Digital Logic Module

Fifth Edition

School of Computing
Faculty of Engineering

UNIVERSITI TEKNOLOGI MALAYSIA

Printed @ 2018

Table of Contents

Modules	Title	Page
	Preface	
1	**Digital Logic Overview**	1
	• Digital and Analog Quantities • Binary Digits, Logic Levels and Digital Waveform • Introduction to Logic Operations • Overview of Logic Functions • Fixed-Function IC • Programmable Logic Devices (PLD)	
2	**Number Systems and Codes**	49

	• Numbering System (Decimal, Binary, Octal and Hexadecimal)
	• Number Conversion between Bases
	• Binary to Octal and Hexadecimal Conversion
	• Codes: BCD, GRAY, Parity, ASCII Arithmetic
	• Operations:
	  ○ Integer (Unsigned, Signed Number)
	  ○ Addition and Subtraction	

3	**Logic Gates : Overview**	89
	• NOT Gate (Inverter) • AND Gate • OR Gate • NAND Gate • NOR Gate • Exclusive-OR (XOR) Gate • Exclusive-NOR (XNOR) Gate	
4	**Boolean Algebra and Logic Simplification**	101

	• Laws and Rules of Boolean Algebra
	• DeMorgan's Theorem
	• Combinational Logic Representation
	  ○ Boolean to Logic Circuit
	  ○ Logic Circuit to Boolean
	  ○ Boolean to Truth Table
	  ○ Logic Circuit to Truth Table
	• Simplification Using Boolean Algebra
	• Standard Forms of Boolean Expressions (SOP and POS Form)
	• Karnaugh Map (K-Map)
	• K-Map Minimisation (SOP and POS)
	• Don't Care Conditions	

5 **Combinational Logic Circuit** 151

- Basic Combinational Logic Circuits (AND-OR, AND-OR-INVERT, XOR, XNOR)
- Universal Property of NAND and NOR
- Dual Symbol Concept
- Design a Combinational Circuit

6 **Functions of Combinational Logic** 175

- Basic and Parallel Binary Adders
- Comparators
- Decoders
- Encoders
- Multiplexer
- Demultiplexer
- Code Converter
- Parity Generator/ Checker

7 **Latches and Flip-Flops** 207

- Introduction to Latch and Flip-Flop
- Latches (SR, Gated SR, Gated D)
- Flip-Flops (SR, JK, D, T)

8 **Counters** 237

- Types of Sequential Circuits
- Counters
- Design and Analysis of Asynchronous Counter
- Operation (Up/Down, Truncated)
- Asynchronous Counter Decoder
- Flip-flop Excitation Table
- Design and Analysis of Synchronous Counter (Up/Down, Truncated)
- Counter for Arbitrary Sequences
- Cascaded Counter
- Analysis of Sequential Circuits

9 **Shift Registers** 283

- Basic Shift Register functions
- Shift Register:
  - SISO, SIPO, PISO, PIPO
- Bidirectional Shift Registers
- Shift Register counters
  - Ring Counter
  - Johnson Counter

Tutorial Questions 303

Tutorial Answers 333

Labs	Title	Page
	Introduction	345
1	**Combinational Logic**	351
2	**Combinational Logic Circuit Design Simulation**	359
3	**Synchronous Digital Counter**	377
4	**[Mini Project] Programmable Logic Device:** Photocopying (Xerox) Machine	385
	Lab References	399
	Lab References Sheet	409

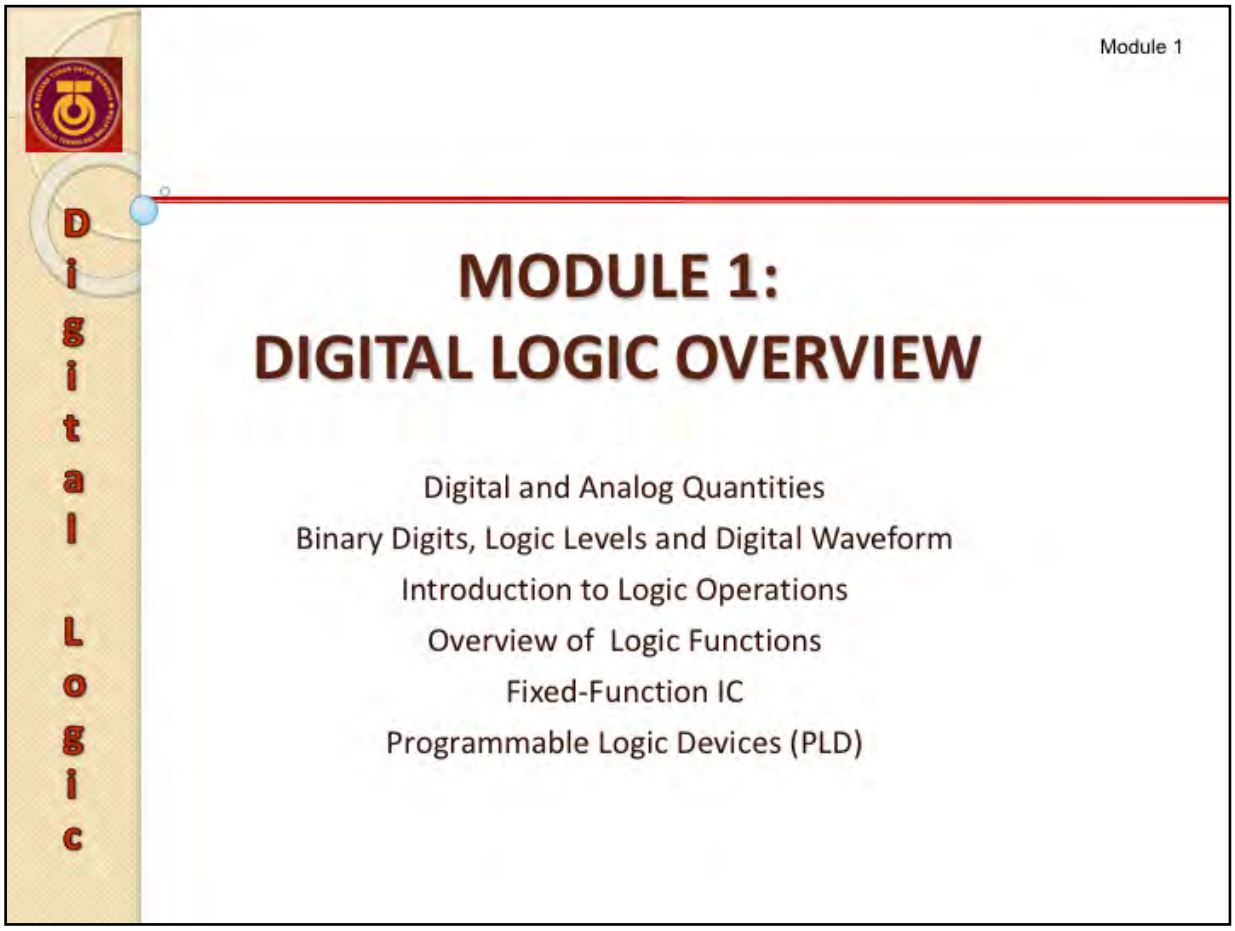
Preface

Digital logic module is a first year student lecture notes taking digital logic course in the School of Computing (SC), Faculty of Engineering, Universiti Teknologi Malaysia (UTM). This module is divided into two main parts, the first part covers combinational logic design and the second part includes a memory element. The first part consists of an introduction to digital logic, number systems and codes, logic basic gates, Boolean algebra and logic simplification, combinational circuit design logic and functional logic combination.

On the other hands the second part covers the introduction to the latch and flip-flops, design counter, and shift registers. Content modules are adapted from some of the contents of the digital logic used by most university students. Among them are *Floyd, T., "Digital Fundamentals"*, *Prentice Hall, USA,* and *Tocci, RJ, "Digital Systems"*, *Prentice Hall, USA*. In addition, the experience of the lecturers who have taught this course in the SC has been taken into account to produce the digital logic teaching modules.

Special appreciation is given mainly to the major contributor to the success of this module, Mr. Abd. Bahrim Yusoff, Assoc. Prof Dr. Mazleena Salleh, Dr. Mohd Fo'ad Rohani, Dr. Ismail Fauzi Isnin, Dr. Raja Zahilah, Mr. Muhalim Mohamed Amin, Ms. Marina Md Arshad, Ms. Rashidah Kadir and Mr. Firoz Yusuf Patel Dawoodi.

6 January 2019
29 Rabiul Akhir 1440h



There are two parts in this module that are:

Part 1 – Introduction to Analog and Digital Systems

Part 2 – Introduction to Digital Operation and Functional Systems

Overall objectives of this module are:

1 To discuss definitions and terms that used in Analog and Digital Systems.

2 To illustrate basic operation of digital logic systems.

3 To discuss technologies of digital Logic ICs and PLDs.

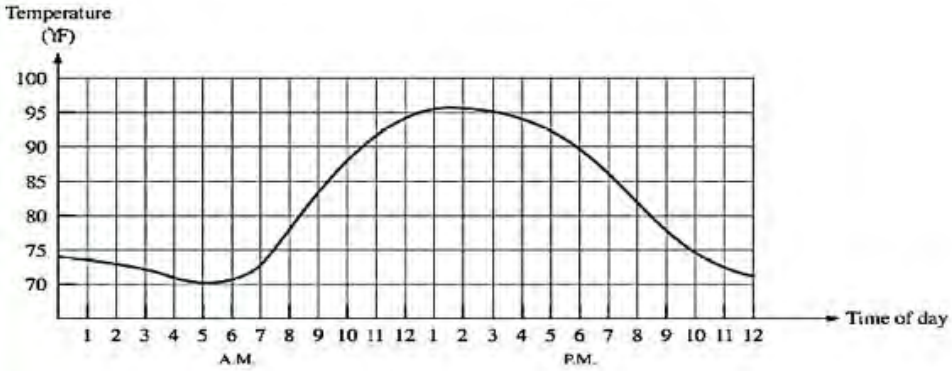
Digital Logic

Digital and Analog Quantities

Definition: Analog & Digital

Digital Logic

- **Analog**
  - Varies over a continuous range of values
    - a variable signal continuous in both time and amplitude
  - Examples of analog quantities : time, pressure, sound.



Example of equations that represent temperature change in analog signal:

$$T = f(t)$$

$$Total\ T = \text{integration of } f(t) \text{ from interval 1s to 1000s}$$

$$= \int_{t=1}^{t=1000} f(t)dt$$

At time = 2:30pm, temperature is equal to 95^0C

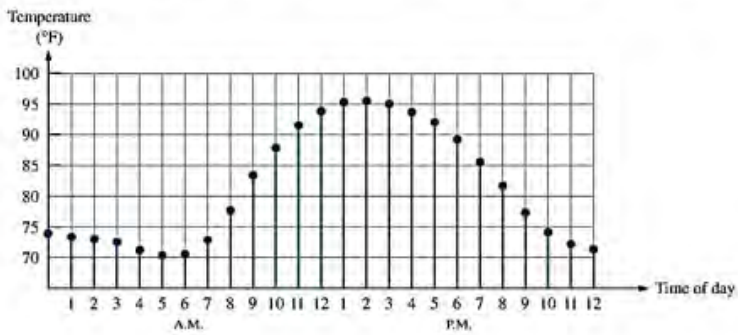
Example of analog quantities measurement (others):

- Time (t)= 10.16s (second)
- Pressure (P)=220.10KPa (Kilo Pascal)
- Energy, Power = 100.5KW (Kilo Watts)

… Definition:

Digital Logic

- **Digital**
  - A discrete (that is, discontinuous) set of values.
  - Varies in discrete (separate) steps.
    - The word *digital* comes from the same source as the word digit and *digitus* (the Latin word for *finger*), as fingers are used for discrete counting.



Example of equations that represent temperature in discrete signal:

$$T_i = \{T(at)\} \ , \ a = 1,2,3,4,\ldots,N$$

$$Total \ T = \sum_{i=1}^{n} T_i \ , i = 1,2,3,4,\ldots,n$$

Example of discrete system measurement:

At time = 2pm, temperature is equal to 96^0C

At time = 2:30pm, there is no measurement

Example of digital system measurement (others):

- Band Width (BW) = 1Mbps (Mega Bits Per Second)
- Storage RAM = 512MB (Mega Byte)
- Hard Disk = 160GB (Giga Byte)

Analog vs Digital

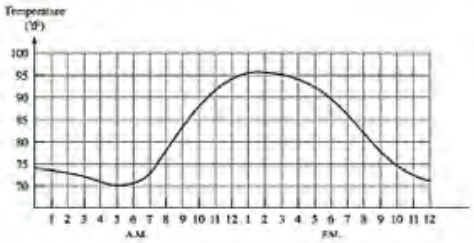
Digital Logic

Analog

- Use base 10 (decimal)
- Represented by 10 different level: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.
- Analog system: A combination of devices that manipulate values represented in analog form

Digital

- Use base 2 (binary)
- Represented by 2 different level: 0 and 1 or low and high.
- Digital system: A combination of devices that manipulate values represented in digital form.



Example of practical analog and digital Systems:

Analog system

Analog Watch

Digital system

Digital Watch

Analog Phone

Mobile Digital Phone

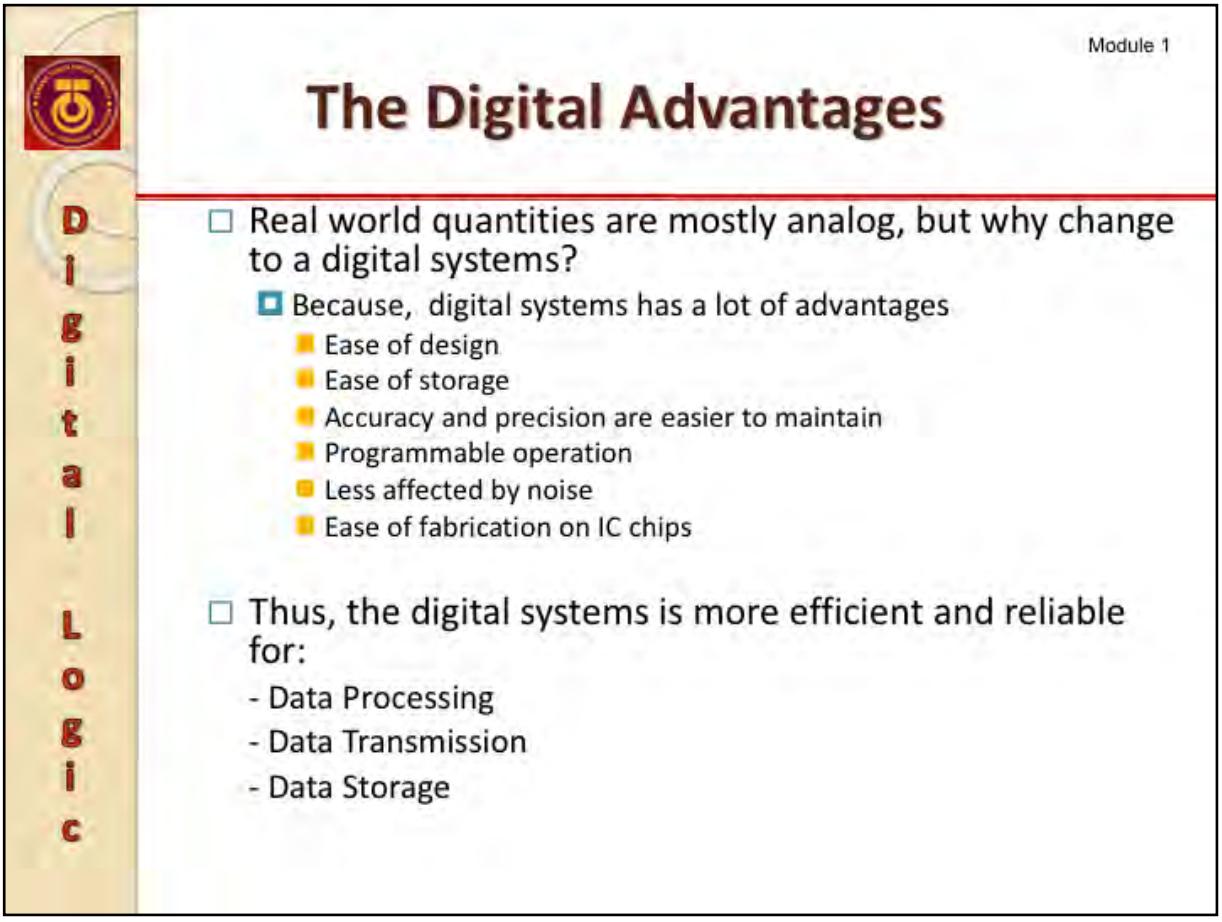
Digital

Digital Logic

- Digital technology is relatively new compared to analog technology, but a lot of analog systems has been changed to a digital systems, Examples:
  - Computers
  - Manufacturing systems
  - Medical Science
  - Transportation
  - Entertainment
  - Telecommunications

Digital Application Systems in real world:

- Computers: CPU and Microprocessors, memory storage
- Manufacturing systems: Micro-controller to control robots for product packaging and conveyor systems
- Medical Science: Medical Imaging Systems that includes digital x-ray, computed tomography (CT), mammography
- Transportation: Speed Micro-Controller, Vehicle Air-Condition micro-controller, Portable Fault Diagnosis Device (trouble shooting)
- Entertainment: Digital television (DTV), DVD, Digital Camera, Media Player
- Telecommunications: Digital phone, digital hand-phone, Digital FM transmitter (i-pod, MP3), **A**symmetric **D**igital **S**ubscriber **L**ine (ADSL Digital Modem) - a new technology that allows a single regular telephone line to serve for normal phone calls and digital data at the same time.

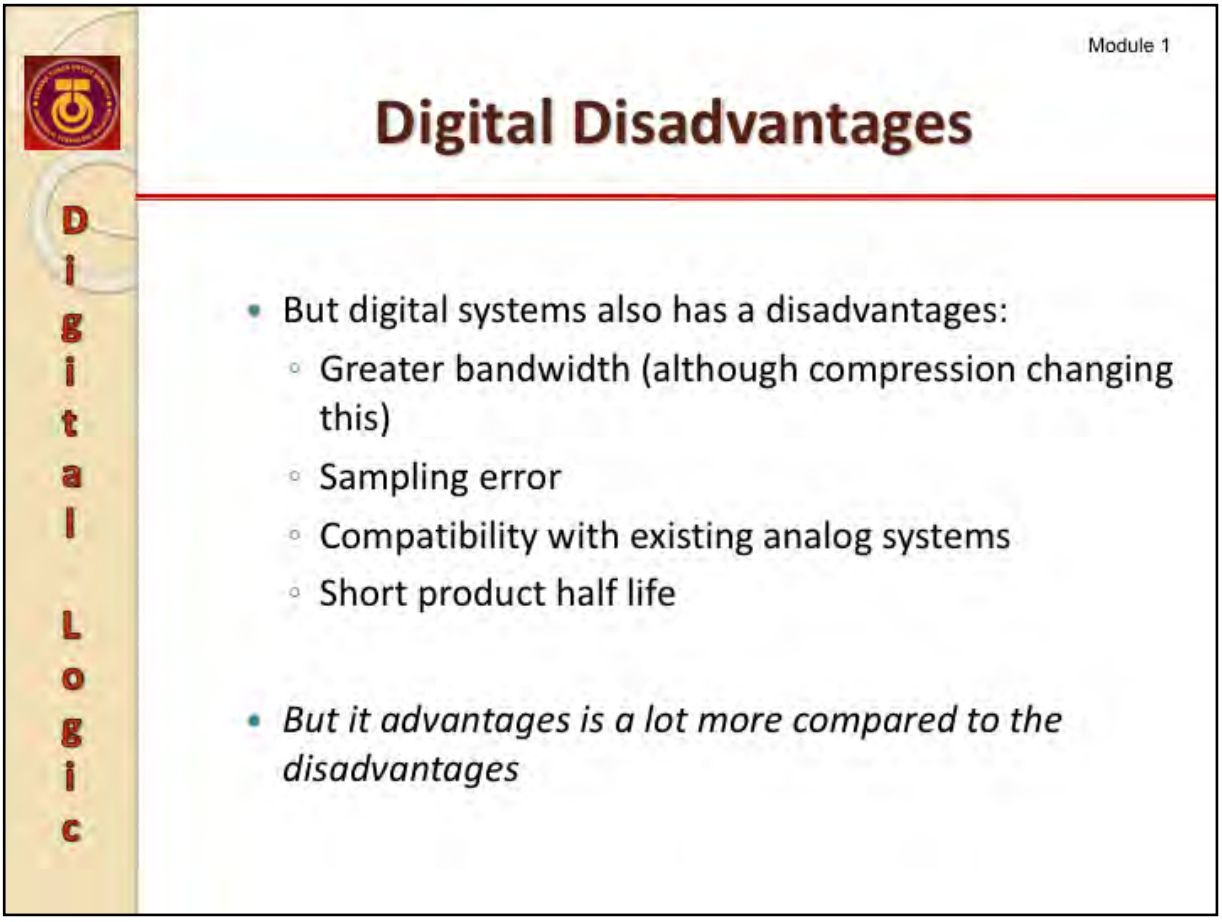


Why digital systems is more preferable?

- Ease of design: only consider 2 state's of voltage level (high or low)
- Ease of storage: we have digital memory device that enable us to easily store bit 0 and 1. Analog does not has any device with equivalent capability.
- Accuracy and precision are easier to maintain: bit data representation is more accurate and easy to maintain
- Programmable operation: Many digital IC circuit can re-programmed to construct different circuit using e.g. using FPGA (Field Programmable Gate Array) technology. Example of research application area includes reconfigurable computing. Unlike analog system is fixed and unchanged from factoring manufacturing.
- Less affected by noise: binary data is representing by range of voltage level to represent high or low. In the middle, between high and low level range, binary signal does not exist. So, it is less affected by noise. Compared to analog system is more affected by noise.
- Ease of fabrication on IC chips : Digital circuit is easy fabricated using silicon technology, which is cheaper and easy to produce.

Example of Practical process that implement digital systems:

- Data Processing: Computer technology that based on binary system is invented to process binary data of bit 0 and 1 in at high speed without compromising the accuracy of the result. Eg. Dual-Core processor at 2GHz speed.
- Data Transmission: Digital data can be transmitted using existing analog telephone line by using modem, or digital data can be transmitted directly without converting to analog signal e.g. by using ISDN (Integrated Service Digital Network) line.
- Data Storage: Digital storage currently offer a huge size capacity e.g. 120GB, 320GB, 512GB, 1TB at low price.



Some drawbacks of Digital Systems:

- Greater bandwidth: Digital TV (HDTV) vs Analog TV (Carrier MHz))

  *A High Definition Television (*HDTV) signal required much more bandwidth (20 MHz) compared to a standard (National Television System Committee) NTSC analog signal (6 MHz)

- Sampling error* – Analog signal will undergo Quantization process before being convert to digital.

  During the quantization process, information loss could occur in the process rounding the signal to integer (e.g rounding from 12.4V or 12.2V will give same result to the nearest integer 12)

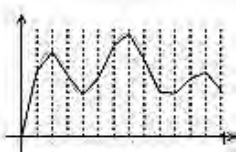
- Compatibility with existing analog systems

  Digital TV (e.g. HDTV – is not compatible with existing transmitting existing signal that use analog carrier. So, new capital investment is needed to provide the digital infrastructure)

- Short product half life

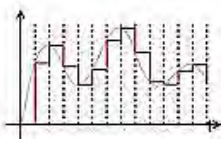
  An analog record player has longer life cycle compared to digital player that usually has shorter product half-life. Due to rapid digital technology improvement , the audio system technology has evolved in relatively short period of time. For instance, from Laser Disc change to VCD technology before improving to DVD technology and recently to Blue Ray Disc.

* Sampling Error (Quantization Error): is derived from Analog to Digital Conversion Process:



Analog

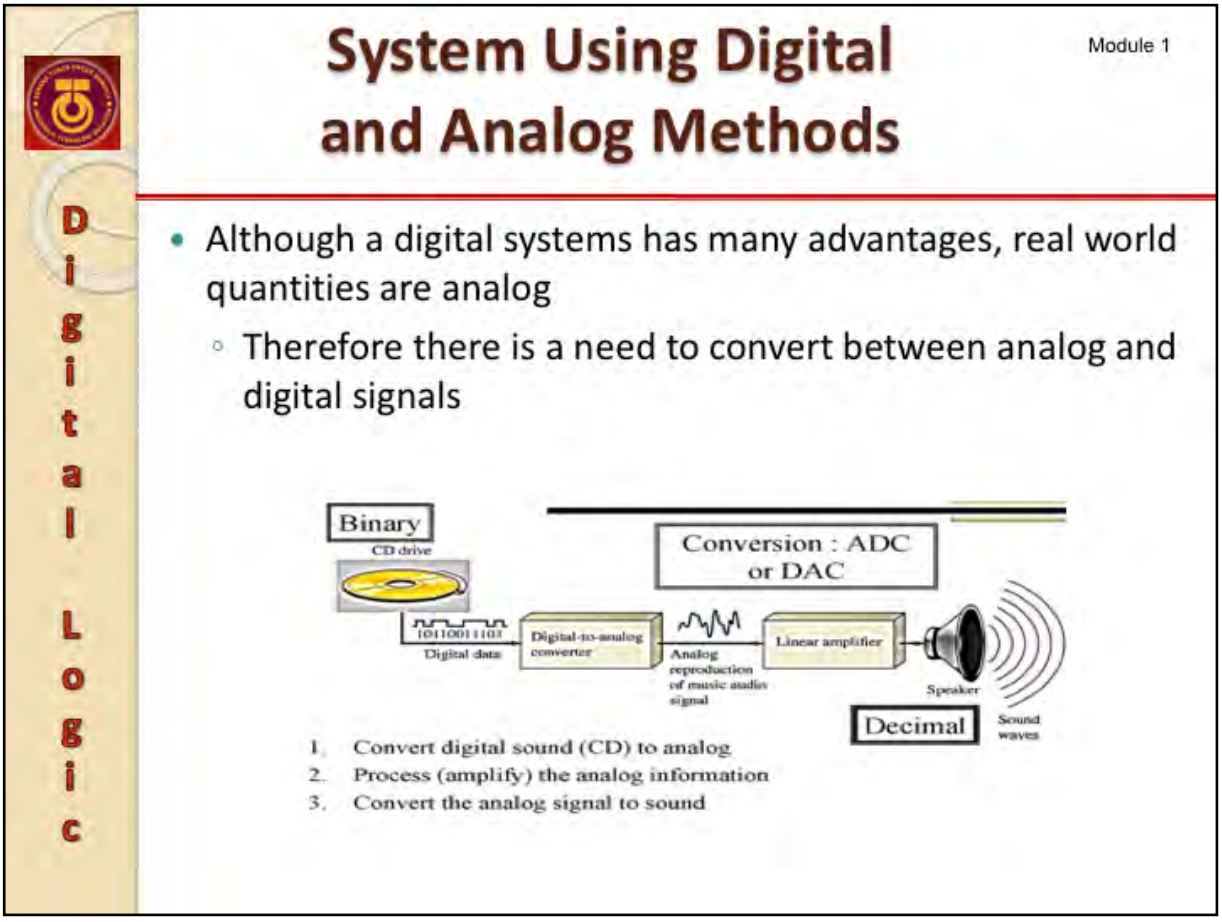
→



Sampling & Quantization

→

Bit 0 and 1



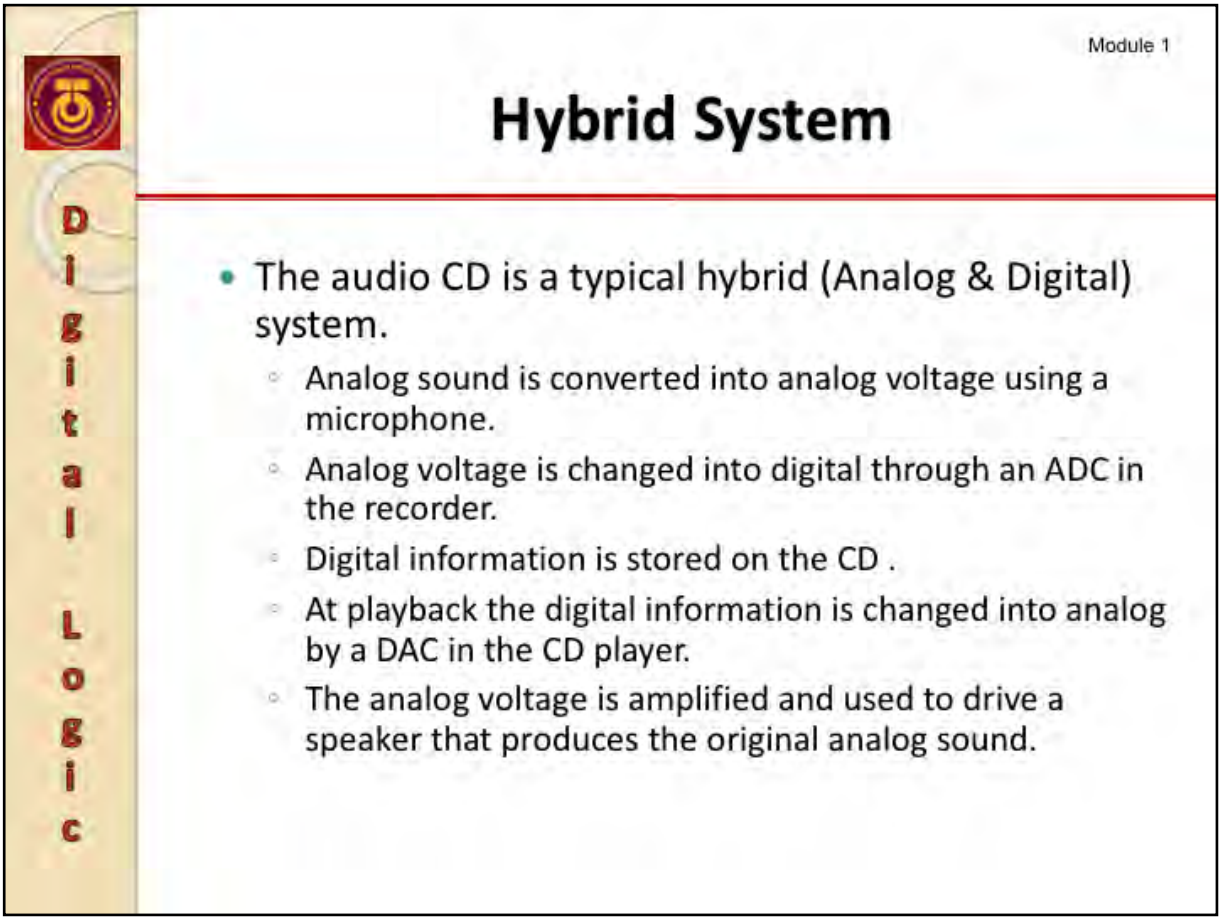
A hybrid of Digital and Analog Systems

An analog-to-digital converter (abbreviated ADC, A/D or A to D) is a device which converts continuous signals to discrete digital numbers. Typically, an ADC is an electronic device that converts an input analog voltage (or current) to a digital number

Digital-to-analog conversion (DAC) is a process in which signals having a few (usually two) defined levels or states (digital) are converted into signals having a theoretically infinite number of states (analog). A common example is the processing, by a modem of computer data into audio-frequency (AF) tones that can be transmitted over a twisted pair telephone line. The circuit that performs this function is a digital-to-analog converter.

