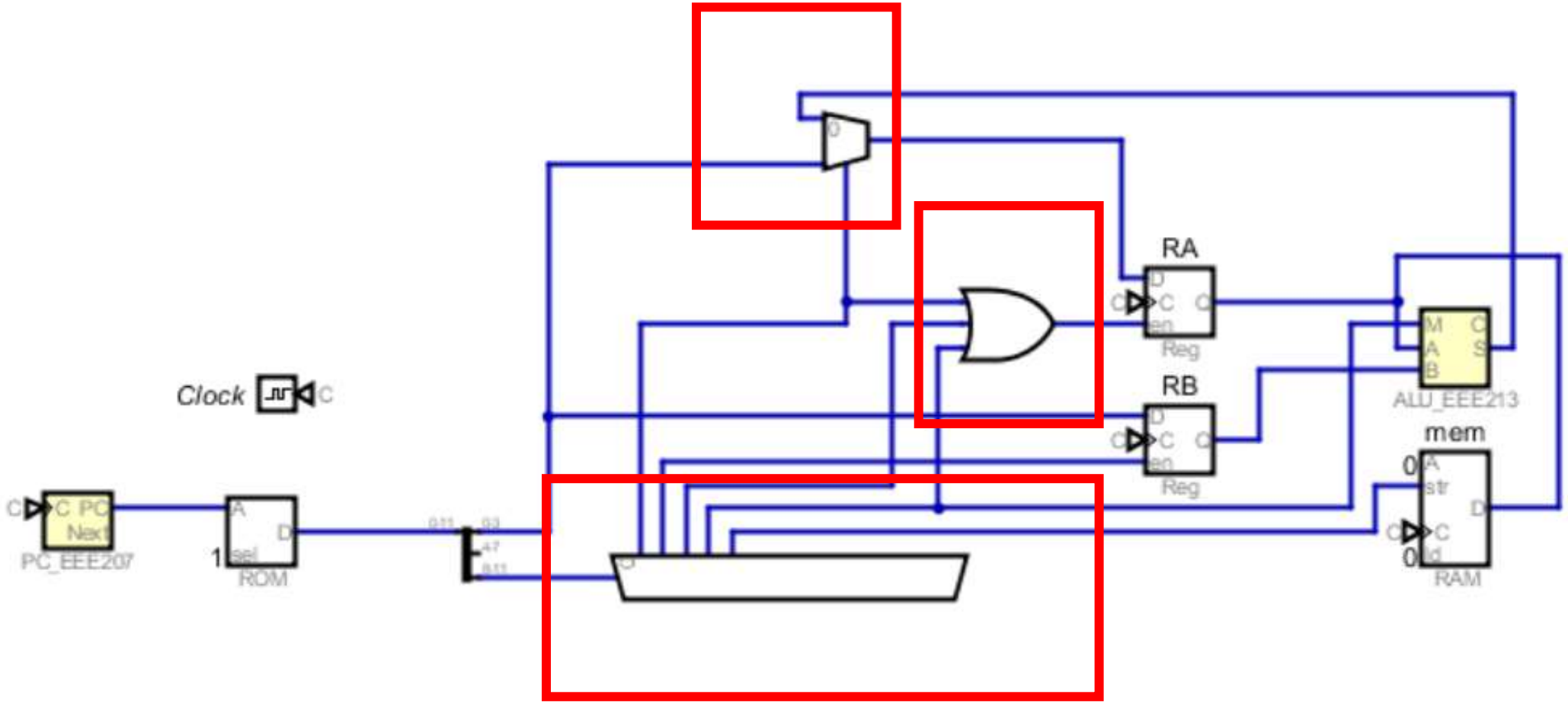


Figure 9-1. Typical application block diagram

Decoder Applications: MCU



BCD to 7 SEGMENT DECODER

BCD decoder are widely used to display binary number in decimal format

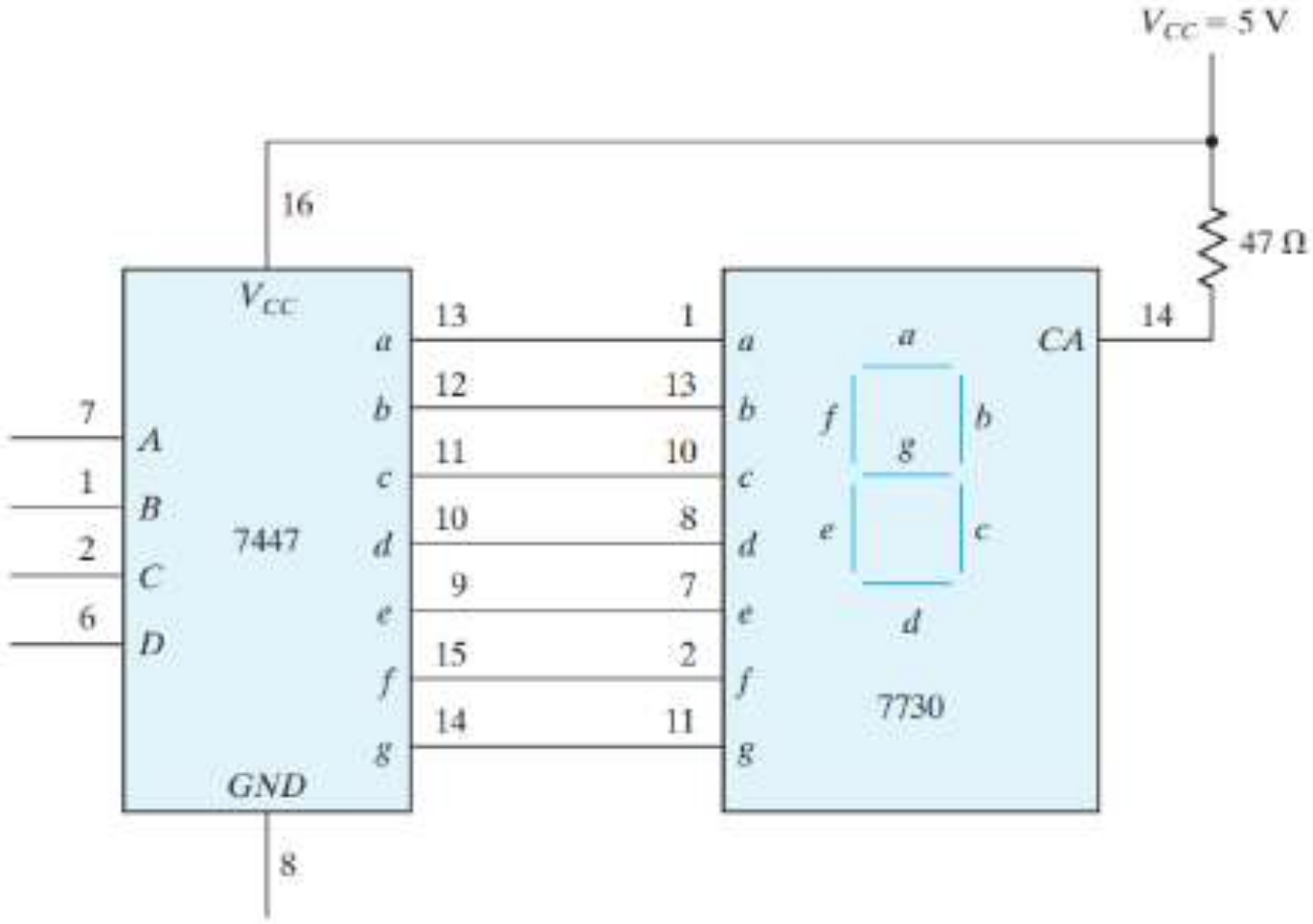


FIGURE 9.8
BCD-to-seven-segment decoder (7447) and seven-segment display (7730)

4.10 Encoders

- encoder is a digital circuit that performs the **inverse operation of a decoder**.
- encoder has 2^n (or fewer) input lines and n output lines.
- An example of an encoder is the octal-to-binary (8 to 3) encoder whose truth table is given in Table 4.7.
- It has eight inputs (one for each of the octal digits) and three outputs that generate the corresponding binary number.
- It is assumed that only one input has a value of 1 at any given time. **WHAT WILL HAPPEN IF NOT ?**

Table 4.7
Truth Table of an Octal-to-Binary Encoder

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

4.10 Encoders

- If two inputs are active simultaneously, the output produces an undefined combination.
- For example, if D3 and D4 are 1 simultaneously, the output of the encoder will be 111 because all three outputs are equal to 1. The output 111 does not represent either binary 3 or binary 4.
- **To resolve this ambiguity, encoder circuits must establish an input priority to ensure that only one input is encoded**

Table 4.7
Truth Table of an Octal-to-Binary Encoder

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

4.10 Encoders

- The operation of the priority encoder is such that if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence
- The truth table of a four-input priority encoder is given in Table 4.8.
- In addition to the two outputs x and y, the circuit has a third output designated by V; this is a valid bit indicator that is set to 1 when one or more inputs are equal to 1. If all inputs are 0, **there is no valid input, and V is equal to 0**

Table 4.8
Truth Table of a Priority Encoder

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

$$x = D_2 + D_3$$
$$y = D_3 + D_1 D_2'$$
$$V = D_0 + D_1 + D_2 + D_3$$

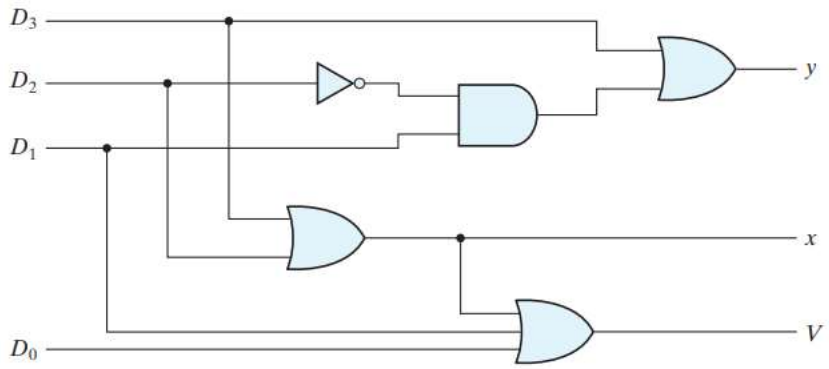


FIGURE 4.23
Four-input priority encoder

SN54147, SN54148, SN54LS147, SN54LS148
SN74147, SN74148 (TIM9907), SN74LS147, SN74LS148
10-LINE TO 4-LINE AND 8-LINE TO 3-LINE PRIORITY ENCODERS

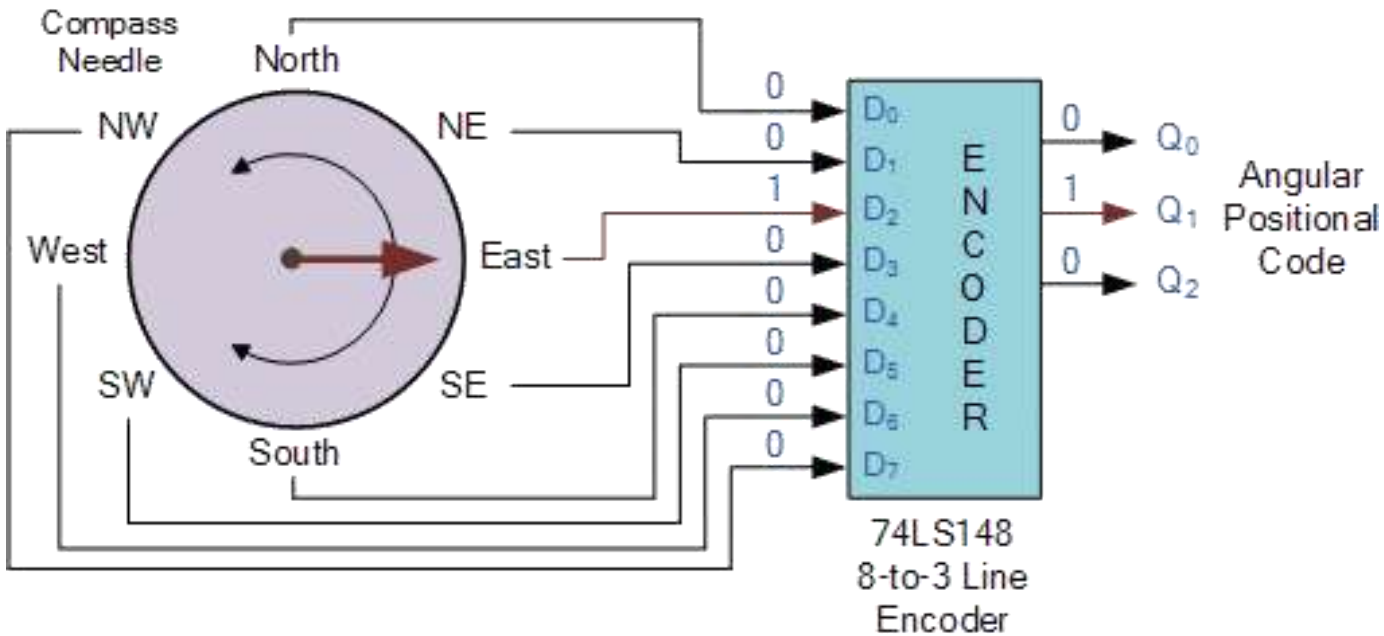
'147, 'LS147

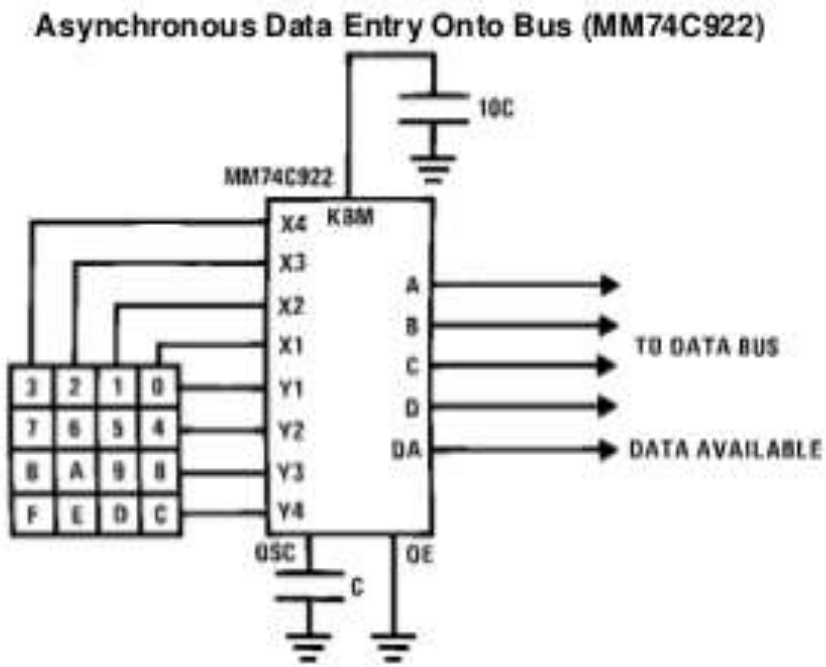
- Encode 10-Line Decimal to 4-Line BCD
- Applications Include:
  - Keyboard Encoding
  - Range Selection

'148, 'LS148

- Encode 8 Data Lines to 3-Line Binary (Octal)
- Applications Include:
  - n-Bit Encoding
  - Code Converters and Generators

decoders are used to reduce number of inputs for microcontroller

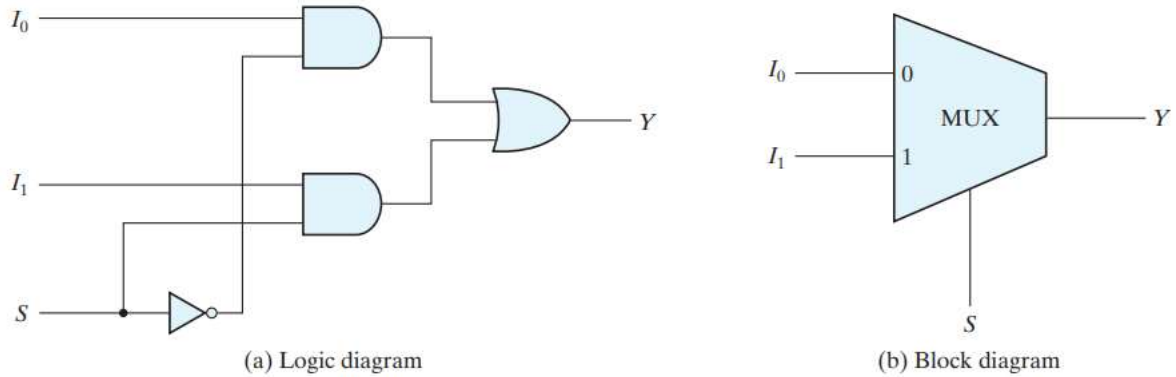




4.11 Multiplexer

- A multiplexer is a combinational circuit that **selects binary information from one of many input lines and directs it to a single output line**.
- The selection of a particular input line is **controlled by a set of selection lines**.
- Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.

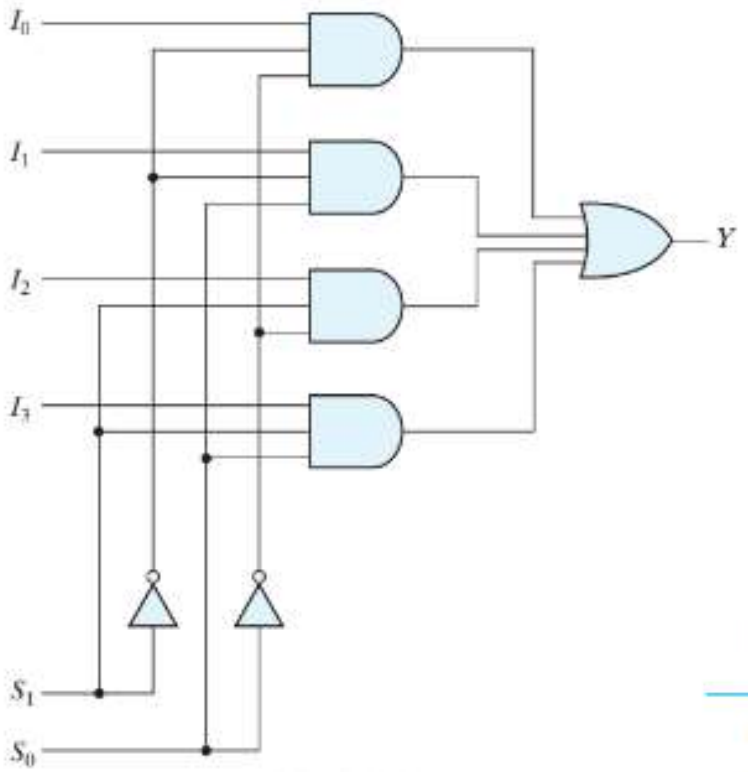
- A two-to-one-line multiplexer connects one of two 1-bit sources to a common destination, as shown in Fig. 4.24.
- The circuit has two data input lines, one output line, and one selection line S.
- When S = 0, the upper AND gate is enabled and I0 has a path to the output.
- When S = 1, the lower AND gate is enabled and I1 has a path to the output.
- The multiplexer acts like an electronic switch that selects one of two sources.



(a) Logic diagram (b) Block diagram

FIGURE 4.24
Two-to-one-line multiplexer

4.11 Multiplexer



(a) Logic diagram

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

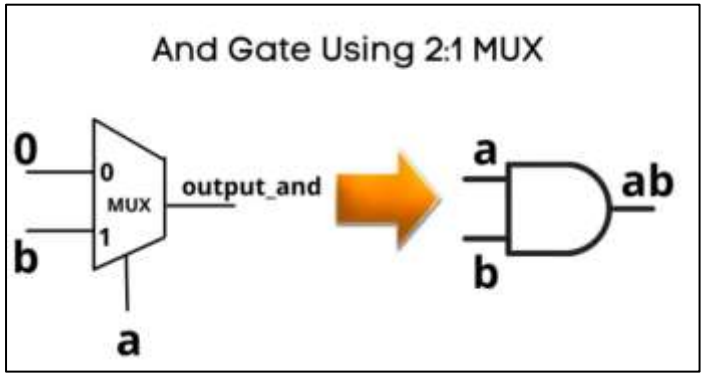
(b) Function table

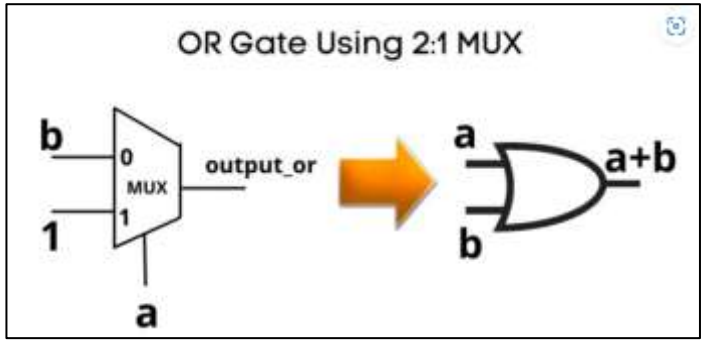
FIGURE 4.25
Four-to-one-line multiplexer

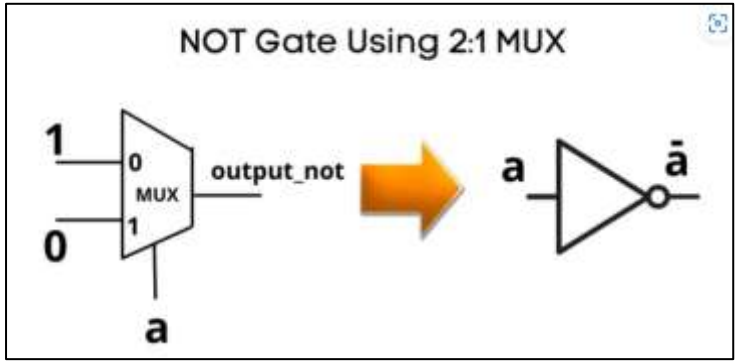
- A four-to-one-line multiplexer is shown in Fig. 4.25.
- Each of the four inputs, I0 through I3, is applied to one input of an AND gate.
- Selection lines S1 and S0 are decoded to select a particular AND gate.
- The outputs of the AND gates are applied to a single OR gate that provides the one-line output

MUX

- using MUX we can create AND OR NOT
- MEANS we can create any combinatorial circuit using only MUX
- remember that in FPGAs





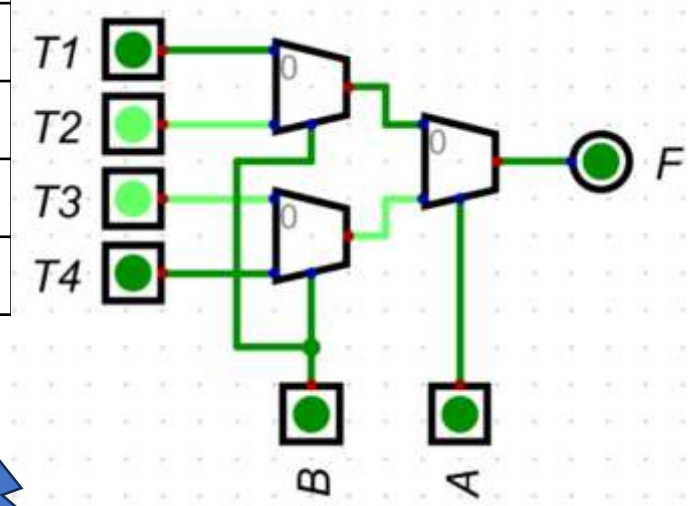


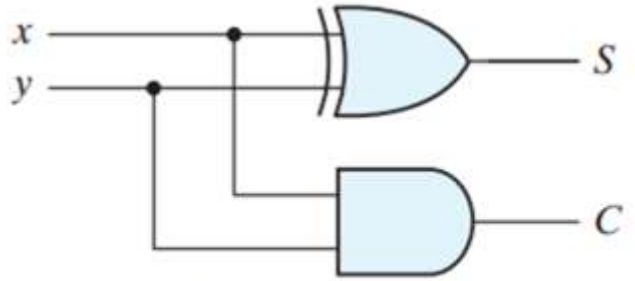
Implementation of Logic Functions using MUX LUT, Lookup Table, FPGA

- MUX architecture is fixed according to number of input bits
- input is coming from the truth table

F=A'B + B'A

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





(b) $S = x \oplus y$
$C = xy$

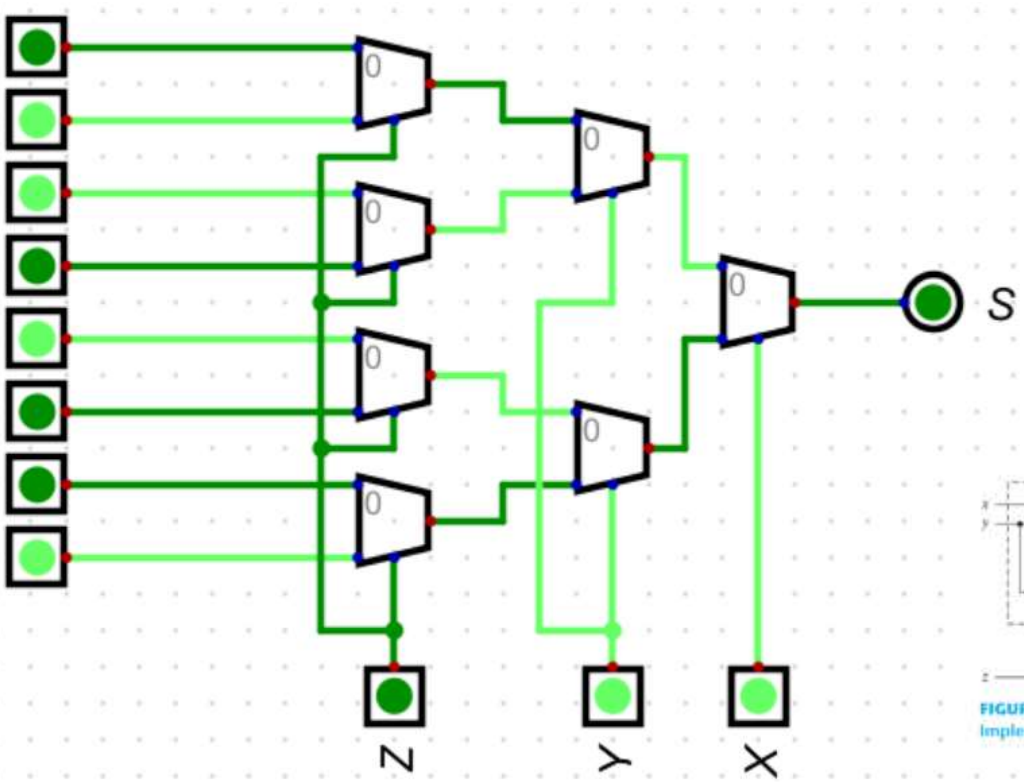
Implementation of Logic Functions using MUX LUT, Lookup Table, FPGA

- MUX architecture is fixed according to number of input bits
- input is coming from the truth table

Table 4.4
Full Adder

x	y	z	C	S	
0	0	0	0	0	T1
0	0	1	0	1	T2
0	1	0	0	1	T3
0	1	1	1	0	T4
1	0	0	0	1	T5
1	0	1	1	0	T6
1	1	0	1	0	T7
1	1	1	1	1	T8

$$S = x'y'z + x'yz' + xy'z' + xyz$$
$$C = xy + xz + yz$$



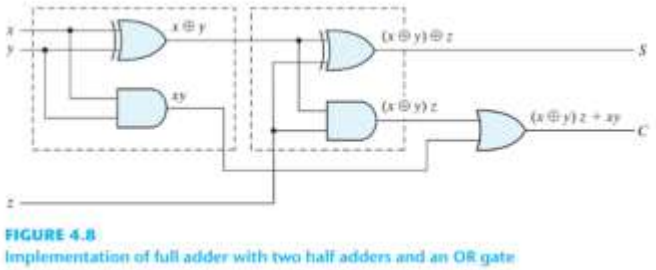


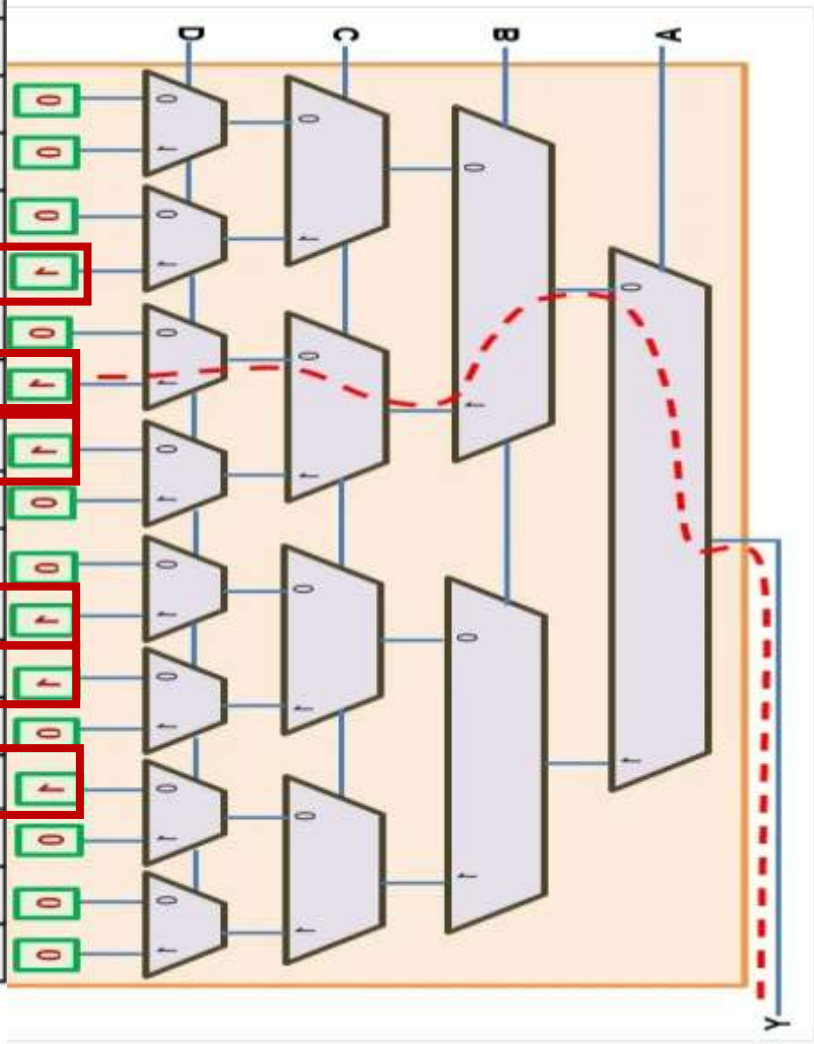
FIGURE 4.8
Implementation of full adder with two half adders and an OR gate

Implementation of Logic Functions using LUT

- Suppose we want to realize a Boolean Function of four input variables A, B, C and D using a 4-input LUT
- While realizing this function using an FPGA, A, B, C, and D will be the inputs to LUT.
- Next, the values of the output variable for each of their combination (available in the last column of the truth table) will be stored in the Flash RAM
- if ABCD = 0101, then the output of the LUT, Y, will take the value of 1 as the content of the sixth memory cell makes its way to the output pin (as shown by the red discontinuous line in Figure 3).