Basically, DAC conversion is the opposite of analog-to-digital conversion. In most cases, if an ADC is placed in a communications circuit after a DAC, the digital signal output is identical to the digital signal input. Also, in most instances when a DAC is placed after an ADC, the analog signal output is identical to the analog signal input.

An Audio linear amplifier is an electronic circuit whose output is proportional to its input, but capable of delivering more power into a load. Generally, an amplifier or simply amp usually increases the amplitude of a signal. The "signal" is usually voltage or current.



Analog sound

Human voice has a continuous sinusoidal wave with frequency within range of 300Hz to 3.4KHz ($3.4X10^3$ cycle per second). This signal will be converted to analog voltage by using microphone.

Analog to Digital Converter (ADC)

Convert analog signal into digital signal using process such as sampling signal, quantization process and digital conversion. Error will occur during the sampling and quantization, hence loss of information can happen.

Digital information

The accuracy of analog to digital conversion is increased when the signal is represented by a higher word size such as 32bit as compared to 8bit. However, the bigger bit size the higher storage is needed to store the digital data.

Digital to Analog Converter (DAC)

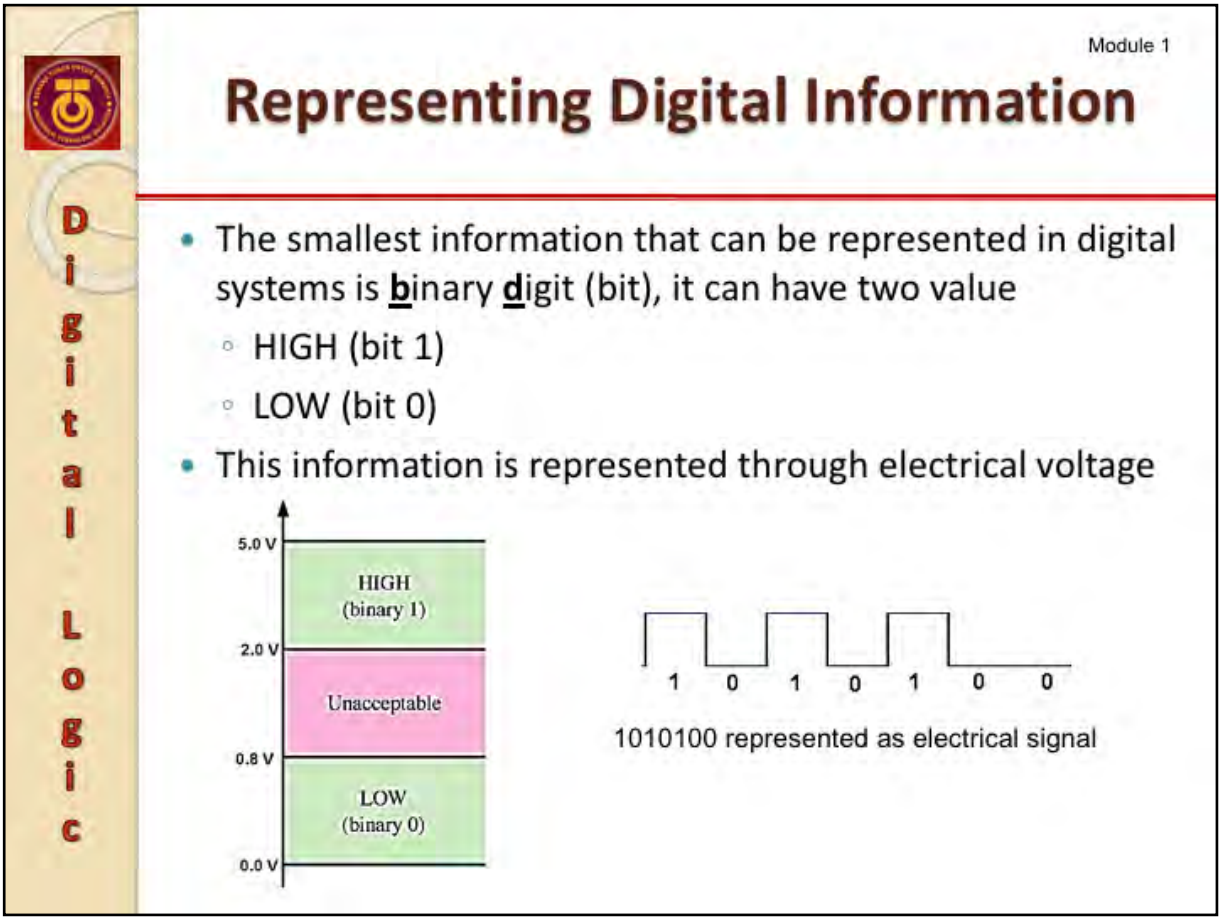
DAC is needed if the speaker is using analog system. We need to convert the digital data to analog signal in order for the speaker works properly and the sound can be heard by human.

Amplifier

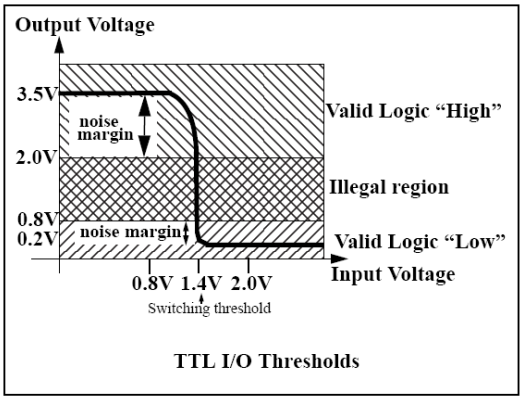
Amplifier will amplified amplitude of the small input signal to increase power output, hence the volume of speaker is increased.

Digital Logic

Digits, Logic Levels and Digital Waveform

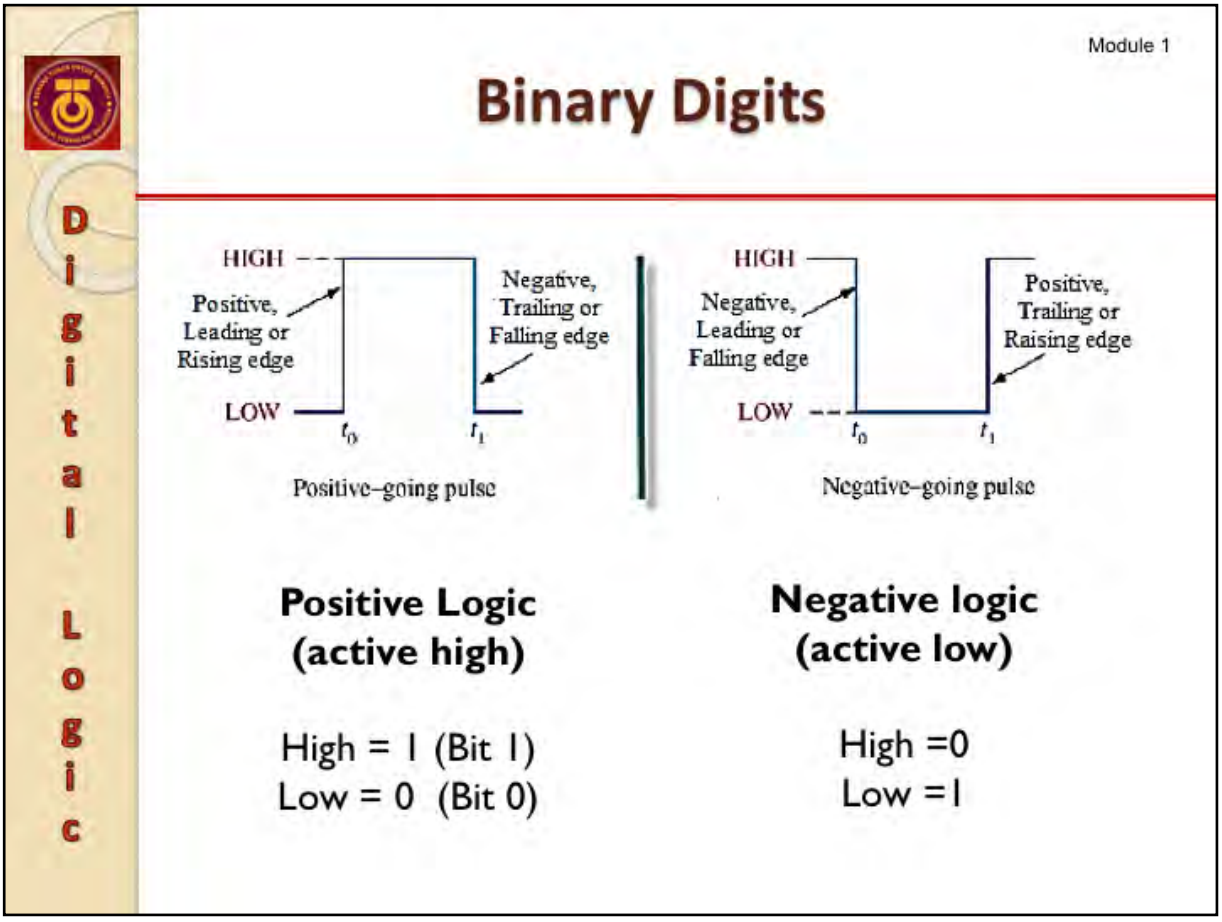


Voltage level to represent bit level :



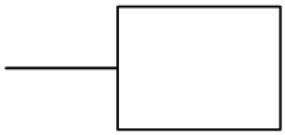
When digital logic has an output voltage of two volts or above its output is considered to be logic "1". An output voltage 0.8 volts or less is considered to be logic "0". Voltages between 0.8 volts and 2.0 volts are considered illegal. The binary logic circuits should never encounter signals within this region except for a quick transition through it when switching states.

In real implementations of these circuits, electrical noise may exist. Therefore the typical output levels are well in excess of the points at which the "1" or "0" decision is made. The distance by which the input voltage exceeds the switching threshold is called the *noise margin.*
Noise levels of up to 0.6 volts may be injected on an input signal and the inverter shown above will still operate correctly.

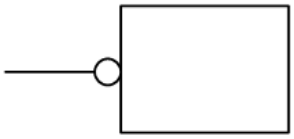


The term *active* means the moment that logic circuit is reacting with its true manner. There are two conditions which control the action of circuit, either active high or active low .

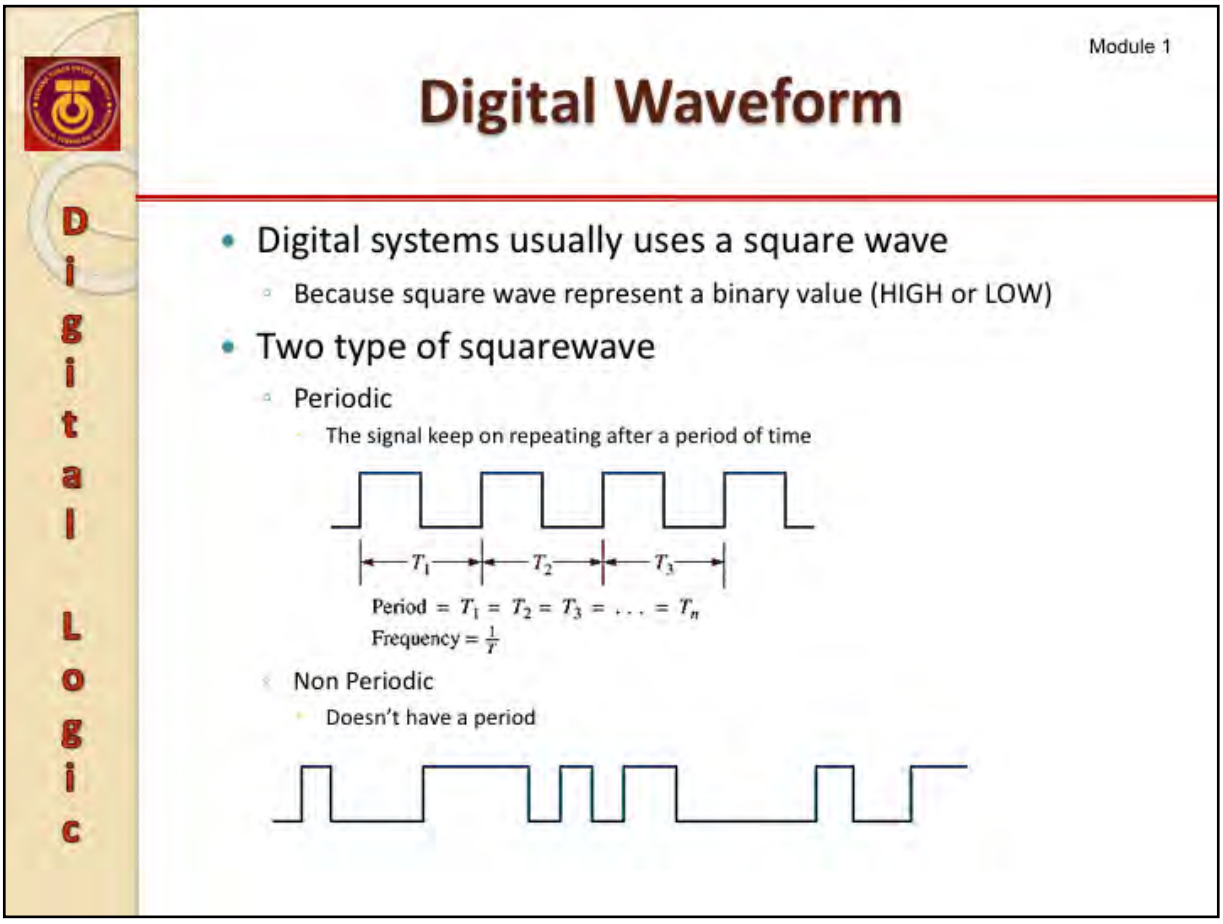
Symbols to show the input state of "active high" *and* "active low":



"active high"



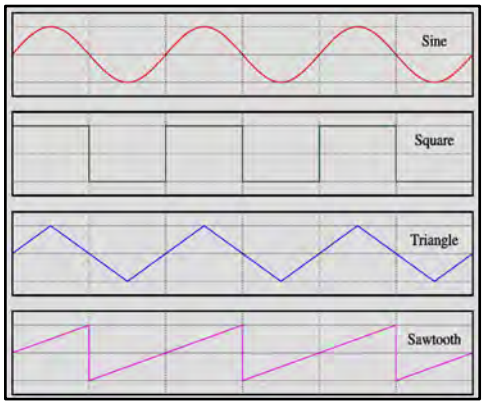
"active low"



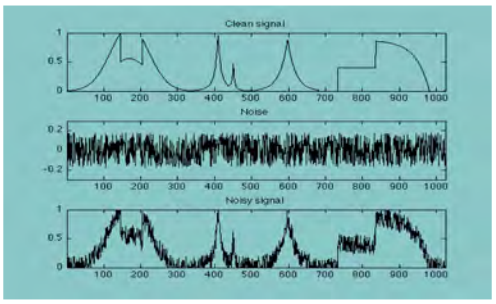
Periodic: In mathematics, a **periodic function** is a function that repeats its values in regular intervals or periods. The most important examples are the trigonometric functions, which repeat over intervals of length 2π. Periodic functions are used throughout science to describe oscillations, waves, and other phenomena that exhibit periodicity.

Everyday examples are seen when the variable is *time*; for instance the hands of a clock or the phases of the moon show periodic behavior. **Periodic motion** is motion in which the position(s) of the system are expressible as periodic functions, all with the *same* period.

Non Periodic: A function that is not periodic is called **aperiodic**.



Periodic signal



Aperiodic signal

Periodic Signal Parameter

Digital Logic

- Frequency (f) is the rate at which the signal repeat itself at a fixed interval. Is measured in cycles per second or Hertz (Hz)

$$f = \frac{1}{T} \text{ Hz}$$

- Period (T) is the time from the edge of one pulse to the corresponding edge of the next pulse. Is measured in second

$$T = \frac{1}{f} \text{ Second}$$

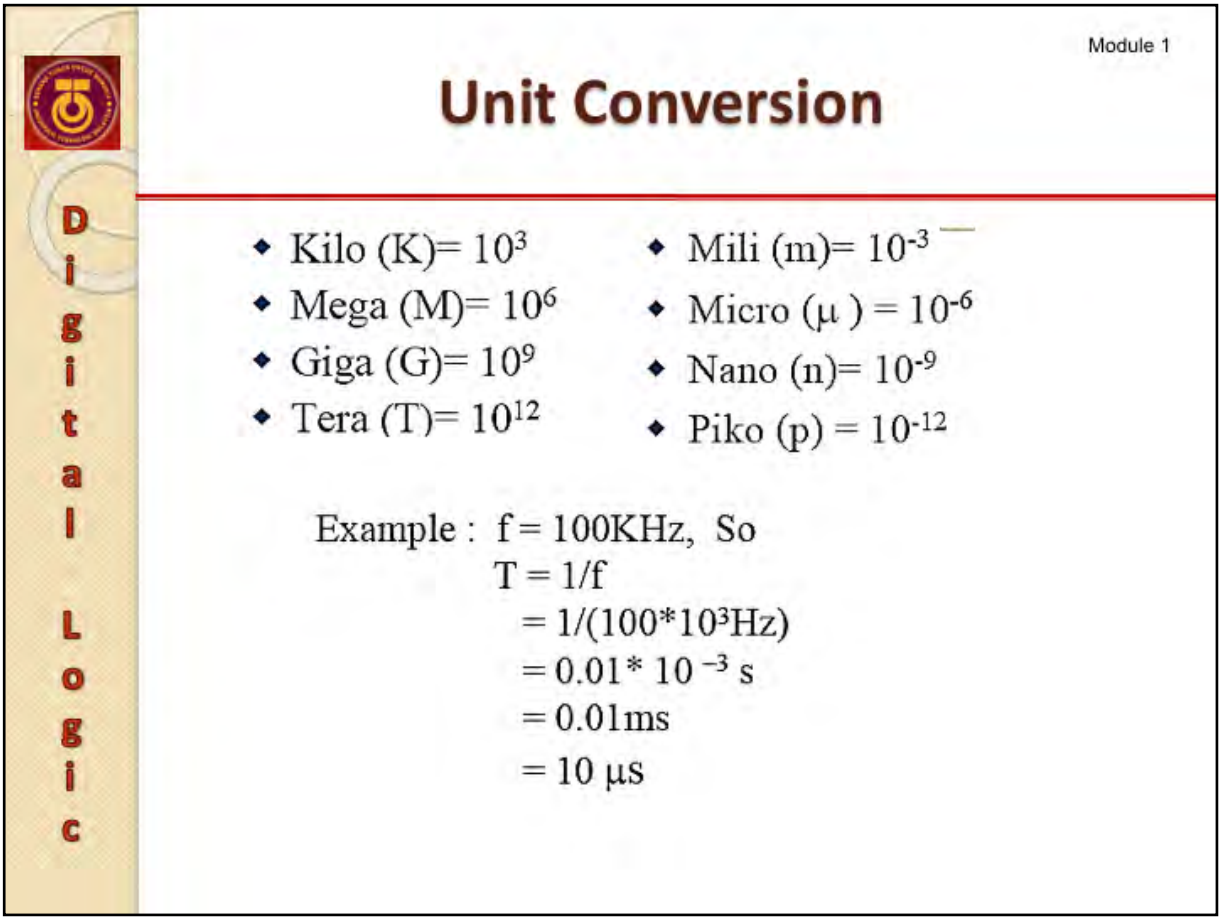
- Example:
  - clock frequency : f = 100Hz,
    so, period : T = 1/100Hz = 0.01s = 10x 10^{-3} = 10 ms

s → ms	(x 10^{3})
ms → s	(x 10^{-3})

Some examples of periodic signal display on the oscilloscope:

(a) Square waveform

(b) Sinusoid waveform



DIY:

1. Calculate frequency of signals if time period are given as the following:

   a) 10ms

   b) 100µs

   c) 100ns

   d) 1000ps

2. Calculate time period of signals if frequency are given as the following:

   a) 1000KHz

   b) 100MHz

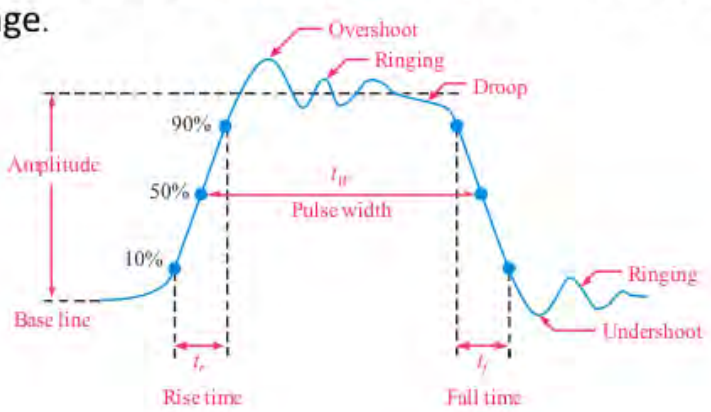
   c) 1000GHz

   d) 100THz

Pulse Width

Digital Logic

- Pulse is a rapid, transient change in the amplitude of a signal from a baseline value to a higher or lower value, followed by a rapid return to the baseline value.
- Pulse width (t_W): A measure of the duration of the pulse.
- Rise time and fall time is a measure of how fast the pulse change.

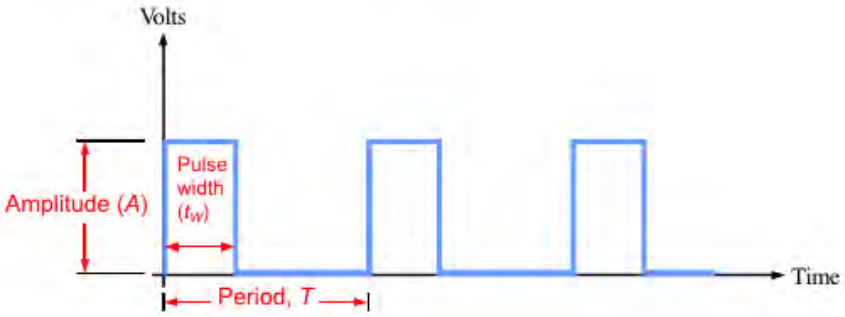


- For digital signal, the rise time (t_r) and fall time (t_f) is usually very short.
- For digital device t_r and t_f must fulfill a certain specified time **to be recognized** as a valid digital logic value.

Repetative Pulse Waveform

Digital Logic

- In addition to frequency and period, repetitive pulse waveforms are described by the **amplitude** (A), **pulse width** (t_W) and **duty cycle**.
- Duty cycle is the ratio of t_W to T.

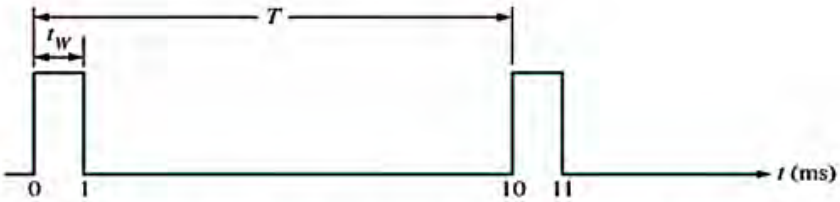


Duty Cycle

Digital Logic

- Duty cycle is the fraction of time that a system is in an "active" state (operated), defined as

$$DutyCycle = (\frac{t_W}{T})100\%$$



Example : a periodic digital waveform has a pulse width (t_w) 1ms and period time (T) 10ms, calculate duty cycle?

Duty cycle = 1ms/10ms * 100% = 10%

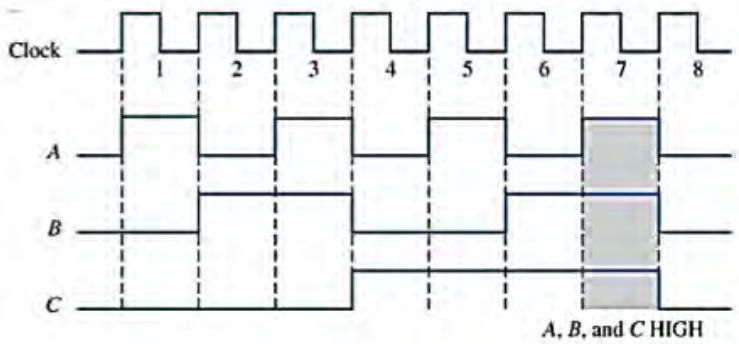
DIY:

Two systems operate with two different ratio of duty cycle 25% and 50% (add 70%, etc..). Given the duration or period for the systems 1000ms. Determine on state and off state for both systems.

Timing Diagram

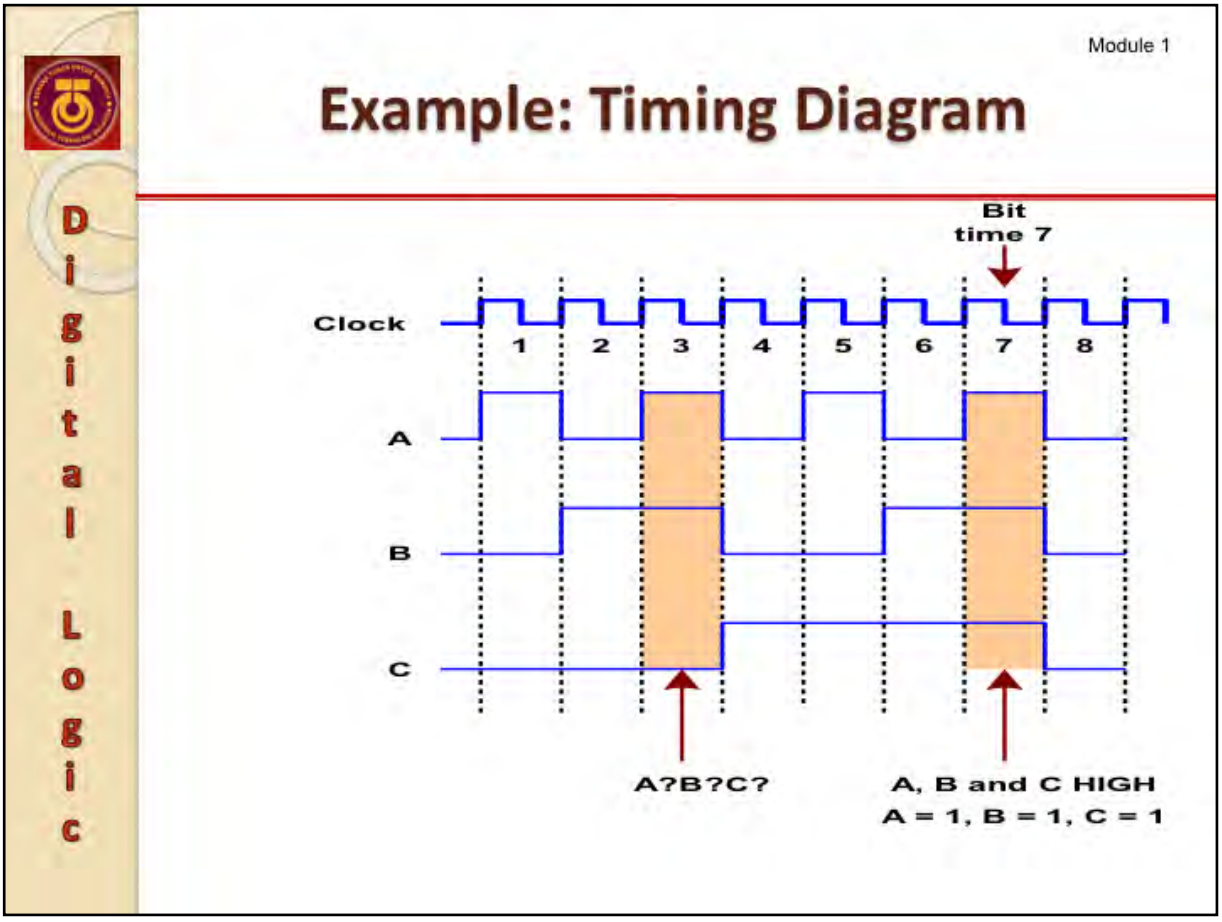
Digital Logic

- Is a graph of digital waveform showing the actual time relationship of two or more waveform and how each waveform changes in relation to the others.
- In digital systems, the emphasis usually the timing not the amplitude because amplitude to represent a bit is predefined.



The importance of timing diagram to digital system:

- ❑ to show the relation of digital value in term of time (discrete value)
- ❑ we are not interested in the actual value (or voltage level)
- ❑ that is why the timing diagram only shows ‘0’ or ‘1’.



Another way of representing the relationship between input and output is by using a truth table.

DIY:

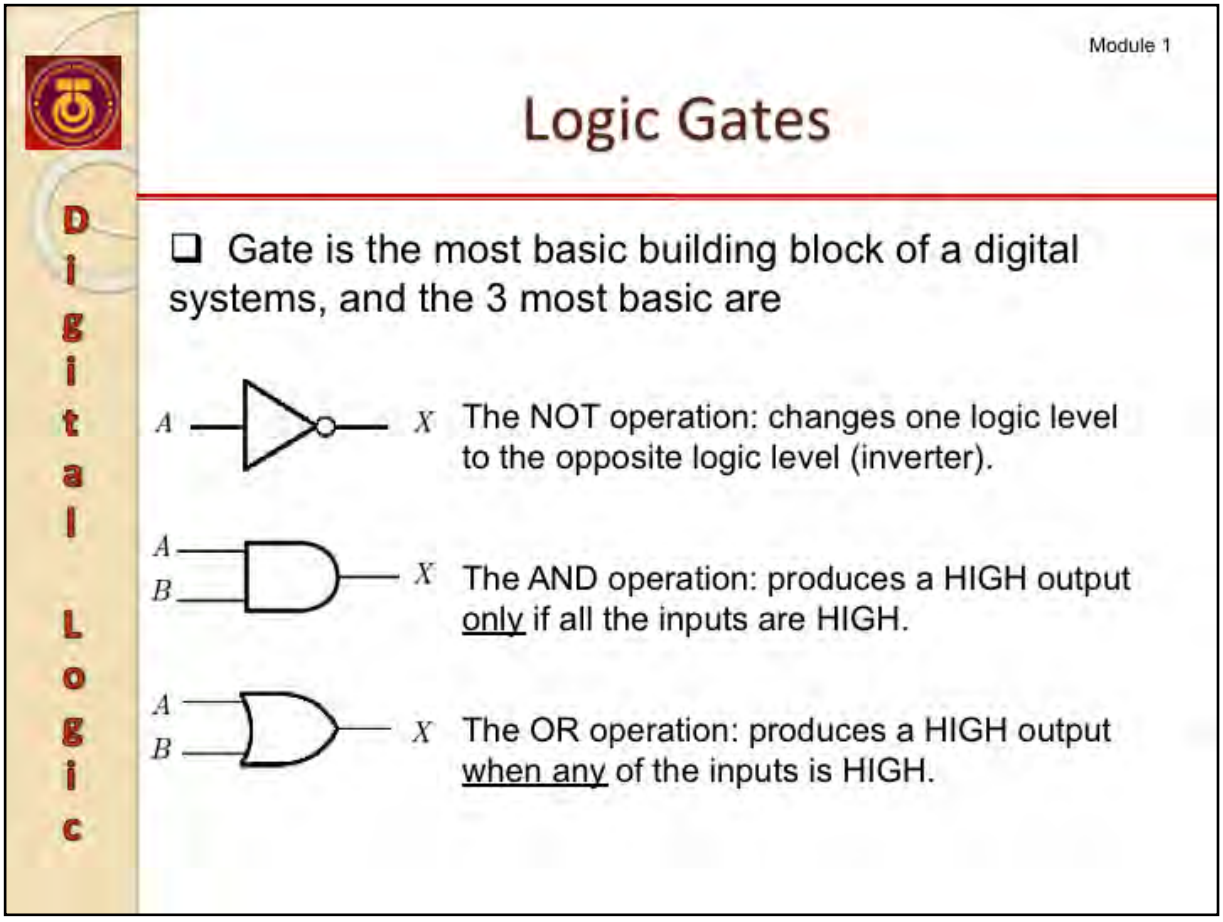
Given : Clock: Active High, Input: A and B, while Output: C

Using Timing Diagram given, convert the state A, B and C into Truth Table form:

Clock (↑)	Input		Output
	A	B	C
1	1	0	0
2			
3	1	1	0
4			
5			
6			
7	1	1	1
8			

Digital Logic

Introduction to Logic Operations



Basic IC (Common):

7404 integrated circuit provides six inverters
7408 integrated circuit provides four two-inputs AND gates
7432 integrated circuit provides four two-inputs OR gates

Others IC (Can be derived from basic IC):

7400 integrated circuit provides four two-inputs NAND gates
7402 integrated circuit provides four two-inputs NOR gates
7486 integrated circuit provides four two-inputs XOR gates.
747266 integrated circuit provides four two-inputs XNOR gates

Logic Gates: NOT

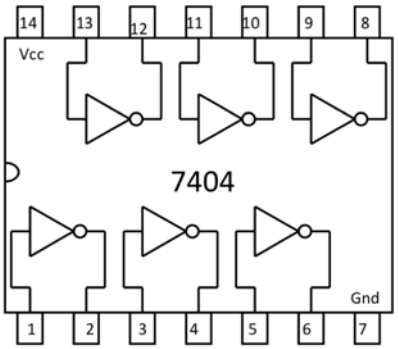
NOT operation

Truth table shows the relationship between output and the input.

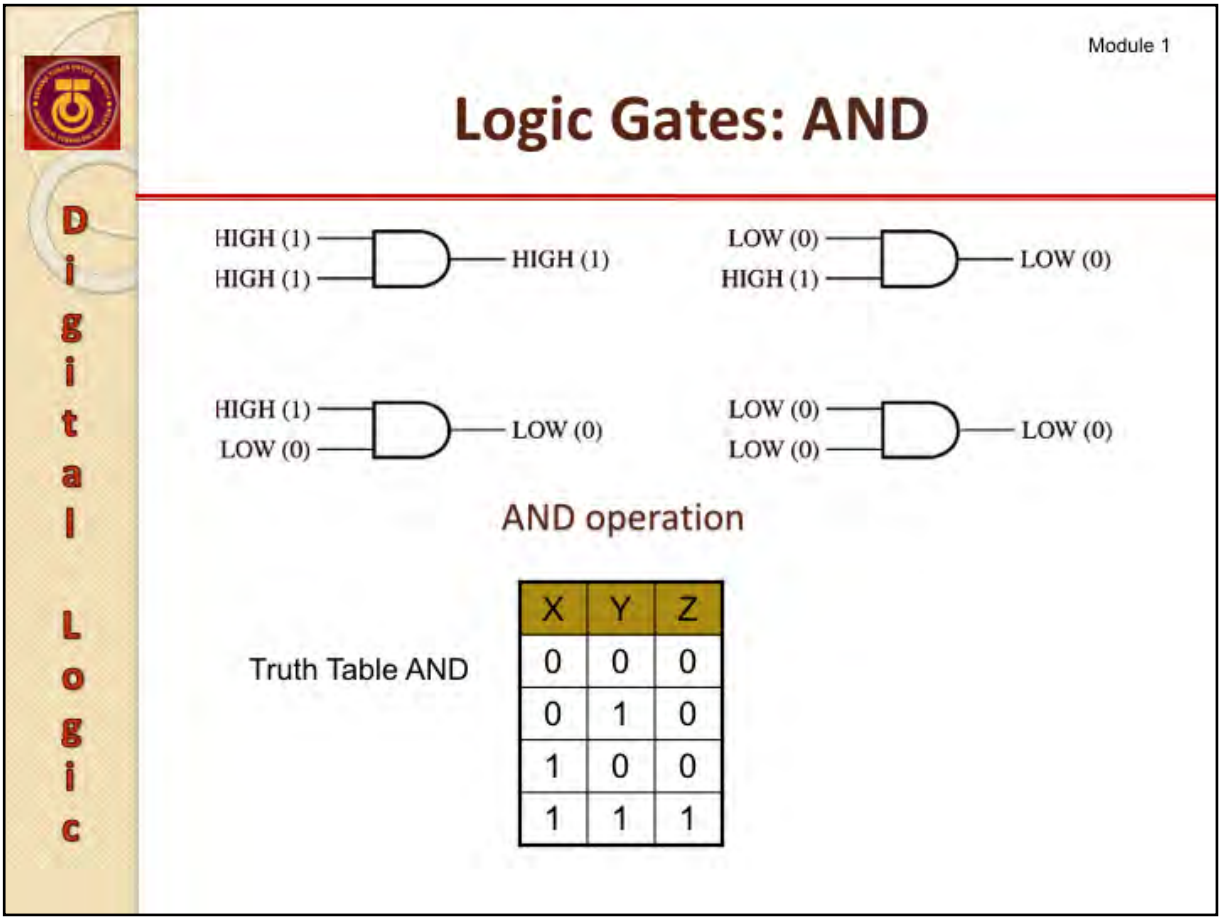
Truth Table for NOT

X	Z
0	1
1	0

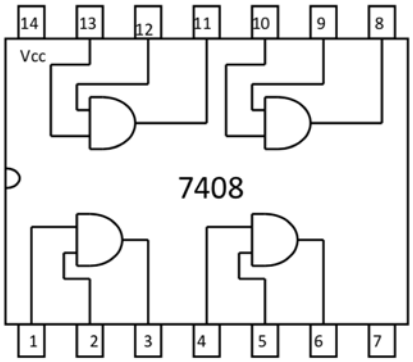
Logic gate IC : NOT



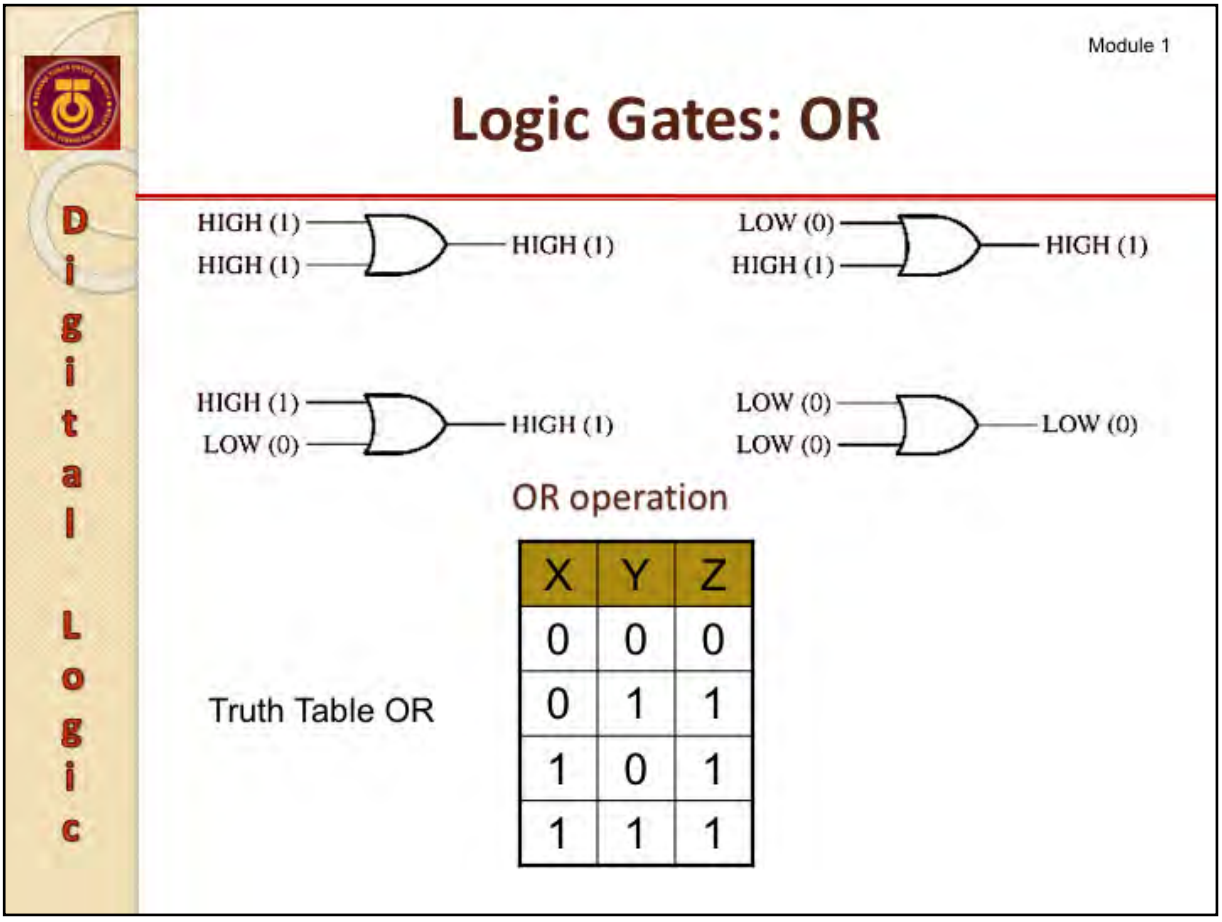
7404 IC six inverters



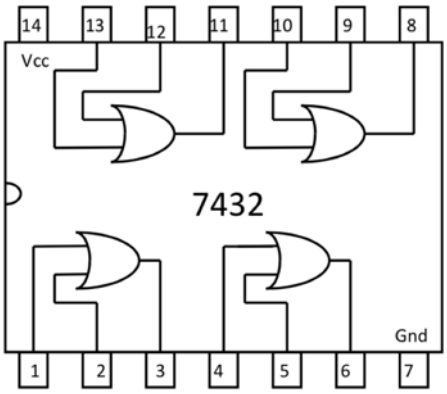
Logic gate IC: AND



7408 IC four (Quad) AND gates



Logic gate IC: OR



7432 integrated circuit provides four (Quad) two-inputs OR gates

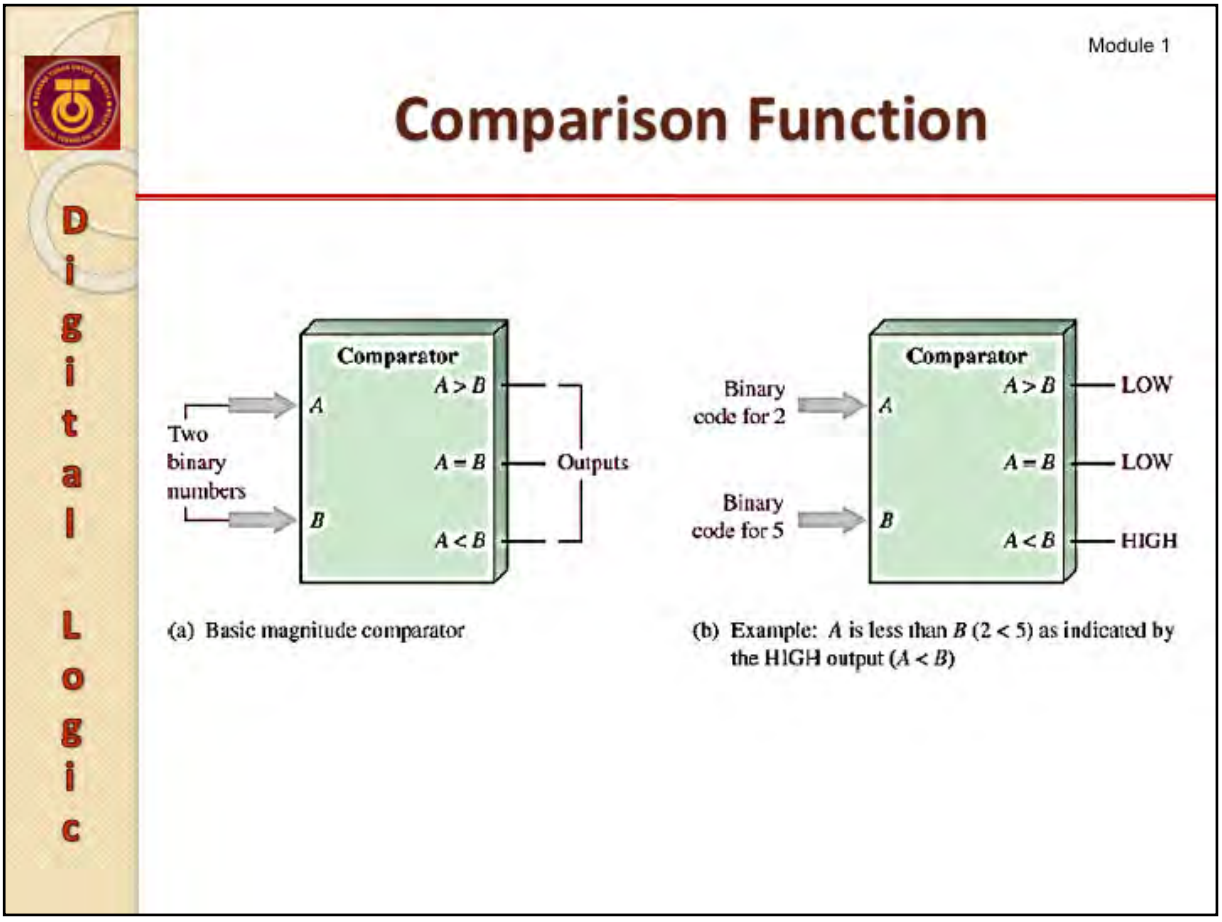
Digital Logic

Overview of Logic Functions

Basic Logic Functions

Digital Logic

- Any digital systems has one or more of the following function.
  - This functions are built from the basic gates.
    - Comparison Function
    - Arithmetic Functions
    - Code conversion function
    - Encoding function
    - Decoding function
    - Data selection function
    - Data storage function
    - Counting function



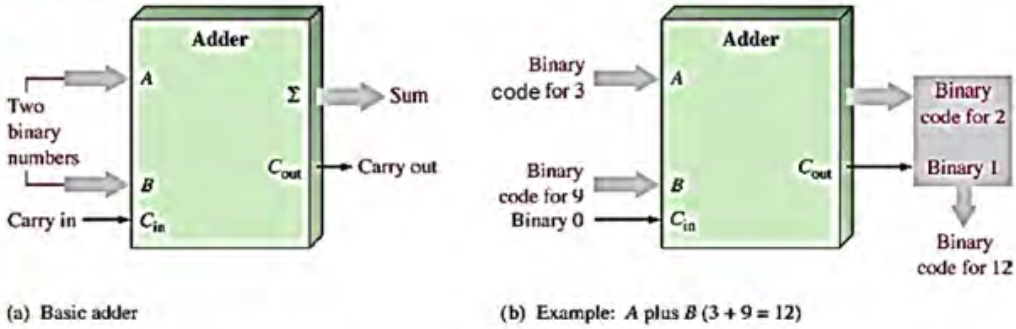
Digital Comparators

Another common and very useful combinational logic circuit is that of the **Digital Comparator** circuit. Digital or Binary Comparators are made up from standard AND, NOR and NOT gates that compare the digital signals at their input terminals and produces an output depending upon the condition of the inputs.

Arithmetic Functions

Digital Logic

- Adder



(a) Basic adder (b) Example: A plus B (3 + 9 = 12)

- Subtractor
- Multiplier
- Division

In electronics, an adder or summer is a digital circuit that performs addition of numbers. In modern computers adders reside in the arithmetic logic unit (ALU) where other operations are performed. Although adders can be constructed for many numerical representations, such as Binary-coded decimal or excess-3, the most common adders operate on binary numbers. In cases where twos complement or ones complement is being used to represent negative numbers, it is trivial to modify an adder into an adder-subtractor. Other signed number representations require a more complex adder.

All the other arithmetic operations can be derived from adder:

- ❑ Subtraction is and addition of negative number such as A–B = A+(-B)
- ❑ Multiplication is a repeated addition such as A*3=A+A+A
- ❑ Division is a repeated subtraction which is a repeated addition such as

  6/3=6-3-3=6+(-3)+-(3)

  - subtract until the remainder = 0
  - total number of subtraction = 2 which is the answer

Code Conversion Function

Digital Logic

- A code is a set of bits arranged in a unique pattern and used to represent specified information.
  - Examples : BCD, ASCII
- The usage of codes allow a faster and more efficient data processing.

BCD:

In computing and electronic systems, binary-coded decimal (BCD) (sometimes called natural binary-coded decimal, NBCD) is an encoding for decimal numbers in which each digit is represented by its own binary sequence. Its main virtue is that it allows easy conversion to decimal digits for printing or display and faster decimal calculations.

In BCD, a digit is usually represented by four bits which, in general, represent the values/digits/characters 0–9. Other bit combinations are sometimes used for a sign or other indications.

Although BCD is not as widely used as it once was, decimal fixed-point and floating-point are still important and continue to be used in financial, commercial, and industrial computing.

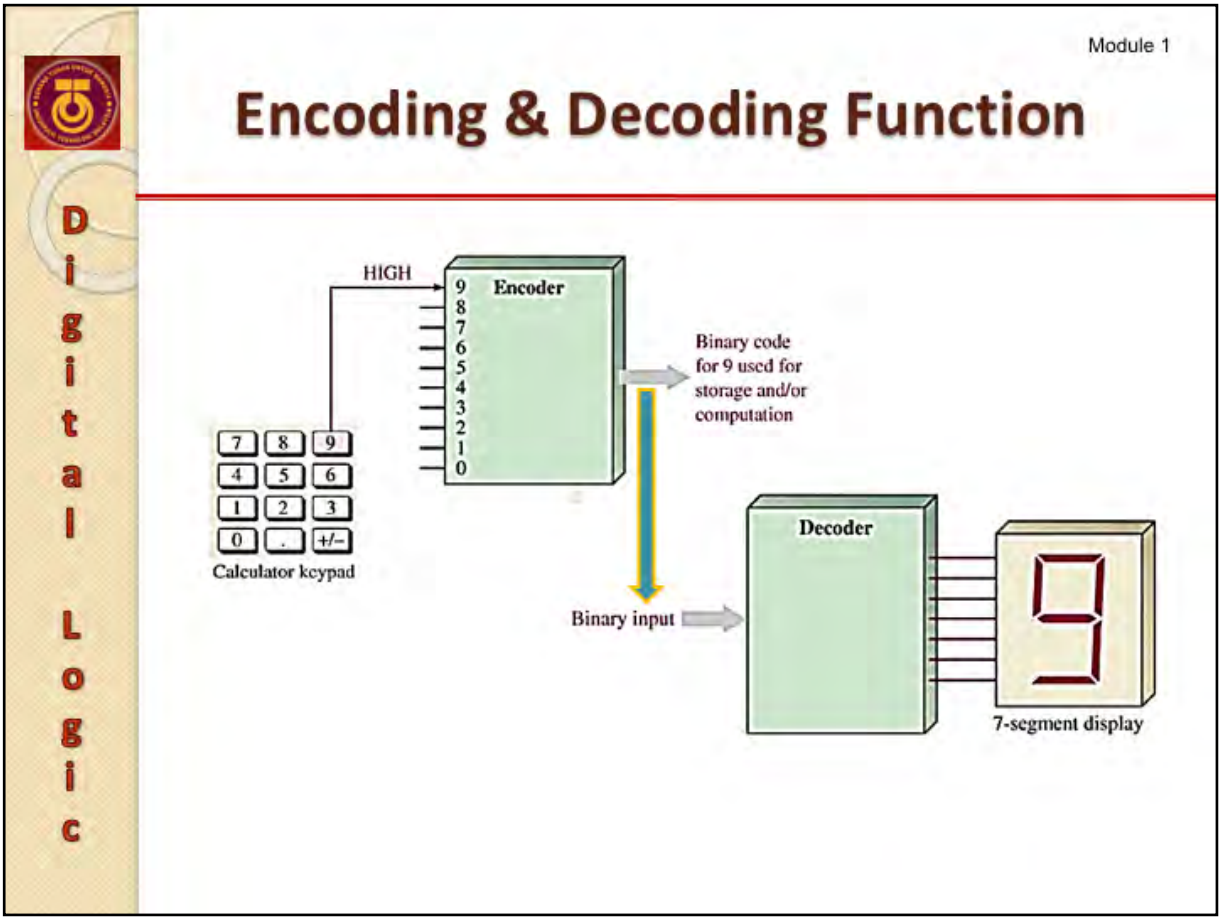
ASCII:

American Standard Code for Information Interchange (ASCII), pronounced /ˈæski/[1] is a character-encoding scheme based on the ordering of the English alphabet. ASCII codes represent text in computers, communications equipment, and other devices that work with text. ASCII developed from telegraphic codes. Its first commercial use was as a seven-bit teleprinter code promoted by Bell data services.

ASCII includes definitions for 128 characters: 33 are non-printing control characters (now mostly obsolete) that affect how text is processed, 94 are printable characters, and the space is considered an invisible graphic.

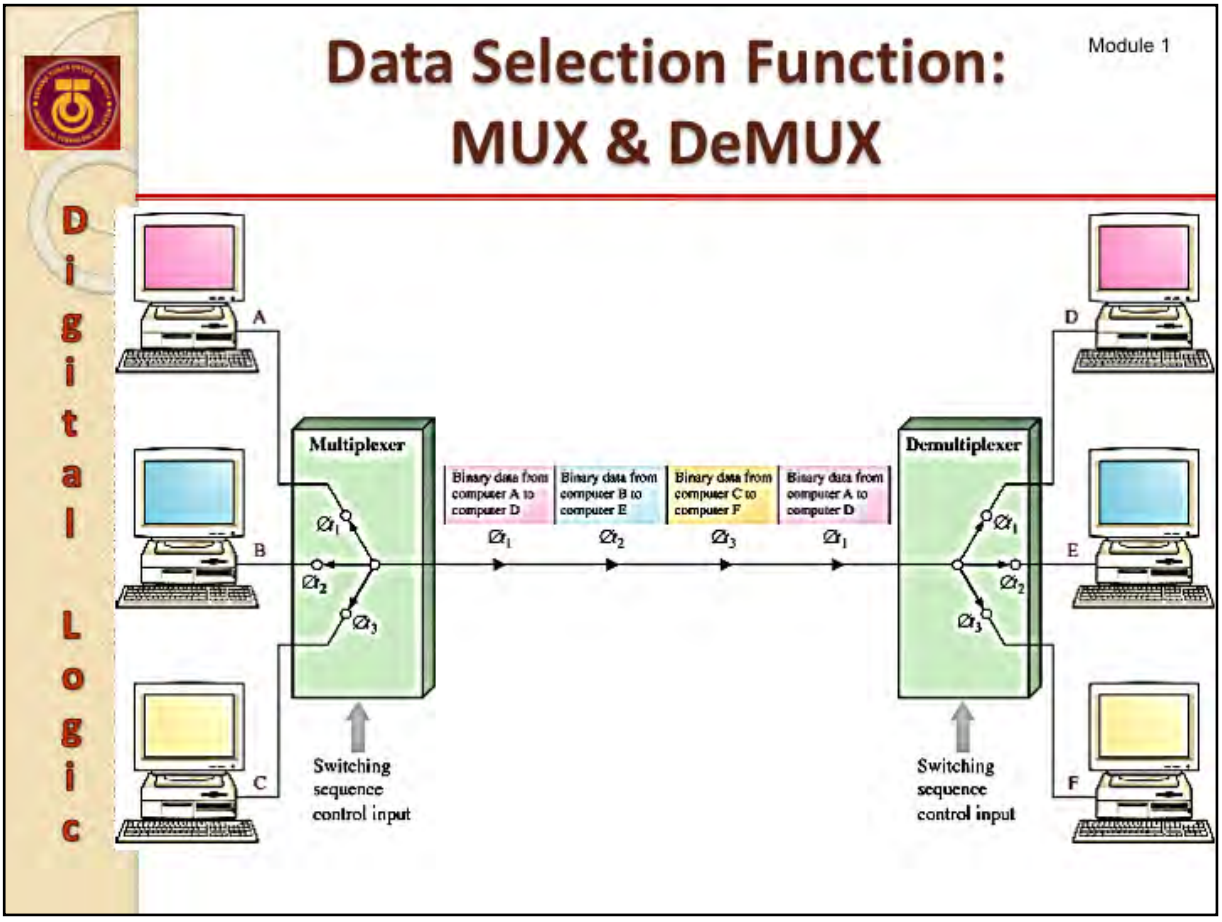
The 95 printable ASCII characters, numbered from 32 to 126 (decimal)

```
 !"#$%&'()*+,-./
0123456789:;<=>?
@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_
`abcdefghijklmno
pqrstuvwxyz{|}~
```



Examples of the Systems:

- A key pad provides decimal value range from 0 – 9.
- Each of the decimal digit can be encoded to its binary representation using Encoder
- The encoded binary data can be decoded back to its original decimal value
- 7 segment display can display the decimal value



Problem:

Many inputs (e.g. A, B and C) wanted to use a single transmission line for their data transmission. How to make sure the data is transferred in a proper manner (issue of cost, synchronization, conflict , crash, loss?)