Truth Table

Inputs				Output
A	B	C	D	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



LUT in FPGA

- there are thousands of LUT in FPGA
- Tang Nano 9K has 8640 LUT4

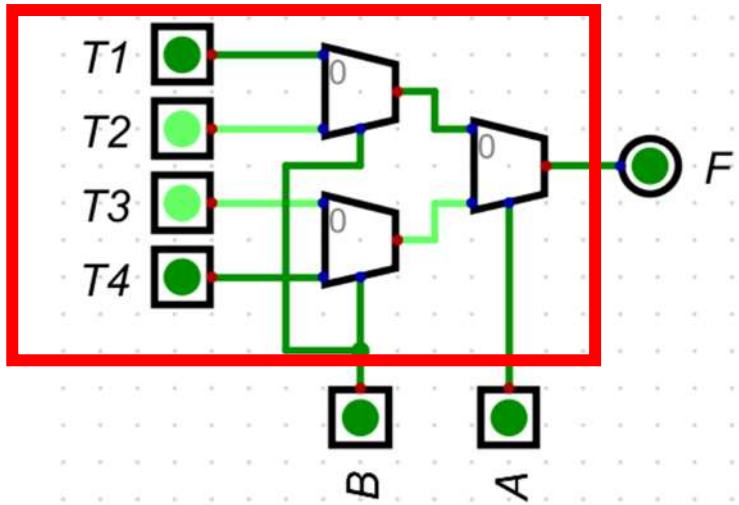
- LUT5-LUT8 can be constructed from LUT4

- example LUT2, and LUT3

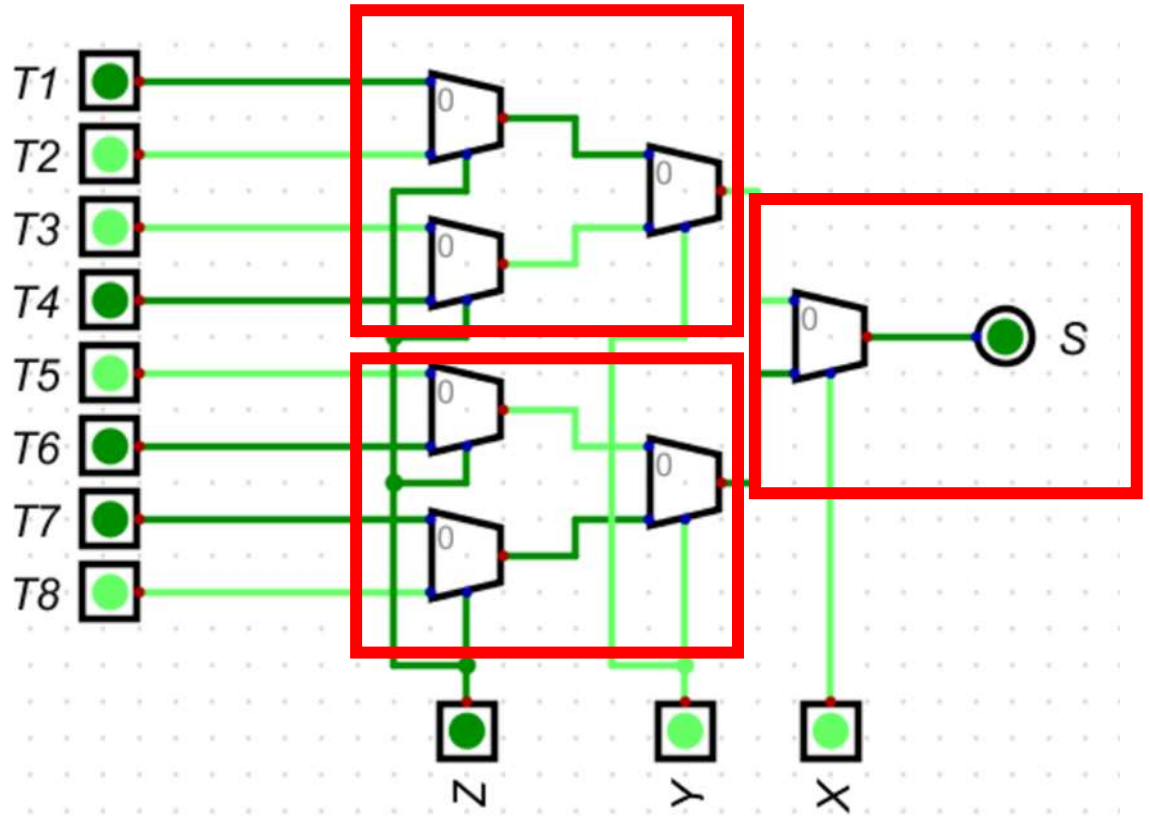
- Truth Table is calculated inside FPGA, A,B is input to LUT

LUT4	4-input Look-up Tables
LUT5	5-input Look-up Tables
LUT6	6-input Look-up Tables
LUT7	7-input Look-up Tables
LUT8	8-input Look-up Tables

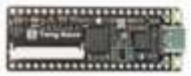
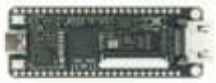
LUT2



LUT3

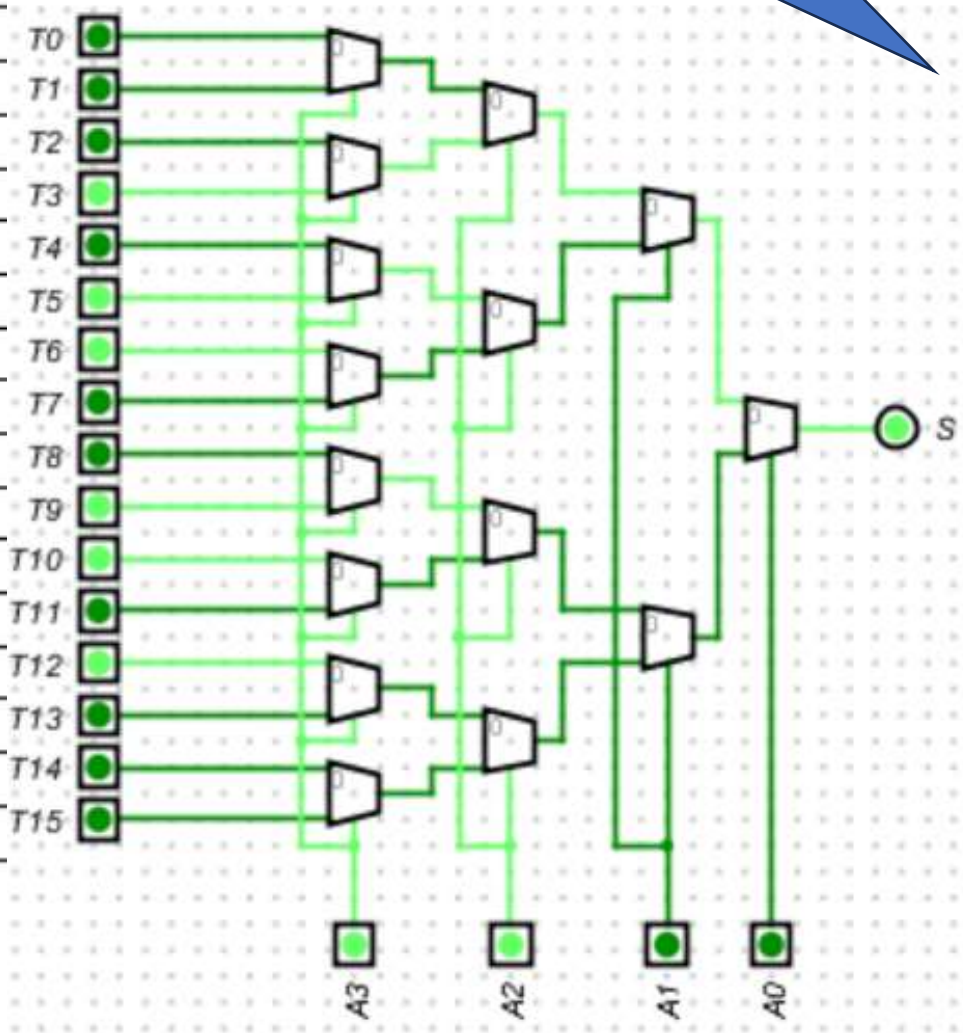


LUT4
Truth Table

Model	Tang Nano 1K	Tang Nano 4K	Tang Nano 9K
Appearance			
Logic Units (LUT4)	1152	4608	8640

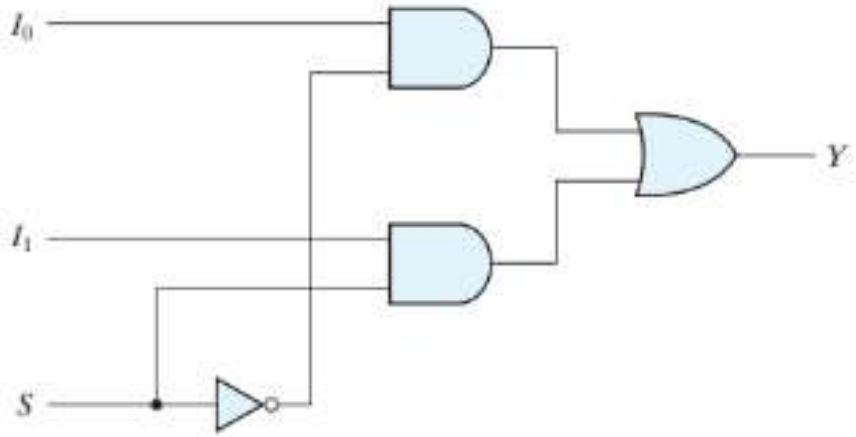
Truth Table

Inputs				Output
A	B	C	D	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0



4.11 Multiplexer

- A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line.
- The selection of a particular input line is controlled by a set of selection lines.
- Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.

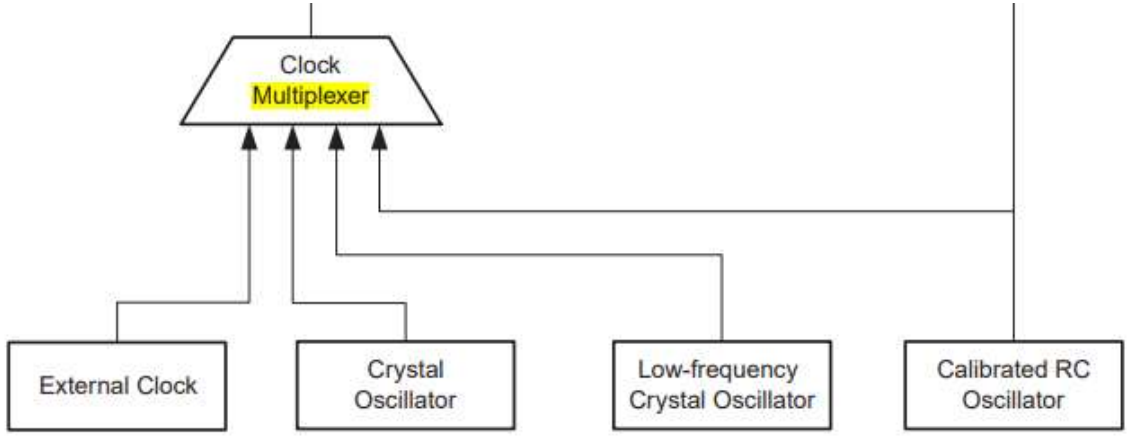


(a) Logic diagram

FIGURE 4.24
Two-to-one-line multiplexer

4.11 Multiplexer

- A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line.
- The selection of a particular input line is controlled by a set of selection lines.
- Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.



10.3 Clock Sources

The device has the following clock source options, selectable by Flash Fuse bits as shown below. The clock from the selected source is input to the AVR clock generator, and routed to the appropriate modules.

Table 10-1. Device Clocking Options Select(1)

Device Clocking Option	CKSEL3:0
Low Power Crystal Oscillator	1111 - 1000
Full Swing Crystal Oscillator	0111 - 0110
Low Frequency Crystal Oscillator	0101 - 0100
Internal 128kHz RC Oscillator	0011
Calibrated Internal RC Oscillator	0010
External Clock	0000
Reserved	0001

Note: 1. For all fuses "1" means unprogrammed while "0" means programmed.

Applications: Multiplexer

9.1 Application Information

The SN74HCS251 is an 8-to-1 data selector/multiplexer. This application shows an example of using the device with all required connections.

9.2 Typical Application

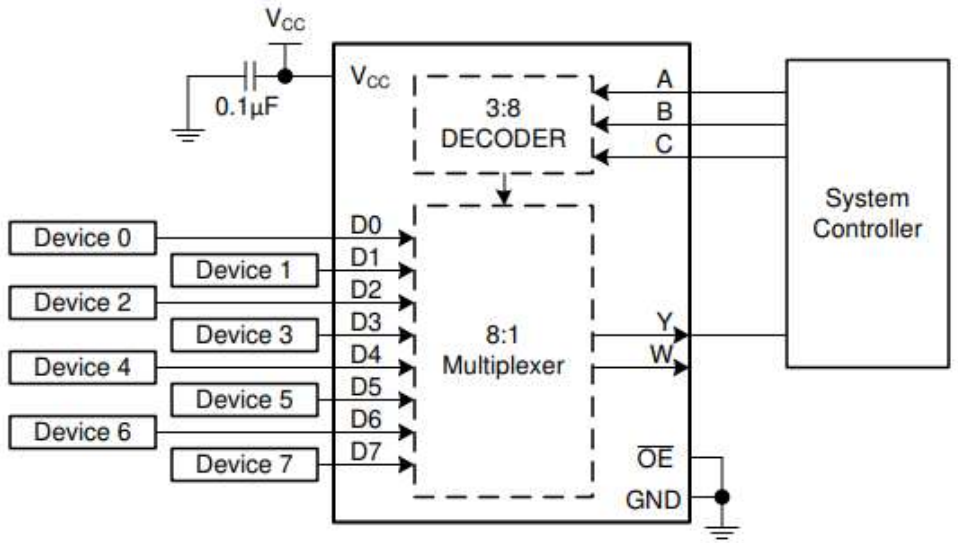
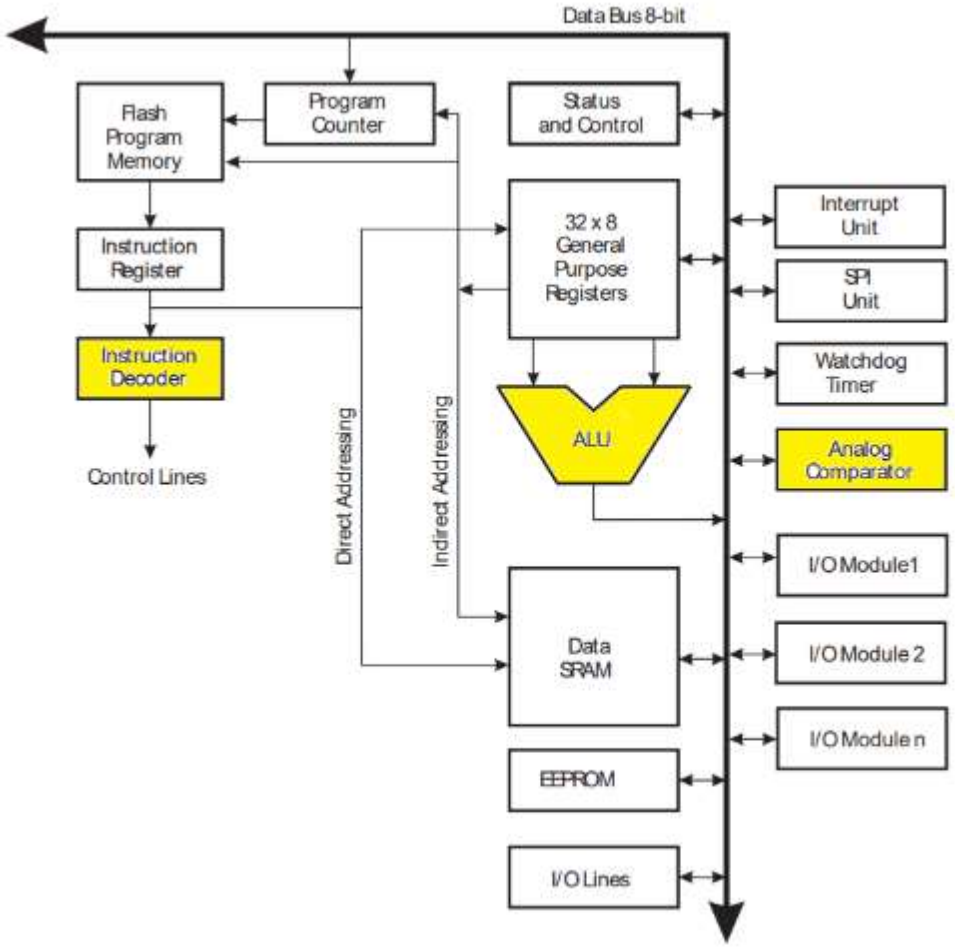


Figure 9-1. Typical application block diagram

ALU+COMPARATOR+DECODER/DEMUX



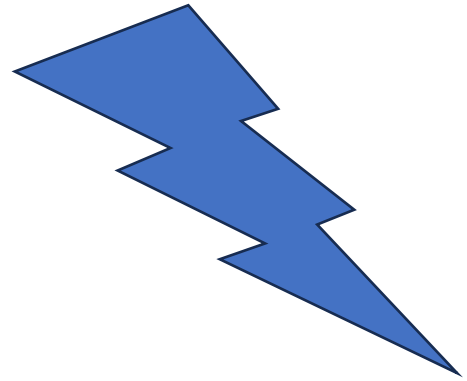
- decoder inside MCU takes the instruction bits, decode it and route to control lines

MICROCHIP

ATmega640

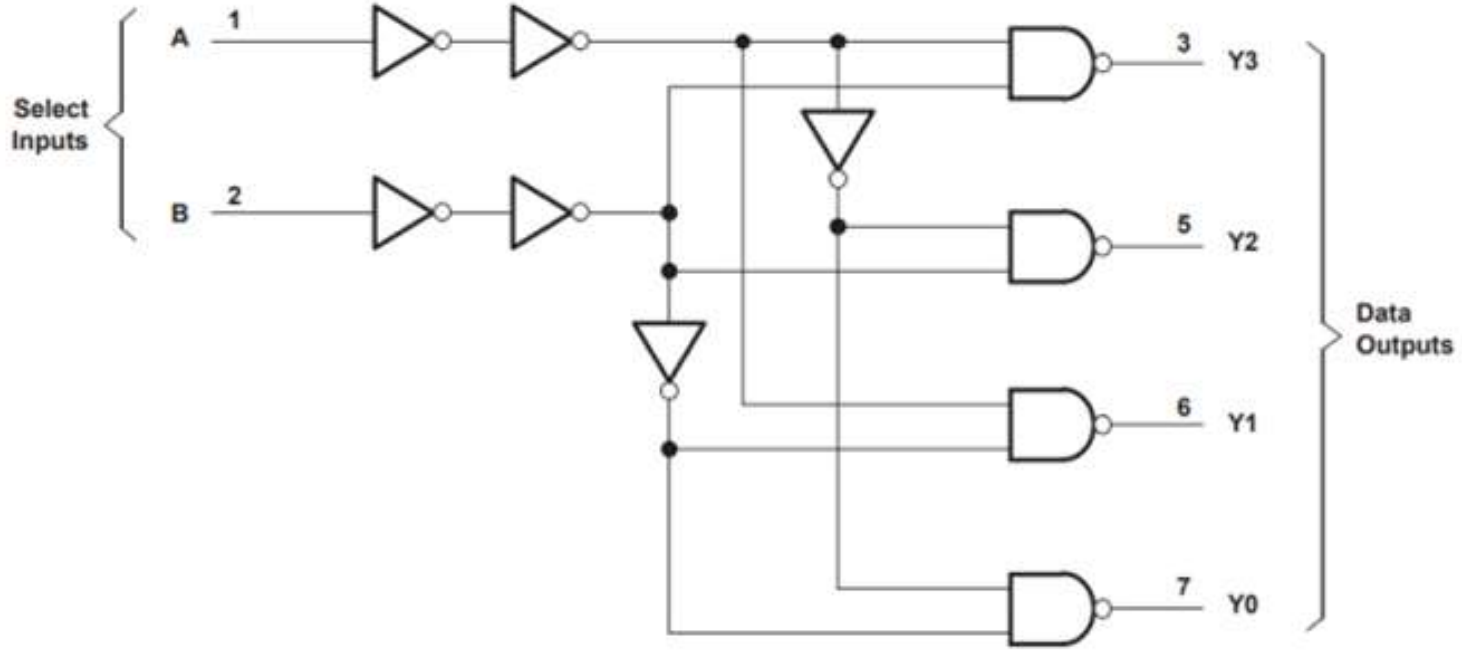
8-bit Microcontroller w

Exercises: Design Guide



1. **Description of function**
2. **Create using Truth Table**
3. **Write logical expression using Truth Table**
4. **Implement logical expression using AND, OR, NOT gates**
5. **Minimize number of gates using Karnaugh map**

Exercises#1



3. (10 points) What is the output Y3 Y2 Y1 Y0 of given logic circuit for inputs A=1, B=0?

Y3 Y2 Y1 Y0=_____

Exercises#2

2. (10 points) Fill truth table of **odd parity** function which adds extra bit to XYZ binary message to make the **number of 1's odd**?

X	Y	Z	F
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

4. (15 points) Write the logical expression for the given truth table of F?

F=_____

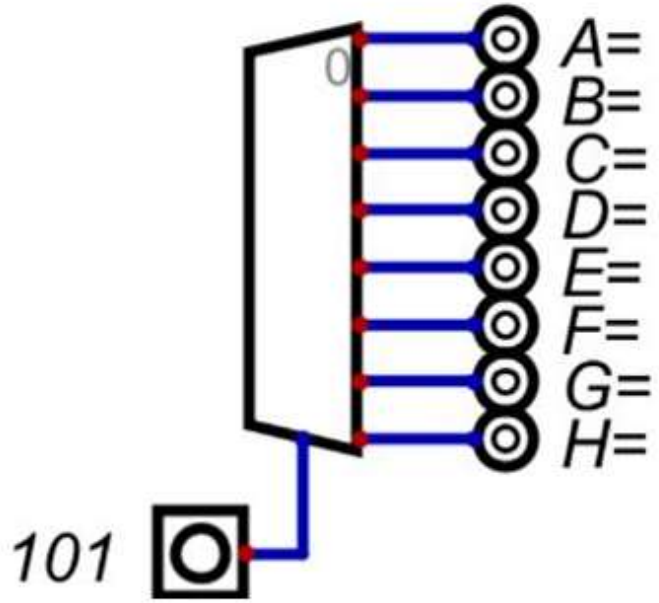
X	Y	Z	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

6. (10 points)
Fill the Karnaugh map for the truth table in question 4.
Write the simplified logical expression:

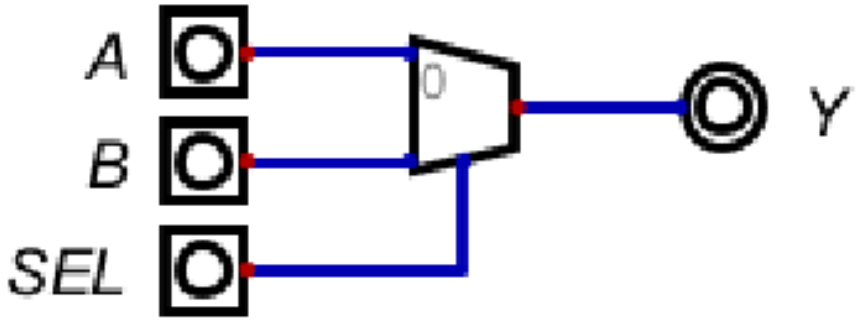
F=

	Y'		Y	
X'				
X				
	Z'	Z		Z'

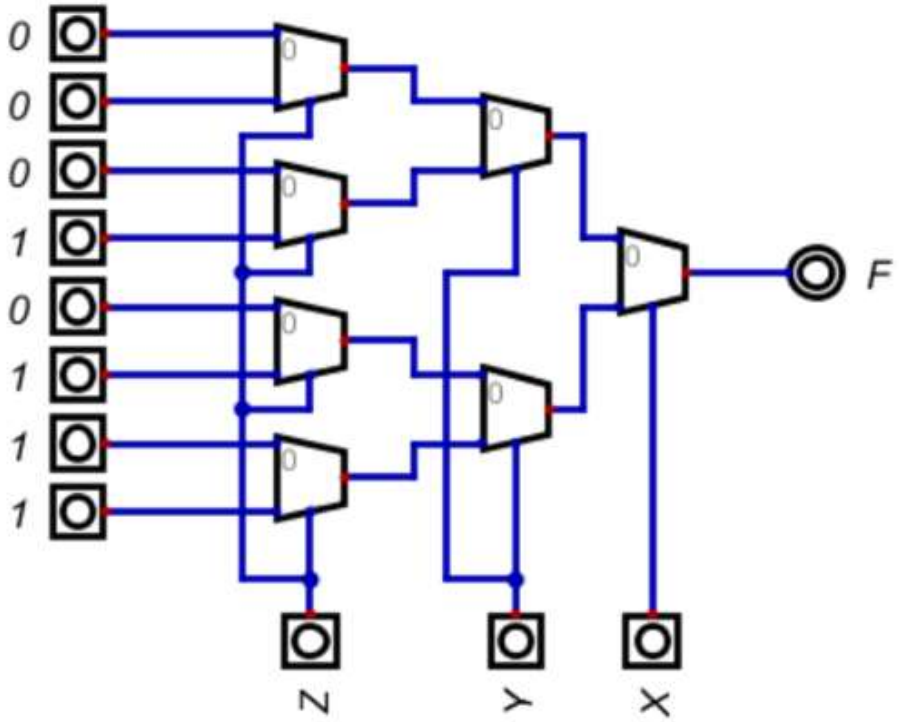
Exercises#3



1. (5 points) 3-bit decoder has 101 input.
Fill the outputs (1 or 0)?



Exercises#4



8. (10 points) ZYX=110, Draw the path from F to input and find the value of F=_____

Exercises#5

9. (10 points) Convert decimal 10 and 3 into 4bit binary. Add these two binary numbers:

Carry					
Augend=10					
Addend=3					
Sum					

5. (10 points) Find 2's complement of decimal number 12

Dec=12				
1's comp				
add 1				1
2's comp=12				

Exercises#6

7. (15 points) Fill in the blanks:

_____, the "father of logic," established the basis for systematic deductive reasoning.

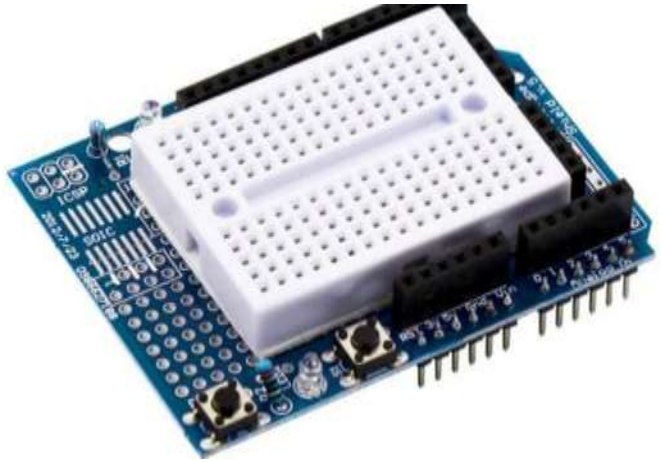
_____ transformed logic into a mathematical framework.

_____ was the pioneer who first applied Boolean algebra to electronics.

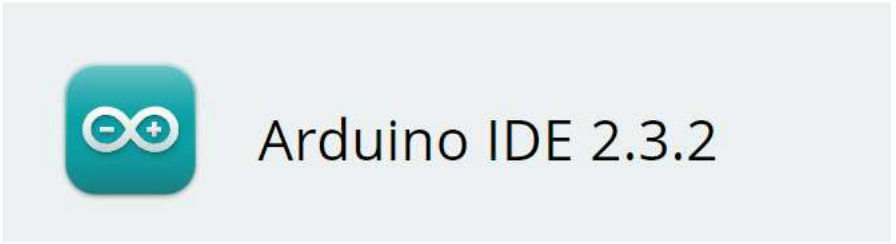
_____ is a fundamental component of a microcontroller responsible for performing arithmetic and logical operations.

Introduction to Arduino

[Arduino UNO R3 Klon USB Kablo Hediyeli - (USB Chip CH340) Satın Al | Robotistan]

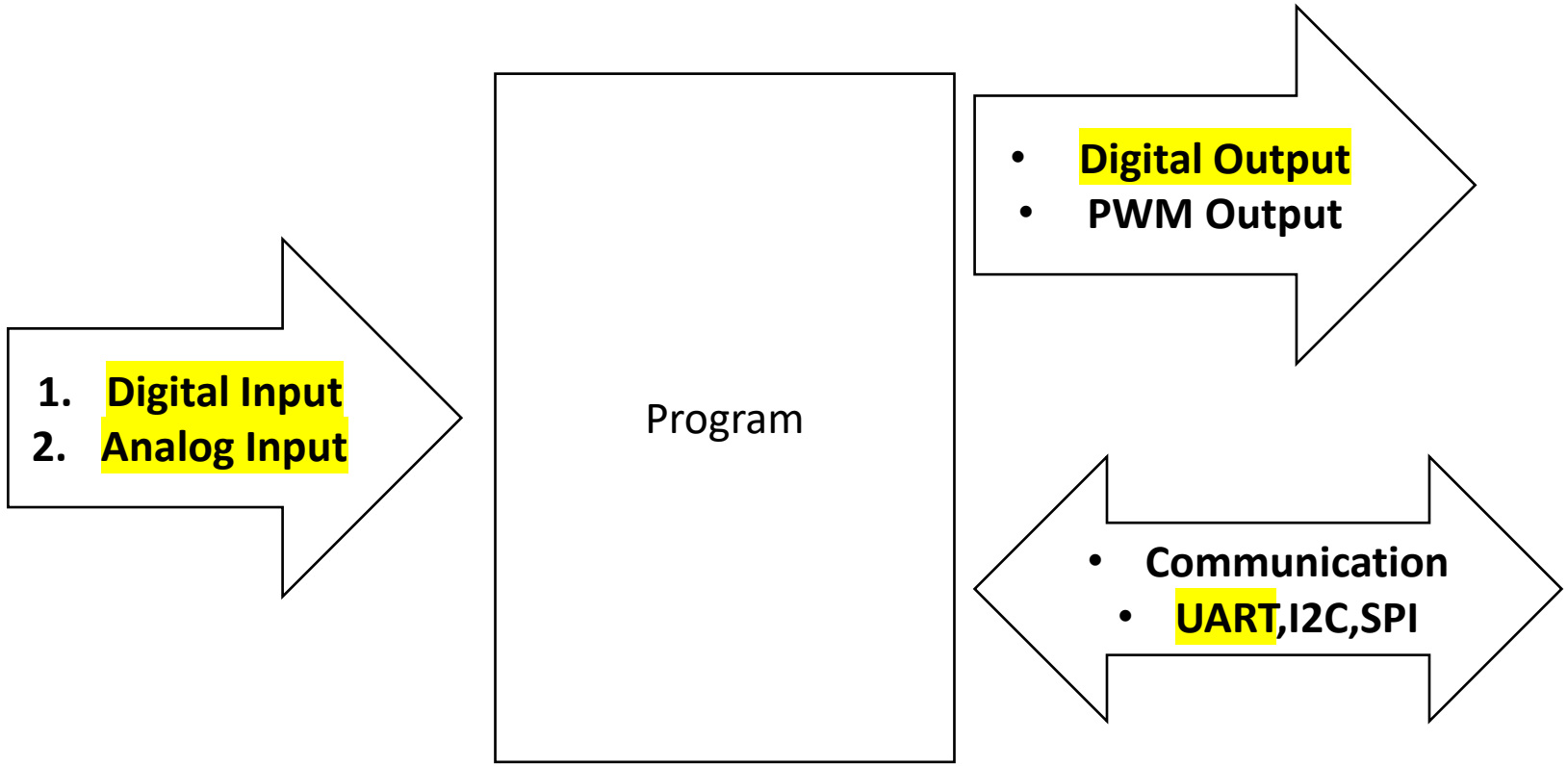


[Mini Breadboardlu Arduino UNO R3 Proto Shield Kiti Satın Al | Robotistan]

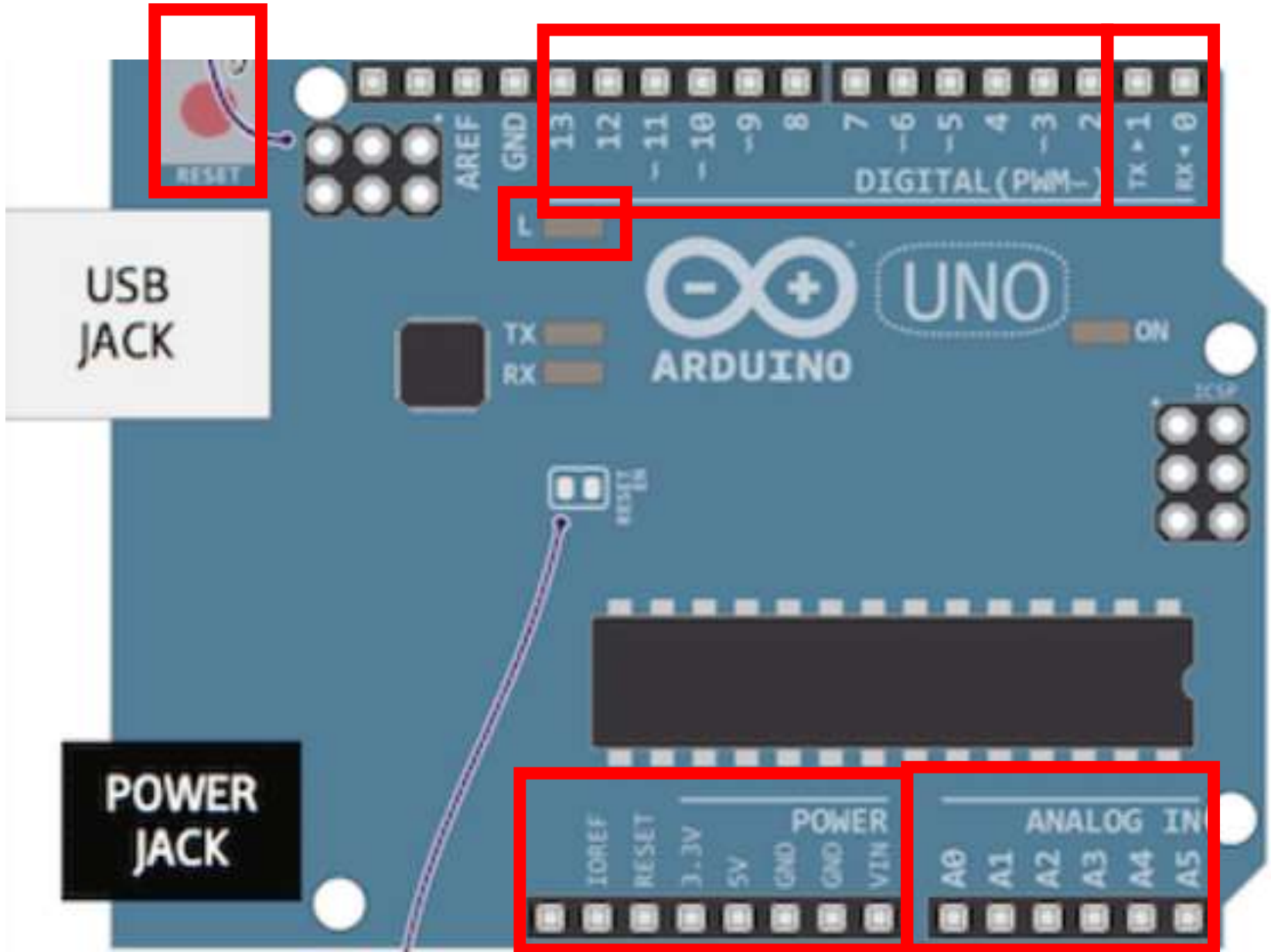


[Software | Arduino]

Introduction to Arduino: Input/Output



Introduction to Arduino for Control Systems



Introduction to Arduino: Coding

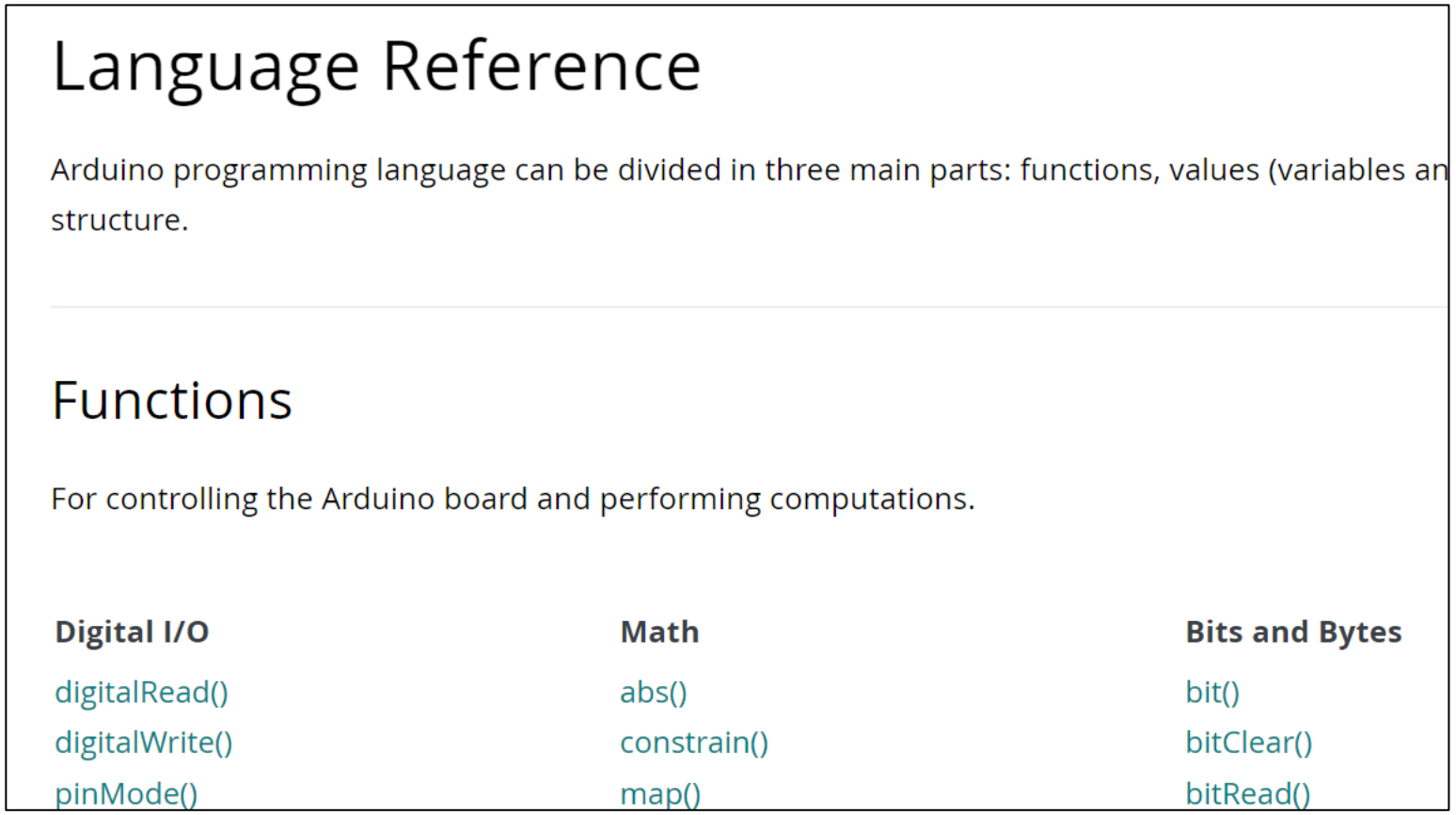
```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Introduction to Arduino: Digital Output

```
// Pin number for the built-in LED
int ledPin = 13;
void setup() {
  // Set the LED pin as an output
  pinMode(ledPin, OUTPUT);
}
void loop() {
  // Turn on the LED
  digitalWrite(ledPin, HIGH);
  delay(100);  // Wait for 1 second
  // Turn off the LED
  digitalWrite(ledPin, LOW);
  delay(100);  // Wait for 1 second
}
```

Introduction to Arduino: Reference

[Arduino Reference - Arduino Reference]



Language Reference

Arduino programming language can be divided in three main parts: functions, values (variables an
structure.

Functions

For controlling the Arduino board and performing computations.

Digital I/O	Math	Bits and Bytes
digitalRead()	abs()	bit()
digitalWrite()	constrain()	bitClear()
pinMode()	map()	bitRead()

Introduction to Arduino : Data Types

[Arduino Reference - Arduino Reference]

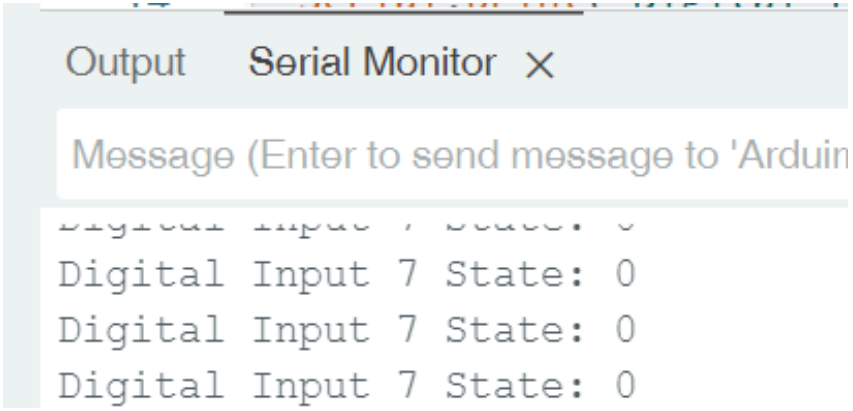
Data Type	Size (Bytes)	Range of Values
void	0	null
bool/boolean	1	True/False
char	1	-128 to +127
unsigned char	1	0 to 255
byte	1	0 to 255
int	2	-32,768 to 32,767
unsigned int	2	0 to 65,535
word	2	0 to 65,535
long	4	-2,147,483,648 to 2,147,483,647
unsigned long	4	0 to 4,294,967,295
float	4	-3.4028235E+38 to 3.4028235E+38
double	4	-3.4028235E+38 to 3.4028235E+38
string	-	character array

mbedgeek.blogspot.com

Introduction to Arduino: Digital Input

FLOATING, State is changing 1/0 if cable is not connected

Connect a jumper cable one end to 5V, and touch to the other end to pin7, check the state

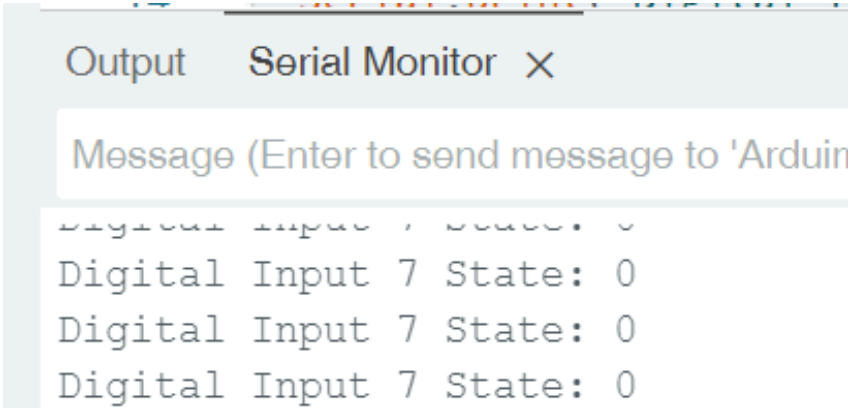



```
// Pin number for digital input
int inputPin = 7;
int inputState;
void setup() {
  // Set the input pin as an input
  pinMode(inputPin, INPUT);
  // Initialize serial communication
  Serial.begin(9600);
}
void loop() {
  // Read the state of the input pin
  inputState = digitalRead(inputPin);
  // Print the input state to the serial
monitor
  Serial.print("Digital Input 7 State: ");
  Serial.println(inputState);
  // Add a short delay to avoid rapid
serial prints
  delay(100);
}
```

Introduction to Arduino: Digital Input INPUT_PULLUP

Connect a jumper cable one end to GND, and connect to the other end to pin7, check the state

FLOATING, State is changing 1/0 if cable is not connected

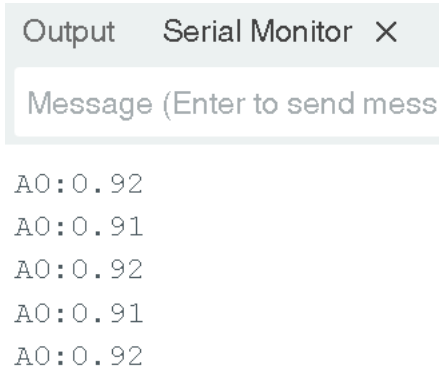



```
// Pin number for digital input
int inputPin = 7; int inputState;
void setup() {
  // Set the input pin as an input with the internal pull-up
resistor enabled
  pinMode(inputPin, INPUT_PULLUP);
  // Initialize serial communication
  Serial.begin(9600);
}
void loop() {
  // Read the state of the input pin
  inputState = digitalRead(inputPin);
  // Print the input state to the serial monitor
  Serial.print("Digital Input 7 State: ");
  Serial.println(inputState);
  // Add a short delay to avoid rapid serial prints
  delay(100);
}
```

Introduction to Arduino: Analog Input

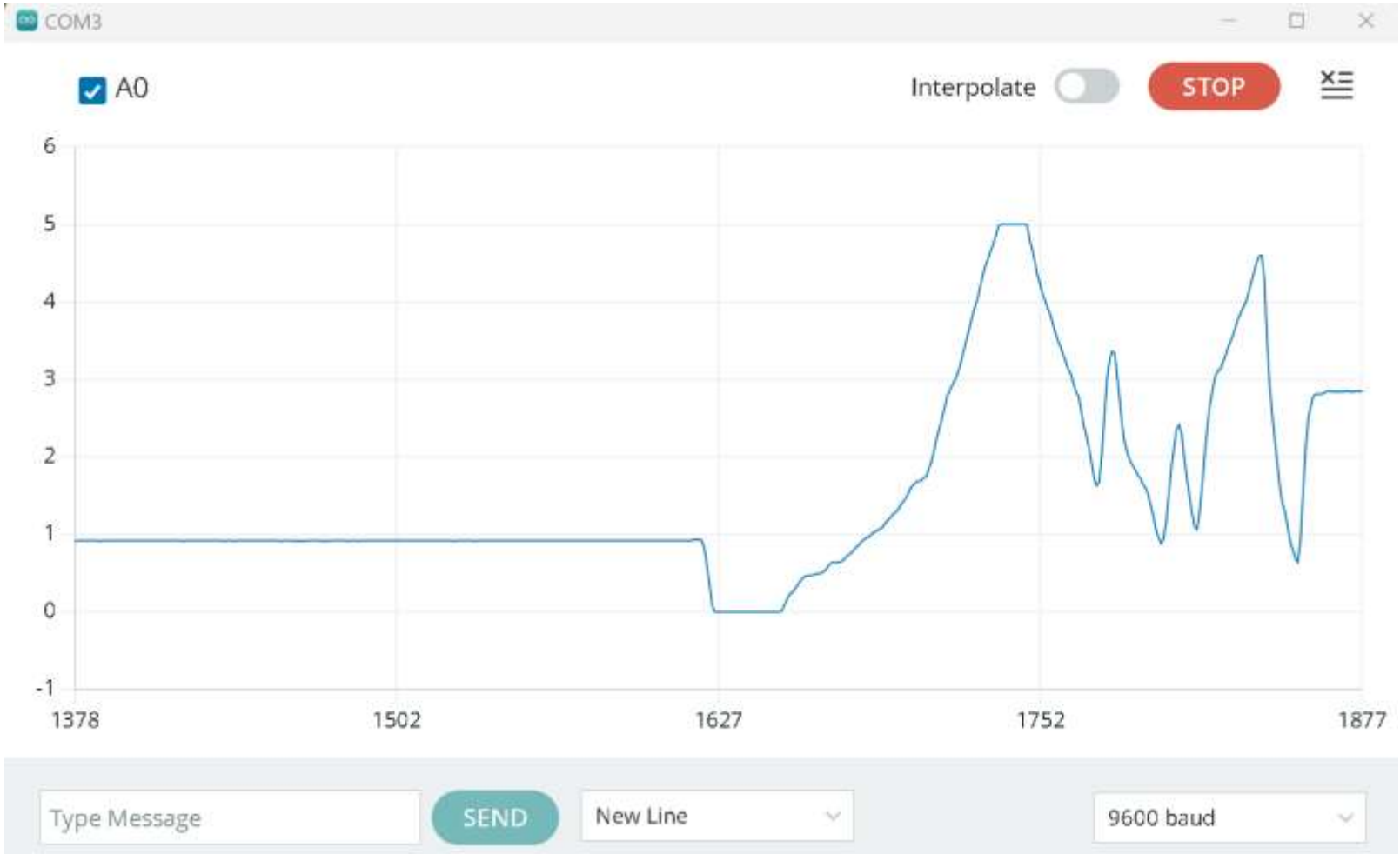
Put a POT on proto shield. connect midle pin to A0 and connect other two pins to5V, and GND. Play with the POT and observe reading. Use serial plot draw the voltage

```
// Pin number for analog input
int analogInputPin = A0;
int analogValue;
float voltage;
void setup() {
  Serial.begin(9600);
}
void loop() {
  analogValue = analogRead(analogInputPin);
  voltage = analogValue * (5.0 / 1023.0);
  Serial.print("A0:");
  Serial.println(voltage);
  delay(100);
}
```



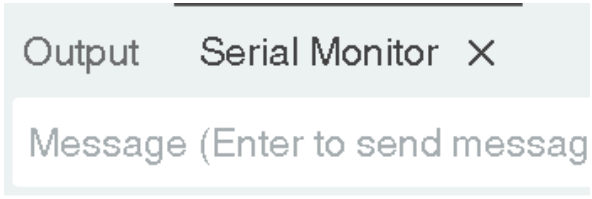
Introduction to Arduino: Serial Plotter

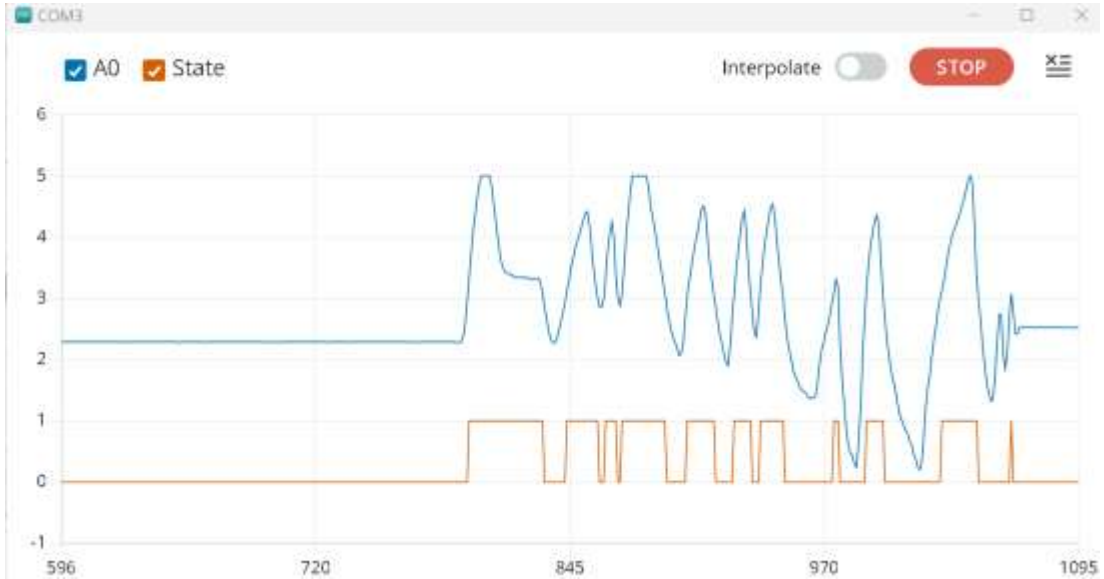
Put a POT on proto shield. connect midle pin to A0 and connect other two pins to5V, and GND. Play with the POT and observe reading. Use serial plot draw the voltage



Introduction to Arduino: Serial Plotter with Multi Variables

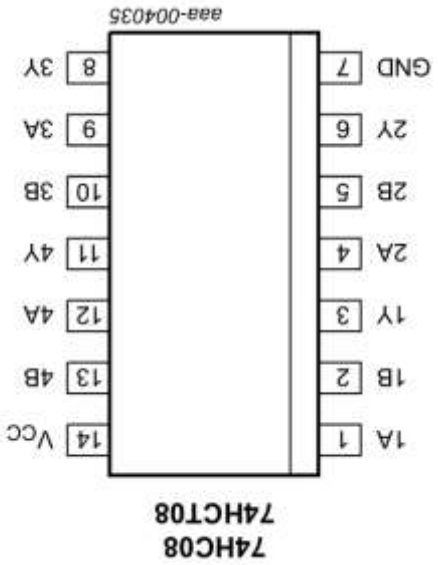
[Using the Serial Plotter Tool | Arduino Documentation]

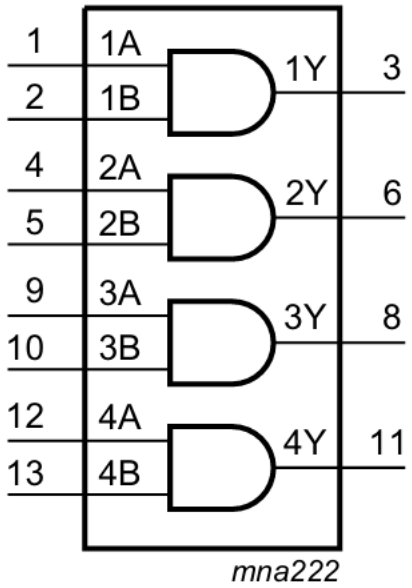





```ino
int analogInputPin = A0;
int state;
int analogValue;
float voltage;
void setup() {
  Serial.begin(9600);
}
void loop() {
  analogValue = analogRead(analogInputPin);
  voltage = analogValue * (5.0 / 1023.0);
  if (voltage > 3) {
    state = 1;
  } else {
    state = 0;
  }
  Serial.print("A0:");
  Serial.print(voltage);
  Serial.print(",State:");
  Serial.println(state);
  delay(100);
}
```

Introduction to Arduino: AND Gate Testing

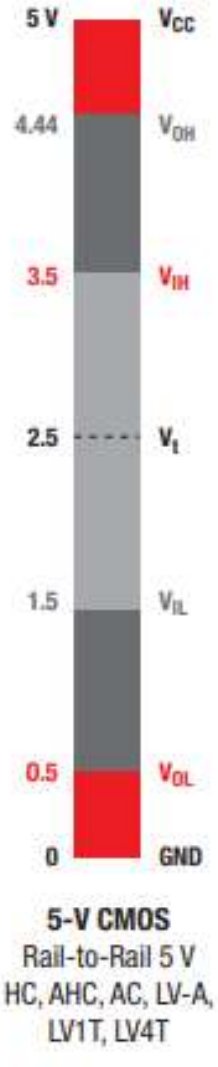


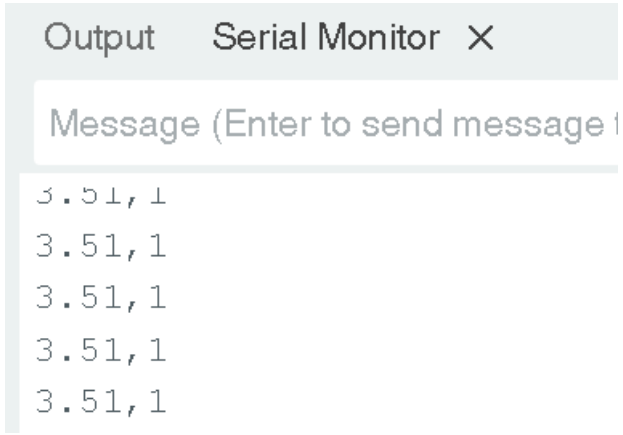


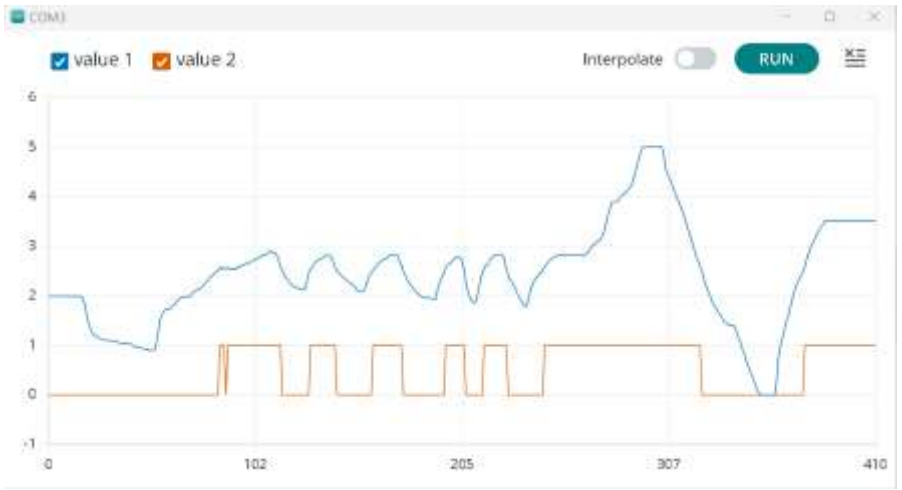

```
AND Gate Testing
A: 0 B: 0 | Y: 0
A: 0 B: 1 | Y: 0
A: 1 B: 0 | Y: 0
A: 1 B: 1 | Y: 1
```

```ino
int inputA = 5;
int inputB = 6;
int outputY = 7;
int stateY;
void setup() {
  pinMode(inputA, OUTPUT);
  pinMode(inputB, OUTPUT);
  pinMode(outputY, INPUT);
  Serial.begin(9600);
}
void loop() {
  Serial.println("AND Gate Testing");
  digitalWrite(inputA, LOW);
  digitalWrite(inputB, LOW);
  stateY = digitalRead(outputY);
  Serial.print("A: 0 B: 0 | Y: ");
  Serial.println(stateY == HIGH ? 1 : 0);
  delay(500);
  digitalWrite(inputA, LOW);
  digitalWrite(inputB, HIGH);
  stateY = digitalRead(outputY);
  Serial.print("A: 0 B: 1 | Y: ");
  Serial.println(stateY == HIGH ? 1 : 0);
  delay(500);
```

Introduction to Arduino: AND Gate Input Voltage