Source (A, B, C) and Destination (D, E, F)

A → D, B → E, C → F

Solution:

MUX: select and permit only one device can use the line and transfer its data at one time.

Data in the transmission line would be arranged as A, B, C

DEMUX: select and route the data to their originate destination

A → D, B → E, C → F

Advantage:

cost effective (reduce cost)

Disadvantage:

can create congestion if many inputs transmit a huge data rate at one time (e.g. Denial of Service (DoS)) that loss of data can occur .

Data Storage Function

Digital Logic

- Flip-flop stores a 1 or 0 only
- Registers
  - Formed by combining several flip-flops
  - 8-bit register ➔ from 8 flip-flops
- Semiconductor Memories
  - e.g. RAM, ROM, Flash
- Magnetic/Optical Memories
  - For mass storage ➔ e.g. hard disk, tape, DVD, Blu-Ray

Examples of Memory Devices in Digital Systems:

Flip-flop:

In digital circuits, a flip-flop is a term referring to an electronic circuit that has two stable states and is capable of serving as one bit of memory. Simple flip-flops can be built around a pair of cross-coupled inverting elements that can be built from vacuum tubes, bipolar transistors, field effect transistors or inverters.

Registers:

A word size memory that may consist of 8 bits, 16 bits, 32 bits or 64bits size. The basic circuit of register is flip-flop.

RAM:

A volatile memory that store current running program. The size varies from 256MB, 512MB, 1GB and 2GB

Flash:

A fast speed ROM technology

ROM:

A non- volatile memory

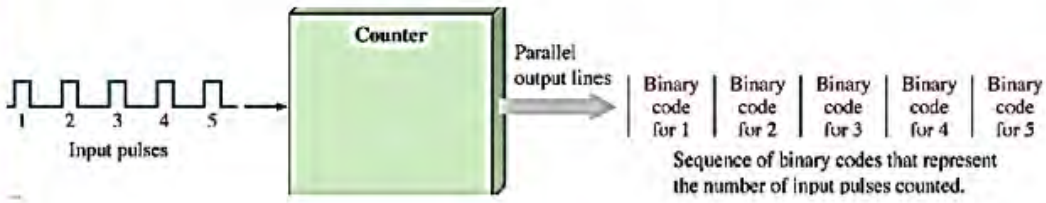
External Memory: Magnetic or Optical

Permanent storage that can stored large quantity of data such as storage size of 60G,120GB, 160GB, 320GB, 512GB or 1TB

Counting Function

Digital Logic

- Counter
  - To count the occurrence at the input.
  - to initiate a controller after a certain count (period).

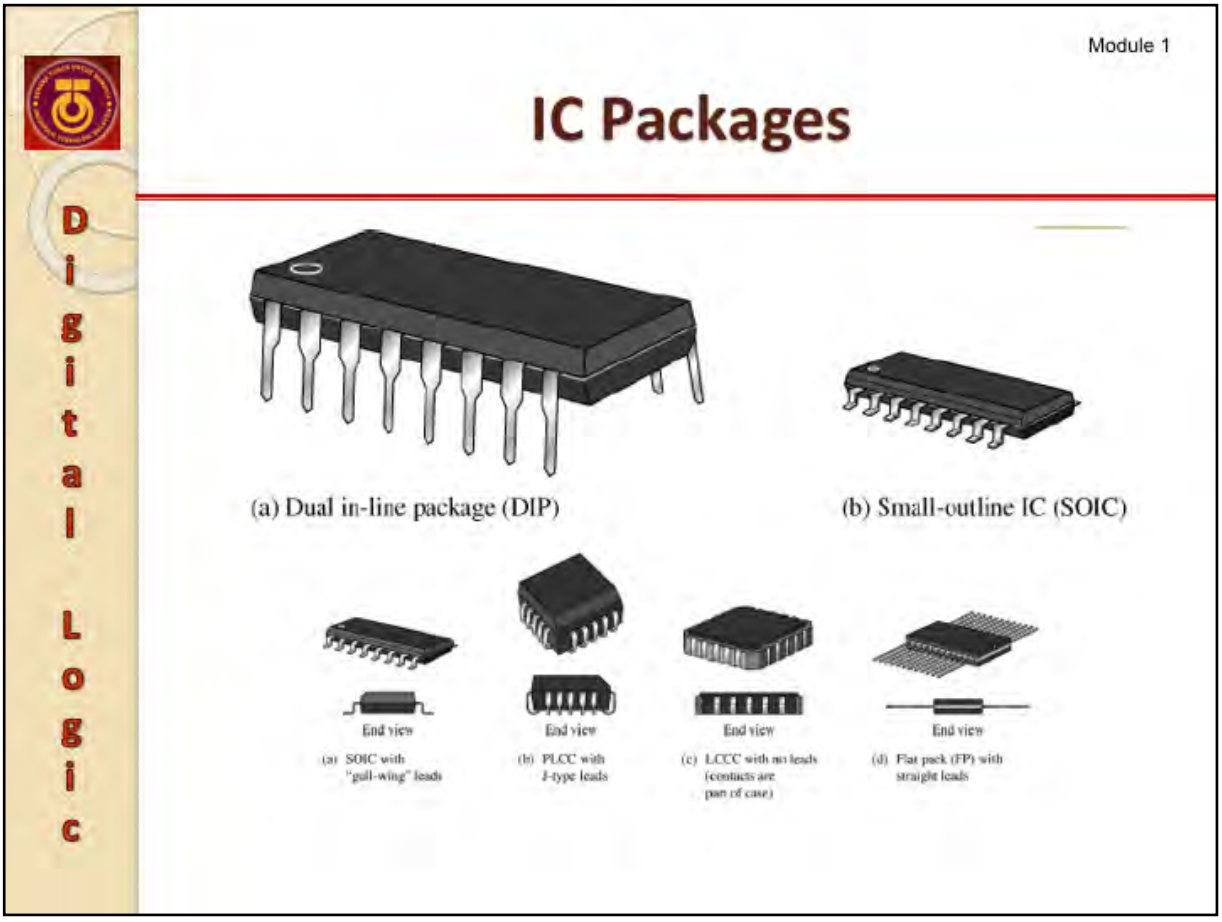


Examples systems:

- Traffic light
- Washing machine
- Vending machine
- Xerox machine
- ATM machine
- etc.

Digital Logic

Fixed-Function Integrated Circuit (IC)

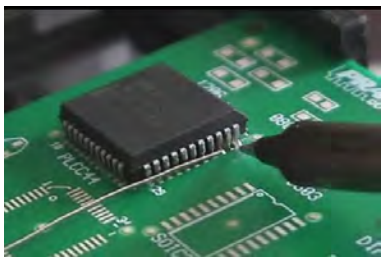


Surface-mount technology (SMT) is a method for constructing electronic circuits in which the components (SMC, or Surface Mounted Components) are mounted directly onto the surface of printed circuit boards (PCBs). Electronic devices so made are called surface-mount devices or SMDs. In the industry it has largely replaced the through-hole technology construction method of fitting components with wire leads into holes in the circuit board.

Through-hole technology, also spelled "thru-hole", refers to the mounting scheme used for electronic components that involves the use of pins on the components that are inserted into holes (PTH - Plated Through-Hole) drilled in printed circuit boards (PCB) and soldered to pads on the opposite side.

Through-hole technology almost completely replaced earlier electronics assembly techniques such as point-to-point construction. From the second generation of computers in the 1950s until surface-mount technology became popular in the late 1980s, every component on a typical PCB was a through-hole component.

IC and conventional **through-hole**

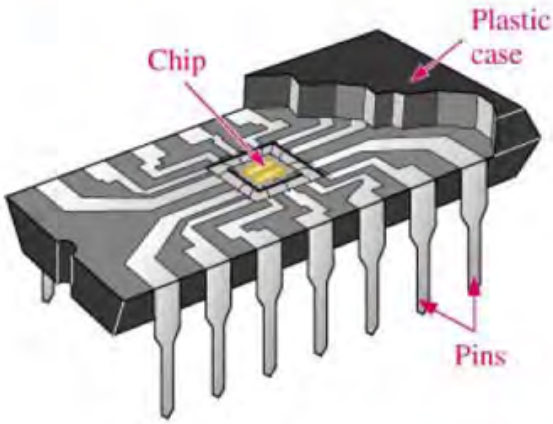


Surface Mount PCB assembly

… IC Packages

Digital Logic

IC package showing the chip mounted inside, with connection to input an output pins.

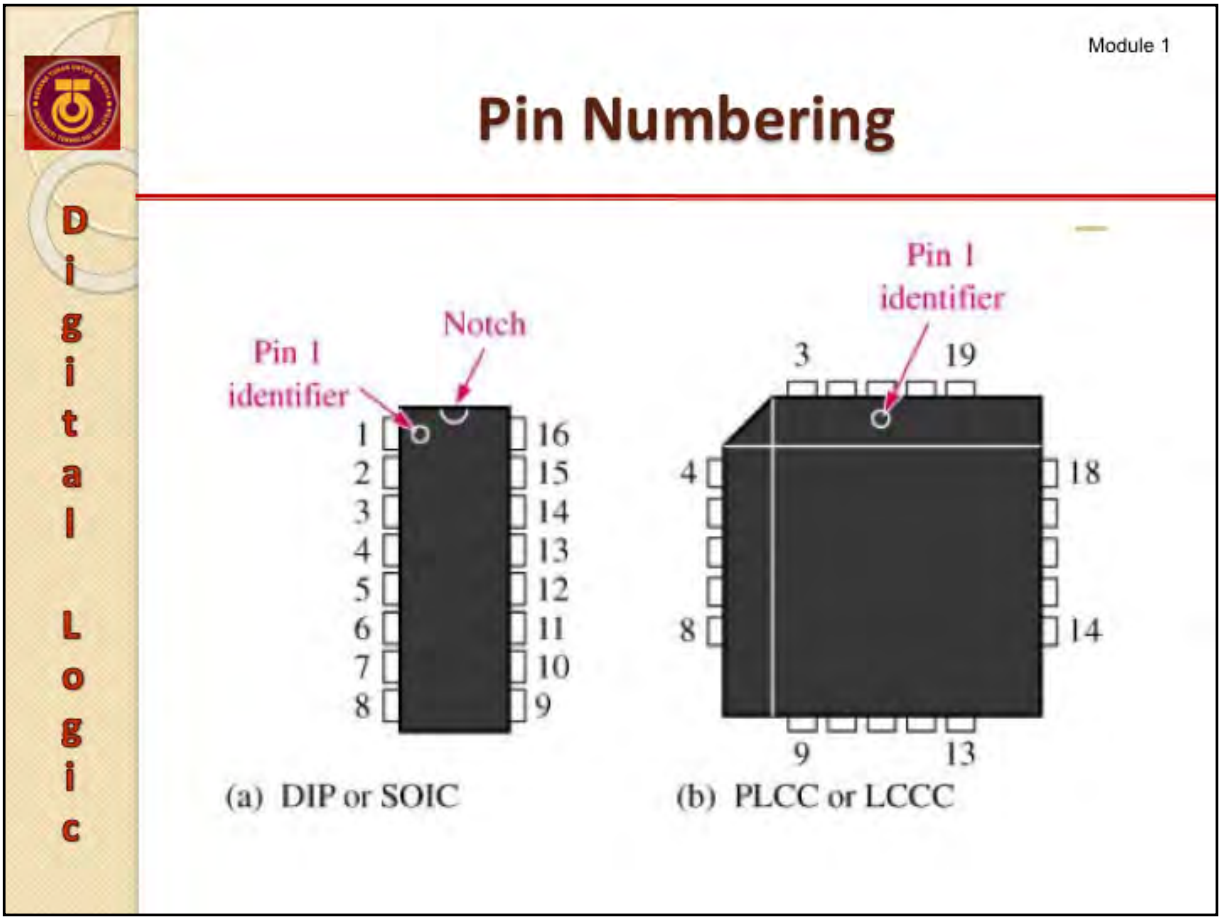


IC Packaging: Why we need packaging?

- To protect the IC (circuit)
- Have a pin system so that can connect to other circuit

Why we have different kind of packaging?

- To accommodate more complex IC and reduce the size of PCB.



Different kind of packaging such as:

Dual in Line Packaging (DIP):

Pin IC occupies both side for instance IC with 16 pins.

Surface Mount:

This is currently most popular in pin connection using surface contact, without hole drilling.

Thru-Hole:

Conventional technology whereby hole needs to be drilled in order to do pin connection to other circuit on the PCB.

Notch:

Pin 1 is denoted from notch and the other pin number sequence are determined in a counter clock wise direction. However, there are other ways to indicate the Pin 1 that depend on the type of packaging.

Some examples of important pins:

VCC, GND

Complexity Classifications for Fixed-Function ICs

Digital Logic

- Small-scale integration (SSI)
  have up to 12 gates on a single chip
- Medium-scale integration (MSI)
  have from 12-99 gates on a single chip
- Large-scale integration (LSI)
  have from 100-9999 gates on a single chip
- Very large-scale integration (VLSI)
  have from 10,000-99,999 gates on a single chip
- Ultra large-scale integration (ULSI)
  have from 100,000 and greater equivalent gates on a single chip

Small Scale Integrated (SSI): AND gate, OR gate, NOR gate, NAND gate, and etc. Example IC:

7404 integrated circuit provides six inverters

7408 integrated circuit provides four two-inputs AND gates

7432 integrated circuit provides four two-inputs OR gates

7400 integrated circuit provides four two-inputs NAND gates

7402 integrated circuit provides four two-inputs NOR gates

7486 integrated circuit provides four two-inputs XOR gates.

MSI (Medium Scale Integrated)

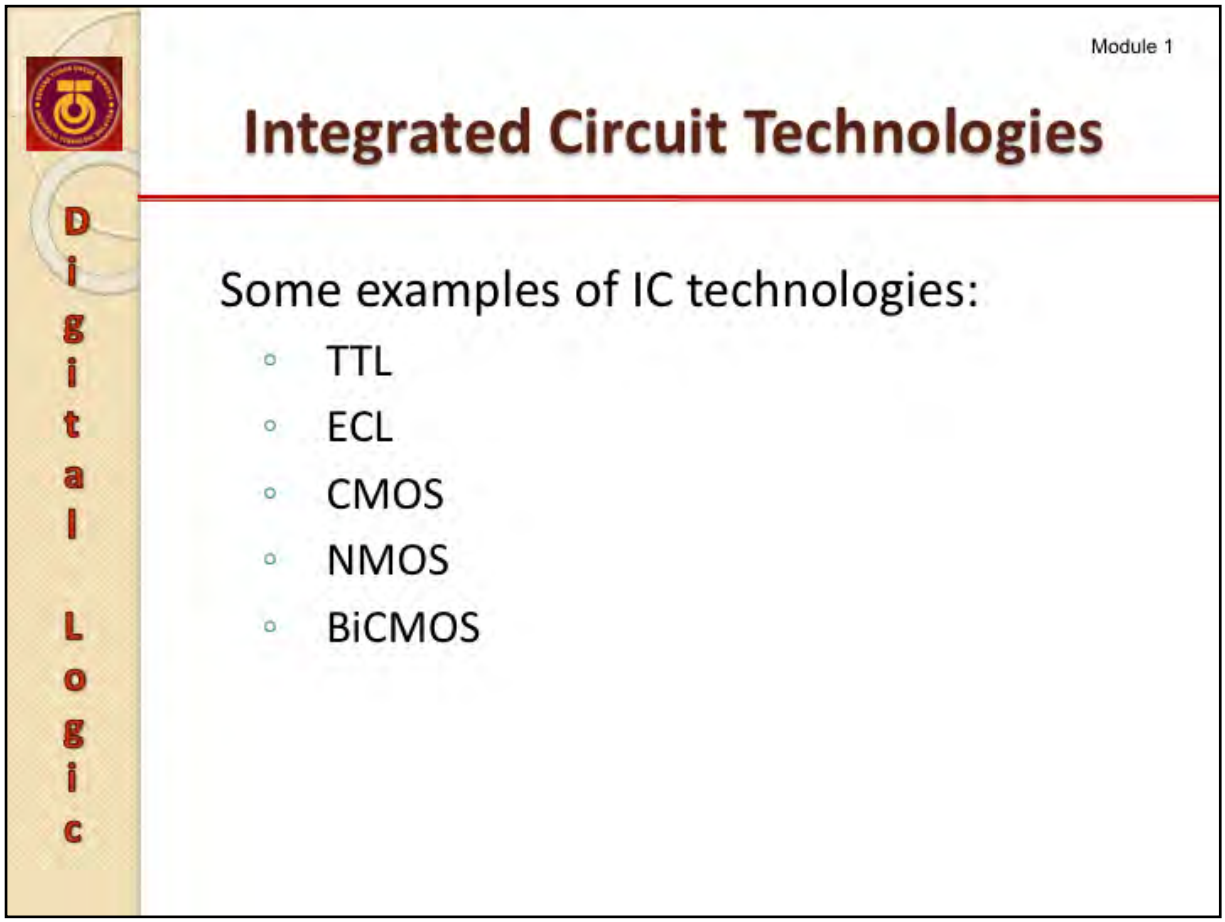
ADDER, COMPARATER, ENCODER, DECODER, Analog Digital Converter (ADC), Arithmetic Unit and etc.

VLSI (Very Large Scale Integrated)

8 Bit Microprocessor (µP)

ULSI (Ultra Large Scale Integrated)

32 bit Microprocessor (µP)



Transistor–transistor logic (TTL) is a class of digital circuits built from bipolar junction transistors (BJT) and resistors. It is called transistor–transistor logic because both the logic gating function (e.g., AND) and the amplifying function are performed by transistors (contrast this with RTL and DTL).

Electronics, Emitter-coupled Logic (ECL) is a logic family in which current is steered through bipolar transistors to implement logic functions. ECL is sometimes called current-mode logic[2] or current-switch emitter-follower (CSEF) logic. The important characteristic of ECL is that the transistors are never in the saturation region and can thus change states at very high speed. Its major disadvantage is that the circuit continuously draws current, which means it requires a lot of power.

Complementary Metal–Oxide–Semiconductor (CMOS) (pronounced /ˈsiːmɒs/ "sea moss"), is a major class of integrated circuits. CMOS technology is used in microprocessors, microcontrollers, static RAM, and other digital logic circuits. CMOS technology is also used for a wide variety of analog circuits such as image sensors, data converters, and highly integrated transceivers for many types of communication.

N-type of Metal–Oxide–Semiconductor (NMOS) : Two important characteristics of CMOS devices are high noise immunity and low static power consumption. Significant power is only drawn when the transistors in the CMOS device are switching between on and off states. Consequently, CMOS devices do not produce as much waste heat as other forms of logic, for example Transistor-Transistor Logic (TTL) or NMOS logic, which uses all n-channel devices without p-channel devices. CMOS also allows a high density of logic functions on a chip.

In integrated circuit technologies, **BiCMOS**, also called BiMOS, refers to the integration of bipolar junction transistors and CMOS technology into a single device. This technology has commercial application in amplifier and discrete component logic design. More recently it has become the technology of choice for power electronics products such as voltage regulators.

Digital Logic

Programmable Logic Devices (PLD)

Overview of PLD

Digital Logic

- **Fixed function**
  - A specific logic function is contained in the IC (hardwired) and can never be changed.
- **PLD**
  - Logic function programmed by the user.
    - Some, can be reprogrammed many times.
  - Advantage
    - More logic circuit can be 'stuffed' into much smaller area.
    - Certain PLD, design can be changed without rewiring or replacing components.
    - Can be implemented faster once the required programming language is mastered.

Hardwired circuit IC :

7404 integrated circuit provides six inverters

7408 integrated circuit provides four two-inputs AND gates

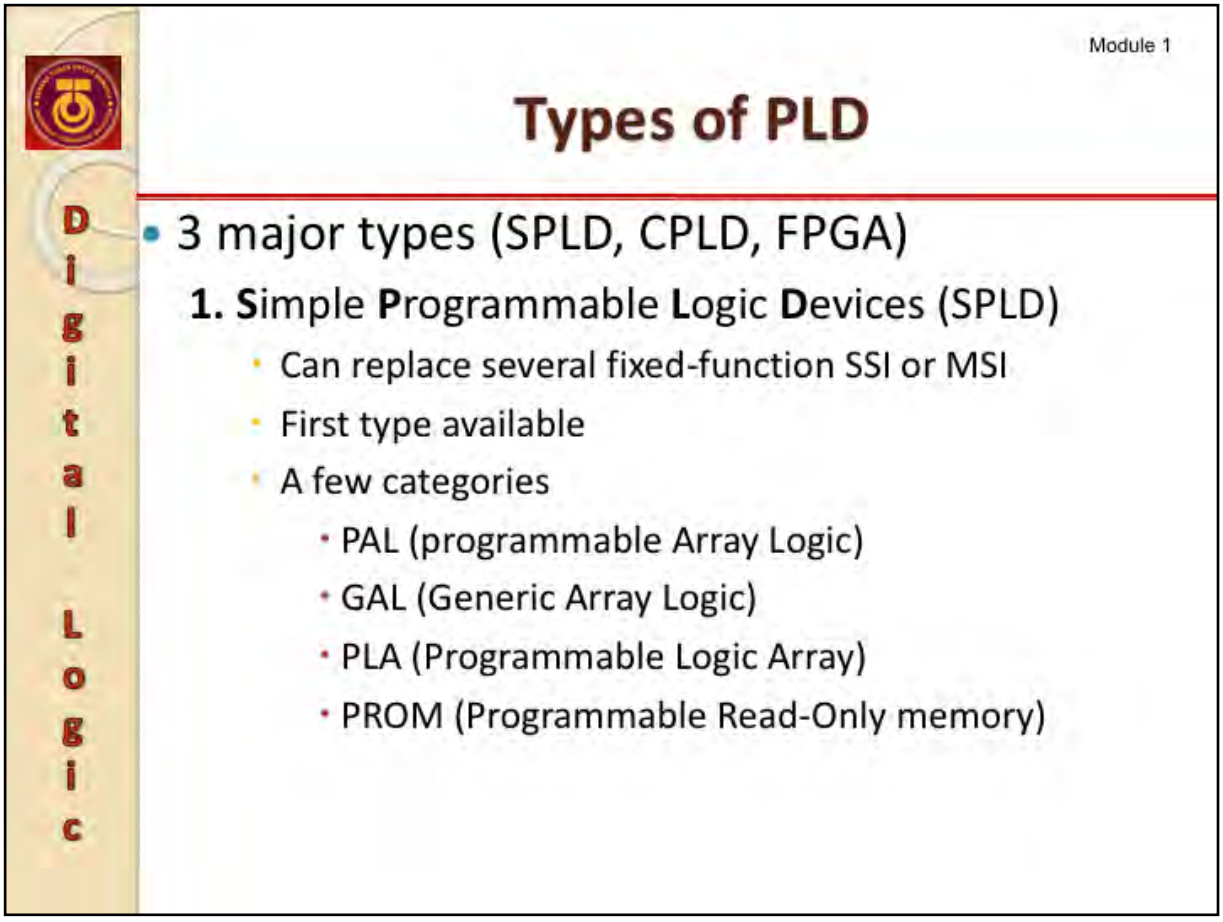
7432 integrated circuit provides four two-inputs OR gates

7400 integrated circuit provides four two-inputs NAND gates

7402 integrated circuit provides four two-inputs NOR gates

7486 integrated circuit provides four two-inputs XOR gates

A programmable logic device or PLD is an electronic component used to build reconfigurable digital circuits. Unlike a logic gate, which has a fixed function, a PLD has an undefined function at the time of manufacture. Before the PLD can be used in a circuit it must be programmed, that is, reconfigured.



PAL (Programmable Array Logic):

The term PAL is used to describe a family of programmable logic device semiconductors used to implement logic functions in digital circuits introduced by Monolithic Memories, Inc. (MMI) in March 1978. PAL devices consisted of a small PROM (programmable read-only memory) core and additional output logic used to implement particular desired logic functions with few components.

Using specialized machines, PAL devices were "field-programmable". Each PAL device was ”One-Time Programmable" (OTP), meaning that it could not be updated and reused after its initial programming. (MMI also offered a similar family called HAL, or "hard array logic", which were like PAL devices except that they were mask-programmed at the factory.)

GAL (Generic Array Logic):

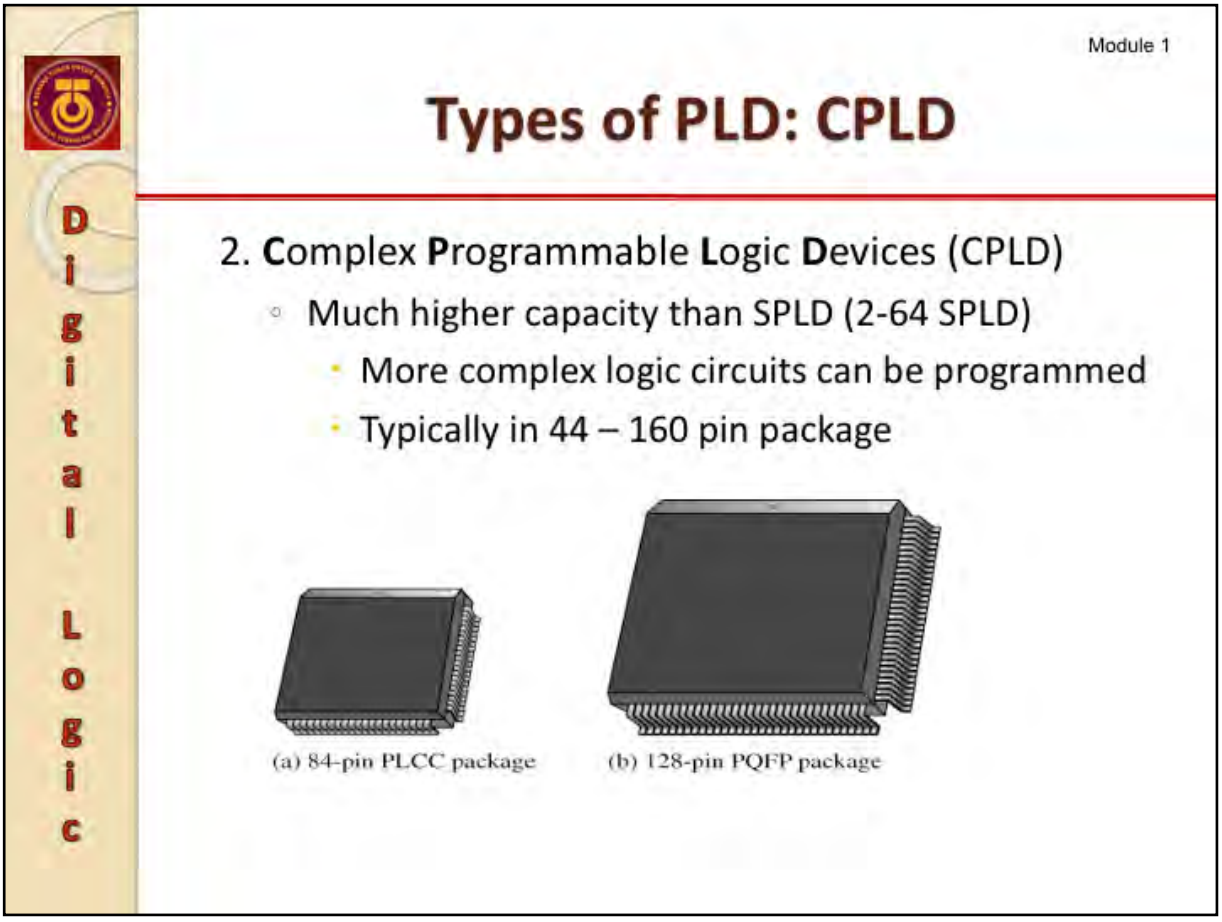
The GAL device was an innovation of the PAL and was invented by Lattice Semiconductor. The GAL was an improvement on the PAL because one device was able to take the place of many PAL devices or could even have functionality not covered by the original range. Its primary benefit, however, was that it was eraseable and re-programmable making prototyping and design changes easier for engineers.

PLA (Programmable Logic Array):

A PLA is a programmable device used to implement combinational logic circuits. The PLA has a set of programmable AND gate planes, which link to a set of programmable OR gate planes, which can then be conditionally complemented to produce an output. This layout allows for a large number of logic functions to be synthesized in the sum of products (and sometimes product of sums) canonical forms.

PROM (Programmable Read-Only Memory):

A PROM is a form of digital memory where the setting of each bit is locked by a fuse or antifuse. Such PROMs are used to store programs permanently. The key difference from a strict ROM is that the programming is applied after the device is constructed. They are frequently seen in video game consoles, or such products as electronic dictionaries, where PROMs for different languages can be substituted.



A complex programmable logic device (CPLD) is a programmable logic device with complexity between that of SPLD and FPGA, and architectural features of both. The building block of a CPLD is the macro cell, which contains logic implementing specialized logic operations.

An Altera MAX 7000-series CPLD with 2500 gates.

Types of PLD: FPGA

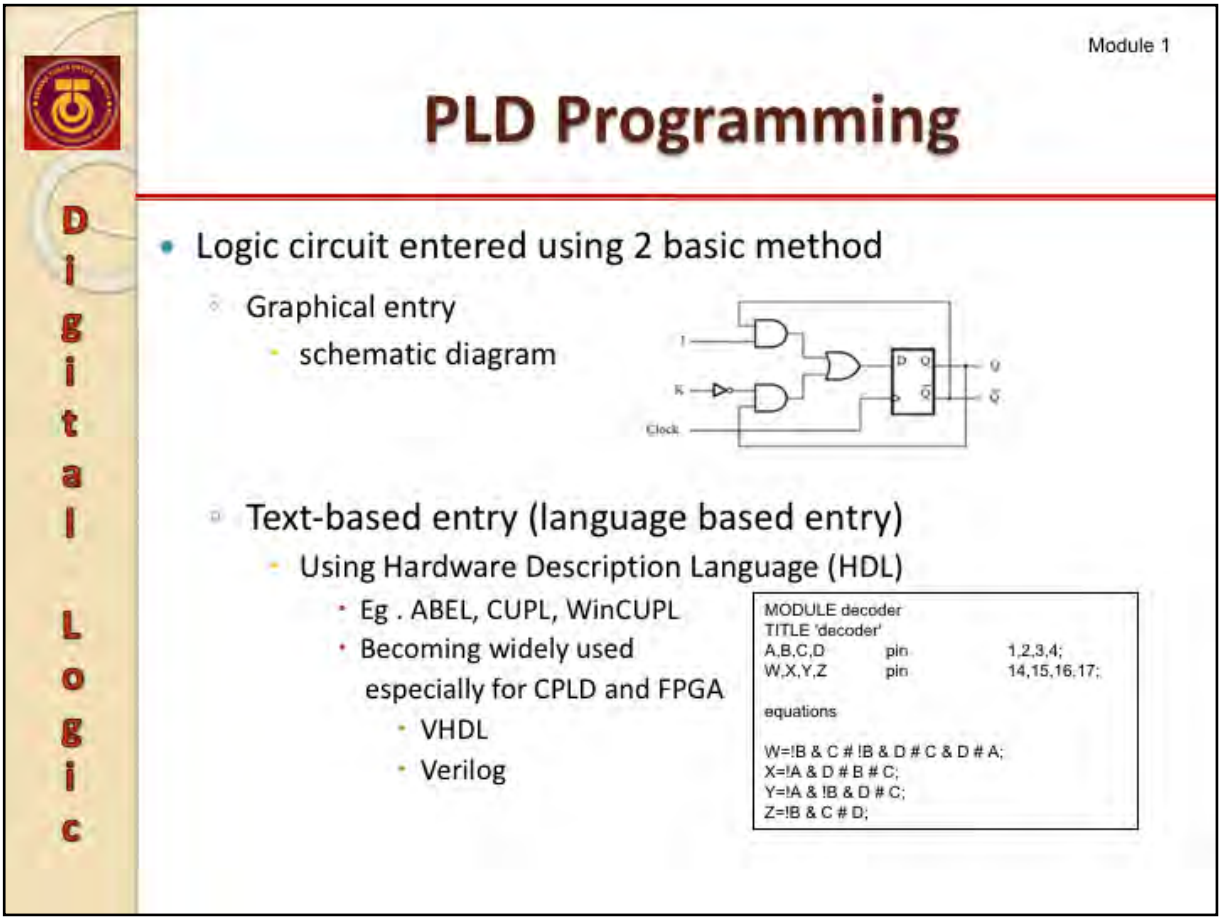
Digital Logic

- 3. **F**ield-**P**rogrammable **G**ate **A**rrays (FPGA)
  - Different internal organization than SPLD and CPLD
  - Greatest logic capacity
    - Consist of 64- thousands logic block (logic gate groups)
  - Classes
    - Fine grain (smaller logic block)
    - Coarse grain (large logic block)

Field-Programmable Gate Array (FPGA) is a semiconductor device that can be configured by the customer or designer after manufacturing—hence the name "field-programmable". FPGAs are programmed using a logic circuit diagram or a source code in a hardware description language (HDL) to specify how the chip will work. They can be used to implement any logical function that an application-specific integrated circuit (ASIC) could perform, but the ability to update the functionality after shipping offers advantages for many applications.

FPGAs contain programmable logic components called "logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together"—somewhat like a one-chip programmable breadboard. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.

An Altera Stratix IV GX FPGA.



Hardware Description Language (HDL)

Hardware description language or HDL is any language from a class of computer languages and/or programming languages for formal description of digital logic and electronic circuits. It can describe the circuit's operation, its design and organization, and tests to verify its operation by means of simulation.

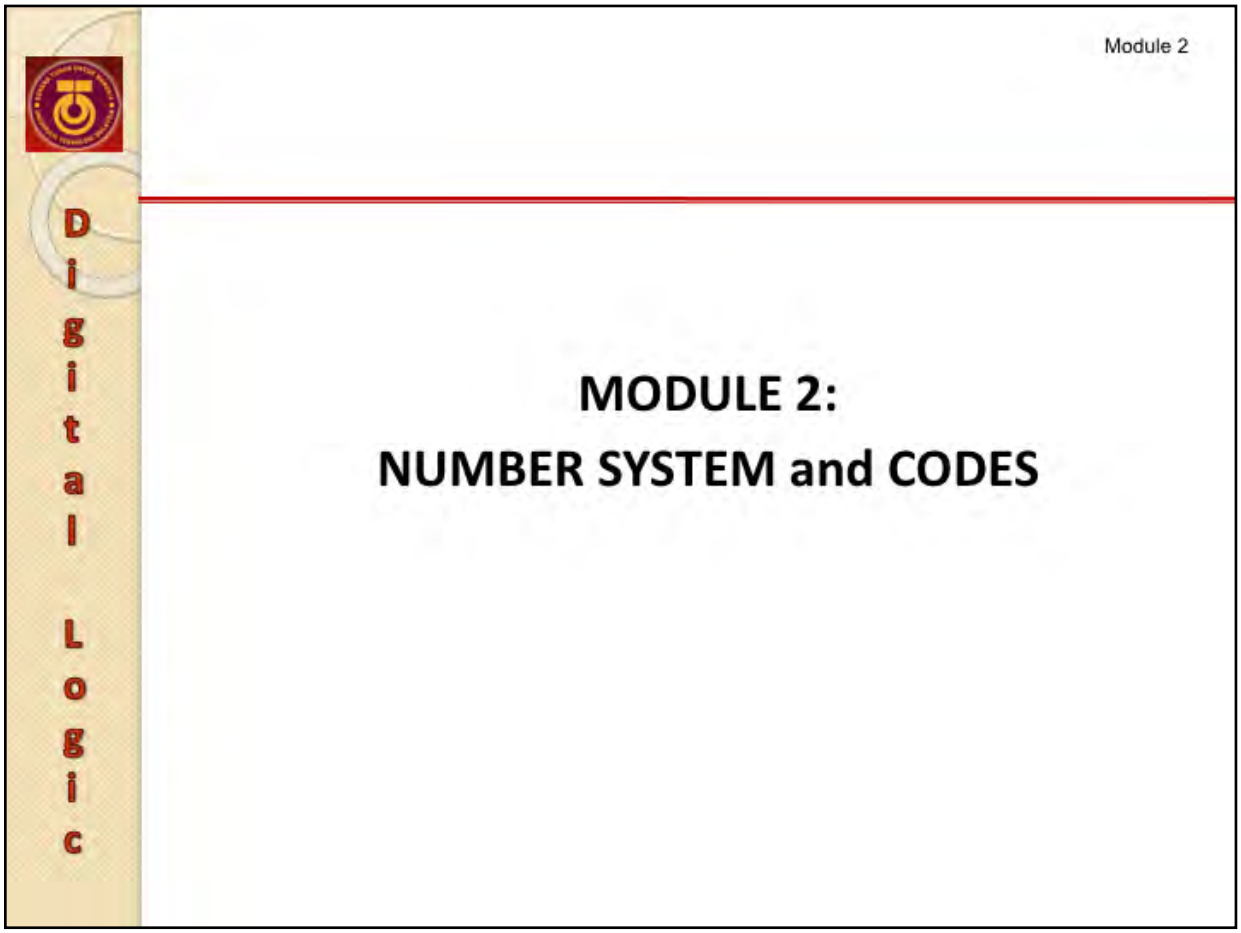
HDLs are standard text-based expressions of the spatial and temporal structure and behavior of electronic systems. In contrast to a software programming language, HDL syntax and semantics include explicit notations for expressing time and concurrency, which are the primary attributes of hardware. Languages whose only characteristic is to express circuit connectivity between a hierarchy of blocks are properly classified as netlist languages used on electric computer-aided design (CAD).

HDLs are used to write executable specifications of some piece of hardware. A simulation program, designed to implement the underlying semantics of the language statements, coupled with simulating the progress of time, provides the hardware designer with the ability to model a piece of hardware before it is created physically. It is this executability that gives HDLs the illusion of being programming languages. Simulators capable of supporting discrete-event (digital) and continuous-time (analog) modeling exist, and HDLs targeted for each are available.

- SPLD/CPLD - Tools & Software:
  - ABEL
  - CUPL
  - WinCUPL
- Other tools that becoming a widely used especially for CPLD and FPGA are:
  - VHDL
  - Verilog

Digital Logic

End of Chapter 1



There are two parts in this module that are:

- Part 1 - Numbering systems and Codes
- Part 2 - Data representation and Basic Arithmetic Operations

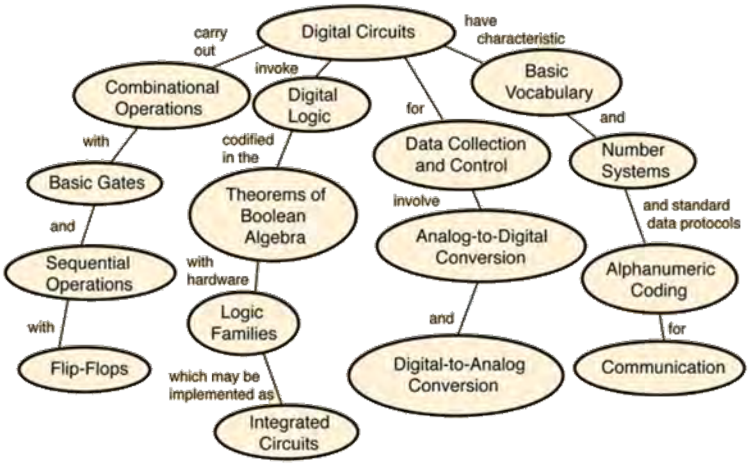
The objectives of this module are:

1 To illustrate the numbering system.
2 To demonstrate the process of converting a value from one numbering system to another.
3 To explain the different coding systems.
4 To illustrate the arithmetic computations.

Module 2, Part 1: Content

Digital Logic

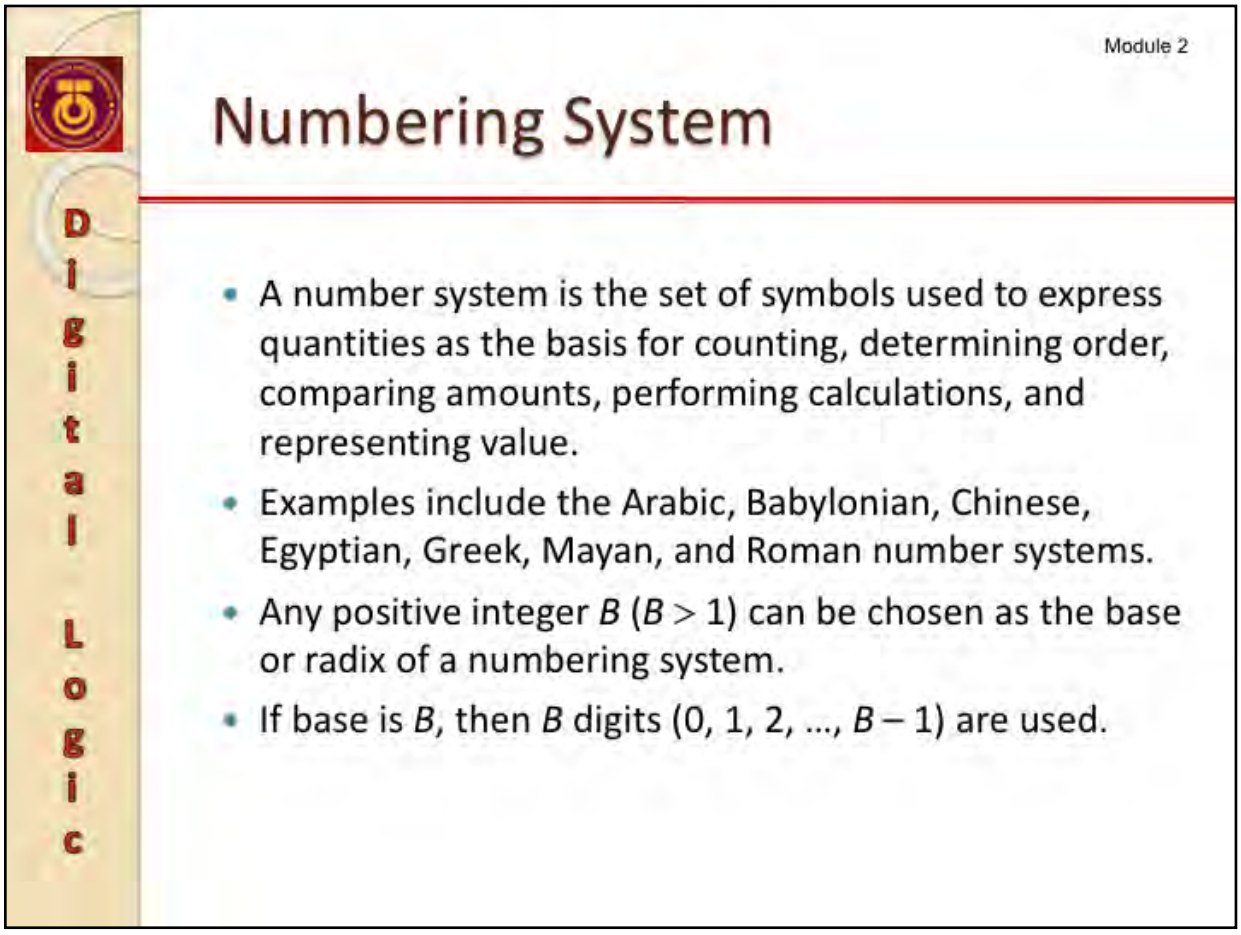
- Decimal Number
- Binary Number
  - Binary-to-Decimal Conversion
  - Decimal-to-Binary Conversion
- Hexadecimal Numbers
- Octal Number
- Binary Coded Decimal (BCD)
- Digital Codes and Parity
- Digital System Application



http://hyperphysics.phy-astr.gsu.edu/hbase/Electronic/ietron/digcktcon.gif

$2^0 = 1$	$2^8 = 256$	$2^{16} = 65{,}536$
$2^1 = 2$	$2^9 = 512$	
$2^2 = 4$	$2^{10} = 1{,}024$	
$2^3 = 8$	$2^{11} = 2{,}048$	
$2^4 = 16$	$2^{12} = 4{,}096$	
$2^5 = 32$	$2^{13} = 8{,}192$	
$2^6 = 64$	$2^{14} = 16{,}384$	
$2^7 = 128$	$2^{15} = 32{,}768$	

$2^{11} = 2K = 2{,}048$	$2^{14} = 16K = 16{,}384$
$2^{12} = 4K = 4{,}096$	$2^{15} = 32K = 32{,}768$
$2^{13} = 8K = 8{,}192$	$2^{16} = 64K = 65{,}536$
$2^{21} = 2M$	
$2^{22} = 4M$	

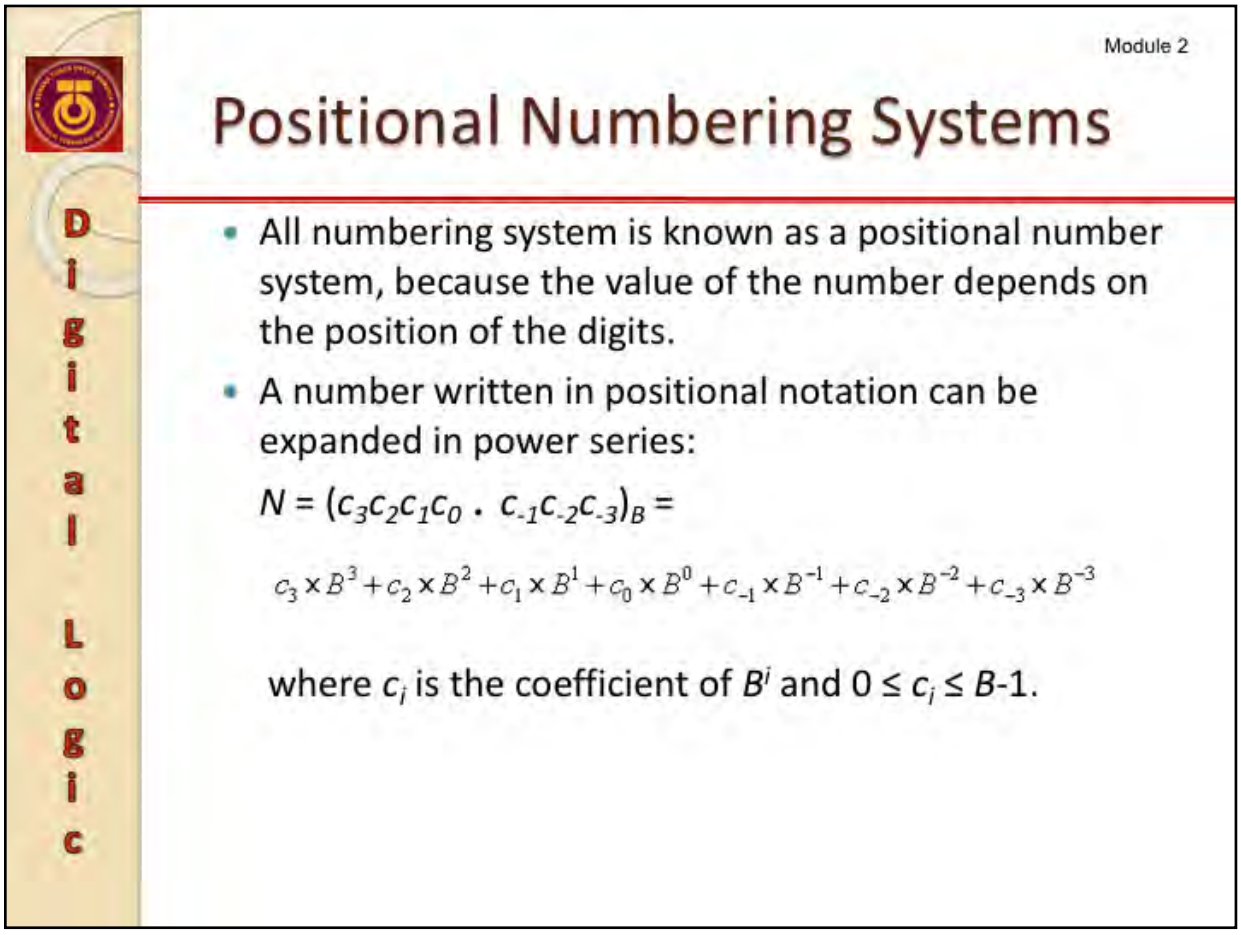


A number system is the set of symbols used to express quantities as the basis for counting, determining order, comparing amounts, performing calculations, and representing value. It is the set of characters and mathematical rules that are used to represent a number. Examples include the Arabic, Babylonian, Chinese, Egyptian, Greek, Mayan, and Roman number systems.

Example of numbering systems are as shown in the table below:

Base/Radix	Name	Numerals
2	Binary	0, 1
3	Trinary	0, 1, 2
4	Quaternary	0, 1, 2, 3
5	Quinary	0, 1, 2, 3, 4
8	Octal	0, 1, 2, 3, 4, 5, 6, 7
10	Decimal	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
16	Hexadecimal	0, 1, 2, 3, 4, 5, 6, 7, 8, 9 A, B, C, D, E, F

Complete list can be accessed at http://en.wikipedia.org/wiki/Numeral_system



Positional Number Systems

All numbering system is known as a positional number system, because the value of the number depends on the position of the digits. For example, a decimal number 789 has a very different value than the number 987, although these same digits are used in both numbers.

In a positional numbering system, the value of each digit is determined by which place it appears in the full number. The lowest place value is the rightmost position, and each successive position to the left has a higher place value.

In decimal numbering system, the rightmost position represents the "ones" column, the next position represents the "tens" column, the next position represents "hundreds", and so forth. Therefore, the number 789 represents **7** hundred and **8** tens and **9** ones, whereas the number 987 represents **9** hundreds and **8** tens and **7** one.

Terms:

Base b number: $N = a_{q-1}b^{q-1} + \ldots + a_0b^0 + \ldots + a_{-p}b^{-p}$

$b > 1, \quad 0 <= a_i <= b\text{-}1$

Integer part: $a_{q-1}a_{q-2} \ldots a_0$

Fractional part: $a_{-1}a_{-2} \ldots a_{-p}$

Most significant digit: a_{q-1}

Least significant digit: a_{-p}

Decimal Number

Digital Logic

- The **decimal numbering** system has 10 digits **0** through **9**.
- The decimal numbering system has a base of 10 with each position **weighted** by a **factor** of **10**

$$\ldots 10^5\ 10^4\ 10^3\ 10^2\ 10^1\ 10^0.\ 10^{-1} 10^{-2}\ 10^{-3}\ 10^{-4}\ 10^{-5} \ldots$$

- Example:

  Express decimal 47 as a sum of the values of each digit.

  $$47_{10} = (4 \times 10^1) + (7 \times 10^0) = 40 + 7$$
  $$= 47$$

The values of each position correspond to powers of the base of the numbering system. So for decimal numbering system, which uses base 10, the place values correspond to powers of 10 are shown below:

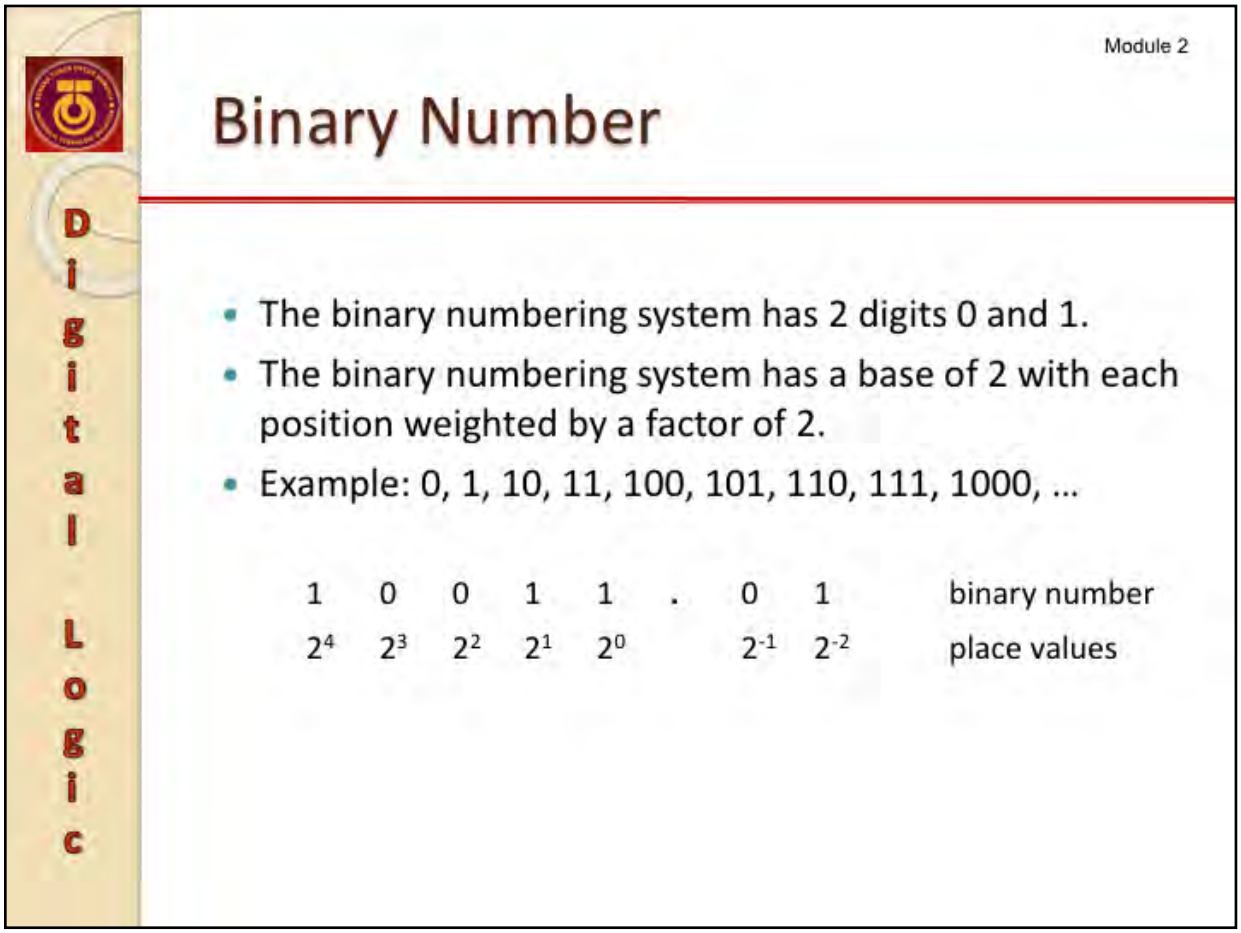
10000	1000	100	10	1	0.1	0.01	Decimal values
10^4	10^3	10^2	10^1	10^0	10^{-1}	10^{-2}	positional values

Example:

Express 1024.68_{10} as a sum of values of each digit

1	0	2	4 .	6	8	number
10^3	10^2	10^1	10^0	10^{-1}	10^{-2}	positional values

$$1024.68_{10} = (1 \times 10^3) + (0 \times 10^2) + (2 \times 10^1) + (4 \times 10^0) + (6 \times 10^{-1}) + (8 \times 10^{-2})$$
$$= (1 \times 1000) + (0 \times 100) + (2 \times 10) + (4 \times 1) + (6 \times 0.1) + (8 \times 0.01)$$
$$= (1000) + (0) + (20) + (4) + (0.6) + (0.08)$$



The binary number system is the most important numbering system in digital systems, beside octal and hexadecimal.

The binary number system uses base 2, so the place values of the digits of a binary number correspond to the power of 2.

For example, the value of the binary number 10011.01_2 is determined by computing the place value of each of the digits of the number the following table.

1	0	0	1	1 .	0	1	binary number
2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	positional values
10000_2	1000_2	100_2	10_2	1_2	0.1_2	0.01_2	binary weight values
16	8	4	2	1	0.5	0.25	decimal values

$$10011.01_2 = (1 \times 2^4) + (0 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) + (0 \times 2^{-1}) + (1 \times 2^{-2})$$
$$= (1 \times 10000_2) + (0 \times 1000_2) + (0 \times 100_2) + (1 \times 10_2) + (1 \times 1_2) + (0 \times 0.1_2) + (1 \times 0.01_2)$$

$$10011.01_2 = (1 \times 2^4) + (0 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) + (0 \times 2^{-1}) + (1 \times 2^{-2})$$
$$= (1 \times 16) + (0 \times 8) + (0 \times 4) + (1 \times 2) + (1 \times 1) + (0 \times 0.5) + (1 \times 0.25)$$
$$= 16 + 2 + 1 + 0.25$$

Octal Number

Digital Logic

- Uses base 8.
- Includes only the digits 0 through 7.
- Based on the binary system with a 3-bit boundary.
- The binary numbering system has a base of 8 with each position **weighted** by a **factor** of **8**.
- Example: 0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 20, …

4	5	3	6	0	.	7	2	octal number
8^4	8^3	8^2	8^1	8^0		8^{-1}	8^{-2}	place values

Octal is sometimes used in computing, perhaps most often in modern times in conjunction with file permissions under Unix systems (see chmod). It has the advantage of not requiring any extra symbols as digits (the hexadecimal system is base-16 and therefore the system needs six additional symbols beyond 0–9). It is also used for digital displays.

At the time when octal originally became widely used in computing, systems such as the ICL 1900 and IBM mainframes employed 24-bit (or 36-bit) words. Octal was an ideal abbreviation of binary for these machines because eight (or twelve) digits could concisely display an entire machine word (each octal digit covering three binary digits). It also cut costs by allowing Nixie tubes, seven-segment displays and calculators to be used for the operator consoles; where binary displays were too complex to use, decimal displays needed complex hardware to convert radixes, and hexadecimal displays needed to display letters.