```ino
int inputA = 5;
int inputB = 6;
int outputY = 7;
int stateY;
int analogInputPin = A0;
int analogValue;
float voltage;
void setup() {
  pinMode(inputA, INPUT);
  pinMode(inputB, OUTPUT);
  pinMode(outputY, INPUT);
  Serial.begin(9600);
  digitalWrite(inputB, HIGH);
}
void loop() {
  analogValue = analogRead(analogInputPin);
  voltage = analogValue * (5.0 / 1023.0);
  stateY = digitalRead(outputY);
  Serial.print(voltage);
  Serial.print(",");
  Serial.println(stateY == HIGH ? 1 : 0);
  delay(100);
}
```

5.2. Sequential Circuits

- Combinational circuits are ONE WAY, no loop
- It consists of a combinational circuit to which **memory elements** are connected to form a feedback path
- The **memory elements** are devices capable of **storing binary information**
- The block diagram demonstrates that the outputs in a sequential circuit are a function not only of the inputs but also of the present state of the storage elements.
- The next state of the storage elements is also a function of external inputs and the present state.
- Thus, a sequential circuit is specified by a time sequence of **inputs, outputs, and internal states**.
- In contrast, the outputs of combinational logic **depend on only the present values of the inputs**

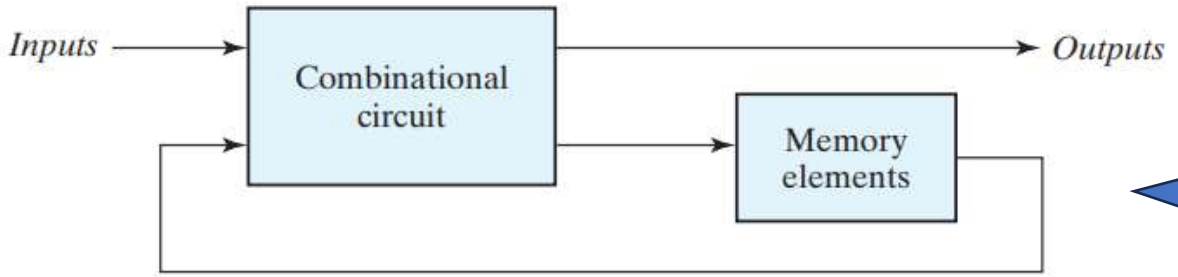
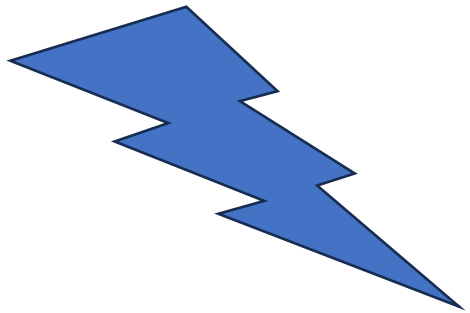
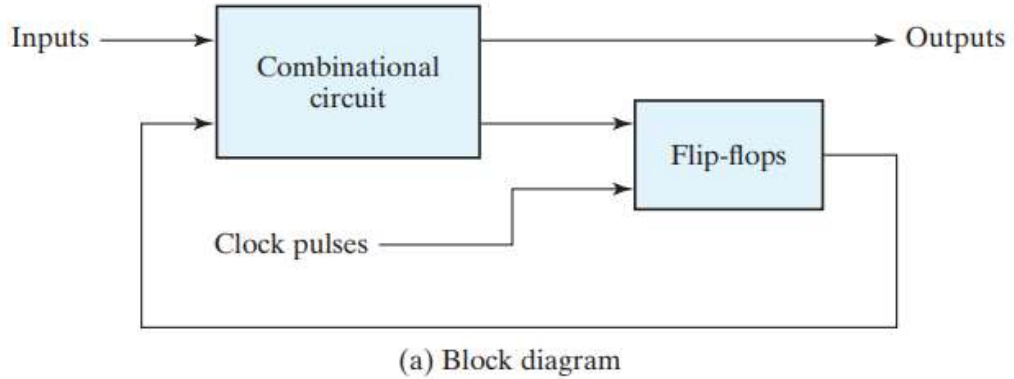


FIGURE 5.1
Block diagram of sequential circuit

5.2. Sequential Circuits

- There are two main types of sequential circuits, and their classification is a function of the timing of their signals
- A **synchronous sequential** circuit employs signals that affect the storage elements at only discrete instants of time. Synchronization is achieved by a timing device called a clock generator, which provides a clock signal having the form of a periodic sequence of clock pulses. The clock signal is commonly denoted by the identifiers clock and clk
- The behavior of an **asynchronous sequential circuit** depends upon the input signals at any instant of time and the order in which the inputs change.





(a) Block diagram

(b) Timing diagram of clock pulses

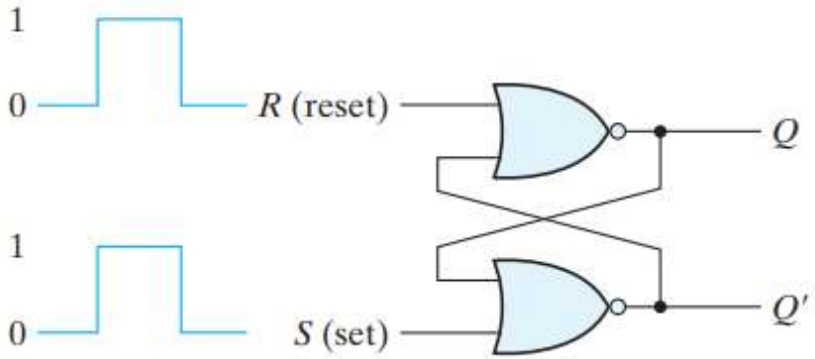
FIGURE 5.2
Synchronous clocked sequential circuit

5.3. Storage Elements: Latches

- A storage element in a digital circuit can **maintain a binary state indefinitely** (as long as power is delivered to the circuit), until directed by an input signal to switch states.
- Storage elements that operate with **signal levels (rather than signal transitions) are referred to as latches**; those **controlled by a clock transition are flip-flops**.
- **Latches are said to be level-sensitive** devices; **flip-flops are edge-sensitive** devices.
- The two types of storage elements are related because **latches are the basic circuits** from which **all flip-flops are constructed.**

SR Latch

- When output Q = 1 and Q’ = 0, the latch is said to be in the **set state**. S=1, R=0
- When Q = 0 and Q’ = 1, it is in the **reset state. S=0, R=1**
- However, when both inputs are equal to 1 at the same time, a condition in which both outputs are equal to 0 (rather than be mutually complementary) occurs.
- If both inputs are then switched to 0 **simultaneously**, the device will enter an unpredictable or undefined state or a metastable state.
- Consequently, in practical applications, **setting both inputs to 1 is forbidden**



(a) Logic diagram

S	R	Q	Q'	
1	0	1	0	
0	0	1	0	(after $S = 1, R = 0$)
0	1	0	1	
0	0	0	1	(after $S = 0, R = 1$)
1	1	0	0	(forbidden)

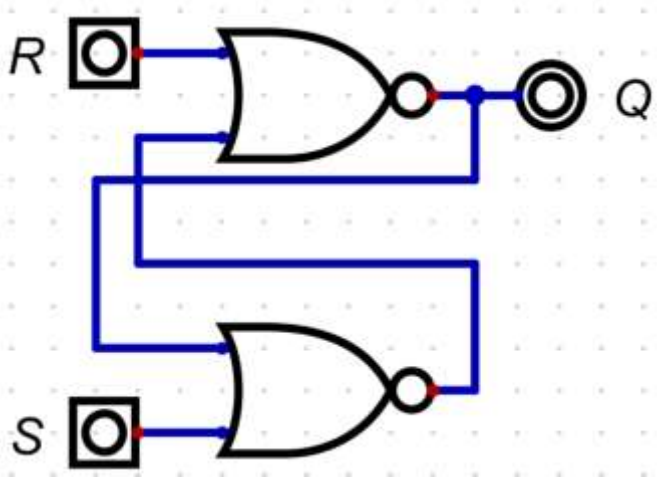
(b) Function table

FIGURE 5.3
SR latch with NOR gates

- **three state of output**
- **S=1 R=0: 1 SET**
- **S=0 R=1: 0 RESET**
- **S=0 R=0: KEEP (memory)**
- **S’=0, R’=0 (S=1, R=1) is not allowed, non stable**

SR Latch

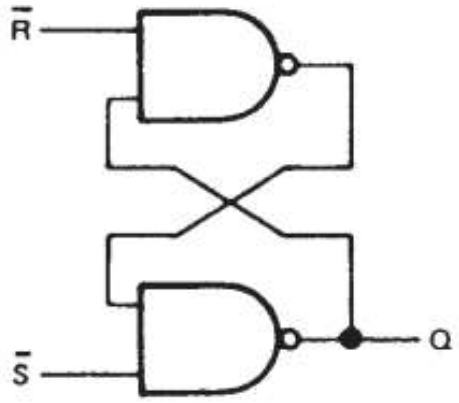
- When output Q = 1 and Q’ = 0, the latch is said to be in the **set state**. S=1, R=0
- When Q = 0 and Q’ = 1, it is in the **reset state. S=0, R=1**
- However, when both inputs are equal to 1 at the same time, a condition in which both outputs are equal to 0 (rather than be mutually complementary) occurs.
- If both inputs are then switched to 0 **simultaneously**, the device will enter an unpredictable or undefined state or a metastable state.
- Consequently, in practical applications, **setting both inputs to 1 is forbidden**



- **three state of output**
- **S=1 R=0: 1 SET**
- **S=0 R=1: 0 RESET**
- **S=0 R=0: KEEP (memory)**
- **S’=0, R’=0 (S=1, R=1) is not allowed, non stable**

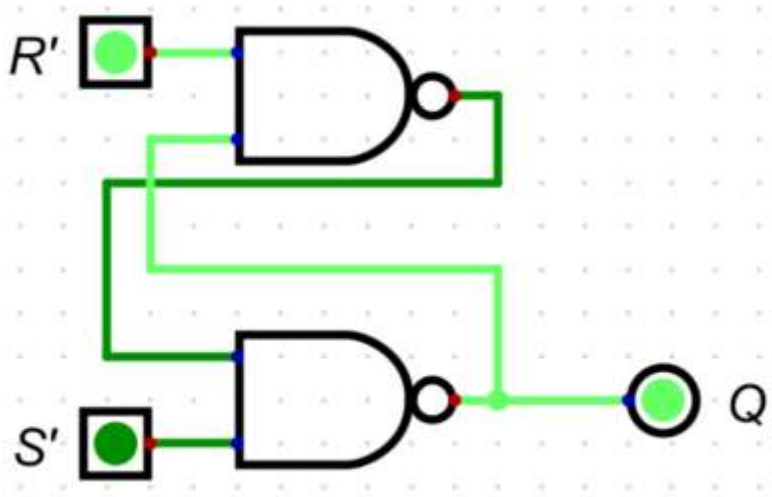
SR Latch (SN74279)

- **three state of output**
- **S=1 R=0: 1 SET**
- **S=0 R=1: 0 RESET**
- **S=0 R=0: KEEP (memory)**
- **S'=0, R'=0 (S=1, R=1) is not allowed, non stable**



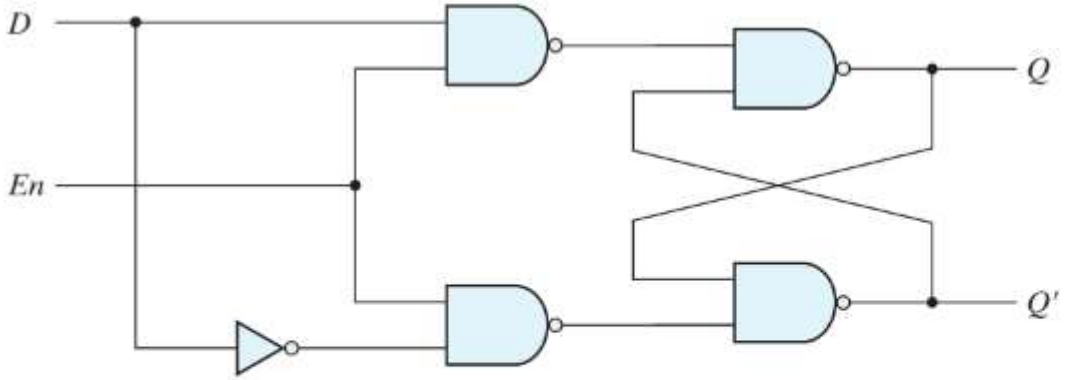
FUNCTION TABLE
(each latch)

INPUTS		OUTPUT
S̄†	R̄	Q
H	H	Q_0
L	H	H
H	L	L
L	L	H‡



D Latch (Transparent Latch)

- One way to eliminate the undesirable condition of the indeterminate state in the SR latch is to ensure that inputs **S and R are never equal to 1, S'R'=00 at the same time**. This is done in the D latch, shown in Fig. 5.6.
- The binary information present at the data input of the **D latch is transferred to the Q output when the enable input is asserted**.
- The output follows changes in the data input as long as the enable input is asserted. This situation provides a path from input D to the output, and for this reason, the circuit is often called a **transparent latch**



(a) Logic diagram

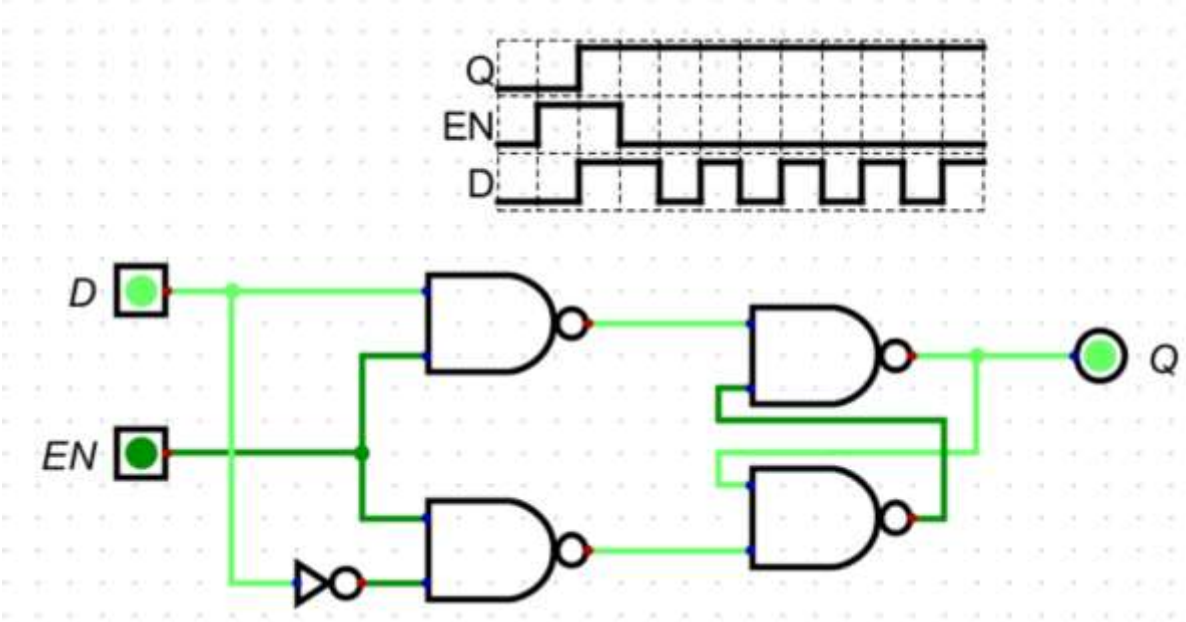
En	*D*	Next state of *Q*
0	X	No change
1	0	$Q = 0$; reset state
1	1	$Q = 1$; set state

(b) Function table

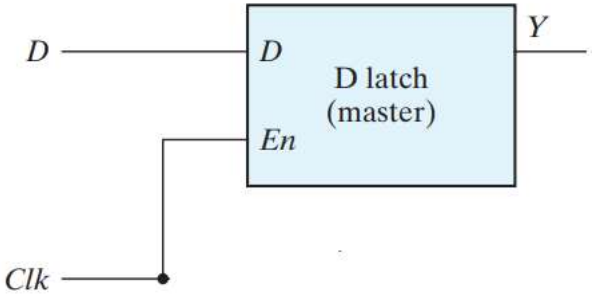
FIGURE 5.6
D latch

D Latch

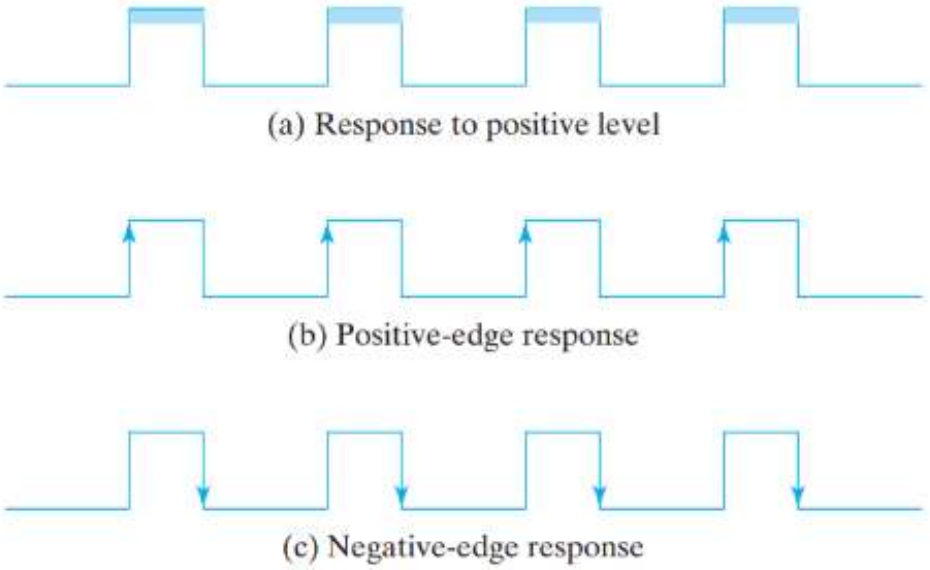
- D latch is a 1-bit register
- First EN=1, then write D-data to register, either 1 or 0
- then EN=0, keeps the value in Q like write protection



Problems of Latches



- Clock signal used for Enable input
- While **clock is ON** if **D changes** then **Y output will be changed** which is NOT something we want
- Because of this unreliable operation, the output of a latch cannot be applied directly or through combinational logic to the input of the same or another latch when **all the latches are triggered by a common clock source**.



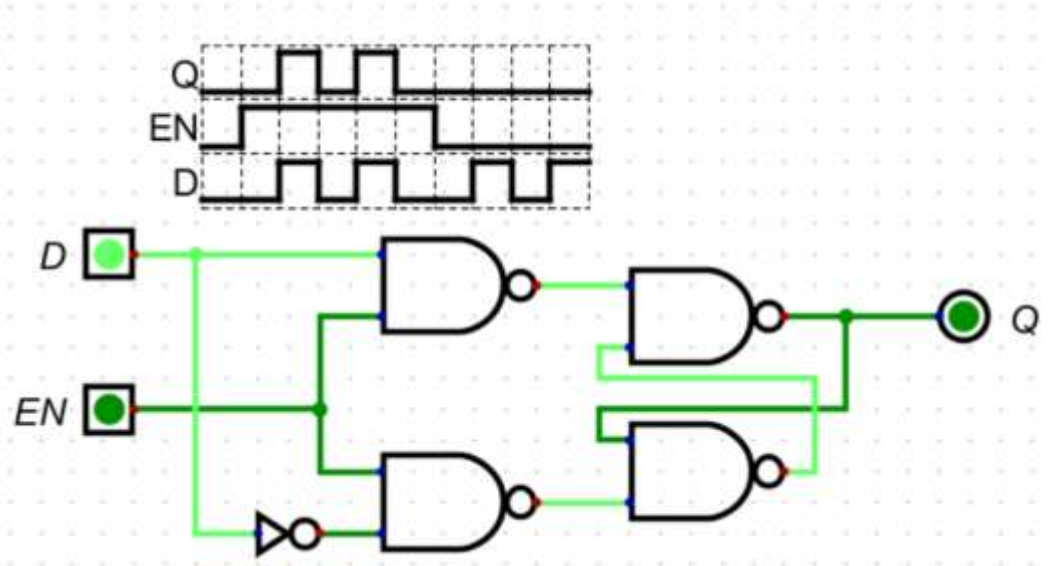


FIGURE 5.8
Clock response in latch and flip-flop

D Flip Flops (Falling Edge)

- Flip-flop circuits are constructed in such a way as to make them operate properly when they are part of a **sequential circuit that employs a common clock**
- The construction of a D flip-flop with **two D latches and an inverter** is shown in Fig. 5.9.
- a change in the output of the flip-flop can be triggered **only by and during the transition** of the clock from 1 to 0 (Falling Edge)
- The slave latch is enabled, and its output Q is equal to the master output Y. The master latch is disabled because Clk = 0.
- When the input (Clk) pulse changes to the logic-1 level, the data from the external D input are transferred to the master. The slave, however, is disabled
- **like two doors entrance**, only one person=D can pass, if you are inside the first door is closed
- There are Rising Edge D Flip-Flops, too.

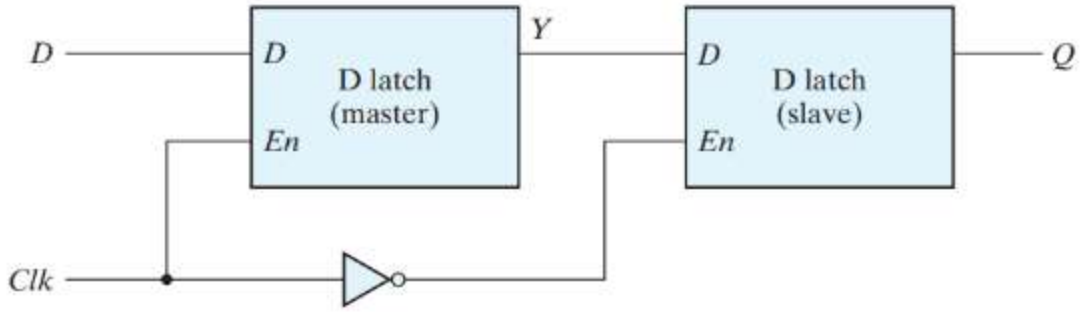


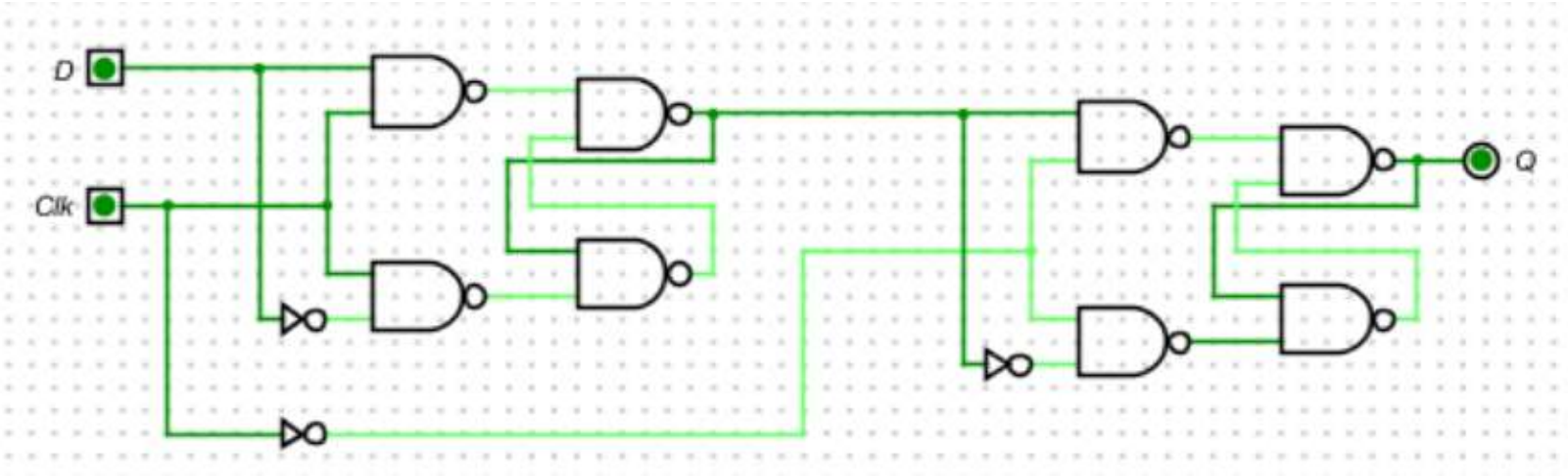
FIGURE 5.9
Master–slave *D* flip-flop

D Flip-Flop

D	Q(t + 1)	
0	0	Reset
1	1	Set

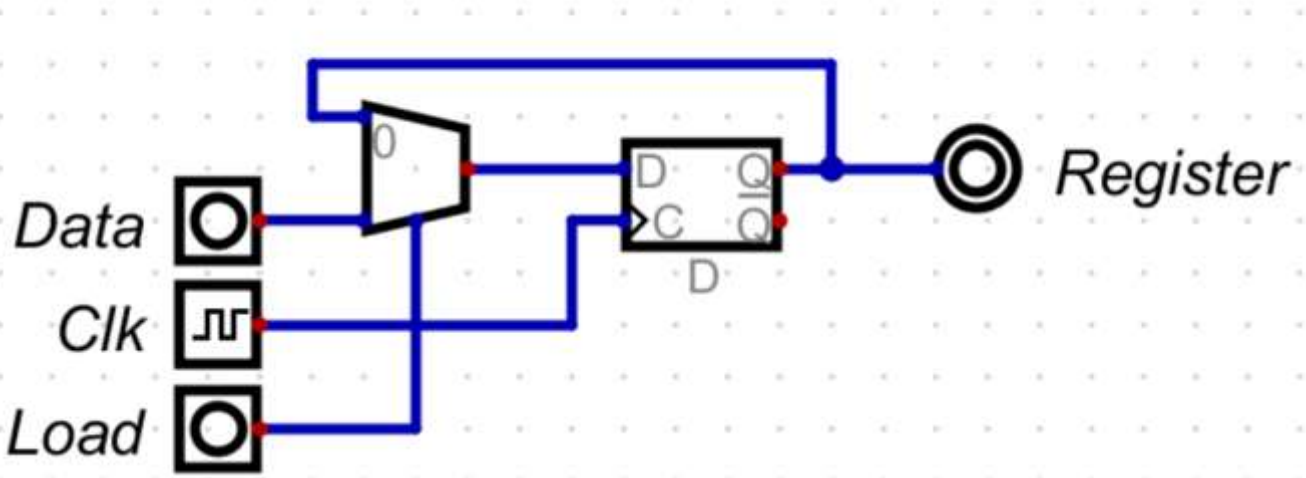
D Flip Flops (Falling Edge)

- D flipflop is constructed by two Dlatch, En signal became Clock
- Q takes the value of D at falling edge of Clk



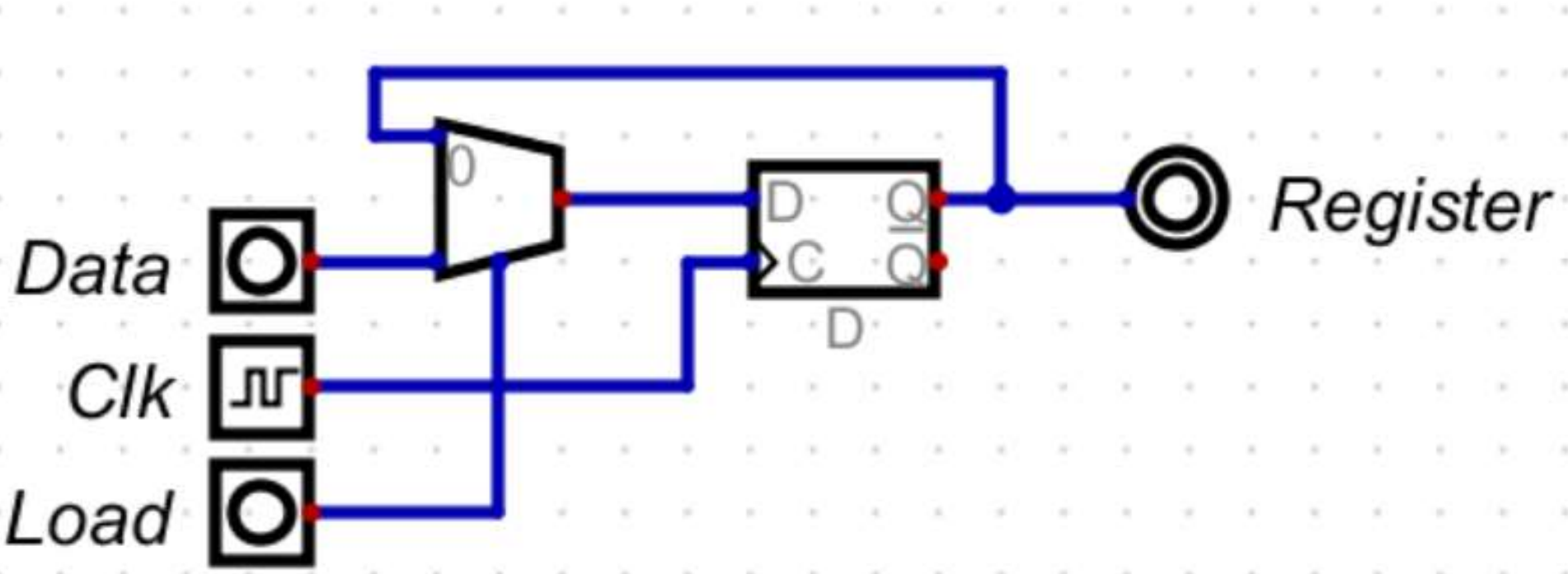
D Flip Flops (Rising Edge)

- En signal became Clock
- Q takes the value of D at rising edge of Clk



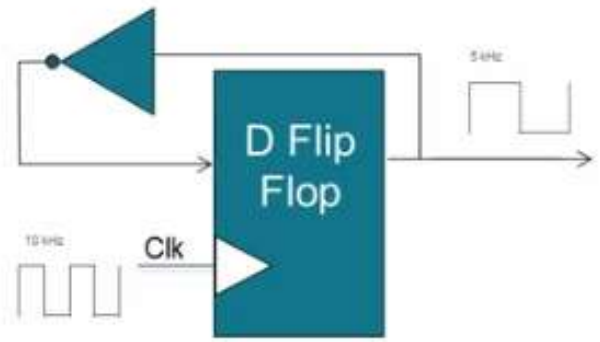
D Flip Flops (1 bit register)

- Load=0, Q keeps the previous value
- Load=1, Data forwarded to Q at clock rising edge



D Flip Flops: Applications

Clock Division using D-Flip Flop

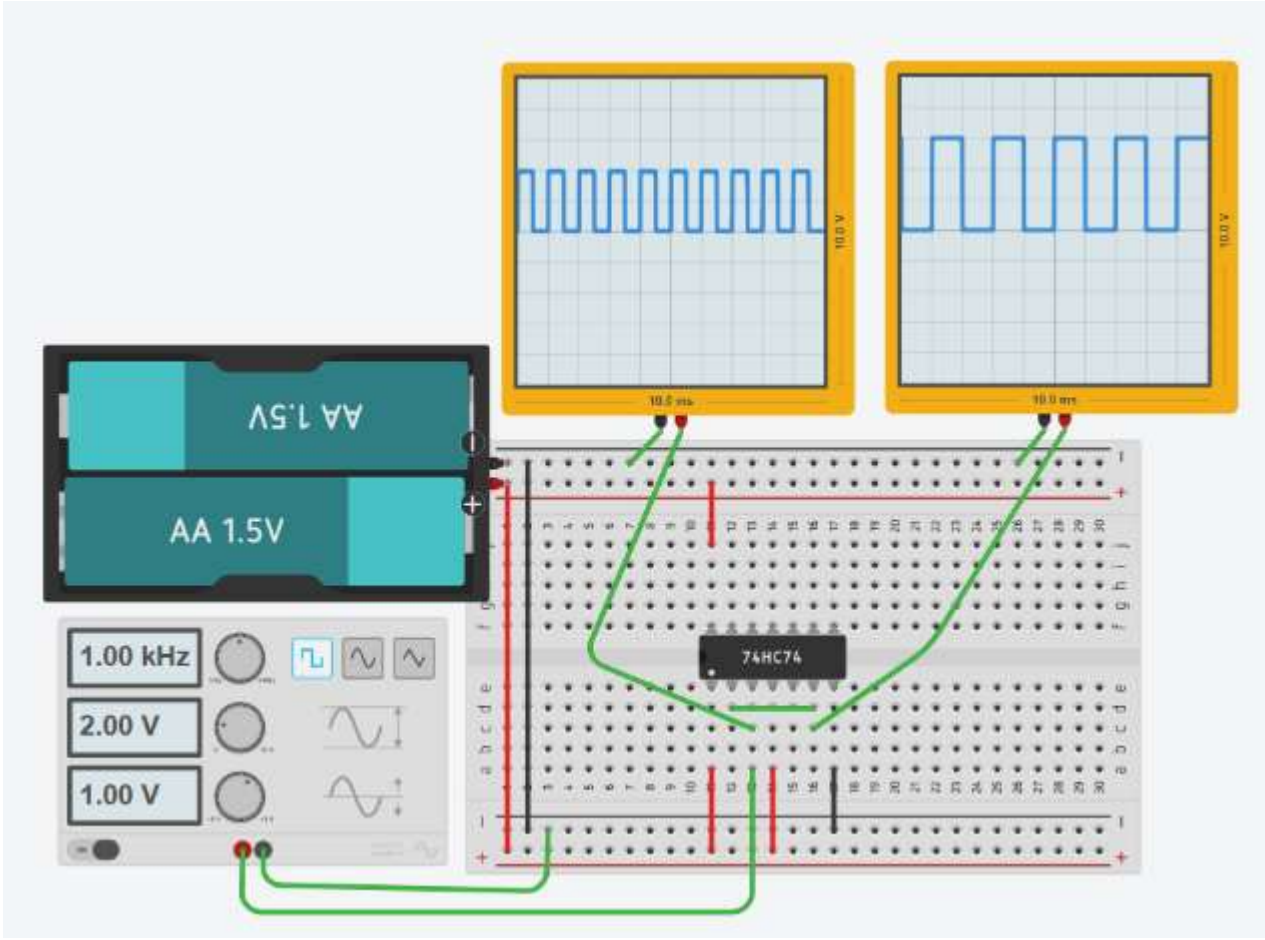


What problem it solves?

- Used when a user wants to decrease clock frequency

Popular Products