The value of the octal number 3706.01_8 is determined by computing the place value of each of the digits of the number as the table below:

3	7	0	6 .	0	1	octal number
8^3	8^2	8^1	8^0	8^{-1}	8^{-2}	positional values
1000_8	100_8	10_8	1_8	0.1_8	0.01_8	octal weighted values
512	64	8	1	0.125	0.015625	decimal values

$$3706.01_8 = (3 \times 8^3) + (7 \times 8^2) + (0 \times 8^1) + (6 \times 8^0) + (0 \times 8^{-1}) + (1 \times 8^{-2})$$
$$= (3 \times 1000_8) + (7 \times 100_8) + (0 \times 10_8) + (6 \times 1_8) + (0 \times 0.1_8) + (1 \times 0.01_8)$$

$$3706.01_8 = (3 \times 8^3) + (7 \times 8^2) + (0 \times 8^1) + (6 \times 8^0) + (0 \times 8^{-1}) + (1 \times 8^{-2})$$
$$= (3 \times 512) + (7 \times 64) + (0 \times 8) + (6 \times 1) + (0 \times 0.125) + (1 \times 0.015625)$$

Module 2

Hexadecimal Number System

- Allows for convenient handling of long binary strings.
- Base 16
  - 16 possible symbols
  - 0-9 and A-F
  - $(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)_{16}$
    $(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15)_{10}$
  - 0, 1, ..., F, 10, 11, 12, ..., 1F, 20, 21, ...

7	9	C	F	3	.	A	1	hex number
16^4	16^3	16^2	16^1	16^0		16^{-1}	16^{-2}	place values

Its primary use is as a human friendly representation of binary coded values, so it is often used in digital electronics and computer engineering.

Most digital systems deal with groups of bits in even powers of 2 such as 8, 16, 32, and 64 bits.

Hexadecimal uses groups of 4 bits.

The value of the octal number $A21C.D_{16}$ is determined by computing the place value of each of the digits of the number as in the following table:.

A	2	1	C .	D	hexadecimal number
16^3	16^2	16^1	16^0	16^{-1}	positional values
1000_{16}	100_{16}	10_{16}	1_{16}	0.1_{16}	hexadecimal weighted values
4096	256	16	1	0.0625	decimal values

$$A21C.D_{16} = (A \times 16^3) + (2 \times 16^2) + (1 \times 16^1) + (C \times 16^0) + (D \times 16^{-1})$$
$$= (A \times 1000_{16}) + (2 \times 100_{16}) + (1 \times 10_{16}) + (C \times 1_{16}) + (D \times 0.1_{16})$$

$$A21C.D_{16} = (A \times 16^3) + (2 \times 16^2) + (1 \times 16^1) + (C \times 16^0) + (D \times 16^{-1})$$
$$= (A \times 4096) + (2 \times 256) + (1 \times 16) + (C \times 1) + (D \times 0.0625)$$
$$= (10 \times 4096) + (2 \times 256) + (1 \times 16) + (12 \times 1) + (13 \times 0.0625)$$

Convert From Any Base To Decimal

Digital Logic

- Recall: $N = (c_3c_2c_1c_0 \,.\, c_{-1}c_{-2}c_{-3})_B =$

$$c_3 \times B^3 + c_2 \times B^2 + c_1 \times B^1 + c_0 \times B^0 + c_{-1} \times B^{-1} + c_{-2} \times B^{-2} + c_{-3} \times B^{-3}$$

where c_i is the coefficient of B^i and $0 \le c_i \le B\text{-}1$.

- The summation of the equation is the value in decimal.

Converting from the other bases to decimal.

Example 1: 10011.001_2 = _____$_{10}$

=. $(1 \times 2^4) + (0 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) + (0 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3})$

= $16 + 0 + 0 + 2 + 1 + 0.125 = 19.125_{10}$

Example 2: 2132.413_5 = _____$_{10}$

= $(2 \times 5^3) + (1 \times 5^2) + (3 \times 5^1) + (2 \times 5^0) + (4 \times 5^{-1}) + (1 \times 5^{-2}) + (3 \times 5^{-3})$

= $(2 \times 125) + (1 \times 25) + (3 \times 5) + (2 \times 1) + (4 \times 0.2) + (1 \times 0.04) + (3 \times 0.008)$

= $250 + 25 + 15 + 2 + 0.8 + 0.04 + 0.024 = 290.864_{10}$

Example 3: 6217_8 = _____ $_{10}$

=. $(6 \times 8^3) + (2 \times 8^2) + (1 \times 8^1) + (7 \times 8^0)$

= $(6 \times 512) + (2 \times 64) + (1 \times 8) + (7 \times 1)$

= $4096 + 128 + 8 + 7 = 4239_{10}$

Example 4: $53D.F1_{16}$ = _____ $_{10}$

=. $(5 \times 16^2) + (3 \times 16^1) + (13 \times 16^0) + (15 \times 16^{-1}) + (1 \times 16^{-2})$

= $(5 \times 256) + (3 \times 16) + (13 \times 1) + (15 \times 0.0625) + (1 \times 0.00390625)$

= $1280 + 48 + 13 + 0.937744140625 + 0.00390625 = 1341.941_{10}$

Simple Deduction: Binary Number

Digital Logic

- Fill in the blank spaces.

(a)			(b)		(c)	
Binary	**Decimal**	2^x	**Binary**	**Decimal**	**Binary**	**Decimal**
	2			1	10**1**	5
	4			3		10
1000	8	2^3		7		17
	16			15		26
	32		11111	31		35

a) Fill in the values for all the blank spaces in the table above. Show all your workings below.

b) Based on the answers that you have calculated in the table, select the correct answer below that best described the deduced observation for (a), (b) and (c).

- An even number will have a zero as the last bit while an odd number will have a one as the last bit.
- The power of two is equivalent to the number of zeroes in the binary representation number.
- A binary number that is equal to $2^x - 1$, will consist of all ones.

Exercises:

Based on the observation above, find the binary representation of:

i. $2^9 =$

ii. $2^{12} =$

iii. $1023_{10} =$

iv. $8191_{10} =$

v. $24_{10} =$

vi. $33_{10} =$

Conversion of Decimal to Other Number Bases

Digital Logic

- Apply method of successive division.
  - Divide the decimal number by the base of the converted value and get the quotient and remainder.
  - Successively divide the quotients and keep the remainder until the quotient is 0. The answer is the string of remainders (read from bottom to up)

Example 1: 82_{10} = _____ $_8$

82/8	=	10	remainder 2
10/8	=	1	remainder 2
1/8	=	0	remainder 1

The answer is formed by taking the remainders in reverse order: 122_8

Example 2: 42_{10} = _____ $_2$

42/2	=	21	remainder 0
21/2	=	10	remainder 1
10/2	=	5	remainder 0
5/2	=	2	remainder 1
2/2	=	1	remainder 0
1/2	=	0	remainder 1

The answer is formed by taking the remainders in reverse order: 101010_2

Example 3: 600_{10} = _____ $_{16}$

600/16	=	37	remainder 8
37/16	=	2	remainder 5
2/16	=	0	remainder 2

The answer is formed by taking the remainders in reverse order: 258_{16}

Module 2

Hex, Octal, Binary and Decimal Numbering System

Digital Logic

Hexadecimal	Octal	Binary	Decimal
0	0	0000	0
1	1	0001	1
2	2	0010	2
3	3	0011	3
4	4	0100	4
5	5	0101	5
6	6	0110	6
7	7	0111	7
8	10	1000	8
9	11	1001	9
A	12	1010	10
B	13	1011	11
C	14	1100	12
D	15	1101	13
E	16	1110	14
F	17	1111	15

Example 4: 32_{10} = _____ $_2$

32/2	=	16	remainder 0
16/2	=	8	remainder 0
8/2	=	4	remainder 0
4/2	=	2	remainder 0
2/2	=	1	remainder 0
1/2	=	0	remainder 1

The answer is formed by taking the remainders in reverse order: 100000_2

Example 5: 512_{10} = _____ $_8$

512/8	=	64	remainder 0
64/8	=	8	remainder 0
8/8	=	1	remainder 0
1/8	=	0	remainder 1

The answer is formed by taking the remainders in reverse order: 1000_8

Example 6: 2047_{10} = _____ $_{16}$

2047/16	=	127	remainder 15 = F
127/16	=	7	remainder 15 = F
7/16	=	0	remainder 7

The answer is formed by taking the remainders in reverse order: $7FF_{16}$

Conversion of Fractions to Other Numbering System

Digital Logic

- Repetitive multiplication
  - Step 1:
    Multiply the fraction number by base of the required numbering system
  - Step 2:
    Separate the whole (part of the answer) and the fraction.
  - Step 3:
    Repeat (1) with the new fraction from (2)
  - Stop when the answer of the multiplication = 0
  - Or until reaching the desired fractional point

Example 1: 0.3125_{10} = _____ $_2$	**Example 2:** 0.6378_{10} = _____ $_5$
0.3125×2 = **0**.625 → **0**	0.6378×5 = **3**.189 → **3**
0.625×2 = **1**.25 → **1**	0.189×5 = **0**.945 → **0**
0.25×2 = **0**.5 → **0**	0.945×5 = **4**.725 → **4**
0.5×2 = **1**.0 → **1**	0.725×5 = **3**.625 → **3**
$0.3125_{10} = .0101_2$	$0.6378_{10} = .3043_5$
Example 3: 0.2340_{10} = _____ $_8$	**Example 4:** 0.798_{10} = _____ $_{16}$
0.2340×8 = **1**.872 → **1**	0.798×16 = **12**.768 → **12 = C**
0.872×8 = **6**.976 → **6**	0.768×16 = **12**.288 → **12 = C**
0.976×8 = **7**.808 → **7**	0.288×16 = **4**.608 → **4**
0.808×8 = **6**.464 → **6**	0.608×16 = **9**.728 → **9**
$0.2340_{10} = .1676_8$	$0.798_{10} = .CC49_{16}$

Whole and Fraction Conversion

Digital Logic

- Given a number $(c_3c_2c_1c_0.c_{-1}c_{-2}c_{-3})_B$
- To convert the number to the base x:
  - Successive division for $c_3c_2c_1c_0$ by x
  - Successive multiplication for $c_{-1}c_{-2}c_{-3}$ by x

Example 1: 96.6875_{10} =_____$_2$

$96.6875_{10} = 96 + 0.6875$

Part 1: Calculate the whole

96/2 =	48	remainder 0
48/2 =	24	remainder 0
24/2 =	12	remainder 0
12/2 =	6	remainder 0
6/2 =	3	remainder 0
3/2 =	1	remainder 1
1/2 =	0	remainder 1

$96_{10} = 1100000$

Part 2: Calculate the fraction

0.6875×2	=	1.375	→ 1
0.375×2	=	0.75	→ 0
0.75×2	=	1.5	→ 1
0.5×2	=	1.0	→ 1

6875_{10} = .1011

$96.6875_{10} = 1100000.1011_2$

Example 2: 128.5_{10} =_____$_{16}$

$128.5 = 128 + 0.5$

Part 1: Calculate the whole

128/16 =	8	remainder 0
8/16 =	0	remainder 8

$128_{10} = 80_{16}$

Part 2: Calculate the fraction

0.5×16	=	8.0	→ 8
0.5_{10}	=	.8	

$128.5_{10} = 80.8_{16}$

Binary to Octal & Hex Conversion

Digital Logic

- Convert from binary to octal by grouping bits in three starting with the LSB.
  - Each group is then converted to the octal equivalent.
- Convert from binary to hex by grouping bits in four starting with the LSB.
  - Each group is then converted to the hex equivalent.
- Leading zeros can be added to the left of the MSB to fill out the last group.

Example 1: 11101101_2 = _____ $_8$

Grouping bits in three starting with the LSB.

011	101	101
3	5	5

$11101101_2 = 355_8$

Example 2: 10011001110_2 = _____ $_{16}$

Grouping bits in four starting with the LSB.

0100	1100	1110
4	12 = C	14 = E

$10011001110_2 = 4CE_{16}$

Binary Fraction to Octal & Hex Conversion

Digital Logic

- For fractional binary number, the grouping of bits start from the radix point :
  - Octal: 3-bit group
  - Hexadecimal: 4-bit group
  - Add the necessary number of 0's
- For whole and fraction binary, the process is divided into two steps:
  - Step 1: Group the whole (integer part) portion starting from the radix point and moving to the left. Add the necessary number of 0's to the left.
  - Step 2: Group the fractional portion starting from the radix point and moving to the right. Add the necessary number of 0's to the right.

Example 1: 0.1100111011_2 = _____ $_8$

Grouping in 3-bits per group starting from the radix point.

110	011	101	**100**
6	3	5	4

$0.1100111011_2 = 0.6354_8$

Example 2: 0.1100111011_2 = _____ $_{16}$

Grouping in 4-bits per group starting from the radix point.

1100	1110	**1100**
12 = C	14 = E	12 = C

$0.1100111011_2 = 0.CEC_{16}$

Example 3: 10001101.1101001_2 = _____ $_{16}$

Part 1:
Group of 4 bits starting from the radix point moving to the left.

1000	1101
8	13 = D

Part 2:
Group of 4 bits starting from the radix point moving to the right.

1101	0010
13 = D	2

$10001101.1101001_2 = 8D.D2_{16}$

Octal & Hex to Binary Conversion

Digital Logic

- Convert every digit into groups of binary bits:
  - Octal: 3 bits
  - Hexadecimal: 4 bits
- When converting octal to hexadecimal and vice-verse, it is advisable to uses binary representative as an intermediate conversion.

Example 1: $124.AB7_{16}$ = _____ $_2$

1	2	4	.	A	B	7
0001	0010	0100	.	1010	1011.	0111

$124.AB7_{16} = 000100100100.101010110111_2$

Example 2: 623.52_8 = _____ $_2$

6	2	3	.	5	3
110	010	011	.	101	011

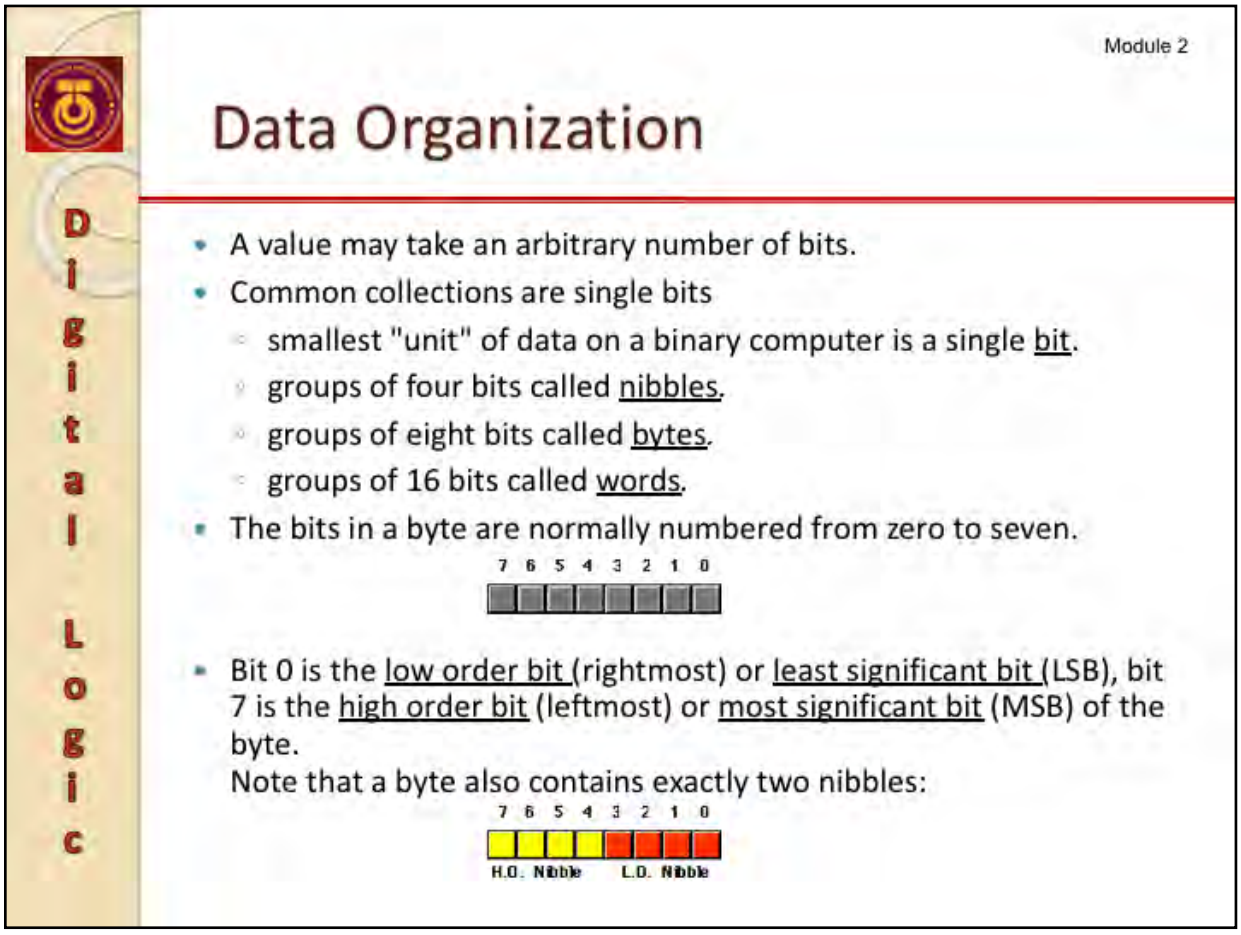
$623.52_8 = 110010011.101011_2$

Example 3: 623.52_8 = _____ $_{16}$

$623.52_8 = 110010011.101011_2$ (Convert to hexadecimal by grouping into 4 bits)

0001	1001	0011	.	1010	11**00**
1	9	3	.	10 = A	12 = C

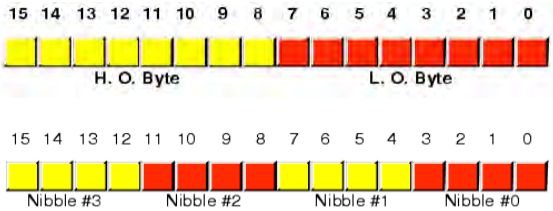
$623.52_8 = 193.AC_{16}$



Nibbles : A *nibble* is a collection of four bits, used for two items that is BCD (*binary coded decimal*) numbers and hexadecimal numbers. Example: 0111, 1011 and 1111.

Bytes : The most important data structure. Since a byte contains eight bits, it can represent 2^8, or 256, different values. Generally, we'll use a byte to represent numeric values in the range 0 ... 255, and signed numbers in the range of -128 to +127.

Words : A word is a group of 16 bits, exactly two bytes. The bits in a word are numbered starting from zero on up to fifteen. Like the byte, bit 0 is the low order bit. For words, bit 15 is the high order bit.



With 16 bits, there can be 2^{16} (65,536) representation of different values. These could be the values in the range 0 ... 65,535 or, as is usually the case, -32,768 to +32,767, or any other data type with no more than 65,536 values. The three major uses for words are signed integer values, unsigned integer values, and UNICODE characters.

What are codes?

Digital Logic

- Code is a representation of information generated by following a certain rules.
- In general, we need code because:
  - Code is unique
  - Code are easier to process
  - Code is easy to represent
  - to enable communication in places where ordinary spoken or written language is difficult or impossible, eg Morse Code
- Due to this, code can simplify the process (such as manipulation and arithmetic operations) of the information in the digital system.

Codes that we learned in this modules are:

i. BCD codes
ii. Gray Codes
iii. ASCII codes
iv. Parity codes/bit

Binary Coded Decimal (BCD)

Digital Logic

- BCD is a way to express each of the decimal digits with a binary code.
- There are only 10 code groups in the BCD system, one for every digit (0000 – 1001)
- Invalid codes are 1010, 1011, 1100, 1101, 1110 and 1111
- Coded values for decimal digit:

Decimal	BCD	Decimal	BCD
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

Binary-coded decimal (BCD) is an encoding for decimal numbers in which each digit is represented by its own binary sequence. Its main virtue is that it allows easy conversion to decimal digits for printing or display and faster decimal calculations. The advantage of BCD has over Binary is that there is no limit to number size. Its drawbacks are the increased complexity of circuits needed to implement mathematical operations and a relatively inefficient encoding—it occupies more space than a pure binary representation. BCD has been in use since the first UNIVAC computer.

Example 1: Convert 3245 to BCD

3245 = 0011 0010 0100 0101

Example 2: Convert 7848 to BCD

7848 = 0111 1000 0100 1000

Example 3: Convert the BCD value 1000100100111 to its equivalent decimal representation.

0001	0001	0010	0111
1	1	2	7

1000100100111 = 1127

Gray Codes

Digital Logic

- Designed to prevent false output from electromechanical switches.
- Are widely used to facilitate error correction in digital communications such as digital terrestrial television and some cable TV systems.
- In modern digital communications, Gray codes play an important role in error correction.
- It is arranged so that every transition from one value to the next value involves only one bit change.
- Sometimes referred to as reflected binary, because the first eight values compare with those of the last 8 values, but in reverse order.

(*http://en.wikipedia.org/wiki/Gray_codes*)

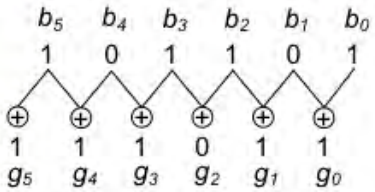
Dec	Gray	Binary
0	000	000
1	001	001
2	011	010
3	010	011
4	110	100
5	111	101
6	101	110
7	100	111

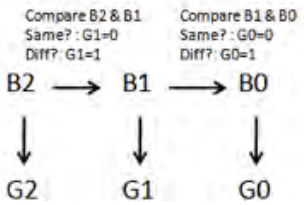
Notice that state 7 can roll over to state 0 with only one switch change. This is called the "cyclic" property of a Gray code. A good way to remember gray coding is by being aware that the least significant bit follows a repetitive pattern of 2. That is 11, 00, 11 etc. and the second digit follows a pattern of fours.

Conversion of Gray Code ←→ Binary

Digital Logic

To convert binary to gray code:



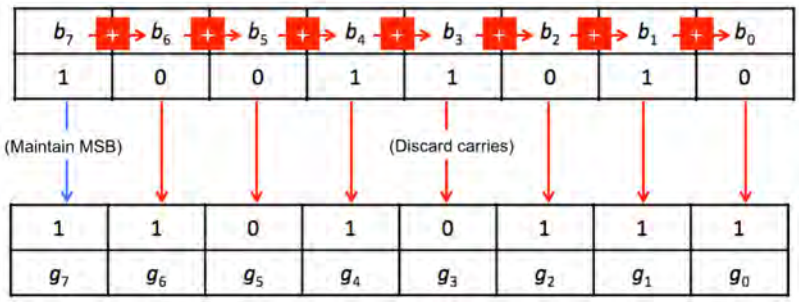


where $b_{i+1}, b_i, b_{i-1}, \ldots, b_0$ is the binary number while $g_{i+1}, g_i, g_{i-1}, \ldots, g_0$

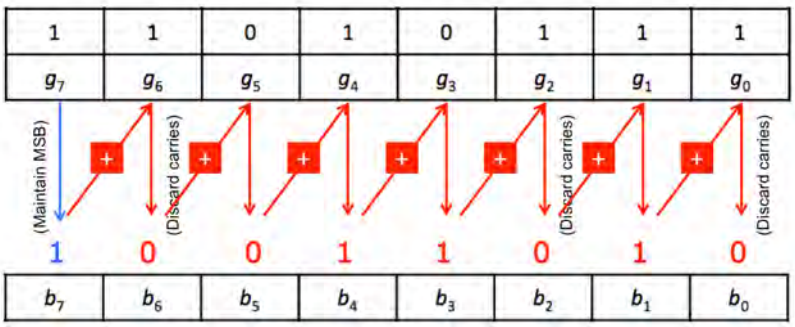
To convert gray code to binary:

- $b_i = g_i$ if no. of 1's preceding g_i is even
- $b_i = \overline{g_i}$ if no. of 1's preceding g_i is odd

Example 1: Convert binary 10011010 to its equivalent gray code value.



Example 2: Convert the Gray code 11010111 to binary.



Parity Code

Digital Logic

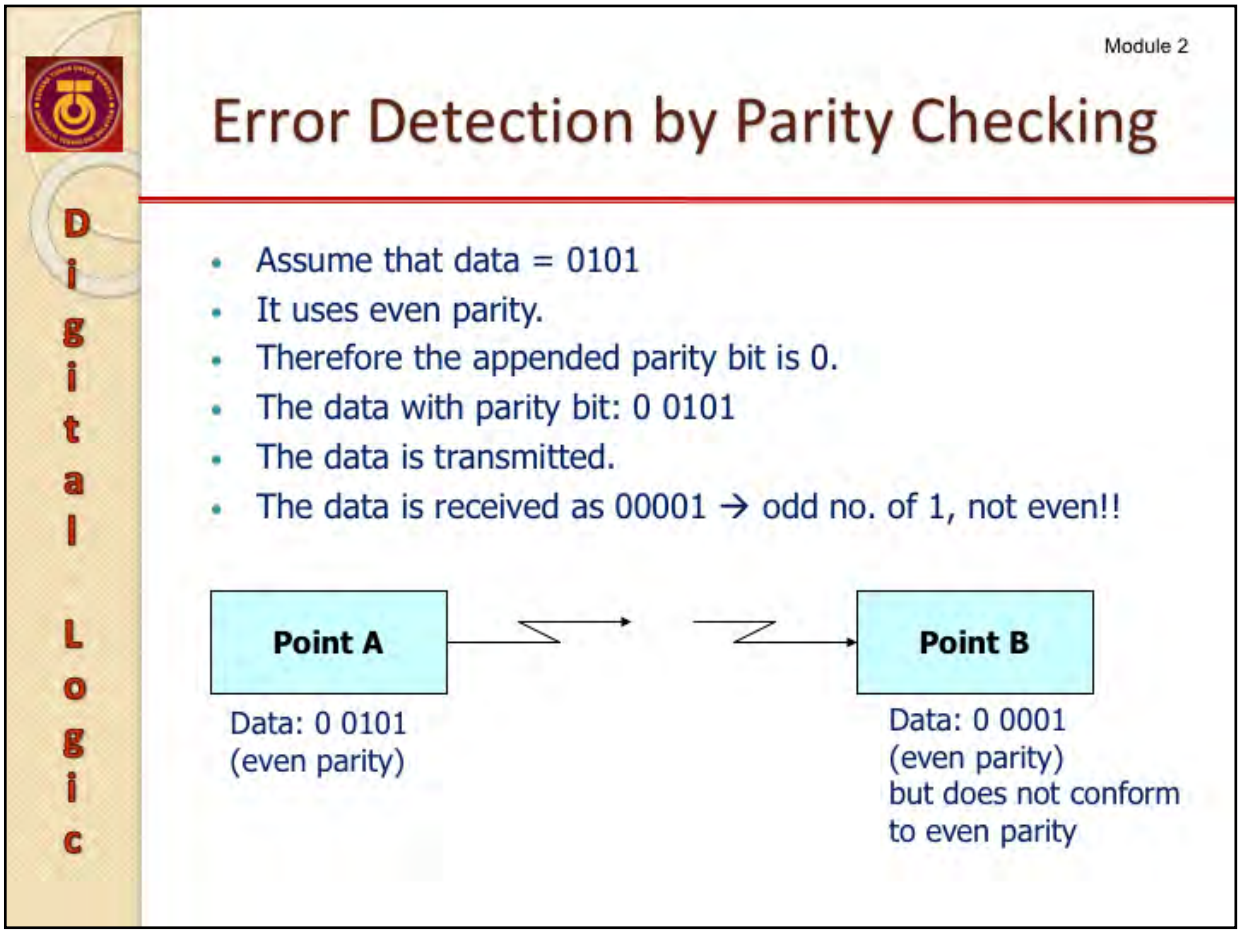
- Parity bit used for bit error detection
  - Even parity – total number of 1s even
  - Odd parity – total number of 1s odd
- Parity bit is append to the code at the leftmost position (MSB) .
- Example: Calculate the parity bit for the codes below.

Code	Number of 1s	Even/Odd	Even Parity	Odd Parity
110010	3	Odd	**1** 110010	**0** 110010
101110	4	Even	**0** 101110	**1** 101110
101000	2	Even	**0** 101000	**1** 101000
110111	5	Odd	**1** 110111	**0** 110111
111111				
100000				

A **parity bit** is a bit that is added to ensure that the number of bits with value of 1's in a given set of bits is always even or odd. Parity bits are used as the simplest error detecting code.

There are two variants of parity bits: **even parity** bit and **odd parity** bit. An even parity bit is set to 1 if the number of ones in a given set of bits is odd (making the total number of ones, including the parity bit, even). An odd parity bit is set to 1 if the number of ones in a given set of bits is even (making the total number of ones, including the parity bit, odd).

Because of its simplicity, parity is used in many hardware applications where an operation can be repeated in case of difficulty, or where simply detecting the error is helpful.



Every byte of data that is stored contains 8 bits of real data, each one a zero or a one. It is possible to count up the number of zeros or ones in a byte. For example, the byte 10110011 has 3 zeros and 5 ones. The byte 00100100 has 6 zeros and 2 ones. Therefore it can be seen here that some bytes will have an even number of ones, and some will have an odd number.