- SN74LVC1G374 | SN74AHC74Q-Q1



SN74AUP1G79 Low-Power Single Positive-Edge-Triggered D-Type Flip-Flop

- The SN74AUP1G79 is a single positive-edgetriggered D-type flip-flop.
- When data at the data (D) input meets the setup-time requirement, the data is transferred to the Q output on the positive-going edge of the clock pulse

8.2 Functional Block Diagram

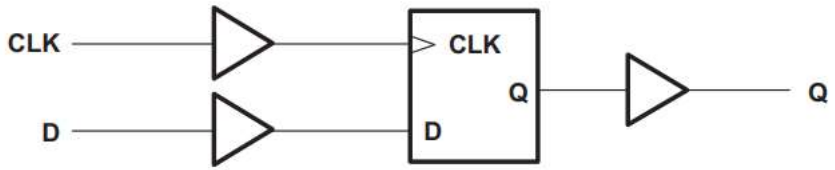
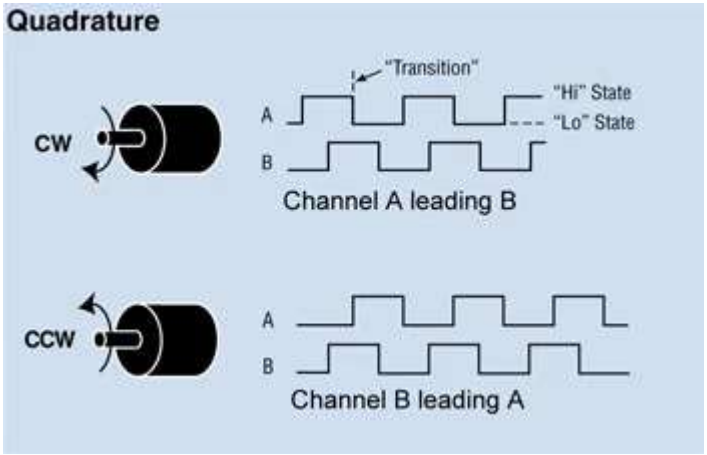


Figure 5. Logic Diagram (Positive Logic)

Table 1. Function Table

INPUTS		OUTPUT
CLK	D	Q
↑	H	H
↑	L	L
L or H	X	Q_0



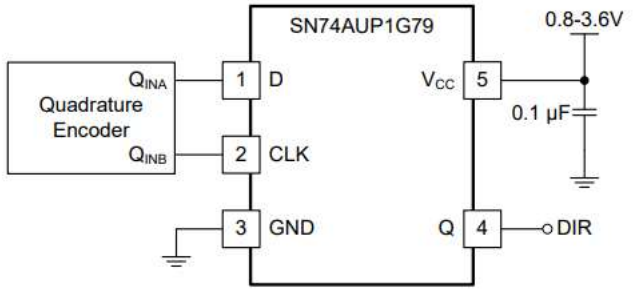
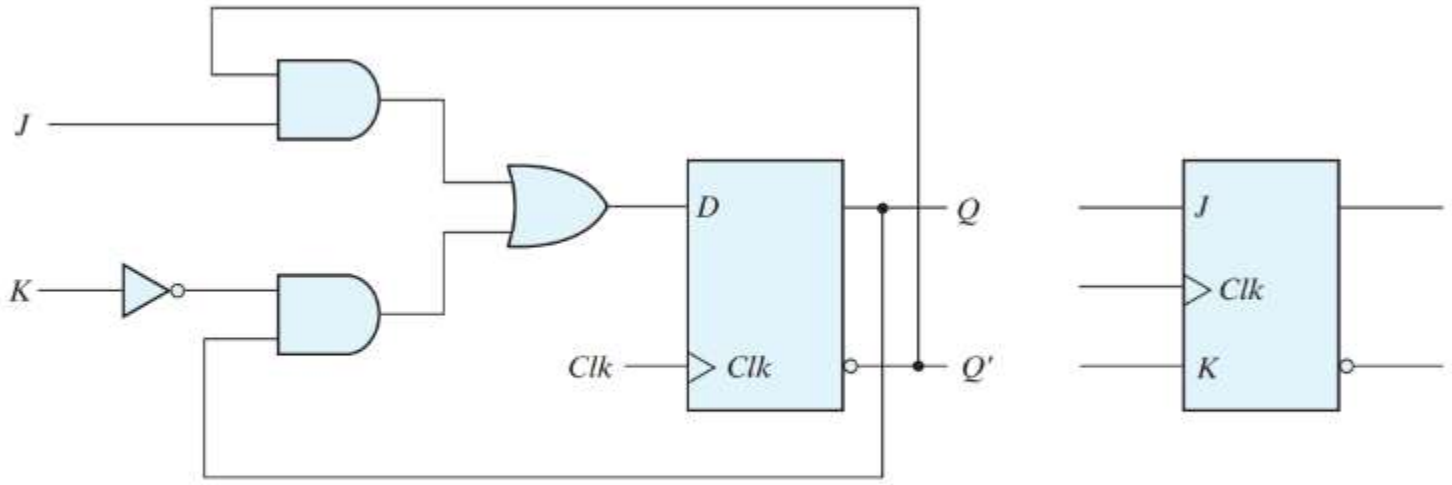


Figure 7. Typical Application Diagram

JK Flip Flops

- With only a single input, the D flip-flop can set or reset the output,
- the JK flip-flop has two inputs and performs three functions : Set it to 1, reset it to 0, or **complement** its output
- The **J input sets** the flip-flop to 1, **the K input resets** it to 0, and when **both inputs** are enabled, the output is complemented.
- This can be verified by investigating the circuit applied to the D input:
- D = JQ’ + K’Q



(a) Circuit diagram (b) Graphic symbol

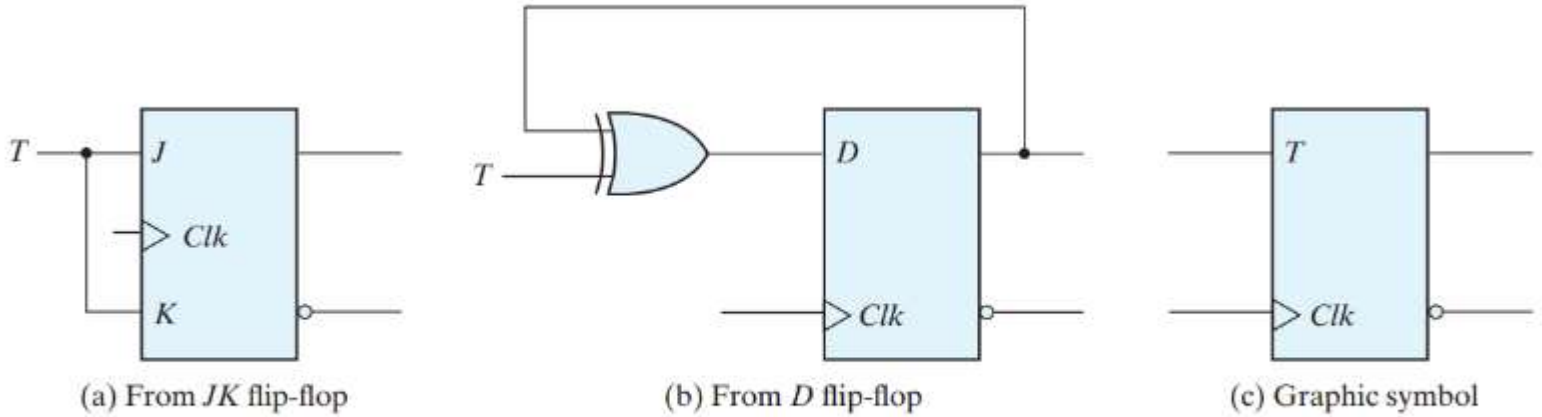
FIGURE 5.12
JK flip-flop

Table 5.1
Flip-Flop Characteristic Tables

***JK* Flip-Flop**			
J	***K***	***Q*(*t* + 1)**	
0	0	$Q(t)$	No change
0	1	0	Reset
1	0	1	Set
1	1	$Q'(t)$	Complement

T Flip Flops

- The T (toggle) flip-flop is a complementing flip-flop and can be obtained from a JK flip-flop when inputs J and K are tied together. This is shown in Fig. 5.13(a).
- When T = 0 (J = K = 0), a clock edge does not change the output. When T = 1 (J = K = 1), a clock edge **complements** the output. The **complementing flip-flop is useful for designing binary counters**.



(a) From *JK* flip-flop (b) From *D* flip-flop (c) Graphic symbol

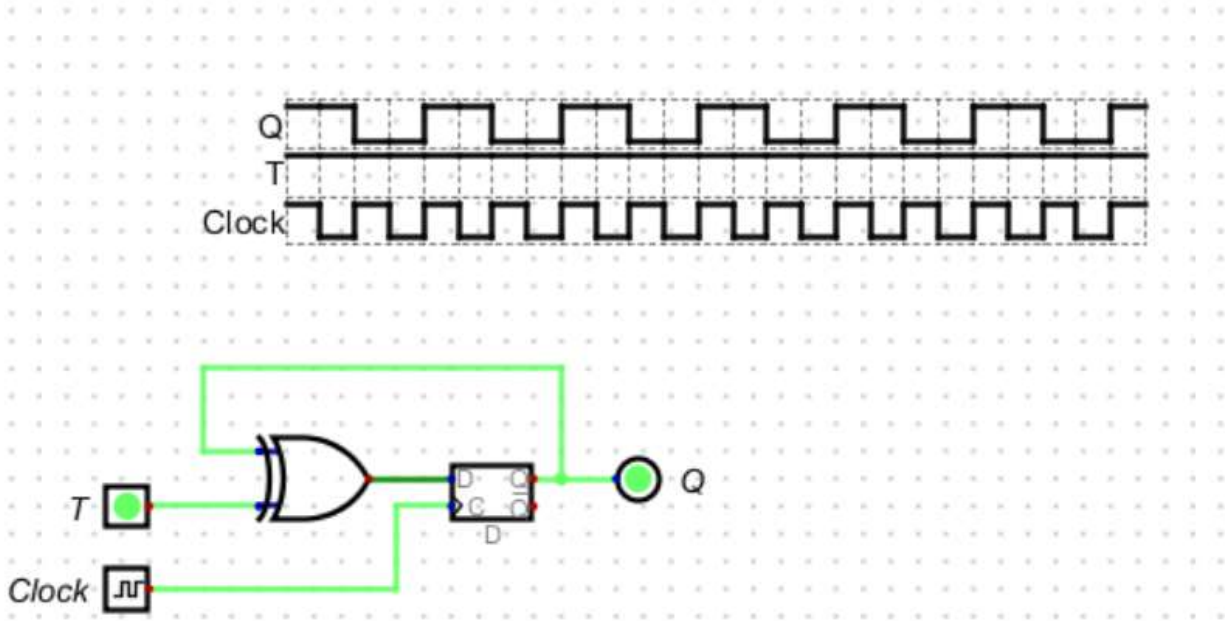
FIGURE 5.13
T flip-flop

T Flip-Flop

T	Q(t + 1)	
0	$Q(t)$	No change
1	$Q'(t)$	Complement

T Flip Flops

- The T (toggle) flip-flop is a complementing flip-flop and can be obtained from a JK flip-flop when inputs J and K are tied together. This is shown in Fig. 5.13(a).
- When T = 0 (J = K = 0), a clock edge does not change the output. When T = 1 (J = K = 1), a clock edge **complements** the output. The **complementing flip-flop is useful for designing binary counters**.

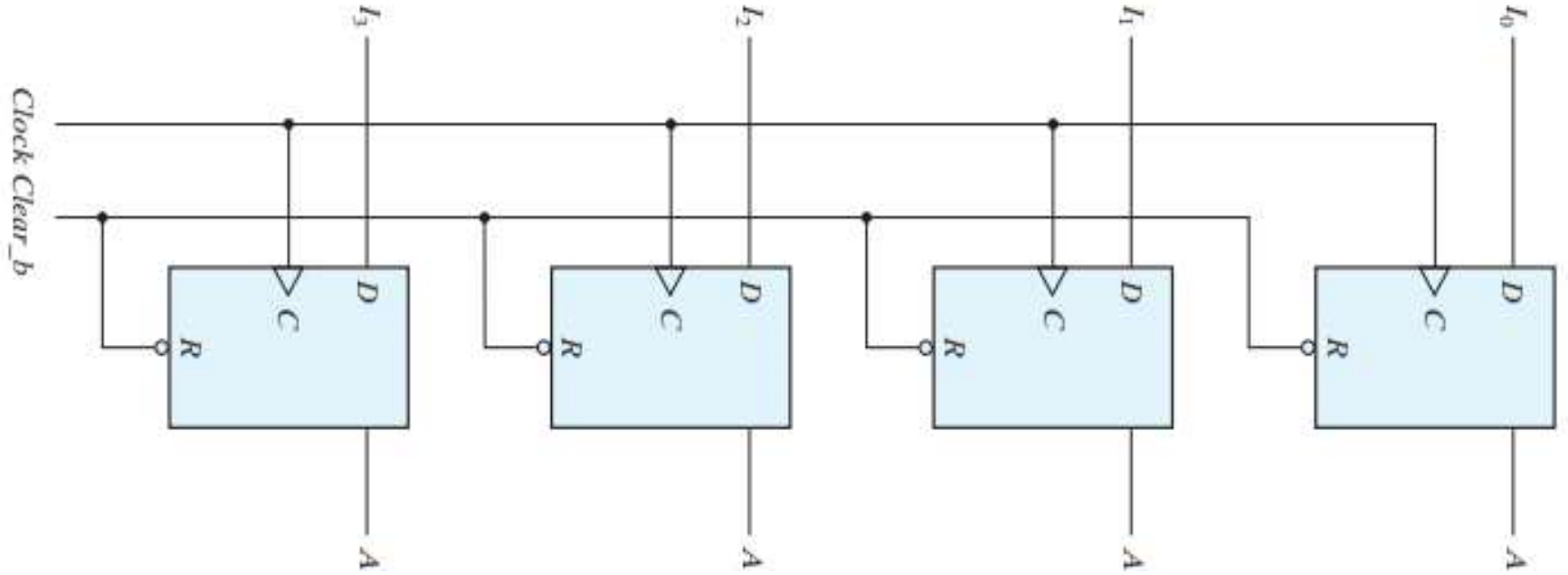


T Flip-Flop

T	Q(t + 1)	
0	$Q(t)$	No change
1	$Q'(t)$	Complement

6.1. Registers with Parallel Load

- A register is a **group of flip-flops**, each one of which shares a common clock and is capable of storing one bit of information
- Registers with **parallel load** are a fundamental building block in digital systems
- If all the bits of the register are **loaded simultaneously with a common clock pulse**, we say that the **loading is done in parallel**
- Performing logic with clock pulses inserts variable delays and may cause the system to go out of synchronism



6.1. Registers

- A four-bit data-storage register with a **load control** input that is directed through gates and into the D inputs of the flip-flops is shown in Fig. 6.2.
- The additional gates implement a two-channel mux whose output drives the input to the register with either the data bus or the output of the register
- The load input to the register determines the action to be taken with each clock pulse.
- **When the load input is 1, the data at the four external inputs are transferred into the register** with the next positive edge of the clock. **Write enable**
- **When the load input is 0, the outputs of the flip-flops are connected to their respective inputs (memory)**
- The feedback connection from output to input is necessary because a D flip-flop does not have a "**no change**" condition

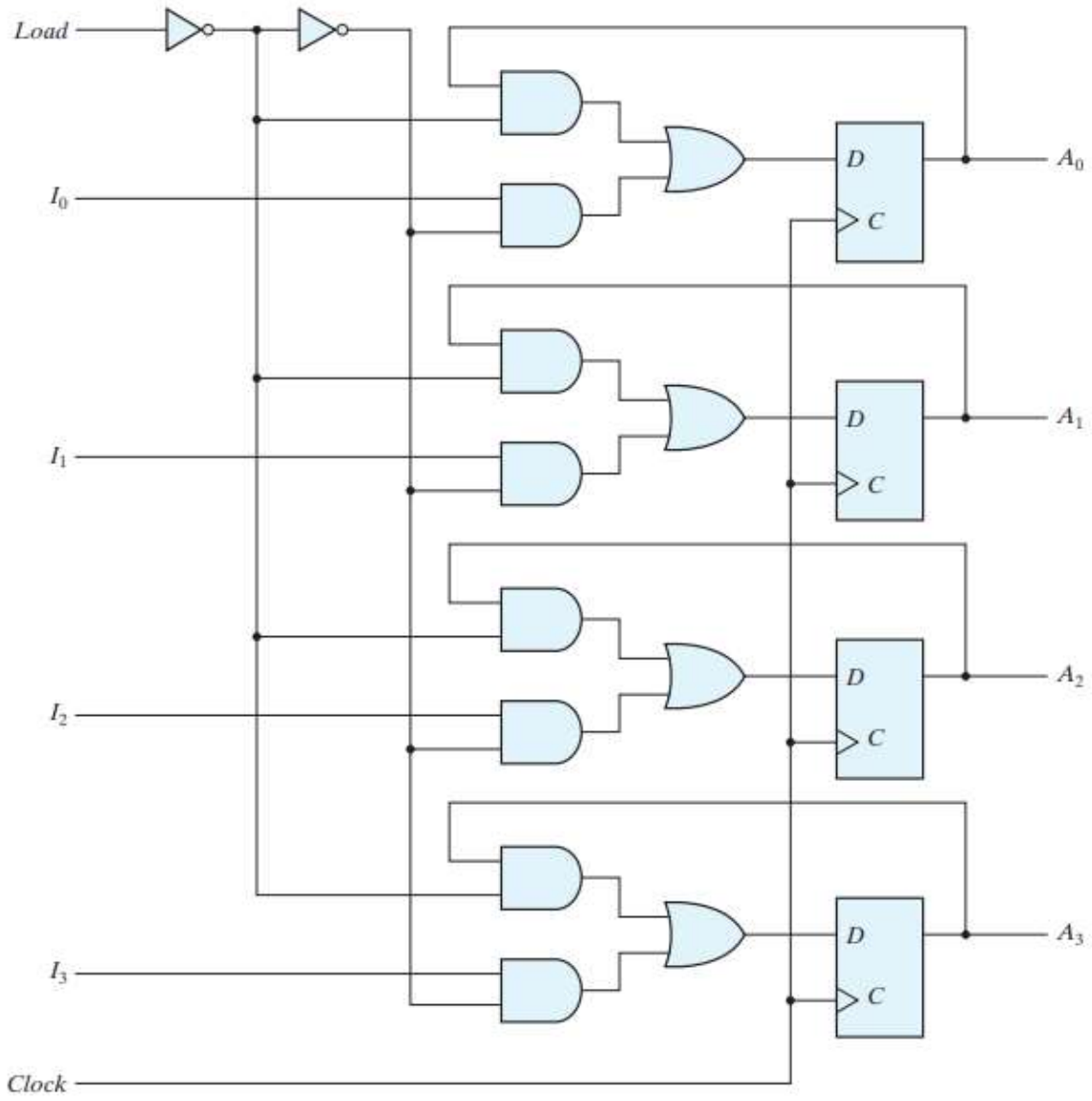
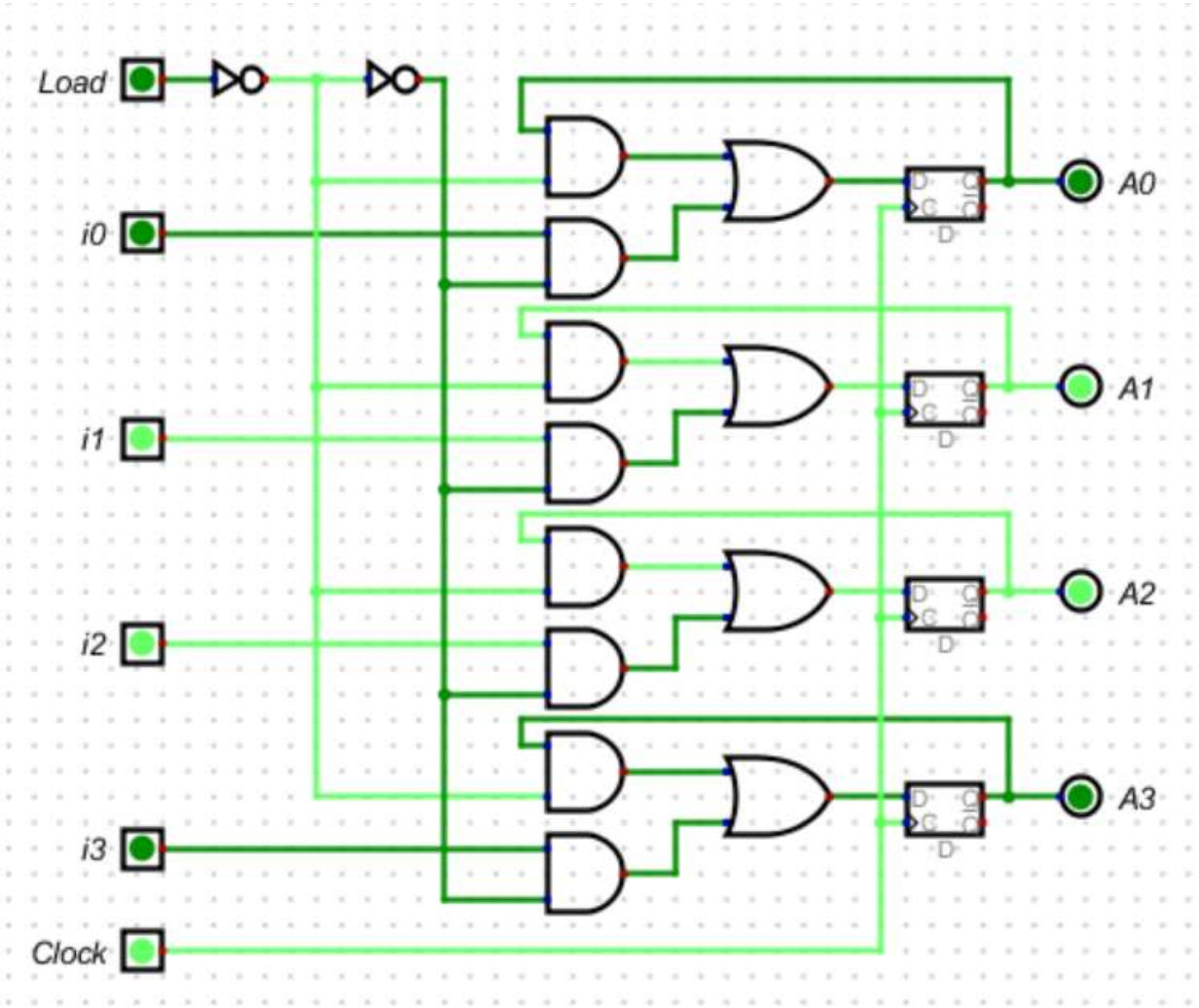


FIGURE 6.2
Four-bit register with parallel load

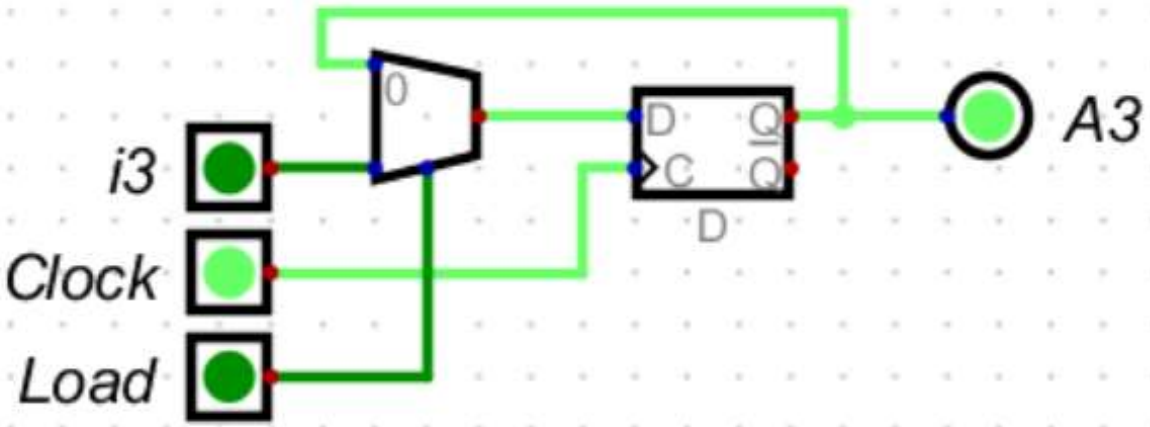
6.1. Registers

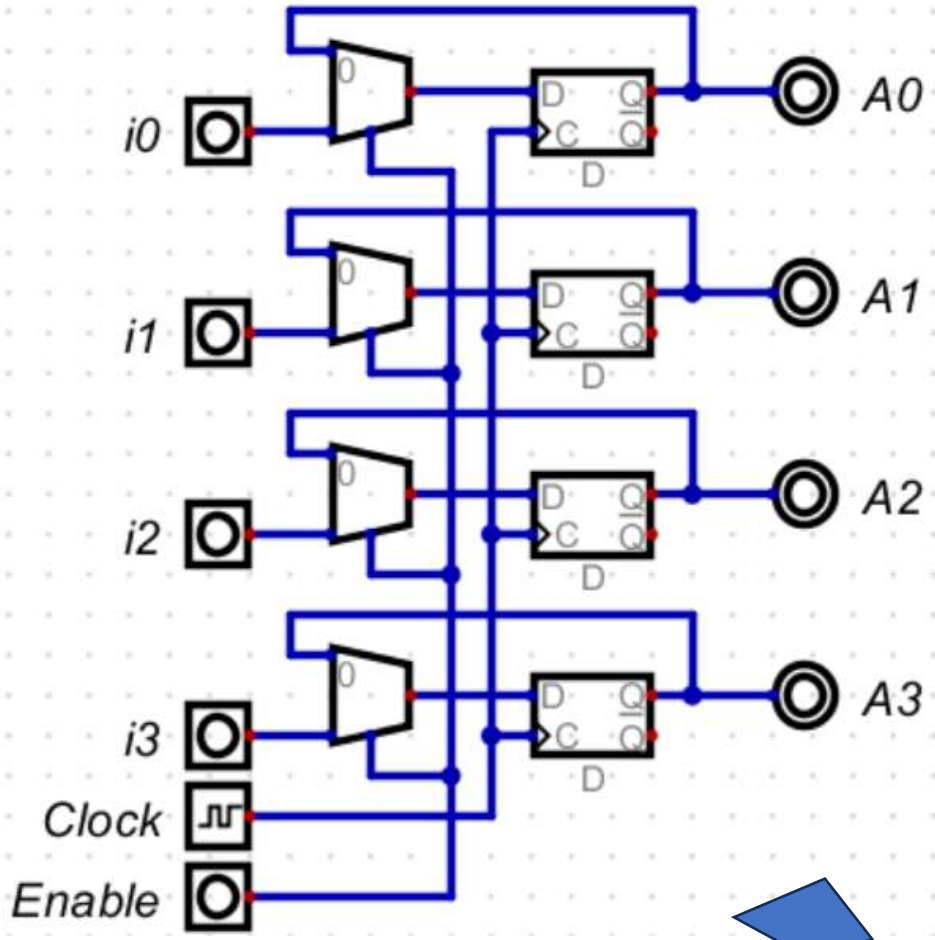
- Load=1, write enable, input transferred into register at rising edge of clock
- **Load=0 write protect. register values are kept**



6.1. Registers

- Enable=1, write enable, input transferred into register at rising edge of clock
- **Enable=0 write protect. register values are kept**





MC14076B

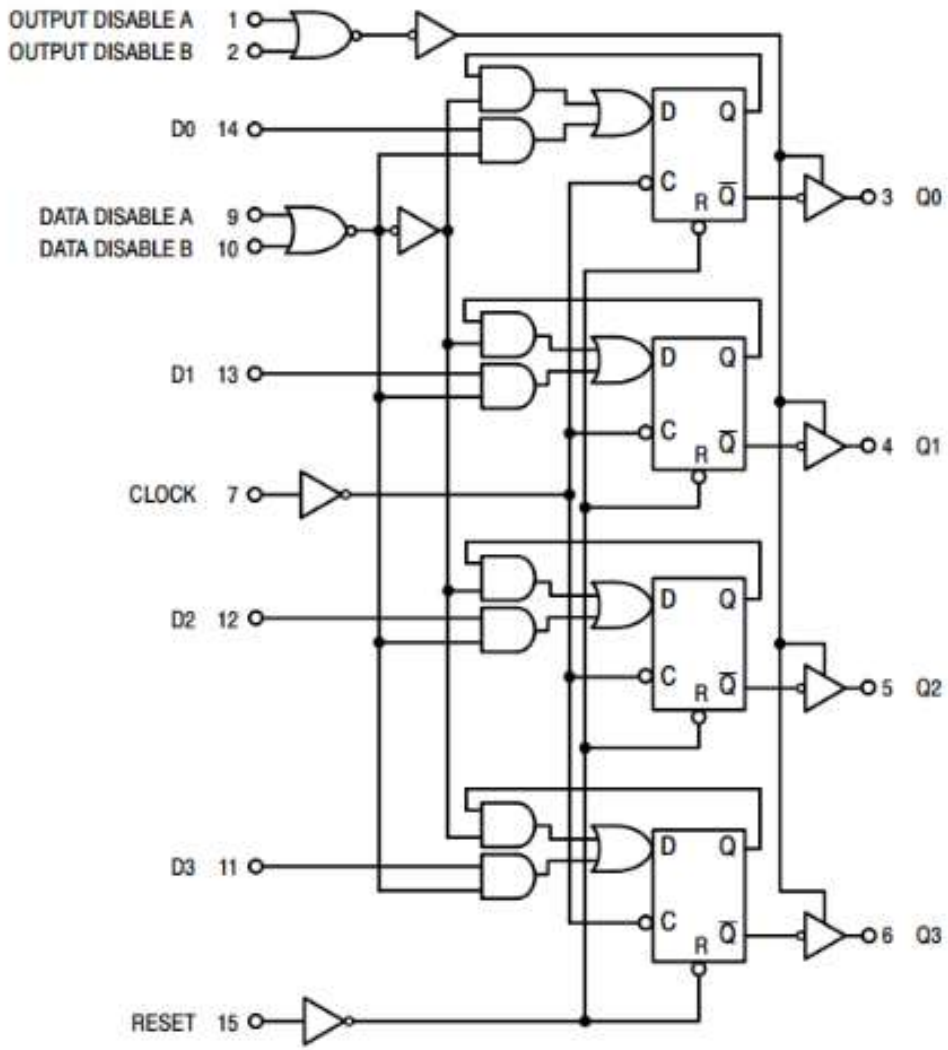
4-Bit D-Type Register with Three-State Outputs

- The MC14076B 4−Bit Register consists of **four D−type flip−flops** operating synchronously from a common clock.
- OR gated **output−disable inputs** force the outputs into a high−impedance state for use in bus organized systems.
- OR gated **data−disable inputs** cause the Q outputs to be fed back to the D inputs of the flip−flops.
- **Thus they are inhibited from changing state while the clocking process remains undisturbed**

FUNCTION TABLE

Inputs					Output
		Data Disable		Data	
Reset	Clock	A	B	D	Q
1	X	X	X	X	0
0	0	X	X	X	Q_n
0	↓	1	X	X	Q_n
0	↓	X	1	X	Q_n
0	↓	0	0	0	0
0	↓	0	0	1	1

When either output disable A or B (or both) is (are) high the output is disabled to the high−impedance state; however sequential operation of the flip−flops is not affected.
X = Don't Care.



6.2. Shift Registers

- A register capable of **shifting the binary information** held in each cell to its neighboring cell, in a selected direction, is called a **shift register**
- The logical configuration of a shift register consists of a **chain of flip-flops in cascade**, with the output of one flip-flop connected to the data input of the next flip-flop. All flip-flops receive **common clock pulses**, which activate the shift of data from one stage to the next.
- **Each clock pulse shifts the contents of the register one bit position to the right**

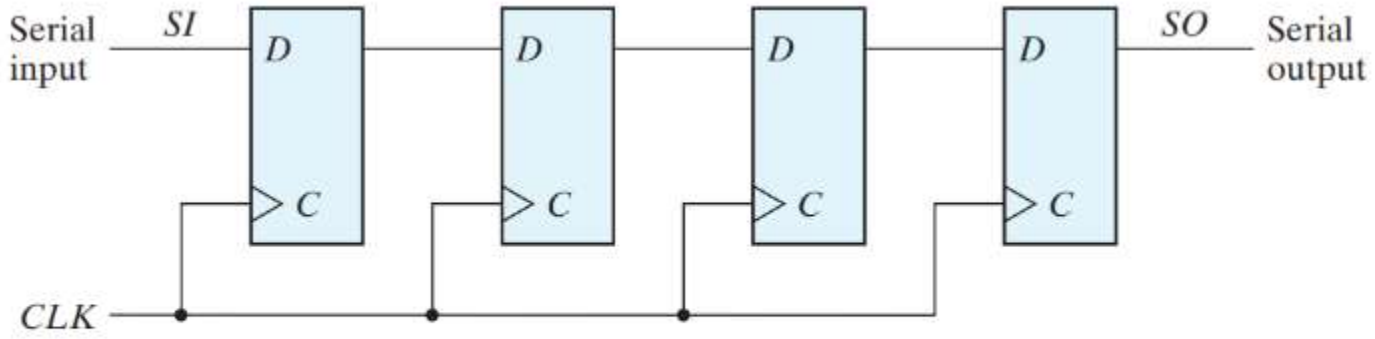
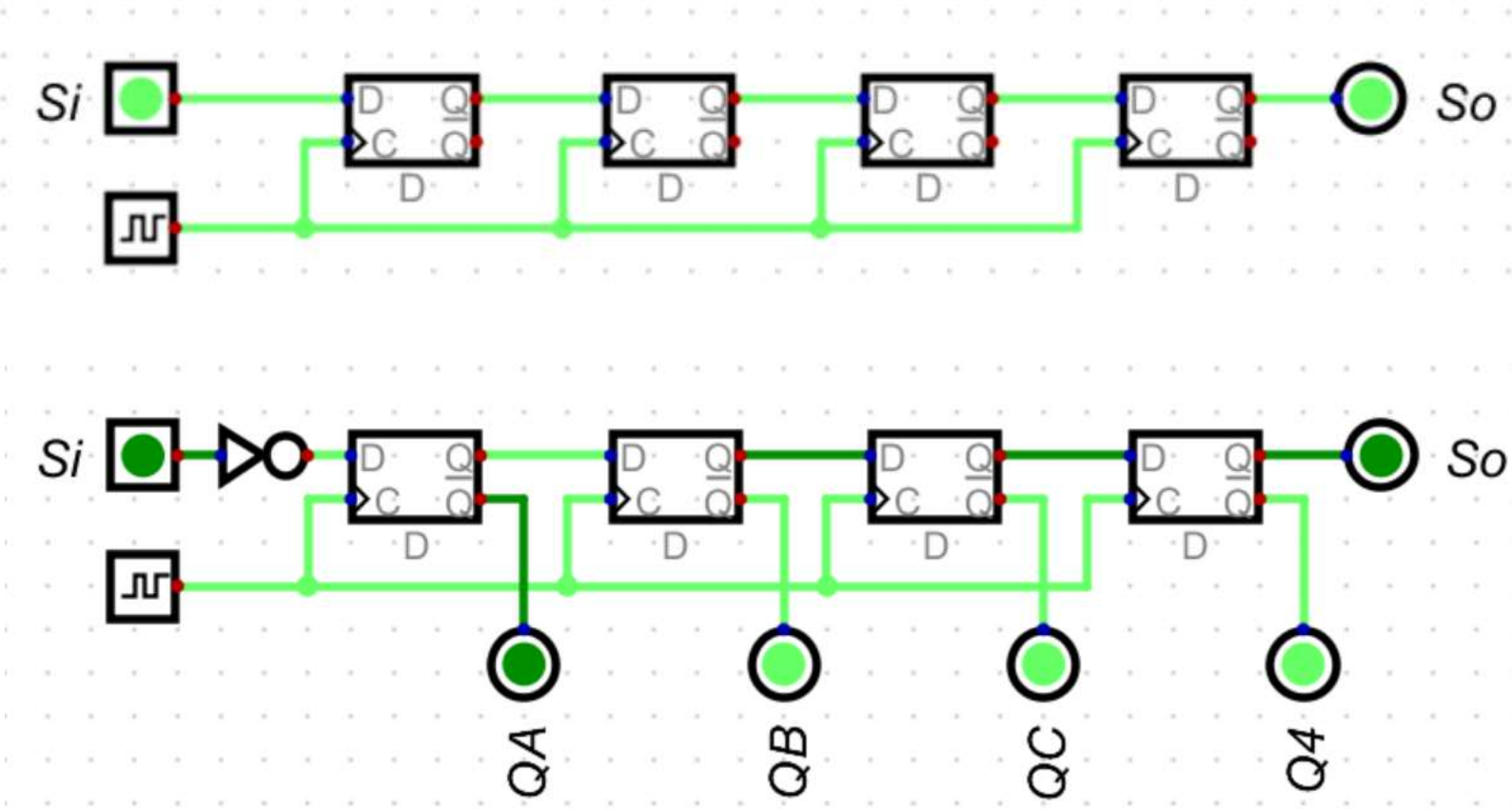


FIGURE 6.3
Four-bit shift register

6.2. Shift Registers

- **Si=1 reaches So after 4 clock cycle**
- **Serial input 4-bit Parallel output**



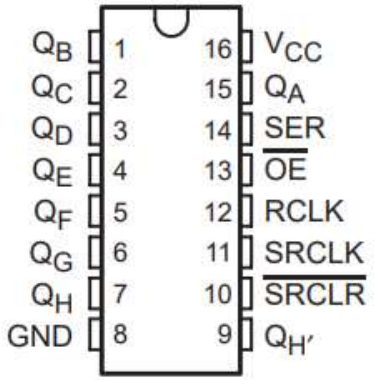
6.2. Shift Registers

CD74HC595
8-BIT SHIFT REGISTERS
WITH 3-STATE OUTPUT REGISTERS

SCHS353 – JANUARY 2004

- 8-Bit Serial-In, Parallel-Out Shift
- Wide Operating Voltage Range of 2 V to 6 V
- High-Current 3-State Outputs Can Drive Up To 15 LSTTL Loads
- Low Power Consumption, 80-μA Max I_{CC}
- Typical t_{pd} = 14 ns
- ±6-mA Output Drive at 5 V
- Low Input Current of 1 μA Max
- Shift Register Has Direct Clear

DW, E, M, NS, OR SM PACKAGE
(TOP VIEW)



description/ordering information

The CD74HC595 device contains an 8-bit serial-in, parallel-out shift register that feeds an 8-bit D-type storage register. The storage register has parallel 3-state outputs. Separate clocks are provided for both the shift and storage registers. The shift register has a direct overriding clear ($\overline{SRCLR}$) input, serial (SER) input, and serial output for cascading. When the output-enable ($\overline{OE}$) input is high, the outputs are in the high-impedance state.

Both the shift register clock (SRCLK) and storage register clock (RCLK) are positive-edge triggered. If both clocks are connected together, the shift register always is one clock pulse ahead of the storage register.

Applications: Shift Registers

9.2 Typical Application

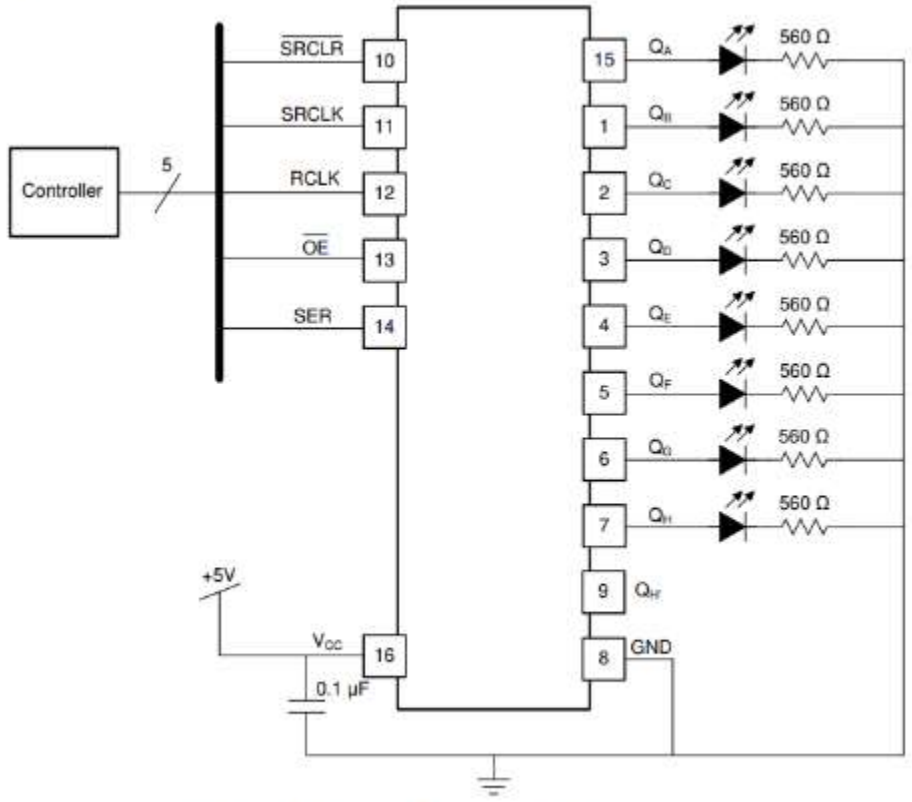
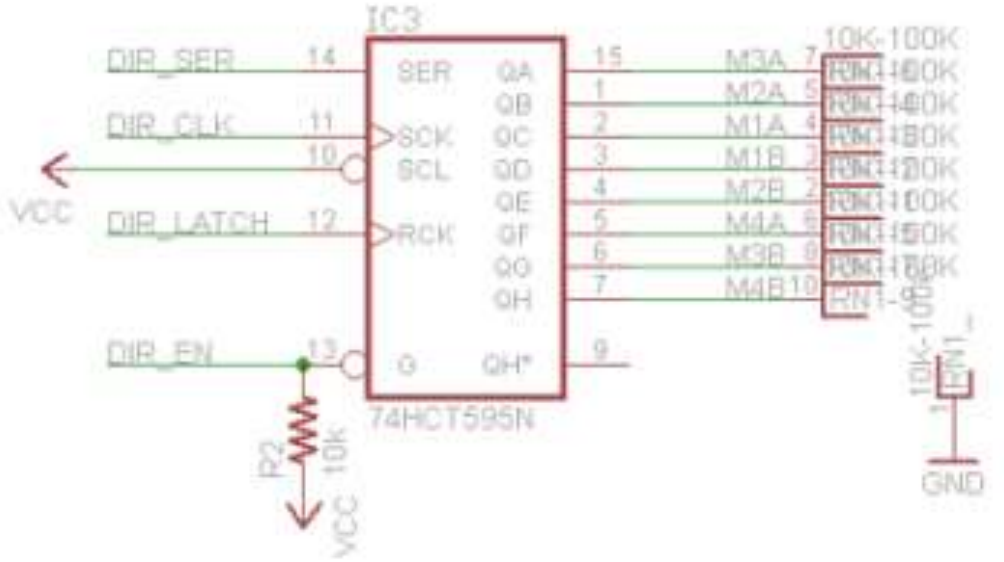


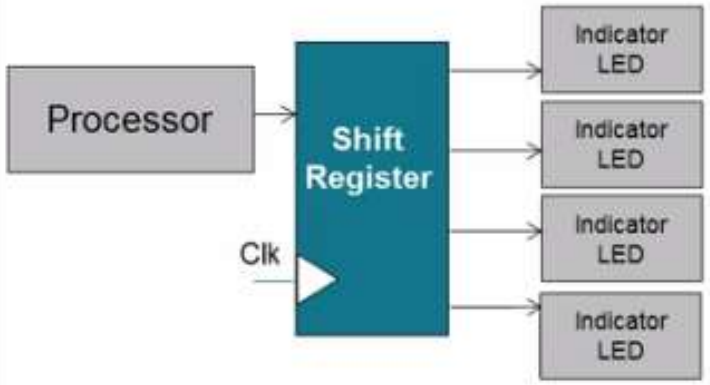
Figure 9-1. Typical Application Schematic



- Arduino Motor Shield uses 74HC595 to control H-Bridge L293D.
- Serial to parallel conversion

Applications: Shift Registers

MCU to Multiple Indicator LED's using a Shift Register



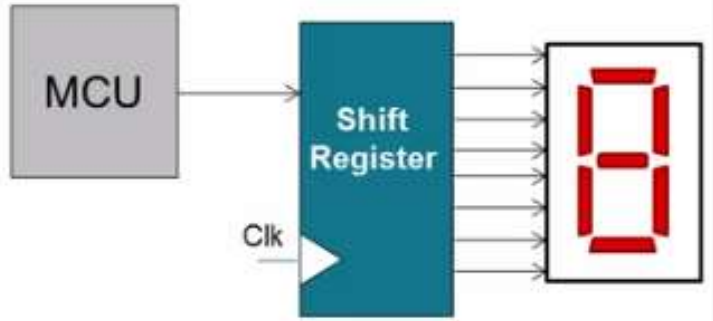
What problem it solves?

- Used to expand the MCU's number of outputs for individual LED control

Popular Products

- SN74AHC595-Q1 | SN74HC595

7-Segment Display using Shift Register



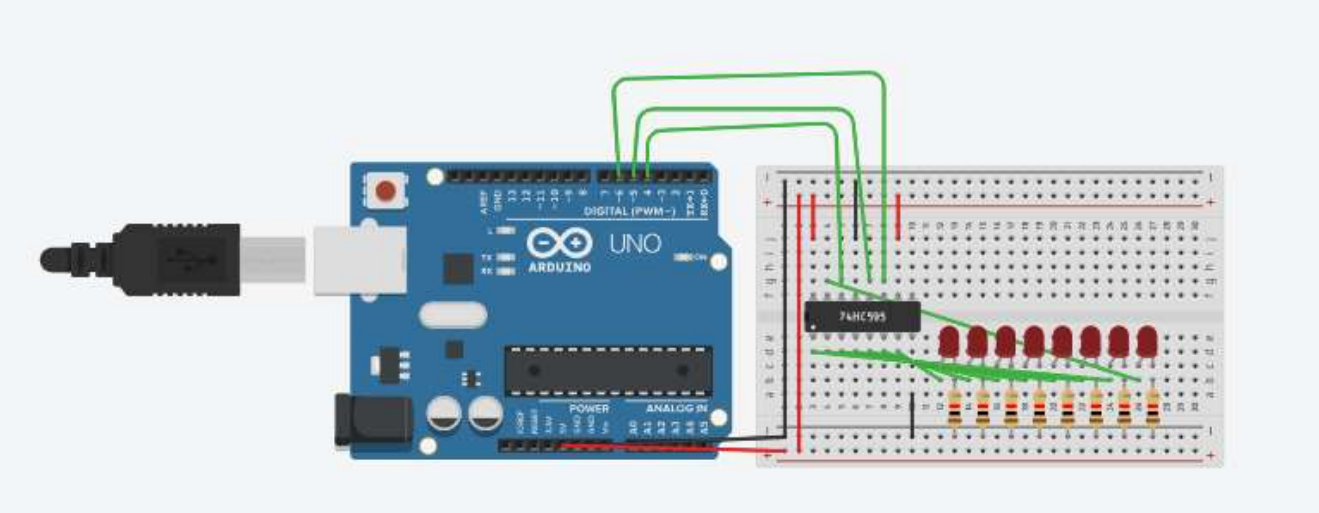
What problem it solves?

- Used to expand the MCU's number of outputs for 7 segment displays

Popular Products

- SN74AHC595-Q1 | CD4021B

6.2. Shift Registers (74HC595)



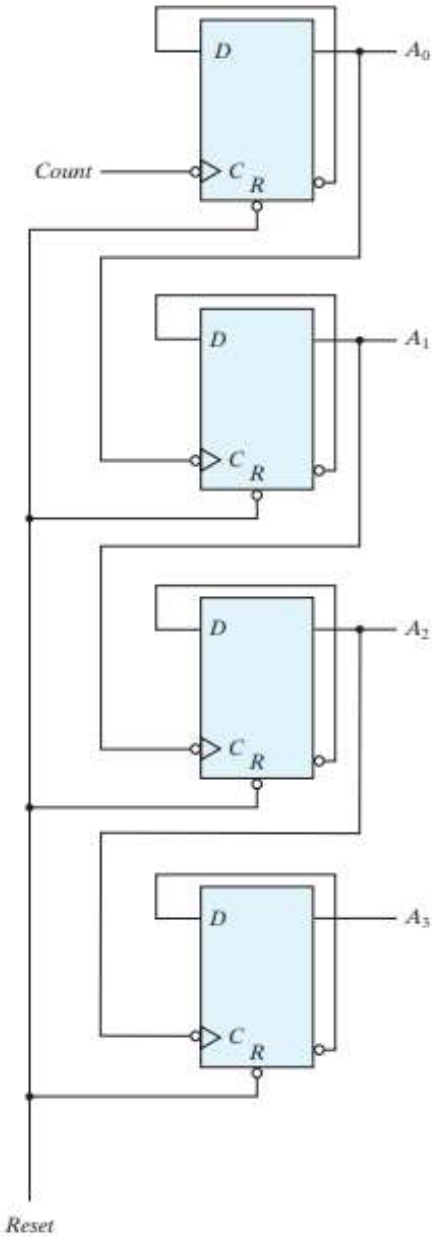

```
//Pin connected to ST_CP of 74HC595
int latchPin = 5;
//Pin connected to SH_CP of 74HC595
int clockPin = 6;
////Pin connected to DS of 74HC595
int dataPin = 4;
void setup() {
//set pins to output so you can control the shif
pinMode(latchPin, OUTPUT);
pinMode(clockPin, OUTPUT);
pinMode(dataPin, OUTPUT);
}
void loop() {
digitalWrite(latchPin, LOW);
shiftOut(dataPin, clockPin, MSBFIRST, 1);
digitalWrite(latchPin, HIGH);
delay(500);
digitalWrite(latchPin, LOW);
shiftOut(dataPin, clockPin, MSBFIRST, 4);
digitalWrite(latchPin, HIGH);
delay(500);
```

Ripple Counters

- A binary ripple counter consists of a **series connection of complementing flip-flops**, with the output of each flip-flop connected to the input of the next higher order flipflop
- A third possibility is to use a D flip-flop with the complement output connected to the D input. In this way, the **D input is always the complement of the present state**, and the **next clock pulse will cause the flip-flop to complement**

Table 6.4
Binary Count Sequence

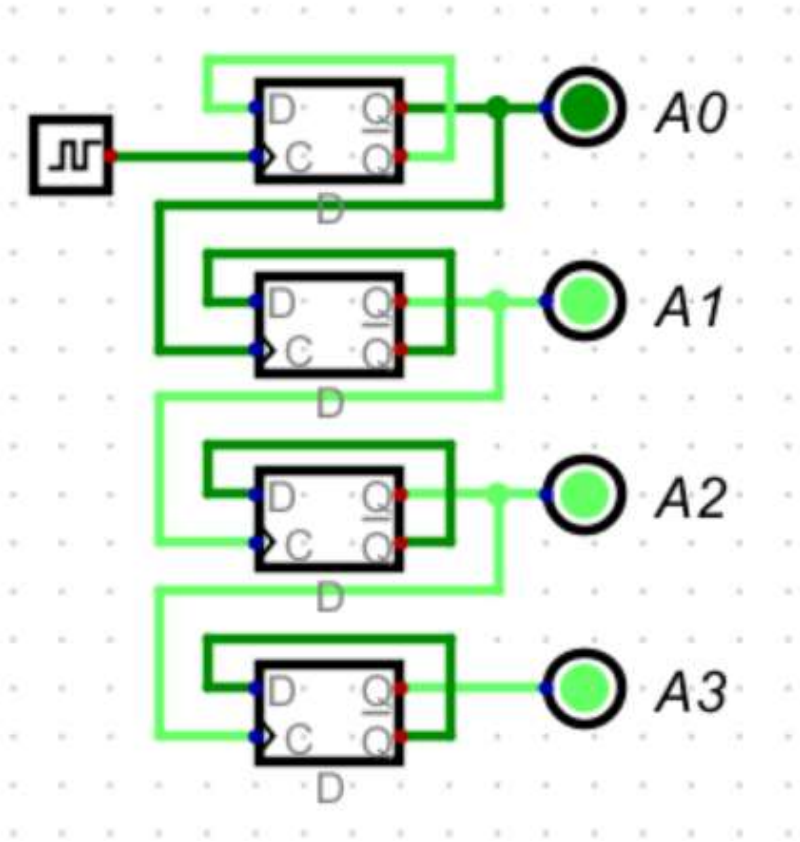
A_3	A_2	A_1	A_0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0



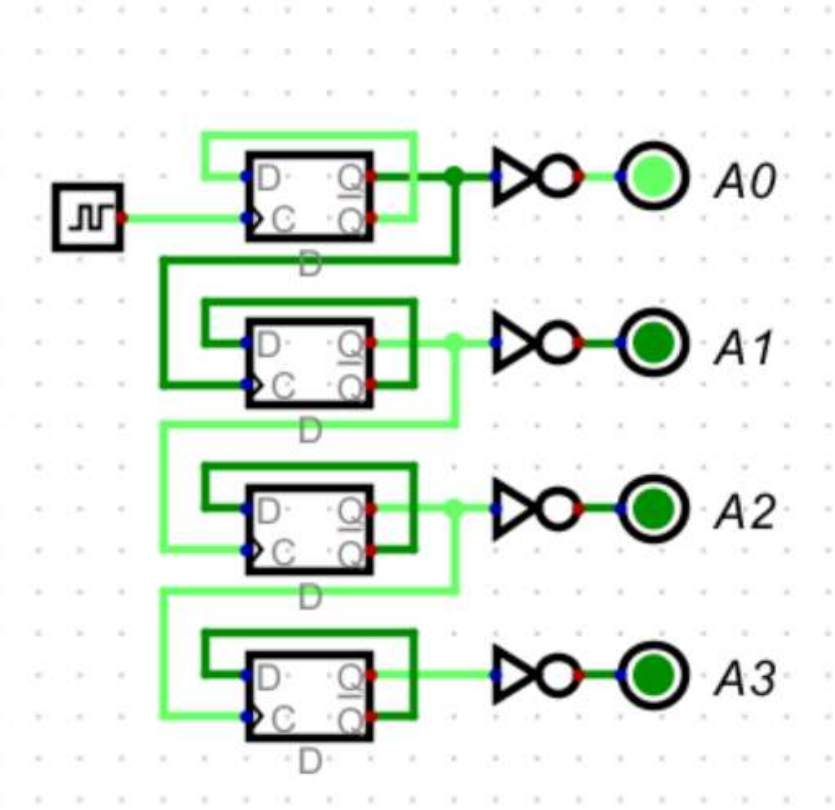
(b) With *D* flip-flops

Ripple Counters: Downcounter - Upcounter

- **4bit down counter 15,14,13…**



- **4bit up counter 0,1,2,3…**



FAIRCHILD
SEMICONDUCTOR

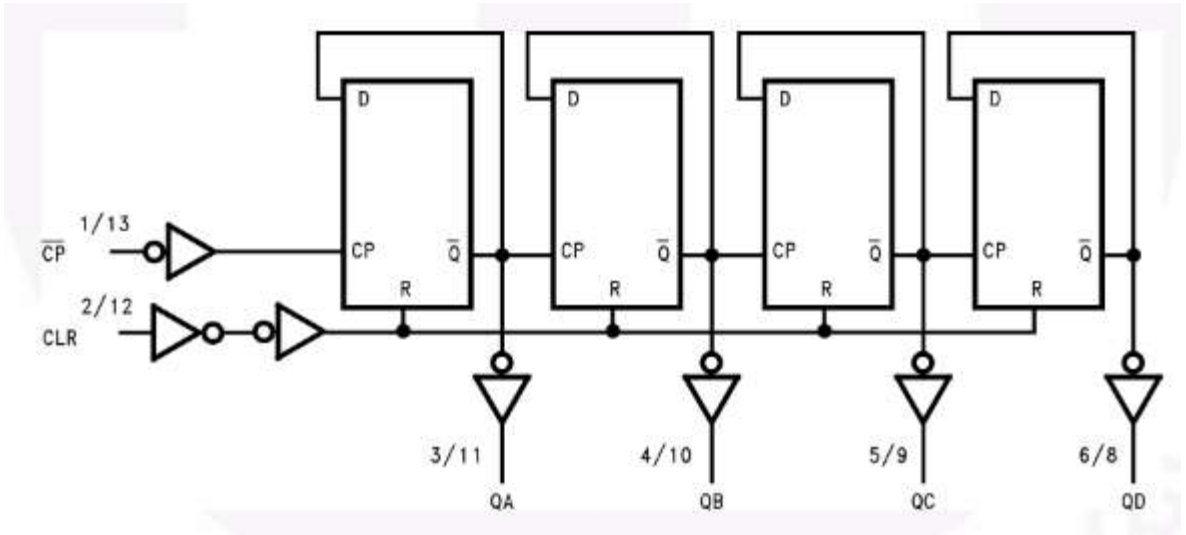
74VHC393
Dual 4-Bit Binary Counter

- It contains two independent counter circuits in one package, so that counting or frequency division of 8 binary bits can be achieved with one IC.
- This device changes state on the negative going transition of the CLOCK pulse.
- The counter can be reset to “0” (Q0–Q3 = “L”) by a HIGH at the CLEAR input regardless of other inputs

Truth Table

Inputs		Outputs			
$\overline{CP}$	CLR	QA	QB	QC	QD
X	H	L	L	L	L
↓	L	Count Up			
↑	L	No Change			

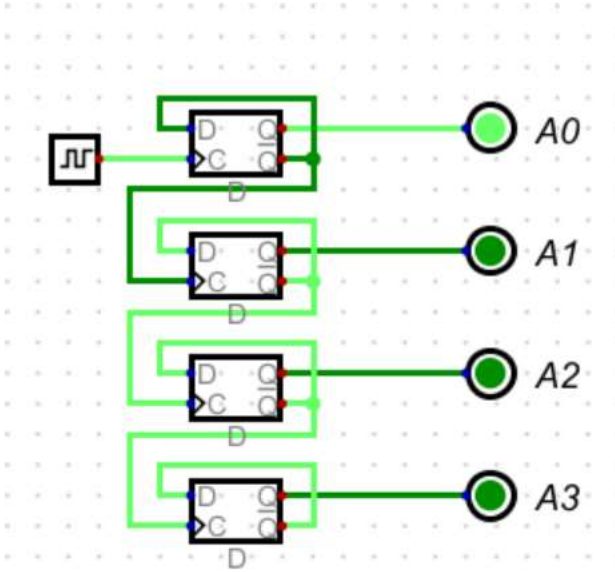
X: Don’t Care



FAIRCHILD
SEMICONDUCTOR

74VHC393
Dual 4-Bit Binary Counter

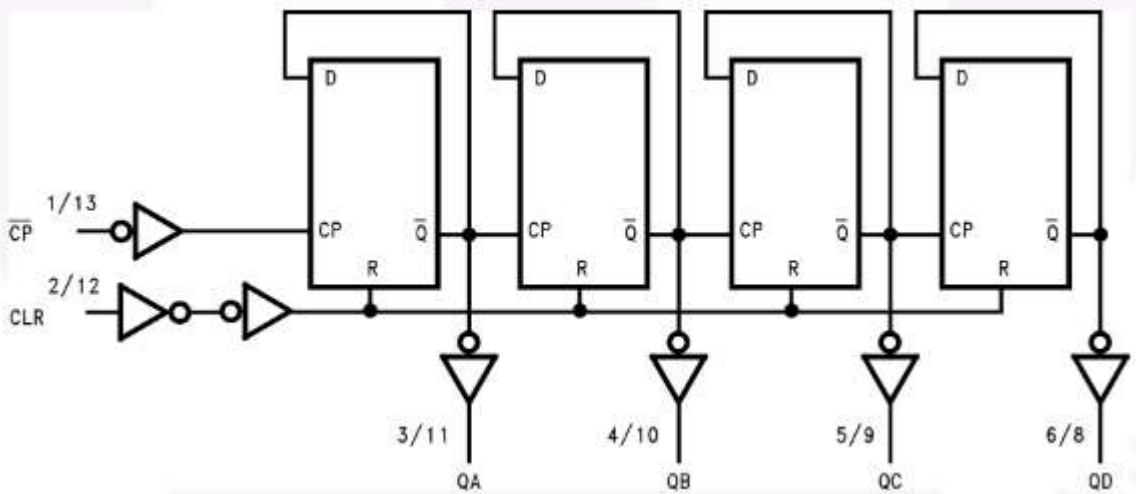
- It contains two independent counter circuits in one package, so that counting or frequency division of 8 binary bits can be achieved with one IC.
- This device changes state on the negative going transition of the CLOCK pulse.
- The counter can be reset to “0” (Q0–Q3 = “L”) by a HIGH at the CLEAR input regardless of other inputs



Truth Table

Inputs		Outputs			
$\overline{CP}$	CLR	QA	QB	QC	QD
X	H	L	L	L	L
↓	L	Count Up			
↓	L	No Change			

X: Don't Care



4 Bit Binary Counter

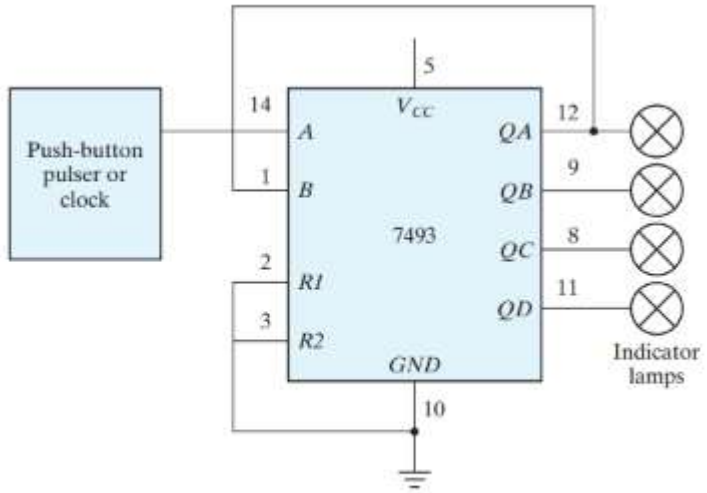
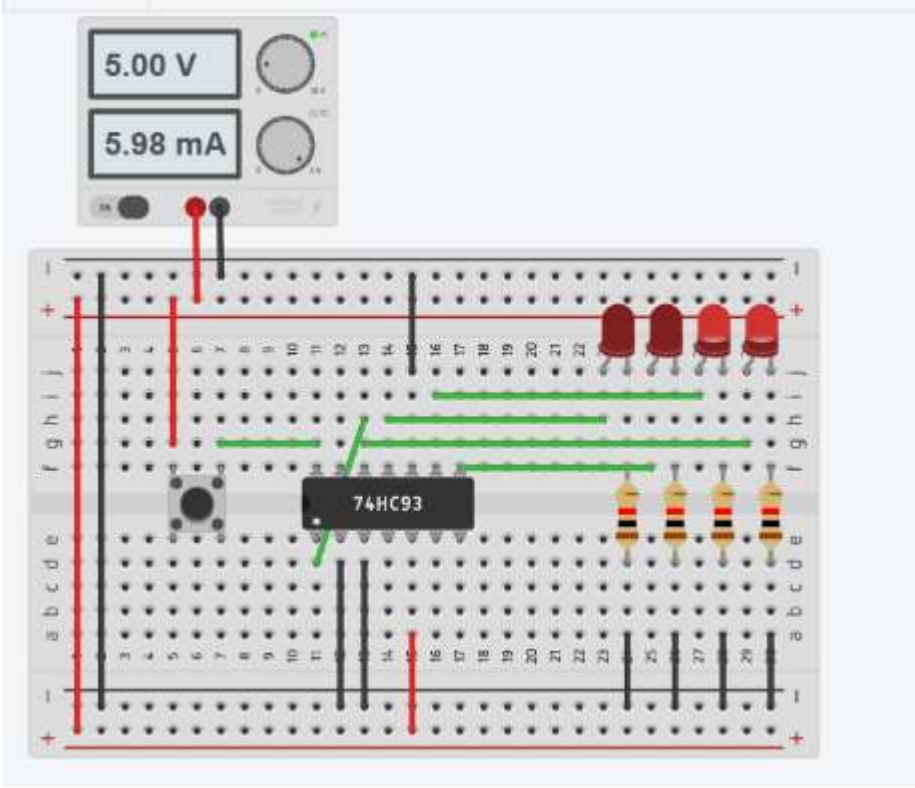


FIGURE 9.3
Binary counter



BCD Counter

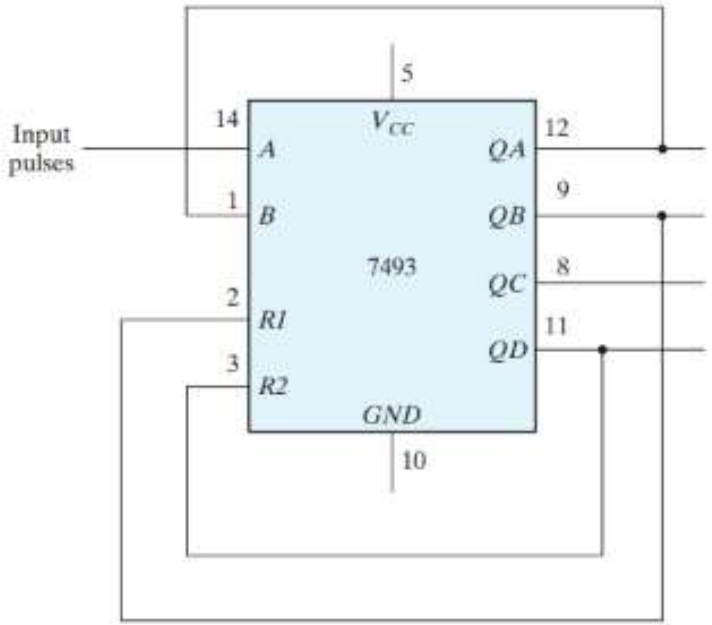
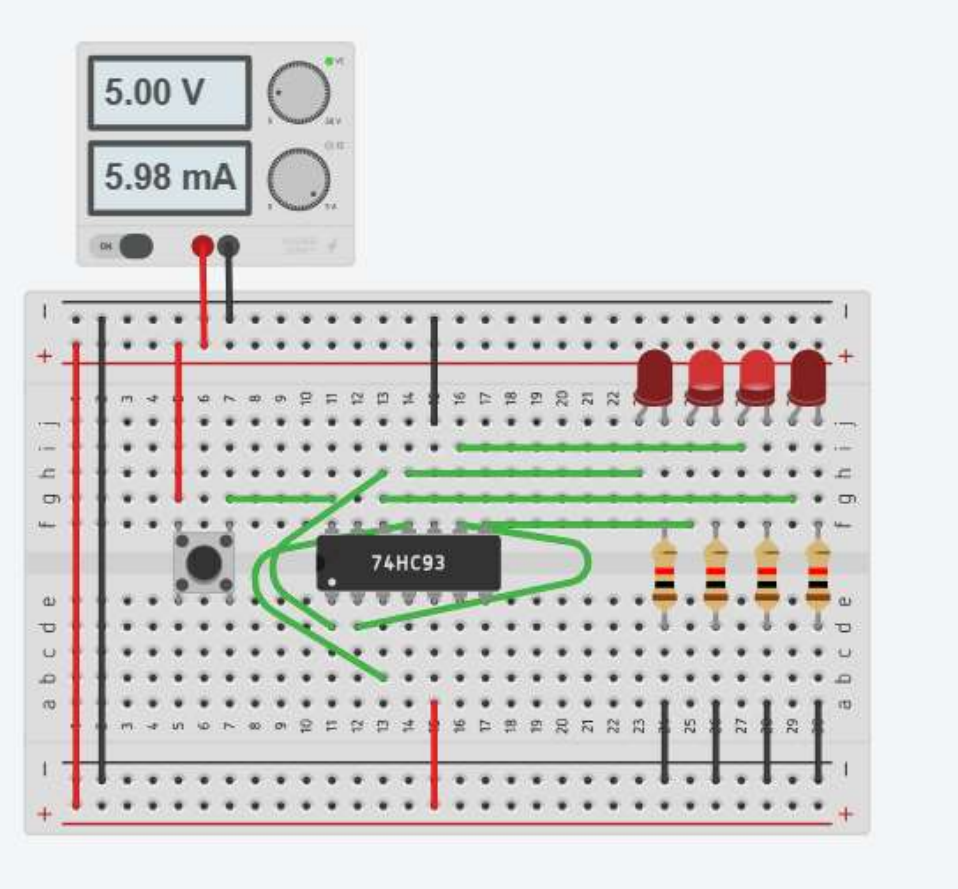


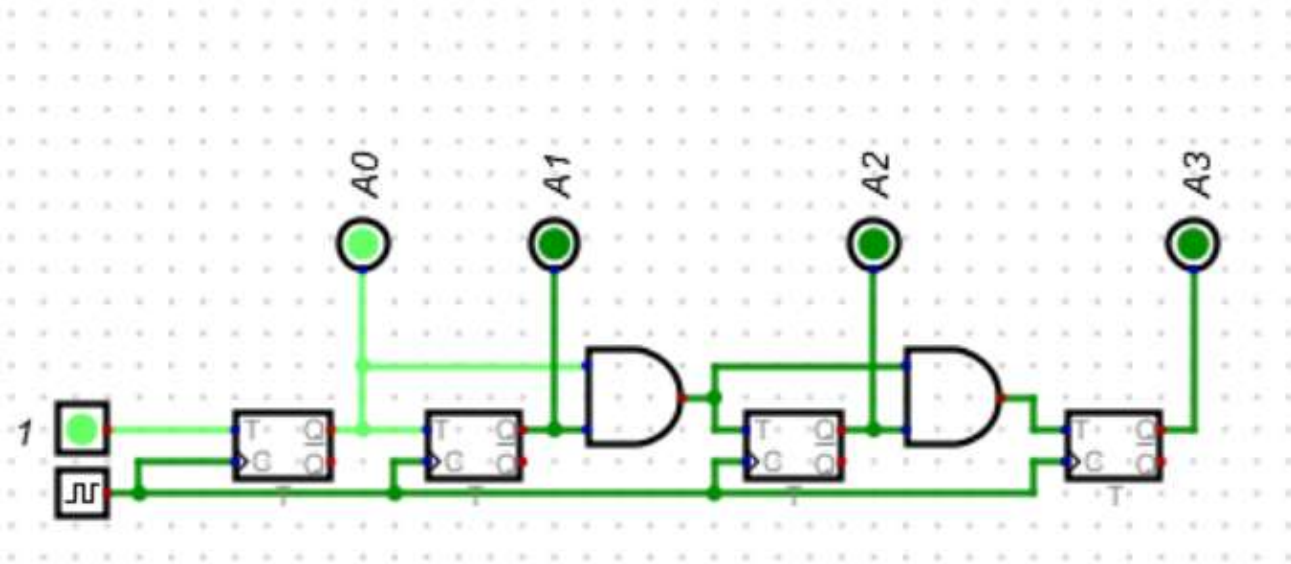
FIGURE 9.4
BCD counter

making the external connections shown in Fig. 9.4. Outputs *QB* and *QD* are connected to the two reset inputs, *R1* and *R2*. When both *R1* and *R2* are equal to 1, all four cells in the counter clear to 0 irrespective of the input pulse. The counter starts from 0, and every input pulse increments it by 1 until it reaches the count of 1001. The next pulse changes the ouput to 1010, making *QB* and *QD* equal to 1. This momentary output cannot be sustained, because the four cells immediately clear to 0, with the result that the output goes to 0000. Thus, the pulse after the count of 1001 changes the output to 0000, producing a BCD count.



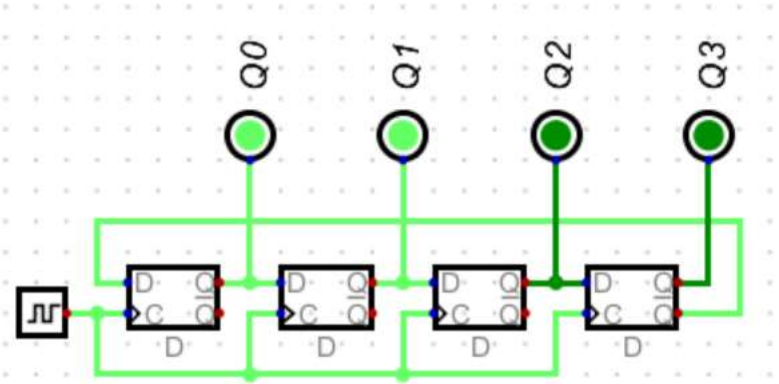
Synchronous Counter

Synchronous counters have common clock
faster, no cascade connection

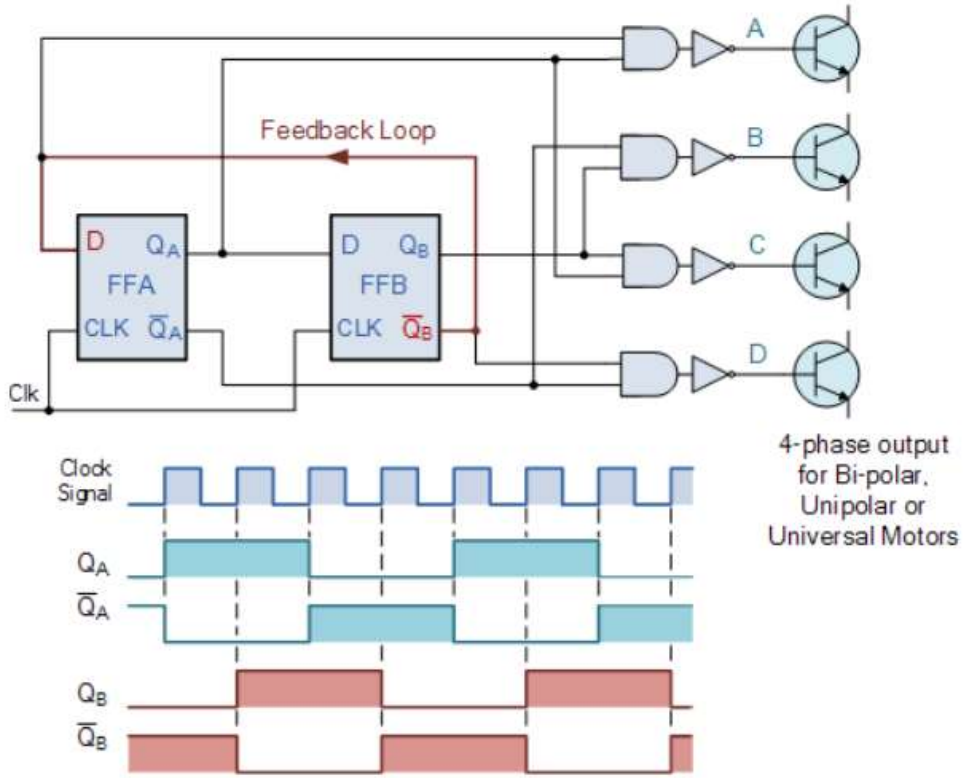


Jhonson Ring Counter

- ring counter is a circular shift register with only one flip-flop being set at any particular time;
- Standard 2, 3 or 4-stage **Johnson Ring Counters** can also be used to divide the frequency of the clock signal by varying their feedback connections and divide-by-3 or divide-by-5 outputs are also available.

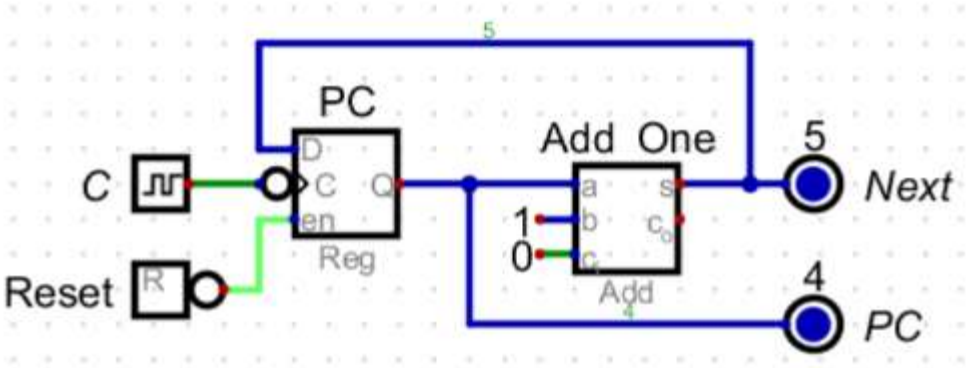


2-bit Quadrature Generator



Program Counter

- Program counter is the **ADRESS OF THE NEXT INSTRUCTION** to be executed
- it is a binary counter increments at each clock
- jumping up/down is possible



Memory

- A memory unit is a device to which binary information is **transferred for storage** and from which information is **retrieved when needed** for processing.
- When data processing takes place, information from memory is **transferred to selected registers** in the processing unit.
- Intermediate and final results obtained in the ALU processing unit are **transferred back to be stored** in memory.
- Binary information received from an input device is stored in memory, and information transferred to an output device is taken from memory
- There are two types of memories that are used in digital systems: **random-access memory (RAM) and read-only memory (ROM).**
- RAM stores new information for later use. The process of storing new information into memory is referred to as a memory write operation. The process of transferring the stored information out of memory is referred to as a **memory read** operation. RAM can perform **both write and read operations**.