When parity checking is enabled, each time a byte is written to memory, a logic circuit called a parity generator/checker examines the byte and determines whether the data byte had an even or an odd number of ones.

What happens if there is an error in two of the bits? Let's say we have a data byte of 00100100 and this is stored as "00100100 1" including the parity bit. Now let's say this is read back as "01100000 1". Here we have two bits that have flipped, one of them from 1 to 0 and the other from 0 to 1. But the number of ones is still odd! As you can see, parity does not protect against double-bit errors.

Module 2

American Standard Code for Information Interchange (ASCII)

Digital Logic

- It has 128 characters and symbols represented in 7-bit binary code
- Example :
- $A = 1000001_2$
- $a = 1100001_2$
- A <u>parity bit</u> is added so that the total number of bits is 8 → a byte.

There is a need to represent alphanumeric data. There are several representation such as :

i. ASCII code - the common code used in the computer system.

ii. EBCDIC - Used only on IBM mainframe.

iii. Unicode - The latest standard that can support multilingual character (Arabic, Chinese, Japanese, Russian, etc.).

The standard ASCII character set uses just 7 bits for each character. ASCII includes definitions for 128 characters: 33 are non-printing, mostly obsolete control characters that affect how text is processed; 94 are printable characters, and the space is considered an invisible graphic.

There are several larger character sets that use 8 bits. The extra characters are used to represent non-English characters, graphic symbols, and mathematical symbols. Several companies and organizations have proposed extensions for these 128 characters. The DOS operating system uses a superset of ASCII called extended ASCII.

Text files stored in ASCII format are sometimes called ASCII files. Text editors and word processors are usually capable of storing data in ASCII format, although ASCII format is not always the default storage format. Most data files, particularly if they contain numeric data, are not stored in ASCII format. Executable programs are never stored in ASCII format.

ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Convert the string SCR1013 to its ASCII hexadecimal value.

SCSR1013 = 53 43 53 52 31 30 31 33

By using even parity coding, calculate the parity bit and insert this bit at the MSB position. Recalculate the ASCII value in its hexadecimal representation.

Character	ASCII Hexadecimal	Binary	Parity Bit
S	53	101 0011	**0** 101 0011 = 53
C	43	100 0011	**1** 100 0011 = C3
S	53	101 0011	**0** 101 0011 = 53
R	53	101 0010	**1** 101 0010 = D2
1	31	011 0001	**1** 011 0001 = B1
0	30	011 0000	**0** 011 0000 = 30
1	31	011 0001	**1** 011 0001 = B1
3	33	011 0011	**0** 011 0011 = 33

SCSR1013 = 53 C3 53 D2 B1 30 B1 33

Part 2: Arithmetic Operations

Digital Logic

- Integer Numbers
  - Unsigned Numbers
  - Signed Numbers
- Addition
- Subtraction

Integer Representation

Digital Logic

- Numbers can be represented as a combination of a value, or magnitude and sign, plus or minus
- Unsigned integer
- Signed integer

Module 2

Unsigned Integer Data

- By unsigned integer, it is mean no negative values.
  - E.g. 0, 1, 2, ..., 254, 255, 256, 257, ... 65535, 65536, 65537, ..., 2000000000, 2000000001, ...
- A **bit** can store unsigned integers from 0 to 1 .
- A **byte** of 8 bits can store unsigned integers from 0 to 255 = $2^8 - 1$.
- In binary arithmetic, if the length of the number is restricted to 8 digits (0s and 1s), the largest value is $1111\ 1111_2$ = 255, and the smallest is 0.
- A **word** of 16 bits can store unsigned integers from 0 to 65535 = $2^{16} - 1$.
- In binary arithmetic, if the length of the number is restricted to 16 digits (0s and 1s), the largest value is $1111\ 1111\ 1111\ 1111_2$ = 65535, and the smallest is 0.

Digital Logic

An integer is a number with no fractional part example 001001, 10111.

The range of number depend on the total number of bits used, *n*.

For positive number yang range is from 0 to 2^n-1.

If a number system comprises of 6 bits, then the number will start at 00 0000 until 11 1111. In other words the range of numbers that can be represented by the this particular numbering system is 00 0000 (0) until 11 1111 (63).

Example: Find the range of binary numbers that can be represented by 10 bits.

Number of bits, $n = 10$

$$00\ 0000\ 0000 \leq x \leq 11\ 1111\ 1111$$

$$0 \leq x \leq 2^{10}-1$$

$$0 \leq x \leq 1023$$

Upper and Lower Bound

Digital Logic

No of Bits	Lower Bound	Upper Bound, $2^n - 1$	Range
4 bits	0	$2^4 - 1 = 15$	0 → 15
8 bits	0	$2^8 - 1 = 255$	0 → 255
10 bits	0	$2^{10} - 1 = 1023$	0 → 1023

Example 1: Find the lower and the upper bound of a 12-bit binary system.

-Lower bound = 0

-Upper bound $=2^n - 1 = 2^{12} - 1 = 4096 - 1 = 4095$

-Therefore the range is 0 → 4095

Example 2: Determine how many bits required to represent 34 in binary.

From the notes (pg 76), $2^n \geq 34$

$\log(2^n) \geq \log(34)$

$n\log 2 \geq \log(34)$

$n \geq (\log(34) / \log 2) \geq 5.087 = 6$

Therefore 6 bits is required to represent 34 in binary.

Signed Numbers

Digital Logic

- Integers that do not have a sign indication are considered as positive numbers and they are referred to as unsigned numbers.
  - 01000, 11101
- However, integers can be positive and negative.
  - +01000, +11101, -10001, -0111001
  - Need for a code to represent '-' and '+'.
- Positive and negative integers use a code system to indicate the sign.
  - Signed bit: **0 (+ve)** or **1 (-ve)** positioned at MSB
  - Positive numbers → 0 01000, 0 11101
  - Negative numbers → 1 10101, 1 0101001
  - This is referred as signed numbers.

Example 1: Change the following decimal numbers to its binary representation.

a) + 4 = + (100) = **0** 100

b) - 12 = - (1100) = **1** 1100

→ This method of representation is called signed and magnitude.

Example 2: Determine if the binary numbers is positive or negative.

a) 0 010001 → Sign bit = 0, therefore number is +ve.

b) 1 0011 → Sign bit = 1, therefore number is –ve.

Signed Numbers Representation

Digital Logic

- Three representations:
  - Sign and magnitude (simple representation)
  - 1's complement
  - 2's complement

Module 2

Sign and Magnitude Representation

Digital Logic

- Simple and fast.
  - Lower bound: - $(2^{n-1} - 1)$
  - Upper bound: $2^{n-1} - 1$
  - Where *n* the total bit
- Example:

  + 01110 = + (01110) = 0 01110

  - 100100 = - (100100) = 1 100100

Decimal	Binary	Sign & Mag
+7	+111	**0** 111
+6	+110	**0** 110
+5	+101	**0** 101
+4	+100	**0** 100
+3	+011	**0** 011
+2	+010	**0** 010
+1	+001	**0** 001
+0	+000	**0** 000
?	?	?
-1	-001	**1** 001
-2	-010	**1** 010
-3	-011	**1** 011
-4	-100	**1** 100
-5	-101	**1** 101
-6	-110	**1** 110
-7	-111	**1** 111

- Use one bit (MSB) to store + or – sign, and the remaining bits to store the magnitude of the number.
- For the sign bit, usually, 0 means plus (+) and 1 means minus (-). This representation is simple and sensible, but there are two zeroes, a +0 and a –0 (a disadvantage).
- The sign must be tested before any addition or subtraction can be carried out, the circuitry to implement sign/magnitude arithmetic is more complicated and the operations are on average slower.

Example 1:

If a number is represented by 11 bits and uses sign and magnitude representation to represent positive and negative numbers. Determine how many bits are assigned for representing the magnitude of the number.

Total bits, n = 11.

One bit is used to represent the sign

Therefore left with (11-1) bits to represent the magnitude.

Example 2:

A 5-bit number system uses sign and magnitude representation for its signed numbers. List all the numbers in the system.

Lower bound = $-(2^{n-1} - 1) = -(2^{5-1} - 1) = -(2^4 - 1) = -(16 - 1) = -15$

Upper bound = $2^{n-1} - 1 = 2^{5-1} - 1 = 2^4 - 1 = 16 - 1 = 15$

The numbers are: **1**1111, **1**1110, **1**1101, …, **1**0000, **0**0000, **0**0001, …, **0**1110, **0**1111

DIY : What is the decimal value of 10001?

Complement of a Number

Digital Logic

- In base *B* arithmetic we can compute two complements: The *B*'s complement, and the (*B*-1)'s complement.
- For binary numbers we can use 2's complement and 1's complement;
- In decimal arithmetic we have 10's complement and 9's complement.
- In general, complements are defined for numbers that have both integer and fractional parts. However in this discussion is restricted to complements of integers and binary numbers only.

The characteristic of a complement is similar to an inverter.

When a positive number is complemented, the result will be a negative number .

When a negative number is complemented, the result will be a positive number .

Complementing twice of a number will result with the same sign as the initial number.

Module 2

1's complement Representation

Digital Logic

- Convert '0' to '1' and '1' to '0'
  - Lower bound: $-(2^{n-1} - 1)$
  - Upper bound: $2^{n-1} - 1$
  - Where *n* the total bit
- Example:

  +01110 = + (01110) = 0 01110

  -100100 = - (100100) = 1 011011

- Example:
  A 4-bit number system uses 1's complement representation for its signed numbers. List all the numbers in the system.
- What is the decimal value of 1111?

Decimal	Binary	1's Comp
+7	+111	0 111
+6	+110	0 110
+5	+101	0 101
+4	+100	0 100
+3	+011	0 011
+2	+010	0 010
+1	+001	0 001
+0	+000	0 000
?	?	?
-1	-001	1 110
-2	-010	1 101
-3	-011	1 100
-4	-100	1 011
-5	-101	1 010
-6	-110	1 001
-7	-111	1 000

The 1's complement of a binary number **N** with **d** digits is $\mathbf{2^d - 1 - N}$.

The 1's complement of 101100 is 1000000 – 1 – 101100 = 111111 – 101100 = 010011 (of course, 1 less than the 2s complement).

101100 has 6 digits, so d = 6, so the number is subtracted from **111111.**

A second way to obtain the 1's complement is: Reverse all 0's and 1's.

The important thing to realise in the preceding discussion is that in all four cases, a subtraction has occurred. This suggests that we can *represent the negative value of a number using its complement.* Notice that taking the complement of the complement restores the original number.

Example 1: Find the 1's complement of the binary numbers.

A = 0100110 1's complement = **1**011001

B = 1111111 1's complement = **0**000000

Note: If MSB is the sign bit, then after converting to its 1's complement, the sign bit will reverse, thus changing a number from positive to negative or negative to positive.

Example 2: Compute -23_{10} to a 7-bit binary system using 1's complement representation.

-(+23) = -(00 10111) = **1**1 01000

2's complement Representation

Digital Logic

- Process:
  - Convert to 1's complement.
  - Add 1.
  - Lower bound: $-(2^{n-1})$
  - Upper bound: $2^{n-1} - 1$
- Example:

  +01110 = + (01110) = 0 01110

  -100100 = - (100100) = 1 011100
- Example:

  A 4-bit number system uses 2's complement representation for its signed numbers. List all the numbers in the system.
- What is the decimal value of 1111 and 1000?

Decimal	Binary	2's Comp
+7	+111	0 111
+6	+110	0 110
+5	+101	0 101
+4	+100	0 100
+3	+011	0 011
+2	+010	0 010
+1	+001	0 001
+0	+000	0 000
-1	-001	1 111
-2	-010	1 110
-3	-011	1 101
-4	-100	1 100
-5	-101	1 011
-6	-110	1 010
-7	-111	1 001
-8	-1000	1 000

Note: If MSB is the sign bit, then after converting to its 2's complement, the sign bit will reverse, thus changing a number from positive to negative or negative to positive.

There are several methods to determine 2's complement:

Method 1: The 2's complement of a binary number **N** with **d** digits is $\mathbf{2^d - N}$
(except that the 2's complement of 0 is 0).

The 2's complement of 101100 is $2^6 - 101100 = 1000000 - 101100 = 010100$.
101100 has 6 digits, so d = 6, so the number is subtracted from **1000000.**

Method 2: Find the 1's complement of the number and then add 1.
The 2's complement of **1**01100 = 010011+ 1 = **0**10100

Method 3: From right to left: leave all trailing zeroes unchanged; keep the first 1; reverse the remaining 0s and 1s.

Example 1: Find the 2's complement of the binary numbers.

A = **0**100110 2's complement = **1**011010

B = **1**111111 2's complement = **0**000001

Example 2: Compute -23_{10} to a 7-bit binary system using 2's complement representation.

-(+23) = -(**0**0 10111) = **1**1 01001

Summarized of Signed Representation

Decimal	Sign & Magnitude	1's Complement	2's Complement
+7	0 111	0 111	0 111
+6	0 110	0 110	0 110
+5	0 101	0 101	0 101
+4	0 100	0 100	0 100
+3	0 011	0 011	0 011
+2	0 010	0 010	0 010
+1	0 001	0 001	0 001
+0	0 000	0 000	0 000
-0	1 000	1 111	-
-1	1 111	1 110	1 111
-2	1 110	1 101	1 110
-3	1 101	1 100	1 101
-4	1 100	1 011	1 100
-5	1 011	1 010	1 011
-6	1 010	1 001	1 010
-7	1 001	1 000	1 001
-8	-	-	1 000

Although sign-magnitude and 1's complement is easier to compute, all arithmetic operations in the computer is performed by using 2's complement. This is because there are two representation of 0 in sign-magnitude (0 000 & 1 000) and 1's complement (0 000 & 1 111) whereas in 2's complement there is only one representation of 0. In addition there is an extra negative value that can be represented using 2's complement.

Example 1: Find 2's complement of -42 using 8-bit number system.

- 42 = - (+42) = - (010 1010) = 101 0110

Extend the sign-bit to form 8 bit number → 1101 0110

Example 2: Find 2's complement of -16 using 10-bit number system.

- 16 = - (+16) = - (01 0000) = 11 0000

Extend the sign-bit to form 8 bit number → 111111 0000

Example 3: Find 2's complement of 12_{10} using 6-bit number system.

+12 = (0 1100)

Extend the sign-bit to form 8 bit number → 00 1100

Example 4: Determine the decimal value of a 2's complement 1101 0110

1101 0110 = - (0010 1010) = -42

Arithmetic Operation: Addition

Digital Logic

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	10

- Example:

```
    1 0 0 1 0
+   0 1 1 0 0
-----------
    1 1 1 1 0
```

```
      1   1
    1 0 1 1 0
+   0 1 1 0 0
-----------
  1 0 0 0 1 0
```

Example:

Use 8-bit, 2's complement representation for the following operation:

a) 127 + 74

+127 = **0**111 1111

+74 = **0** 1001010

0111 1111 + **0** 1001010 **= 1**1001001 (note the sign of the result)

When two positive numbers are added the result should be positive. However the answer above is not positive. This is because the result is out of range of the valid available numbers that can be represented by the numbering system. The actually answer is **0** 11001001 (+201) and it requires 9-bit representation.

Note: The range of 2's complement 8-bit number is between -2^{8-1} to $+2^{8-1}-1$ that is -128 to 127.

b) 60 + (-30)

+60 = **0**011 1100

-30 = -(+30) = - (**0**001 1110)

Use 2's complement : - (**0**001 1110) = (**1**110 0010)

0011 1100 + **1**110 0010 = 1 0001 1110 = 0001 1110 (+30)

(Note that the carry bit form MSB is ignored)

Arithmetic Operation: Subtraction

Digital Logic

- In digital system, subtraction is performed by using 2's complement and addition.
- Carry from the MSB (signed bit) is deleted.
- Example:

010011 – **0**01111 = **0**10011 + (– **0**01111)
= **0**10011 + (110001)
= **0**00100

	1			1	1	
	0	1	0	0	1	1
+	**1**	1	0	0	0	1
~~1~~	**0**	0	0	1	0	0

Examples:

Perform the operations below using 6-bit 2's complement signed number.

(a) 24 – 17 (b) -9 – 15

Solutions:

(a) 24 – 17 = 24 + (-17) = +7

24 = + 24 = **0** 11000

-17 = - (+ 17) = - (**0** 10001) = **1** 01111

24 – 17 = 24 + (-17) = **0** 11000 + **1** 01111 = **0** 00111 = +7

(b) -9 – 15 = -9 + (-15) = -24

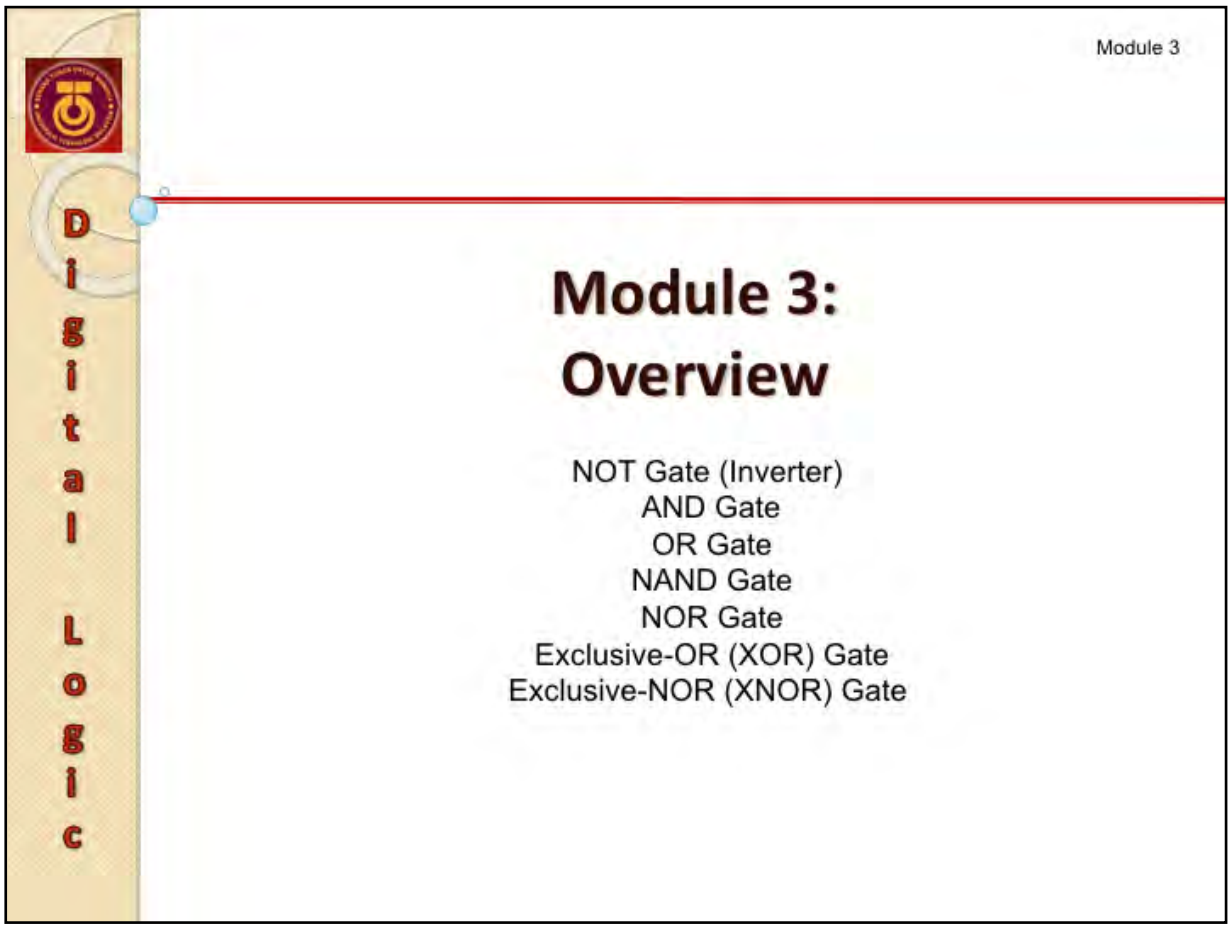
-9 = - (+9) = - (**0** 1001) = **1** 0111 = **11** 0111

-15 = - (+15) = - (**0** 1111) = **1** 0001 = **11** 0001

-9 – 15 = -9 + (-15) = **11** 0111 + **11** 0001 = **1**0 1000 = - 24

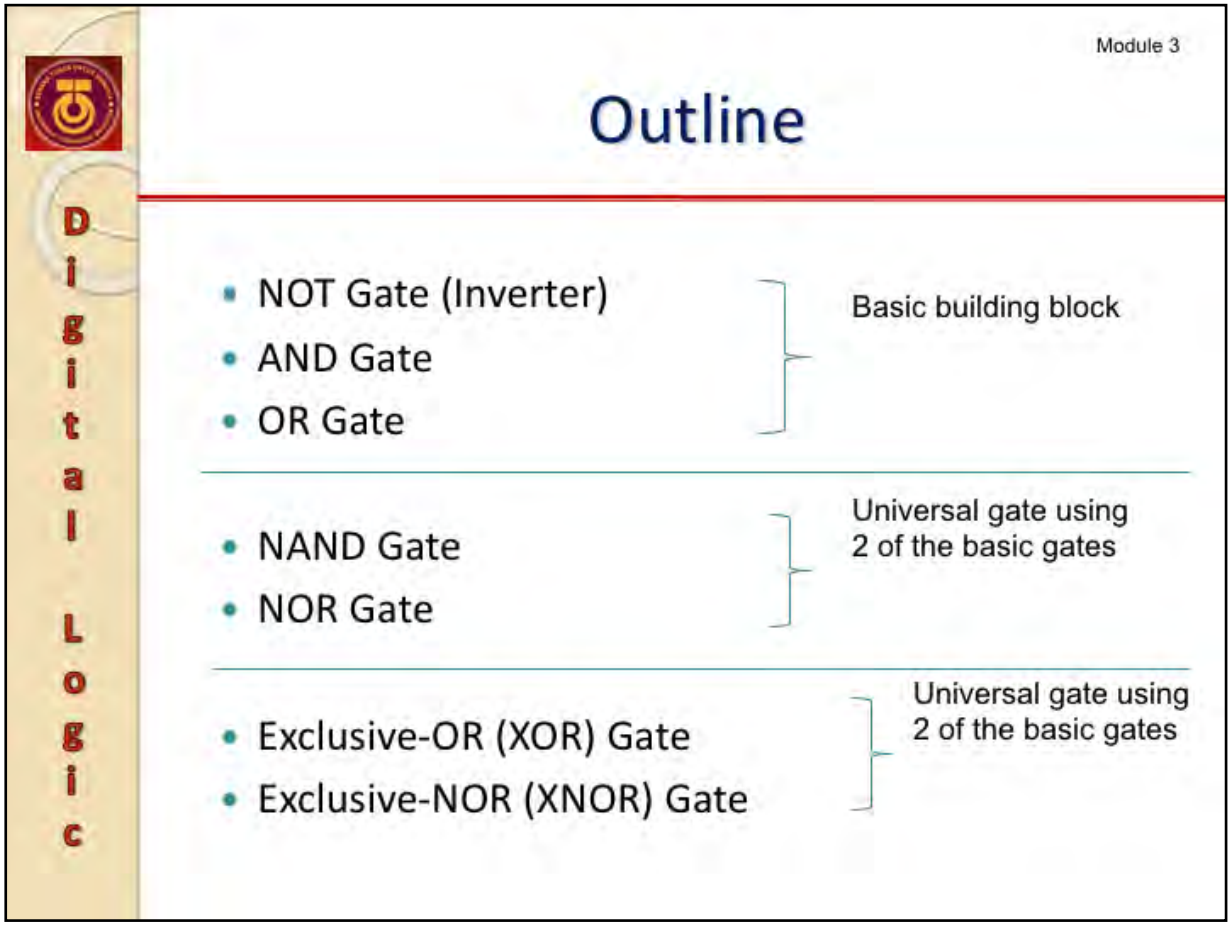
Digital Logic

End of Module 2



The objectives of this module:

1. To introduce the basic gates (NOT, AND and OR)
2. To derive the universal gates from the basic gates (NAND, NOR, XOR and XNOR).
3. To discuss the characteristics of each gates.



- Any digital circuit can be built from the most basic building block which are NOT, AND and OR gates
- But any digital circuit can also be built from a universal gates such as NAND, NOR, XOR and XNOR gates
- For each gates, we will emphasize on the following:
  i. Characteristics
  ii. Symbols
  iii. Operator
  iv. Boolean expression
  v. Truth Table
  vi. Timing Diagram

- For every logic circuit the input and the output can only have two logic values either 0 or 1.

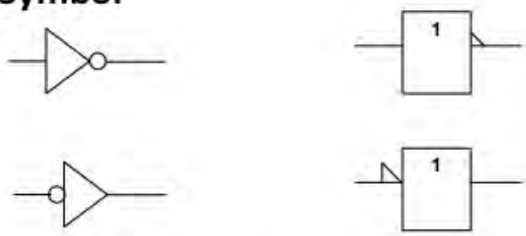
NOT Gate (Inverter)

Digital Logic

- **Characteristics**

  Performs inversion or complementation
  - Changes a logic level to the opposite
  - 0(LOW) → 1(HIGH) ; 1 → 0

- **Symbol**



(a) Distinctive shape symbols with negation indicators

(b) Rectangular outline symbols with polarity indicators

Our class will use the following symbols.

(a) (b)

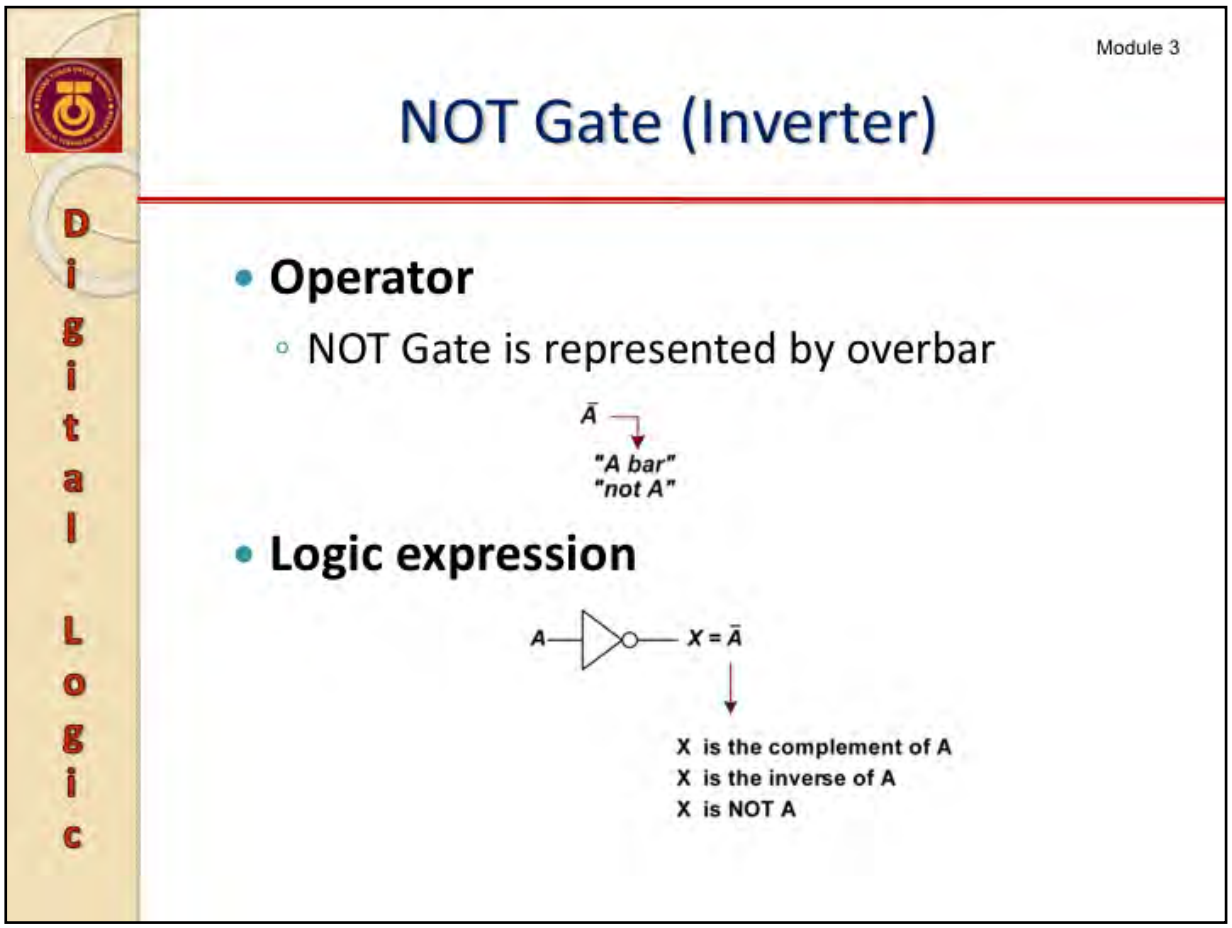
You must be able to "read" the symbol as follows:

"Output will be 0 if the input is 1" for (a)

"Output will be 1 if the input is 0" for (b)

- There are two symbols for each gate , normally symbol (a) will be used but sometimes symbol (b) is used if the symbol can describe the desired operation of the circuit. This will be discussed in the later module.