- **ROM can perform only the read operation**.

Read-Only Memory (ROM)

- A read-only memory (ROM) is essentially a memory device in which permanent binary information is stored
- it stays within the unit even when power is turned off and on
- electrically erasable PROM (**EEPROM**). programmed data can be erased with an electrical signal
- **Flash memory** devices are **similar to EEPROMs**, but have additional built-in circuitry to selectively program and erase the device in-circuit, without the need for a special programmer

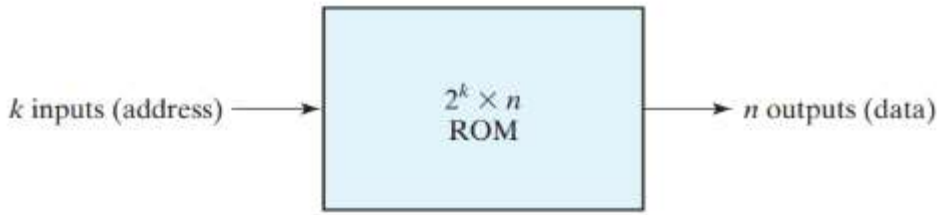
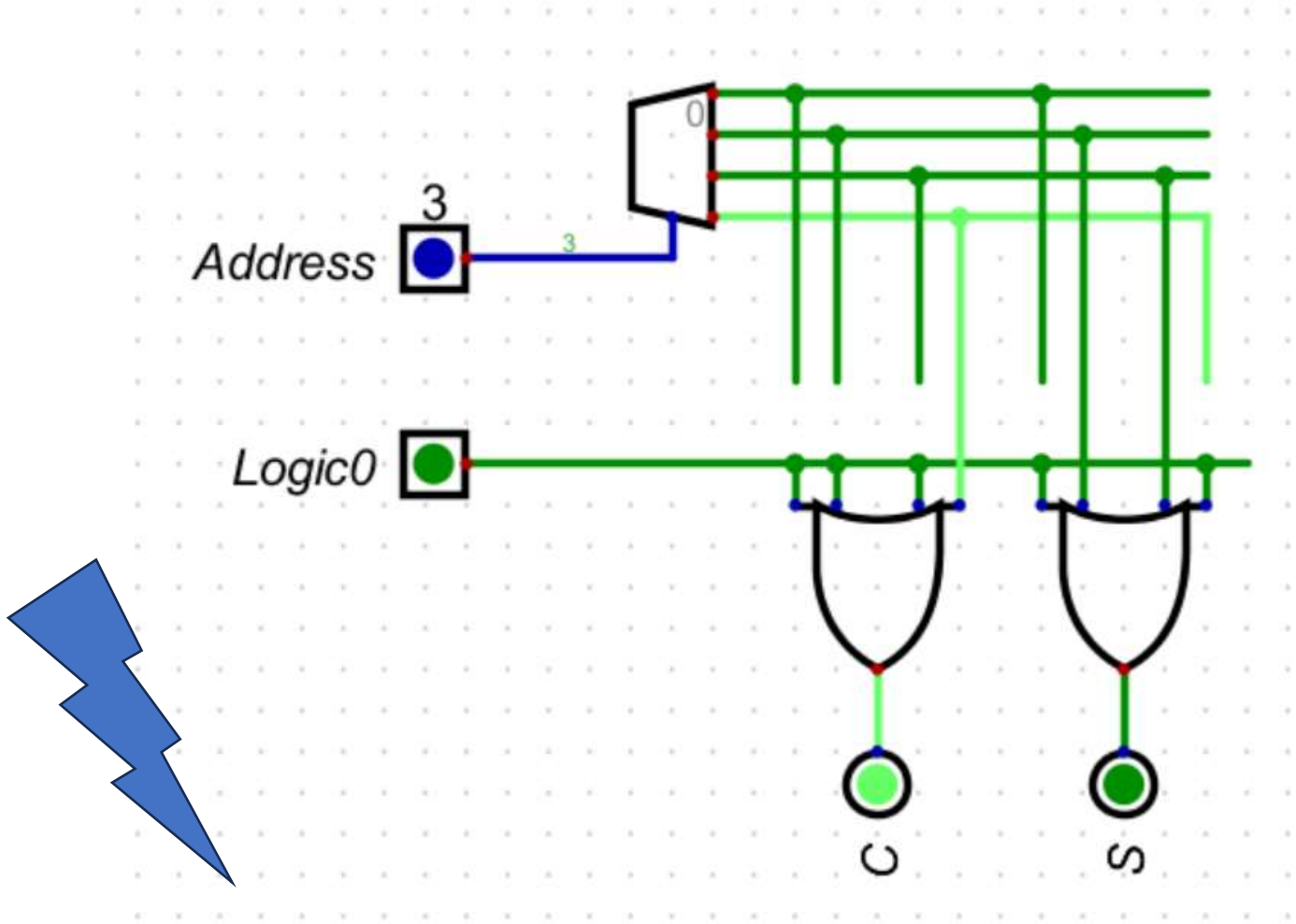


FIGURE 7.9
ROM block diagram

Read-Only Memory (ROM)

- **We can represent any combinatorial circuit with truth table MEANS that we can represent any combinatorial circuit with ROM**
- **lets have a look half adder**

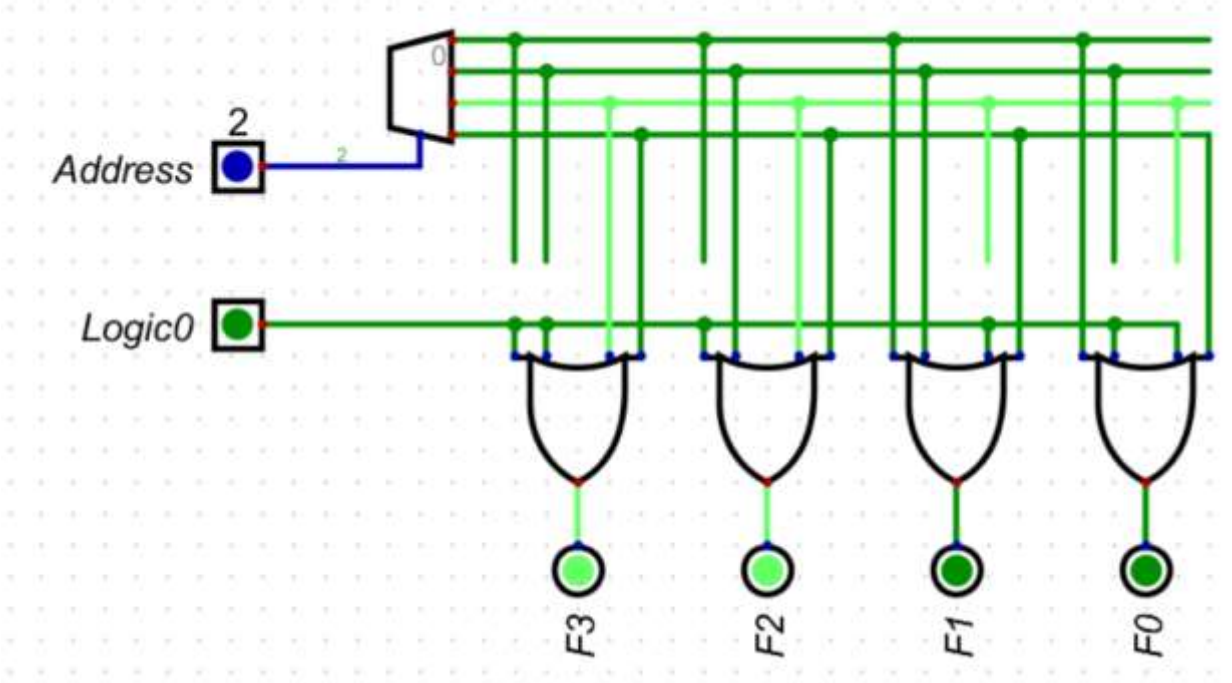
Address		Output	
A1	A0	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Read-Only Memory (ROM)

- Consider, for example, a 2 * 4 ROM. The unit consists of 2 words of **4 bits each**.
- The two inputs are decoded into 4 distinct outputs by means of a 2 * 4 decoder
- connections to OR gates performed (programmed) according to the truth table for each address line
- open connection are connected to ground (logic0)
- address is input, ROM will give output in truth table at selected address

Address		Output			
A1	A0	F3	F2	F1	F0
0	0	0	0	1	1
0	1	0	1	1	0
1	0	1	1	0	0
1	1	1	1	1	1



Read-Only Memory (ROM)

MICROCHIP

24C01C

1K 5.0V I²C™ Serial EEPROM

Features:

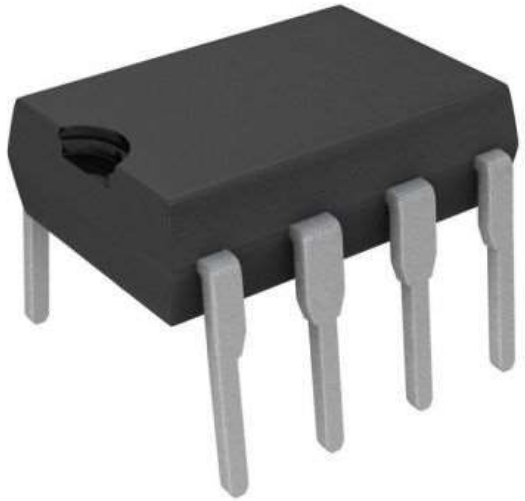
- Single Supply with Operation from 4.5V to 5.5V
- Low-Power CMOS Technology:
  - Read current 1 mA, max.
  - Standby current 5 μA, max.
- 2-Wire Serial Interface, I²C™ Compatible
- Cascadable up to Eight Devices
- Schmitt Trigger Inputs for Noise Suppression
- Output Slope Control to Eliminate Ground Bounce
- 100 kHz and 400 kHz Clock Compatibility
- Page Write Time 1 ms max.
- Self-Timed Erase/Write Cycle
- 16-Byte Page Write Buffer
- ESD Protection >4000V
- More than 1 Million Erase/Write Cycles
- Data Retention >200 Years

Description:

The Microchip Technology Inc. 24C01C is a 1K bit Serial Electrically Erasable PROM with a voltage range of 4.5V to 5.5V. The device is organized as a single block of 128 x 8-bit memory with a 2-wire serial interface. Low-current design permits operation with max. standby and active currents of only 5 μA and 1 mA, respectively. The device has a page write capability for up to 16 bytes of data and has fast write cycle times of only 1 ms for both byte and page writes. Functional address lines allow the connection of up to eight 24C01C devices on the same bus for up to 8K bits of contiguous EEPROM memory. The device is available in the standard 8-pin PDIP, 8-pin SOIC (3.90 mm), 8-pin 2x3 DFN and TDFN, 8-pin MSOP and TSSOP packages. The 24C01C is also available in the 6-lead SOT-23 package.

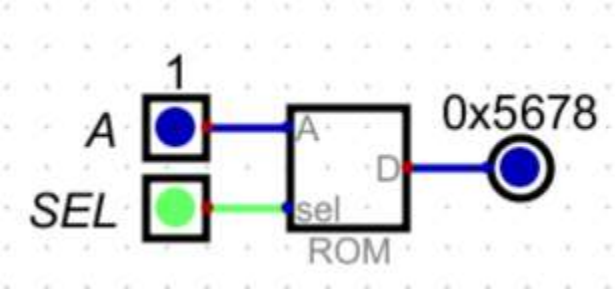
Block Diagram

the device is organized as a single block of 128 x 8-bit memory with a 2-wire serial interface.



Read-Only Memory (ROM)

- A read-only memory (ROM) is essentially a memory device in which permanent binary information is stored
- it stays within the unit even when power is turned off and on
- electrically erasable PROM (EEPROM). programmed data can be erased with an electrical signal
- Flash memory devices are similar to EEPROMs, but have additional built-in circuitry to selectively program and erase the device in-circuit, without the need for a special programmer



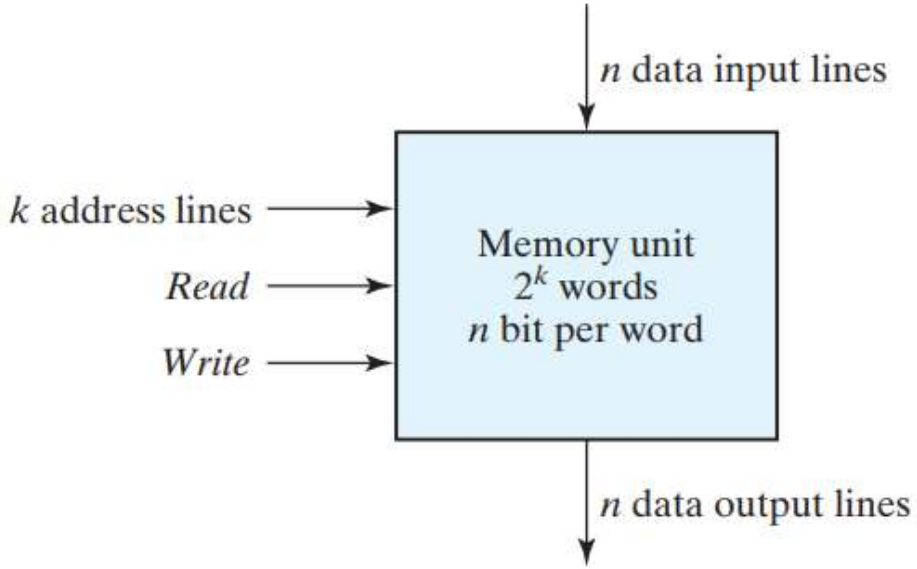
Data

File

Address	0x0000	0x0001	0x0002	0x0003	0x000
0x0000	0x1234	0x5678	0x9999	0	
0x0010	0	0	0	0	
0x0020	0	0	0	0	

Random Access Memory (RAM)

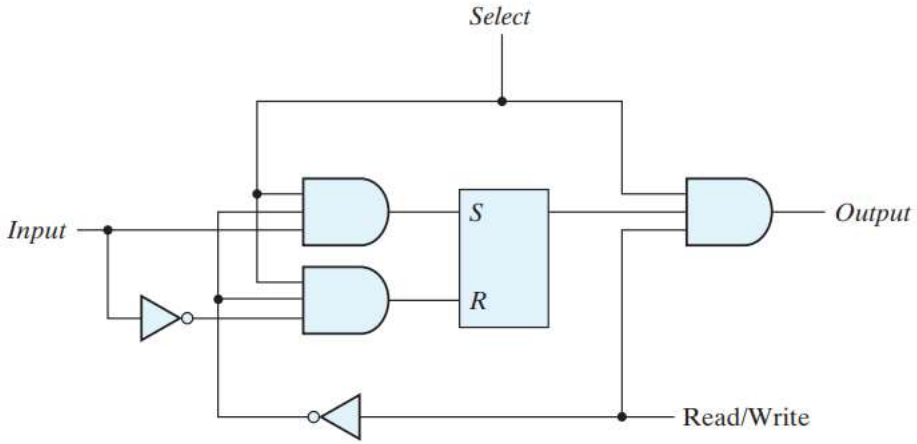
- A memory unit is a collection of storage cells, together with associated circuits needed to transfer information **into and out of a device**.
- **time** it takes to transfer information to or from any desired random location is **always the same**—hence the name random-access memory, abbreviated RAM
- A memory unit stores binary information in groups of bits called **words**. A group of 8 **bits is called a byte**.
- Most computer memories use words that are multiples of 8 bits in length. Thus, a 16-bit word contains two bytes, and a 32-bit word is made up of four bytes. The capacity of a memory unit is usually stated as the **total number of bytes**



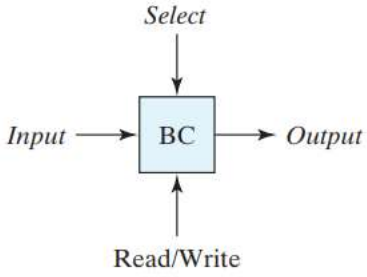
Memory address		Memory content
Binary	Decimal	
0000000000	0	1011010101011101
0000000001	1	1010101110001001
0000000010	2	0000110101000110
⋮	⋮	⋮
1111111101	1021	1001110100010100
1111111110	1022	0000110100011110
1111111111	1023	1101111000100101

Random Access Memory (RAM)

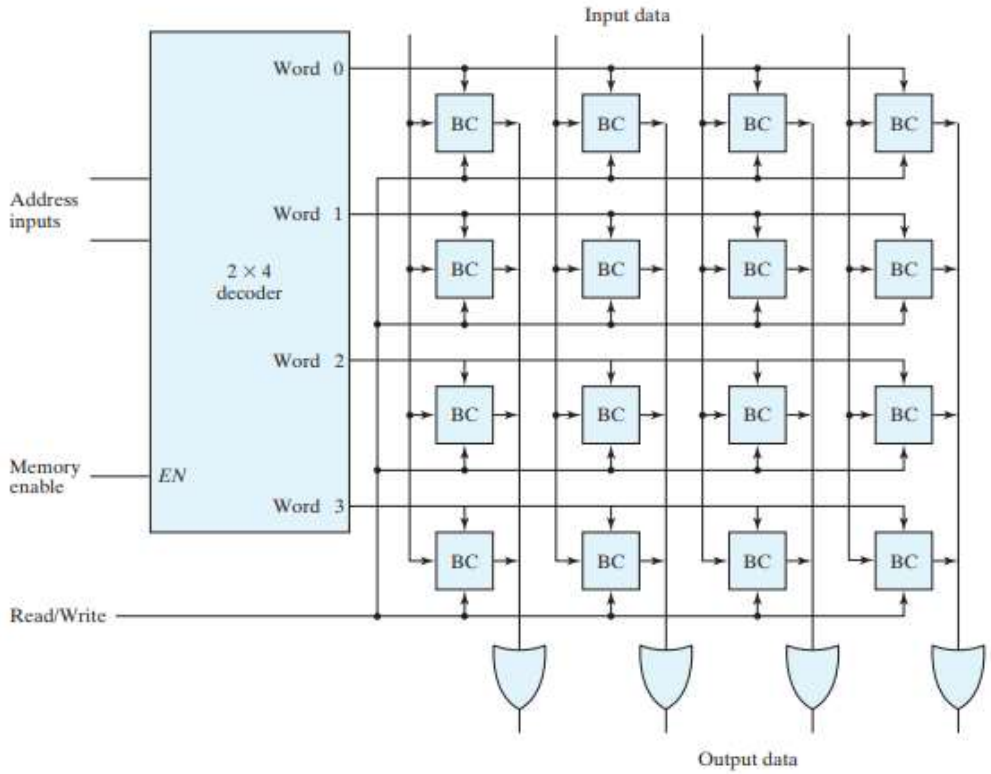
- This RAM consists of four words of four bits each and has a total of 16 binary cells.
- The small blocks labeled BC represent the **binary cell** with its three inputs and one output
- A memory with four words needs two address lines. The two address inputs go through a 2 * 4 decoder to select one of the four words
- When the memory enable is 0, all outputs of the decoder are 0 and none of the memory words are selected
- During the read operation, the four bits of the selected word go through OR gates to the output terminals



(a) Logic diagram

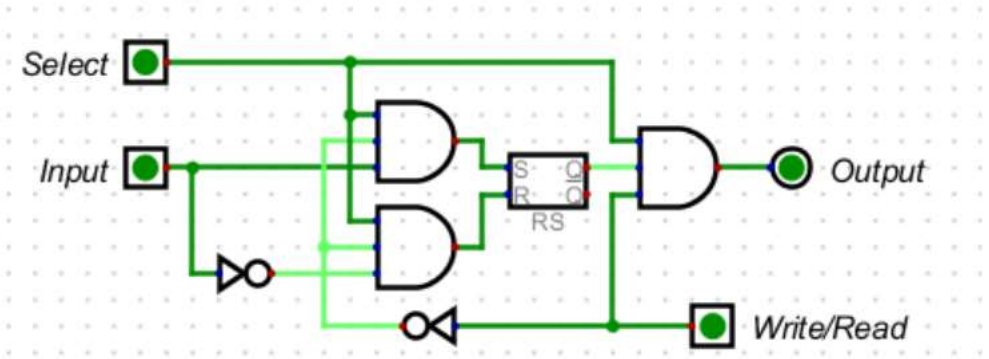


(b) Block diagram



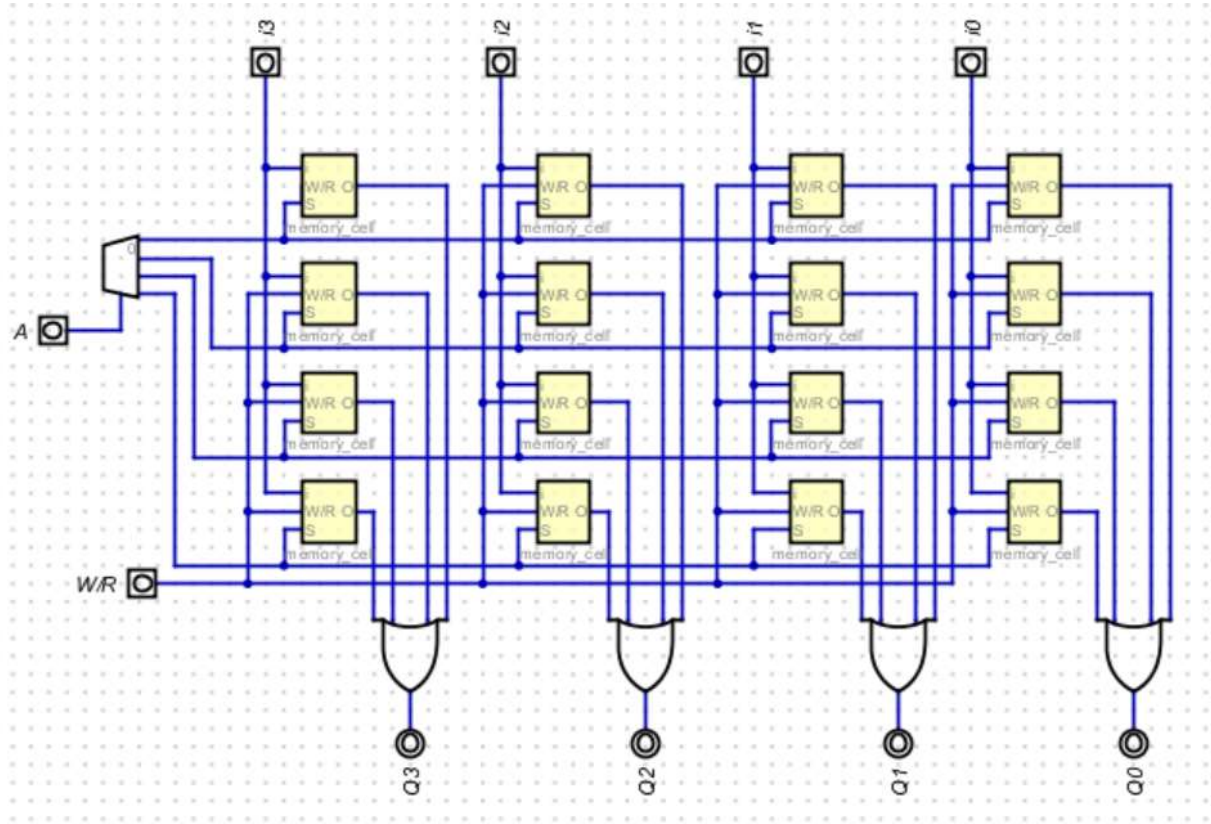
Random Access Memory (RAM)

- memory cell is 1 bit register



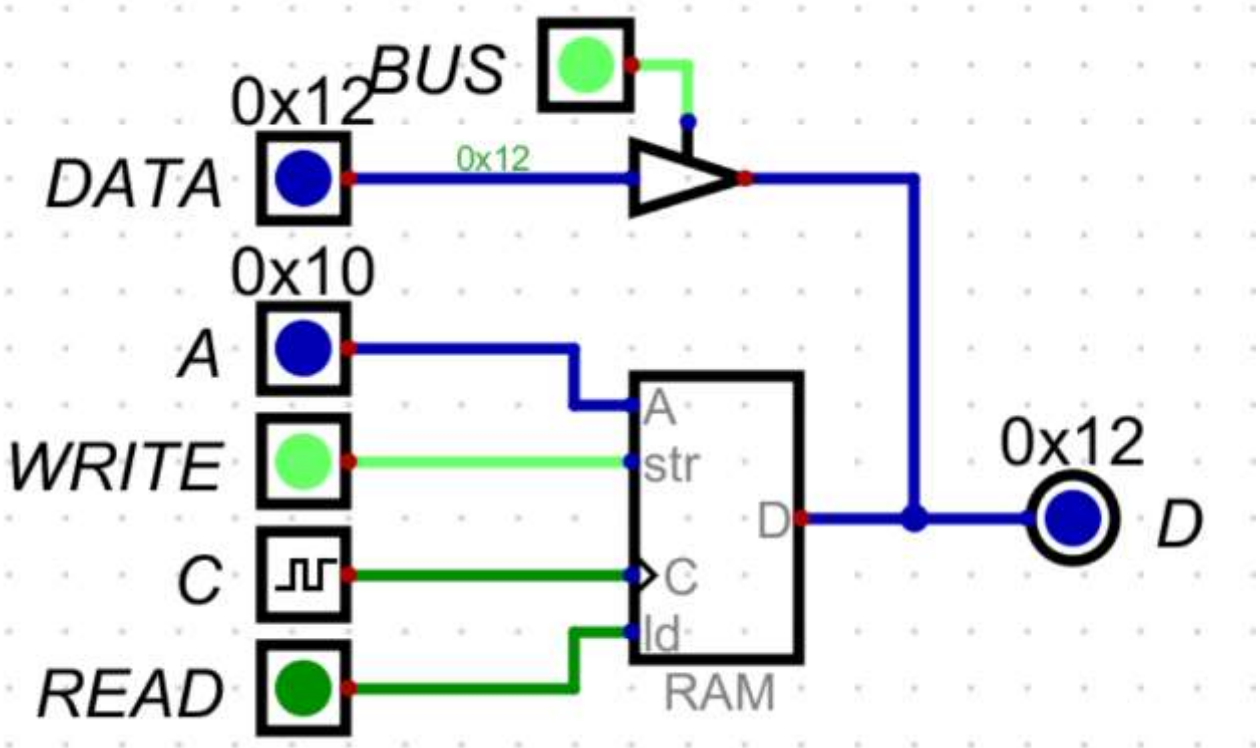
Random Access Memory (RAM)

- 4x4 RAM
- Static RAM (SRAM) consists essentially of internal latches that store the binary information. The stored information remains valid as long as power is applied to the unit.
- Dynamic RAM (DRAM) stores the binary information in the form of electric charges on capacitors provided inside the chip by MOS transistors.



Random Access Memory (RAM)

- **We can read or write to RAM, giving Adress and Data inputs**
- **RAM is empty power on**
- **write: load data and adress, at clock cycle it will write data to the adress**
- **read: reads the adress transfers to the output**

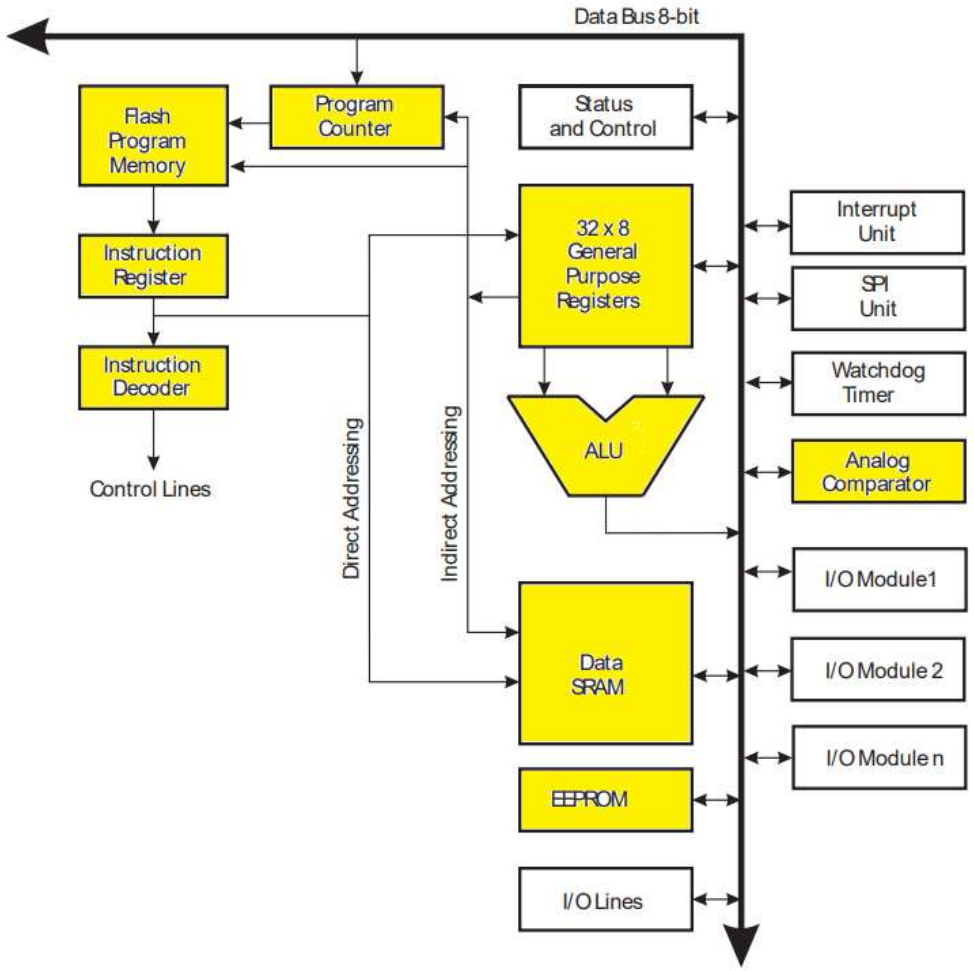


RAM

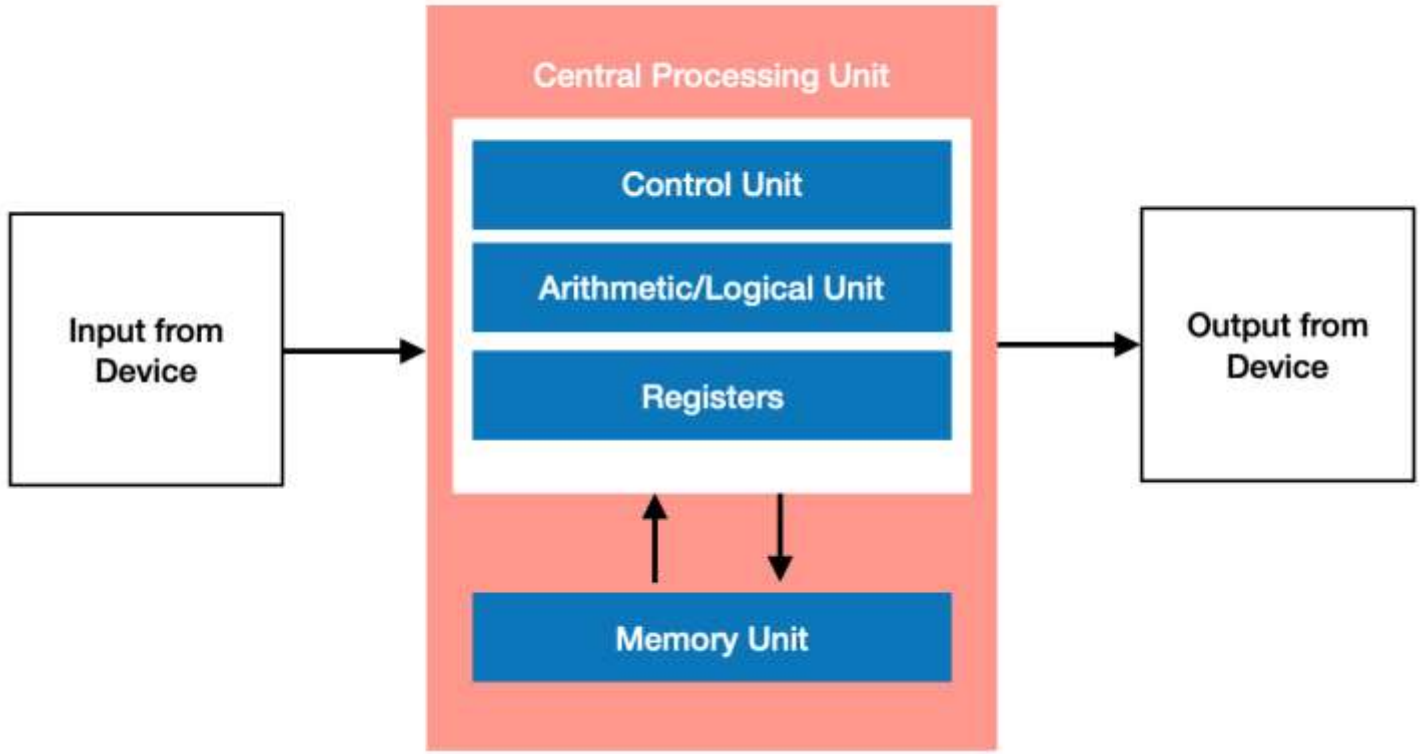
Address	0x0000	0x0001	0
0x0000	0	0	
0x0010	0x12	0	
0x0020	0	0	
0x0030	0	0	

Arduino MEGA Microcontroller (MCU)

- **We almost covered boxes of an Arduino MEGA ATMEGA640 MCU**
- **NOW we can design our own microcontroller**

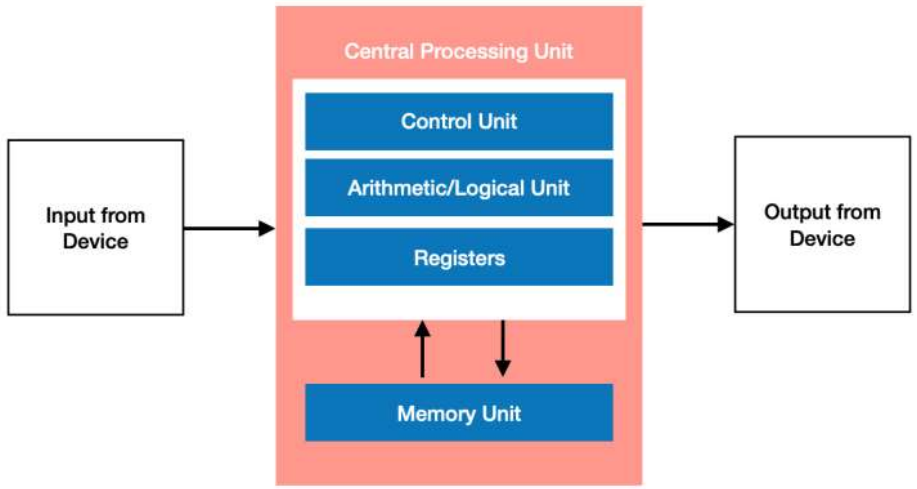


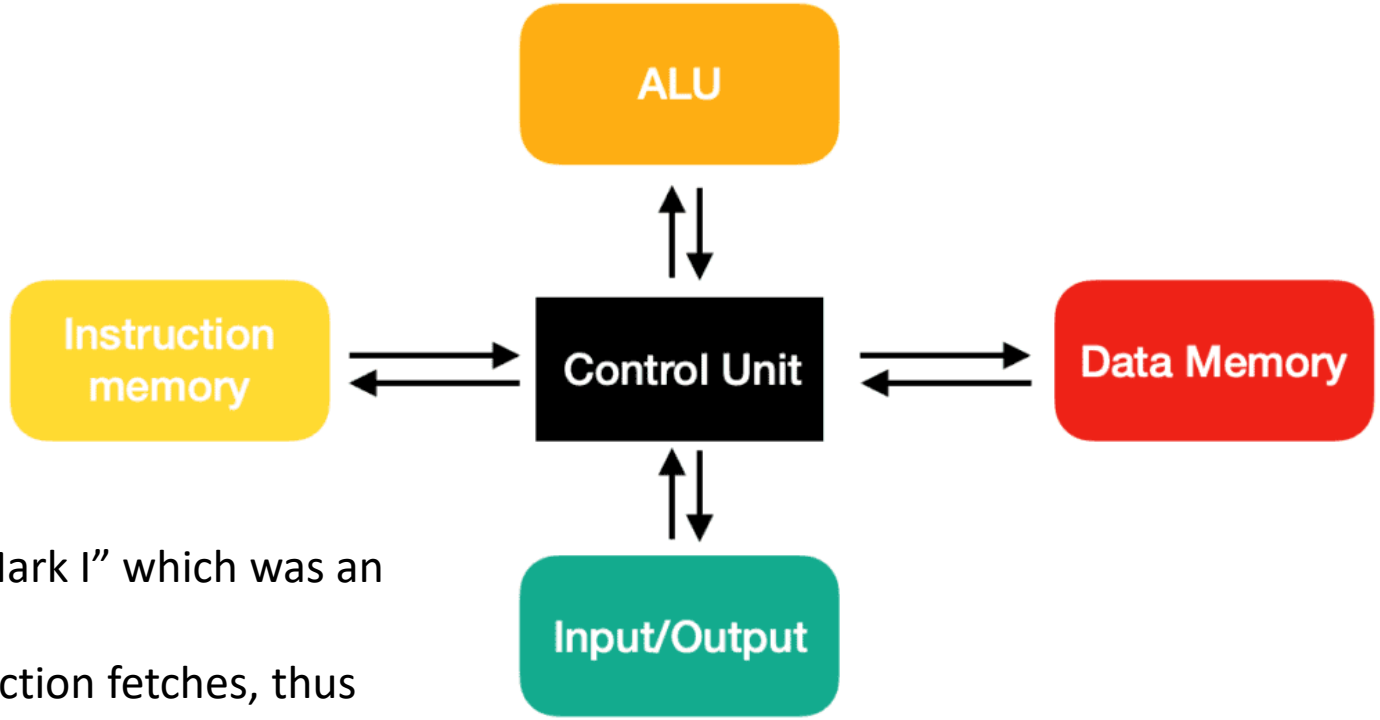
Microcontroller (MCU)



The **von Neumann** architecture—also known as the von Neumann model or Princeton architecture—is a computer architecture based on a 1945 description by John von Neumann, and by others, in the First Draft of a Report on the EDVAC.

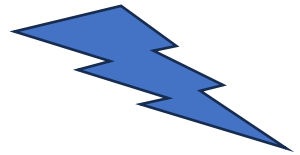
Microcontroller (MCU): Harvard Architecture





- Harvard architecture is named after the “Harvard Mark I” which was an IBM computer in the University of Harvard
- two buses, one for data transfers and one for instruction fetches, thus faster

EEE213 MCU DESIGN

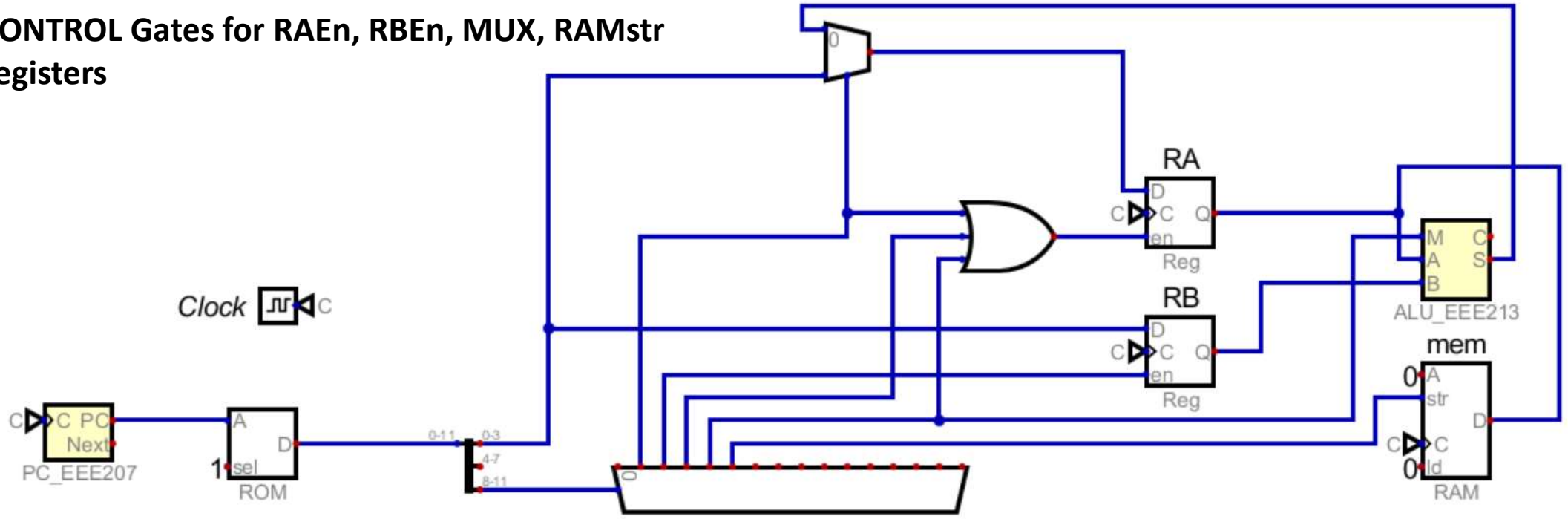


- **6 INSTRUCTIONS (4bit), 4bit DATA/ADDRESS**
- **MSB 6-bits for instructions and LSB 4 bits for DATA/ADRESS**
- **Mod, MUX, enA, enB, str (5bits) has to be controlled**
- **STS, store register value inside RAM address**

INS	ROM ABCD 0000 XXXX	FUNCTION	Mod	MUX	enA	enB	str
LDA	0001 0000 0011=0x103	LDA 3	x	1	1	0	0
LDB	0010 0000 0010=0x202	LDB 2	x	x	0	1	0
ADD	0011 0000 0000=0x300	A+B=3+2	0	0	1	0	0
ADD	0011 0000 0000=0x300	A+B=5+2	0	0	1	0	0
SUB	0100 0000 0000=0x400	A-B=7-2	1	0	1	0	0
STS	0101 0000 0000=0x500	STS 0x00 RA=5	x	x	0	0	1
NOP	0000 0000 0000=0x000		x	x	0	0	0

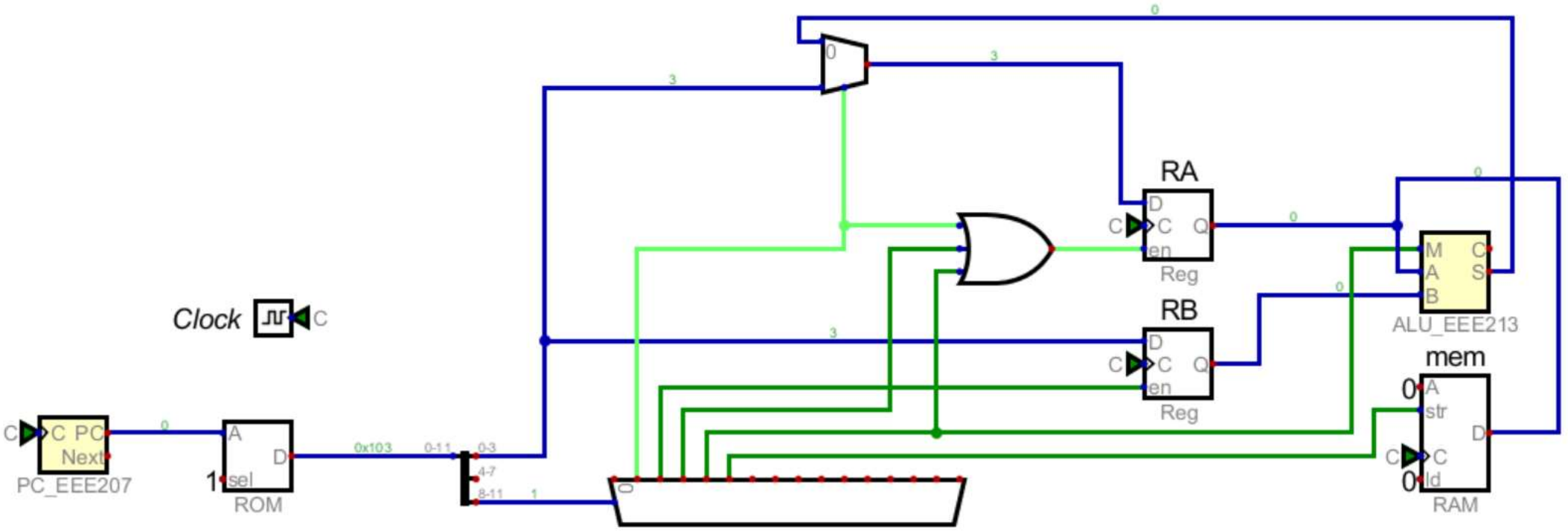
EEE213 MCU DESIGN

- **General Architecture of MCU**
- **Program Counter PC**
- **ROM**
- **LOGIC CONTROL Gates for RAEn, RBEn, MUX, RAMstr**
- **RA,RB registers**
- **ALU**
- **RAM**



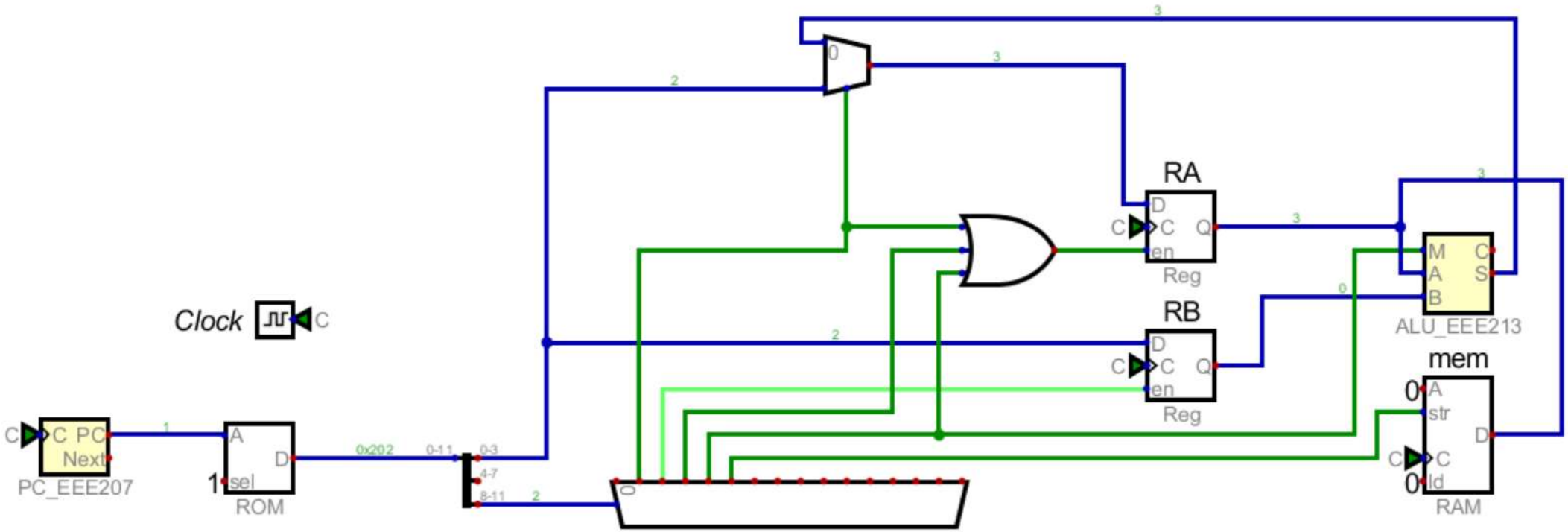
EEE213 MCU DESIGN

- **TURN ON**
- **PC=0, ROM address 0 is 0x103 loaded**
- **RAEn=1, Mux=1 ready to take the value into RA in next cycle**



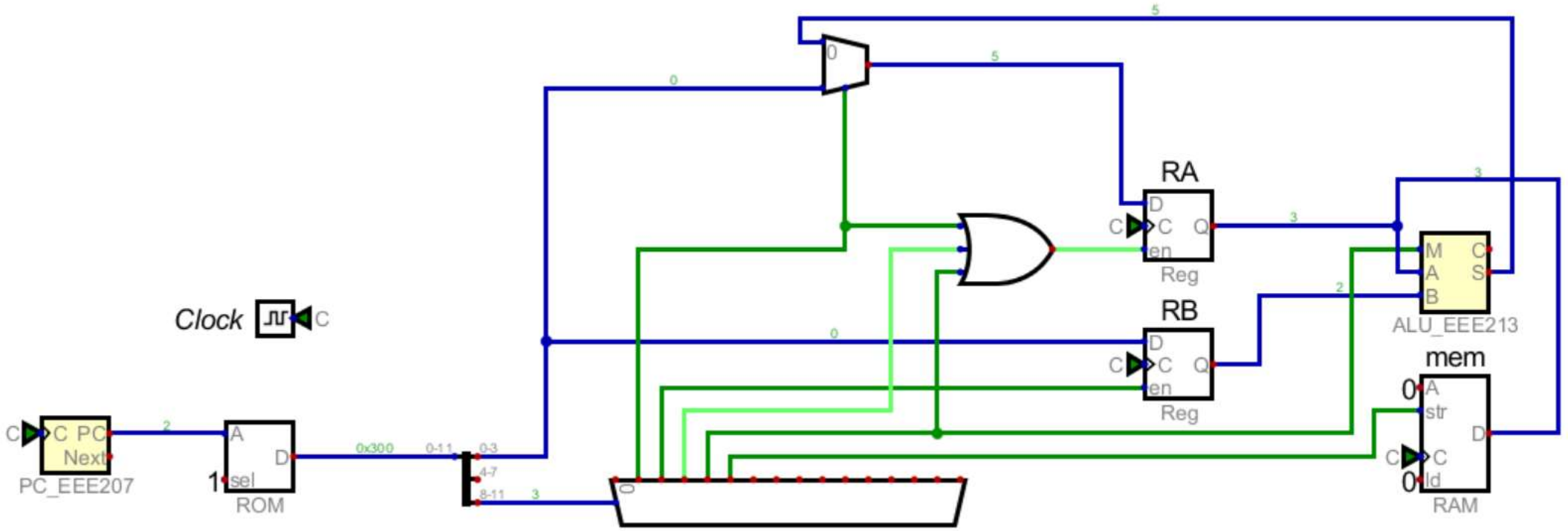
EEE213 MCU DESIGN

- **1st clock cycle**
- **PC=1 ROM address 1 is 0x202 loaded**
- **RA=3 loaded**
- **ALU=3+0=3**
- **RBEn=1, ready to take the value into RB in next cycle**



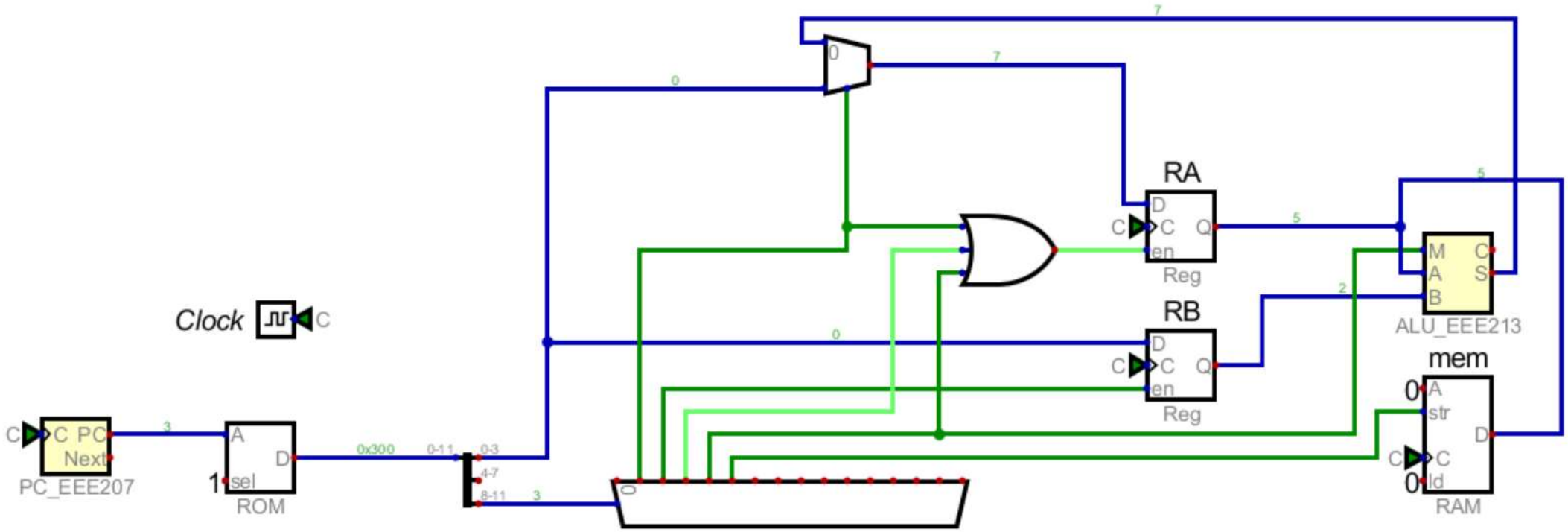
EEE213 MCU DESIGN

- **2st clock cycle**
- **PC=2 ROM address 2 is 0x300 loaded**
- **RB=2 loaded**
- **ALU=3+2=5**
- **RAEn=1, ready to take the ALU value into RA in next cycle**



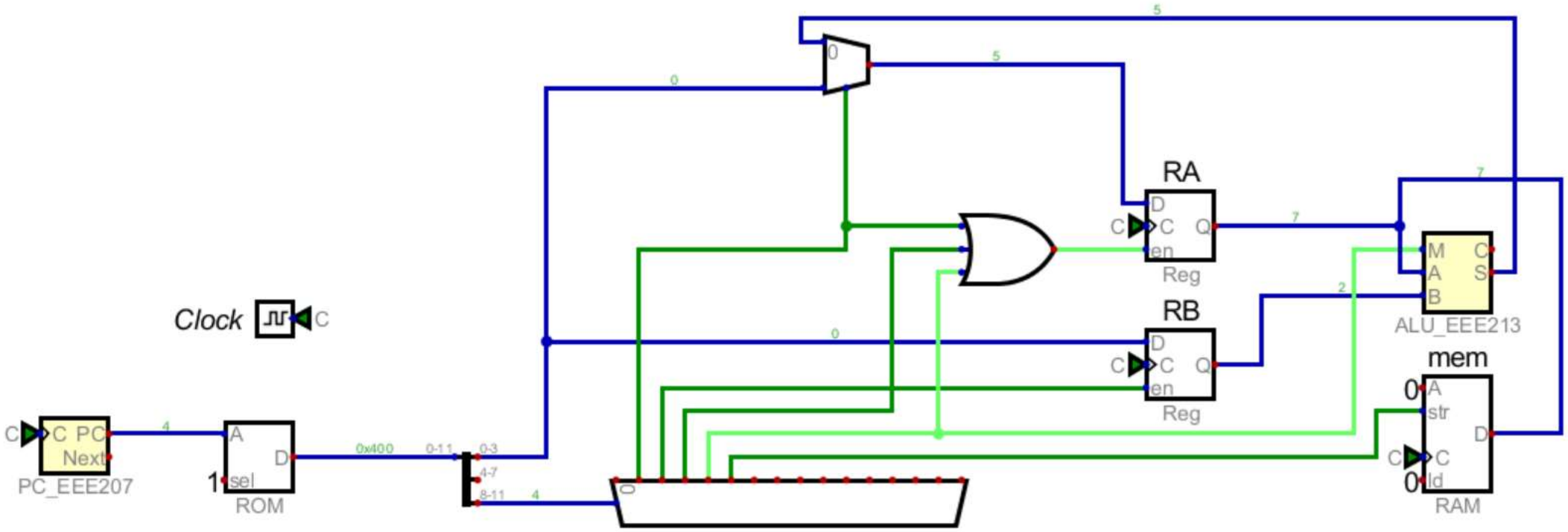
EEE213 MCU DESIGN

- **3rd clock cycle**
- **PC=3 ROM address 3 0x300 loaded**
- **ALU=5+2=7**
- **RAEn=1, ready to take the ALU value into RA in next cycle**



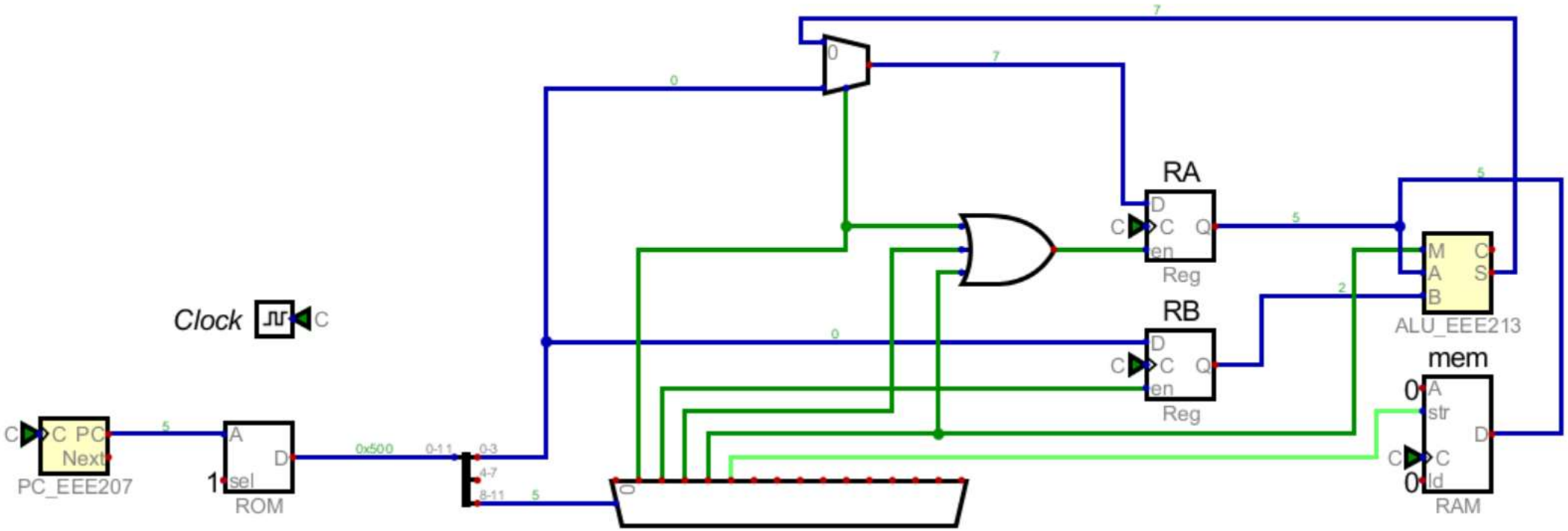
EEE213 MCU DESIGN

- **4th clock cycle**
- **PC=4 ROM address 4 0x400 loaded**
- **ALU=7-2=5**
- **RAMstr=1, ready to take RA the value into RAM in next cycle**



EEE213 MCU DESIGN

- **5th clock cycle**
- **PC=5 ROM address 5 0x500 loaded**
- **RAM address=0x0 loaded with RA=5 in next cycle**

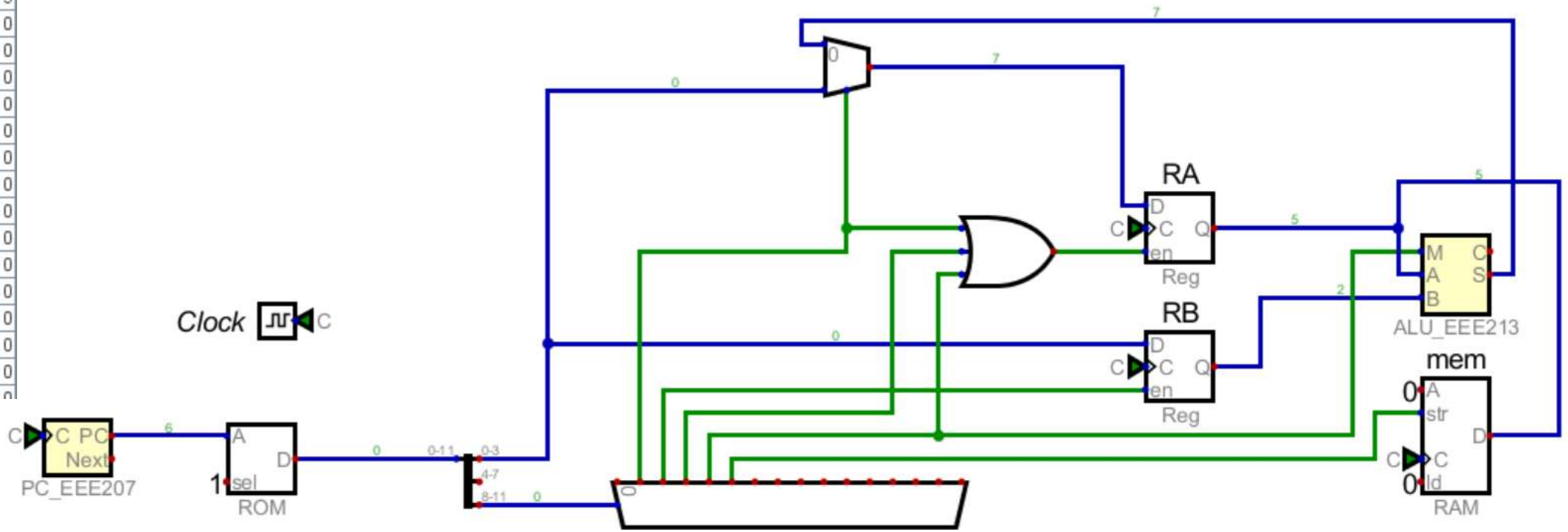


EEE213 MCU DESIGN

- **6th clock cycle**
- **PC=6 ROM address 5 0x0 loaded**
- **RAM address=0x0 loaded with RA=5**

mem

Address	Value
0x0	5
0x1	0
0x2	0
0x3	0
0x4	0
0x5	0
0x6	0
0x7	0
0x8	0
0x9	0
0xA	0
0xB	0
0xC	0
0xD	0
0xE	0
0xF	0



EEE213 MCU DESIGN

- **Similar instructions in Arduino Assembly Language**

```
volatile byte a=0;
void setup() {
  Serial.begin(9600);
  asm (
    "ldi r26, 3  \n"
    "ldi r27, 2  \n"
    "add r26, r27 \n"
    "add r26, r27 \n"
    "sub r26, r27 \n"
    "sts (a), r26 \n"
  );
  Serial.print("a = ");
  Serial.println(a);
}
void loop() { }
```

```
1
2  // Arduino ASM example
3  // Senol Gulgonul
4  volatile byte a=0;
5  void setup() {
6    Serial.begin(9600);
7    asm (
8      "ldi r26, 3  \n"
9      "ldi r27, 2  \n"
10     "add r26, r27 \n"
11     "add r26, r27 \n"
12     "sub r26, r27 \n"
13     "sts (a), r26 \n"
14   );
15   Serial.print("a = ");
16   Serial.println(a);
17 }
18 void loop() { }
19
20
```

Serial Monitor

```
a = 5
```