- However, you must take note that the following symbols below also exist.



Logic expression

X = Ā or X=A'

The variable on left hand side of the expression (e.g. *X*) is the OUTPUT

The variable on the right hand side of the expression (e.g. *A*) is the INPUT

For Inverter, we can say:

OUTPUT is the complement of INPUT

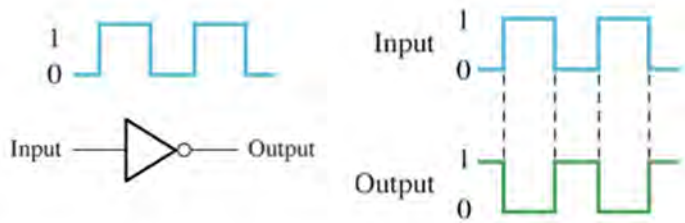
OUTPUT is the inverse of INPUT

OUTPUT is NOT INPUT

Truth Table

A	X
0	1
1	0

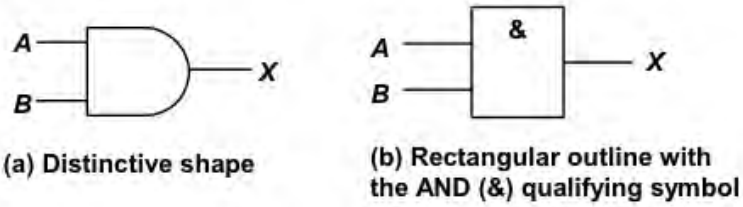
Timing diagrams



AND Gate

Digital Logic

- **Characteristics**
  - Performs 'logical multiplication'
    - If all of the input are HIGH, then the output is HIGH.
    - If any of the input are LOW, then the output is LOW.
      - *AND gate must at least have two (2) INPUTs, and must always have 1 (one) OUTPUT. The AND gate can have more than two INPUTs*
- **Symbols**



(a) Distinctive shape **(b) Rectangular outline with the AND (&) qualifying symbol**

Operator

AND operator can be represented by a dot sign "."

AND gate performs Boolean multiplication

Boolean multiplication follows the same basic rule as binary multiplication:

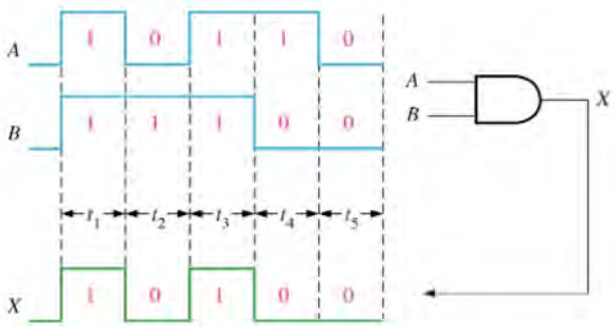
0 . 0 = 0
0 . 1 = 0
1 . 0 = 0
1 . 1 = 1

Logical Expression $X = A \bullet B$

Truth Table

A	B	X = A • B
0	0	0
0	1	0
1	0	0
1	1	1

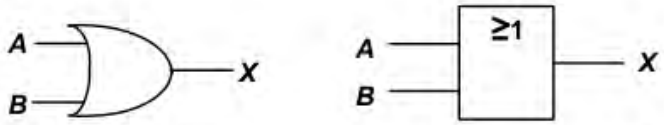
Timing Diagram



OR Gate

Digital Logic

- **Characteristics**
  - Performs 'logical addition'.
    - If any of the input are HIGH, then the output is HIGH.
    - If all of the input are LOW, then the output is LOW.
- **Symbols**



(a) Distinctive shape

(b) Rectangular outline with the OR (≥1) qualifying symbol

Operator

OR operator can be represented by a plus sign "+"

OR gate performs Boolean addition

Boolean addition follows the basic rules as follows:

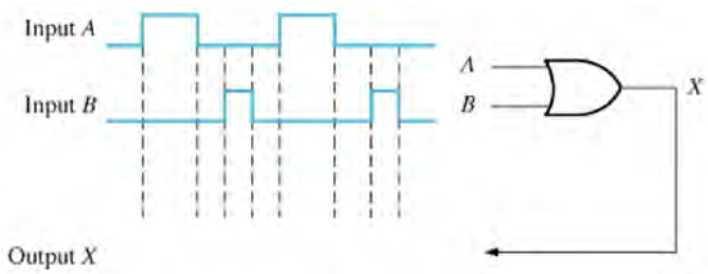
0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 1

Logical Expression X = A + B

Truth Table

A	B	X = A + B
0	0	0
0	1	1
1	0	1
1	1	1

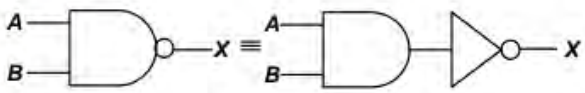
Timing Diagram (DIY)



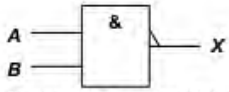
NAND Gate

Digital Logic

- NAND → NOT-AND ⇨combines the AND gate and an inverter
- Used as a universal gate
  - Combinations of NAND gates can be used to perform AND, OR and inverter operations
  - If all or any of the input are LOW, then the output is HIGH.
  - If all of the input are HIGH, then the output is LOW.
- Symbol used:



(a) Distinctive shape: 2 input NAND gate and its NOT/AND equivalent



(b) Rectangular outline: 2 input NAND gate with polarity indicator

Operator

NAND operator can be represented by a dot with an overbar symbol →

$\overline{0.0} = \overline{0} = 1$

$\overline{0.1} = \overline{0} = 1$

$\overline{1.0} = \overline{0} = 1$

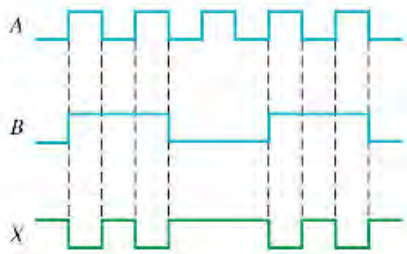
$\overline{1.1} = \overline{1} = 0$

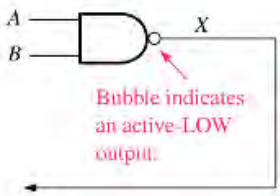
Logical Expression $X = \overline{A \bullet B}$

Truth Table

INPUTS A	B	OUTPUTS $AB \rightarrow \overline{AB} = X$
0	0	$\overline{0.0} = \overline{0} = 1$
0	1	$\overline{0.1} = \overline{0} = 1$
1	0	$\overline{1.0} = \overline{0} = 1$
1	1	$\overline{1.1} = \overline{1} = 0$

Timing Diagram (DIY)

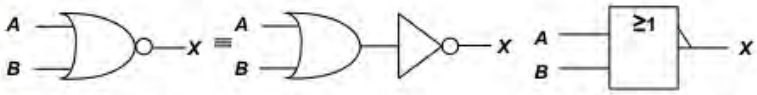




NOR Gate

Digital Logic

- NOR → NOT-OR ⇨ combines the OR gate and an inverter
- Used as a universal gate
  - Combinations of NOR gates can be used to perform AND, OR and inverter operations
  - If all or any of the input are HIGH, then the output is LOW.
  - If all of the input are LOW, then the output is HIGH.
- Symbol used:



(a) Distinctive shape: 2 input NOR gate and its NOT/OR equivalent

(b) Rectangular outline with the OR (≥1) qualifying symbol

Operator

NOR operator can be represented by a plus with an overbar symbols

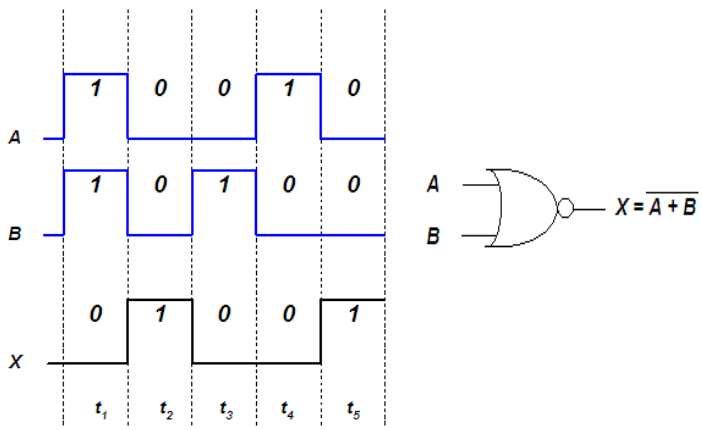
$$\overline{0+0} = \overline{0} = 1$$
$$\overline{0+1} = \overline{1} = 0$$
$$\overline{1+0} = \overline{1} = 0$$
$$\overline{1+1} = \overline{1} = 0$$

Logical Expression $X = \overline{A + B}$

Truth Table

INPUTS A	INPUTS B	OUTPUTS $A+B \rightarrow \overline{A+B} = X$	
0	0	0	$\overline{0+0} = \overline{0} = 1$
0	1	1	$\overline{0+1} = \overline{1} = 0$
1	0	1	$\overline{1+0} = \overline{1} = 0$
1	1	1	$\overline{1+1} = \overline{1} = 0$

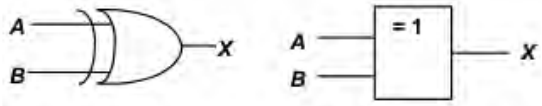
Timing Diagram (DIY)



XOR Gate

Digital Logic

- **Characteristics**
  - Combines basic logic circuits of AND, OR and Inverter.
  - Has only 2 inputs
  - Used as a universal gate
    - Can be connected to form an adder that allows a computer to do perform addition, subtraction, multiplication and division in ALU (Arithmetic and Logic Unit).
    - If both of the input are at the same logic level, then the output is LOW.
    - If both of the input are at opposite logic levels, then the output is HIGH
- **Symbol**



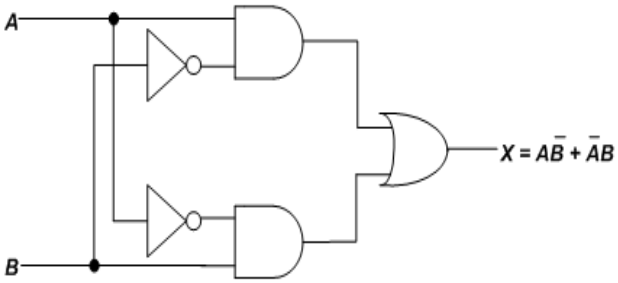
(a) Distinctive shape (b) Rectangular outline

Operator

XOR operator can be represented by a $\oplus$ symbol.

Logical Expression

$$X = A \oplus B$$
$$= A\overline{B} + \overline{A}B$$

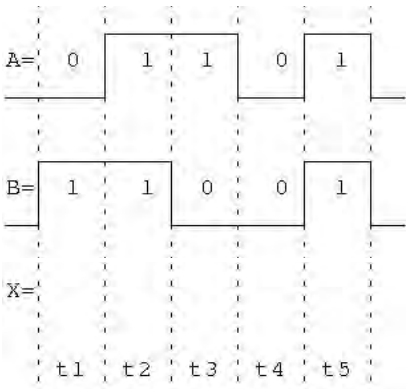


XOR is a combination of 2 AND gates, 1 OR gate and 2 inverters.

Truth Table

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

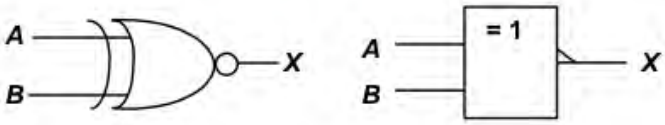
Timing Diagram (DIY)



XNOR Gate

Digital Logic

- **Characteristics**
  - Has only 2 inputs, but output of XNOR is the opposite of XOR
    - If both of the input are at the same logic level, then the output is HIGH.
    - If both of the input are at opposite logic levels, then the output is LOW.
- **Symbol**



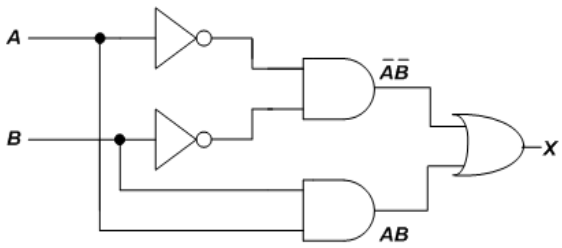
(a) Distinctive shape (b) Rectangular outline

Operator

XNOR operator can be represented by a $\odot$ symbol .

Logical Expression

$$X = A \odot B$$
$$= \overline{A \oplus B}$$
$$= \overline{A\bar{B} + \bar{A}B}$$
$$= AB + \bar{A}\bar{B}$$



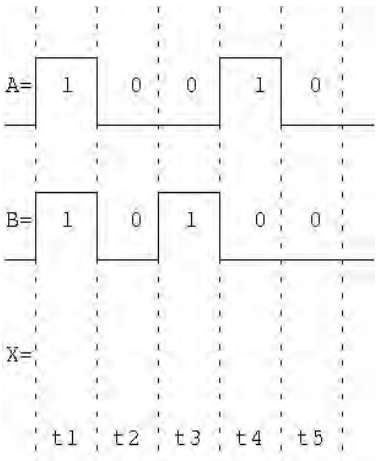
XNOR is a combination of 2 AND gates, 1 OR gate and 2 inverters

.

Truth Table

A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

Timing Diagram (DIY)

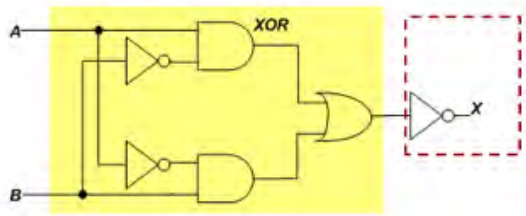


XNOR Implementation

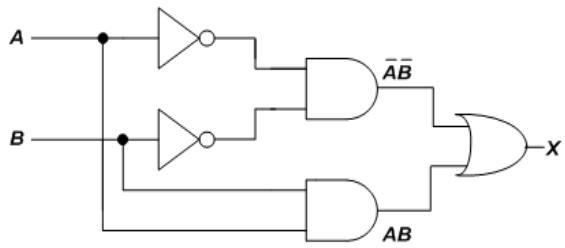
Digital Logic

- There are two ways in implementing XNOR Gate:
  - The first implementation by **complementing (inverting)** XOR circuit

$$X = \overline{A\overline{B}+\overline{A}B}$$



Second implementation is by directly implementing the expression: $X = AB+\overline{A}\,\overline{B}$



The characteristic of the above circuit can be proven by the following truth table:

A	B	AB	$\overline{A}$	$\overline{B}$	$X = AB + \overline{A}\,\overline{B}$
0	0	1	1		1
0	1	0	0		0
1	0	0	0		0
1	1	1	0		1

Digital Logic

End of Module 3

Digital Logic

Module 4:
BOOLEAN ALGEBRA & LOGIC SIMPLIFICATION

Laws and Rules of Boolean Algebra

Constructing Truth table from Boolean Expression

Standard Form of Boolean Expression

Determining standard Expression from truth table

Logic Simplification using:

- Boolean algebra
- Karnaugh Map

Objectives of this module :

-To introduce Boolean Algebra.

-To simplify the digital circuit using Boolean Algebra and Karnaugh Map.

-To apply the Boolean Algebra in designing and analyzing digital circuit.

Important terms in Boolean Algebra

Digital Logic

- **Variable**
  - A symbol that represents a logical quantity
  - Usually italic uppercase (*A*, *B*, *C*, *D*)
  - A single variable can have a 1 or 0 value
- **Complement**
  - The inverse of a variable
  - Indicated by an overbar ($\bar{A}$) or prime (A′)
  - If A = 1, then $\bar{A}$ = 0
- **Literal**
  - Both variable and its complement in a term.
  - $\bar{A}+B+C$ ➔ 3 literals

Example 1:

$f = x'y' + x'y$

Total Number of Literals = 4 which are x', y', x' and y

Total Number of variables = 2 which are x and y

Total Number of terms = 2 which are $x'y'$ and $x'y$

Example 2:

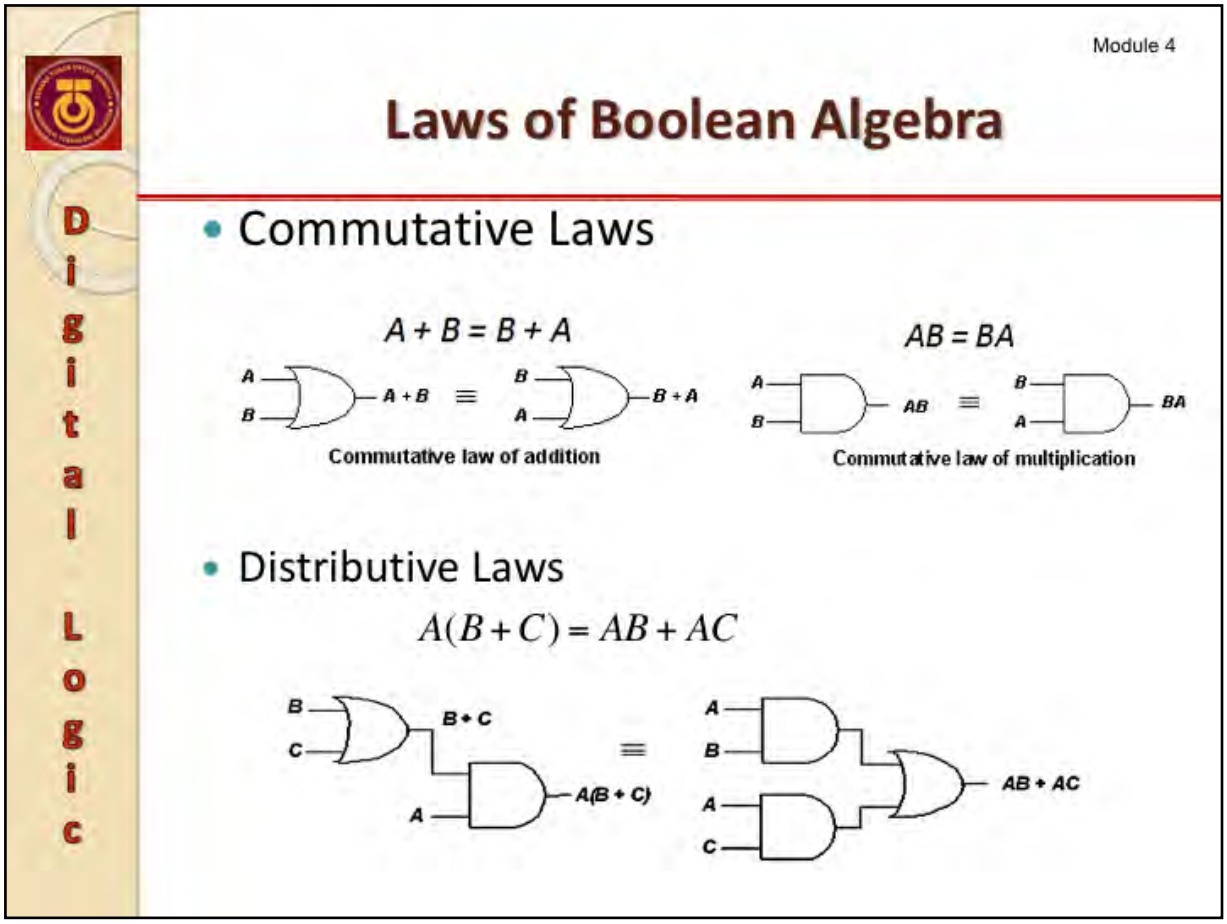
$f = x'$

`

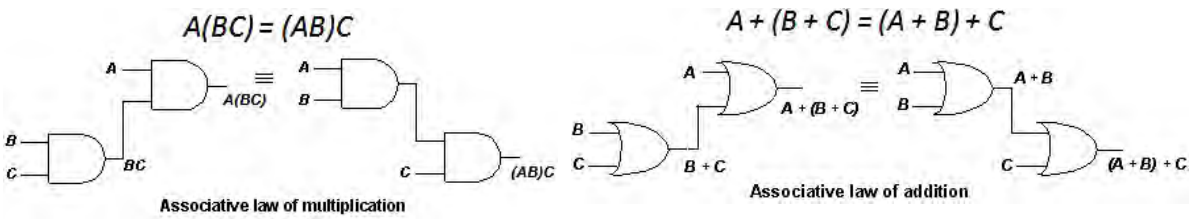
Total Number of Literals = 1 which is x'

Total Number of variables = 1 which is x

Total Number of terms = 1 which is x'



Associative Laws:



Numerical proving of Associative Law using truth table

$$A + (B + C) = (A + B) + C$$

A	B	C	A + B	B + C	A + (B + C)	(A + B) + C
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	1	1	1	1
0	1	1	1	1	1	1
1	0	0	1	0	1	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

DIY:

Do the same procedure to prove the Associative Law for *A(BC) = (AB)C*

Rules of Boolean Algebra

Digital Logic

1	$A + 0 = A$
2	$A + 1 = 1$
3	$A \bullet 0 = 0$
4	$A \bullet 1 = A$
5	$A + A = A$
6	$A + \overline{A} = 1$
7	$A \bullet A = A$
8	$A \bullet \overline{A} = 0$
9	$\overline{\overline{A}} = A$
10	$A + AB = A$
11	$A + \overline{A}B = A + B$
12	$(A + B)(A + C) = A + BC$

The rules can be proven by using **Truth table**

Rule 10

A	B	AB	$A + AB$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

equal

and **Boolean algebra laws and rules**.

Rule 10:

$A + AB = A(1 + B)$ Factoring (distributive law)

$= A \,.\, 1$ Rule 2: (1 + B) = 1

$= A$ Rule 4: A . 1 = A

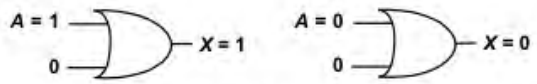
Rules of Boolean Algebra

Digital Logic

1	$A + 0 = A$
2	$A + 1 = 1$
3	$A \bullet 0 = 0$
4	$A \bullet 1 = A$

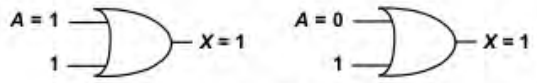
The characteristic of the OR gate follows the rule 1 and 2

Rule 1:



$X = A + 0 = A$

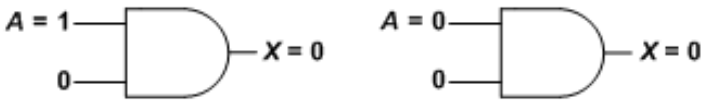
Rule 2:



$X = A + 1 = 1$

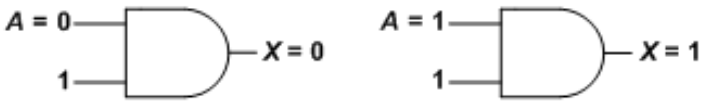
The characteristic of the AND gate follows the rule 3 and 4

Rule 3:



$X = A \bullet 0 = 0$

Rule 4:



$X = A \bullet 1 = A$

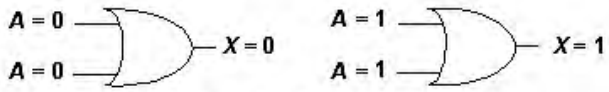
Rules of Boolean Algebra

Digital Logic

5	$A + A = A$
6	$A + \overline{A} = 1$
7	$A \bullet A = A$
8	$A \bullet \overline{A} = 0$

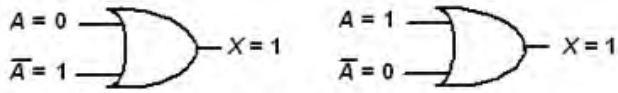
The characteristic of the OR gate follows the rule 5 and 6

Rule 5:



$X = A + A = A$

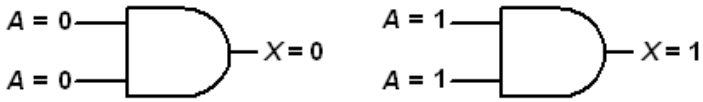
Rule 6:



$X = A + \overline{A} = 1$

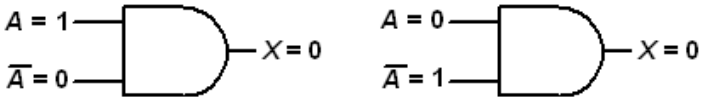
The characteristic of the AND gate follows the rule 7 and 8

Rule 7:



$X = A \bullet A = A$

Rule 8:



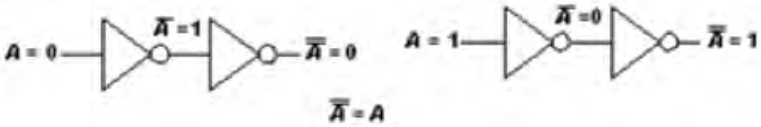
$X = A \bullet \overline{A} = 0$

Rules of Boolean Algebra: Proof …

Digital Logic

9	$\overline{\overline{A}} = A$
10	$A + AB = A$
11	$A + \overline{A}B = A + B$
12	$(A + B)(A + C) = A + BC$

Rule 9:



Rule 10:

$A + AB = A(1 + B)$	Factoring (distributive law)
$= A \,.\, 1$	Rule 2: (1 + B) = 1
$= A$	Rule 4: A . 1 = A

Rule 11:

$A + \overline{A}B = (A + AB) + \overline{A}B$	Rule 10: A = A + AB
$= (AA + AB) + \overline{A}B$	Rule 7: A = AA
$= AA + AB + A\overline{A} + \overline{A}B$	Rule 8: adding $A\overline{A} = 0$
$= (A + \overline{A})(A + B)$	Factoring
$= 1 \,.\, (A + B)$	Rule 6: $A + \overline{A} = 1$
$= A + B$	Rule 4: drop the 1

Rule 12:

$(A + B)(A + C) = AA + AC + AB + BC$	Distributive law
$= A + AC + AB + BC$	Rule 7: AA = A
$= A(1 + C) + AB + BC$	Factoring (distributive law)
$= A \,.\, 1 + AB + BC$	Rule 2: 1 + C = 1
$= A(1 + B) + BC$	Factoring (distributive law)
$= A \,.\, 1 + BC$	Rule 2: 1 + B = 1
$= A + BC$	Rule 4: A . 1 = A

DeMorgan's Theorems

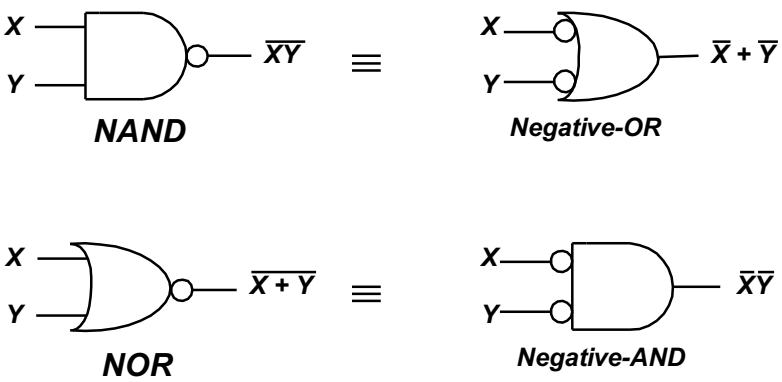
Digital Logic

- Why DeMorgan's Theorem?
  - Because it is useful to minimize the variety and number of logic gates IC.
    - Convert between AND ↔OR representation, so that we can reduce the variety of gates in the circuit.
  - To convert the Boolean expression to its simplified form, it is sometime necessary to apply DeMorgan Theorem before the expression can be simplified using Boolean algebra laws and rules.
- **DeMorgan's theorem 1**: $\overline{XY} = \overline{X} + \overline{Y}$
  - The complement of a product of variables is equal to the sum of the complements of the variables
- **DeMorgan's theorem 2**: $\overline{X + Y} = \overline{X}\,\overline{Y}$
  - The complement of a sum of variables is equal to the product of the complements of the variables

DeMorgan's Theorem also provides mathematical verification for:

NAND ≡ negative-OR

NOR ≡ negative-AND



DeMorgan's Theorem Application

Digital Logic

DeMorgan's Theorems I

$$\overline{XYZ} = \overline{X} + \overline{Y} + \overline{Z}$$

$$\overline{WXYZ} = \overline{W} + \overline{X} + \overline{Y} + \overline{Z}$$

DeMorgan's Theorem II

$$\overline{X + Y + Z} = \overline{X}\,\overline{Y}\,\overline{Z}$$

$$\overline{W + X + Y + Z} = \overline{W}\,\overline{X}\,\overline{Y}\,\overline{Z}$$

Examples of applying DeMorgan's Theorem

$$\overline{(AB + C)(A + BC)} = \overline{(AB + C)} + \overline{(A + BC)}$$
$$= (\overline{AB})\overline{C} + \overline{A}(\overline{BC})$$
$$= (\overline{A} + \overline{B})\overline{C} + \overline{A}(\overline{B} + \overline{C})$$

** Note:
$X = AB + C$
$Y = A + BC$

Point to be highlighted:

- when to use DeMorgan's Theorem? When ***overbar*** covers more than 1 literal

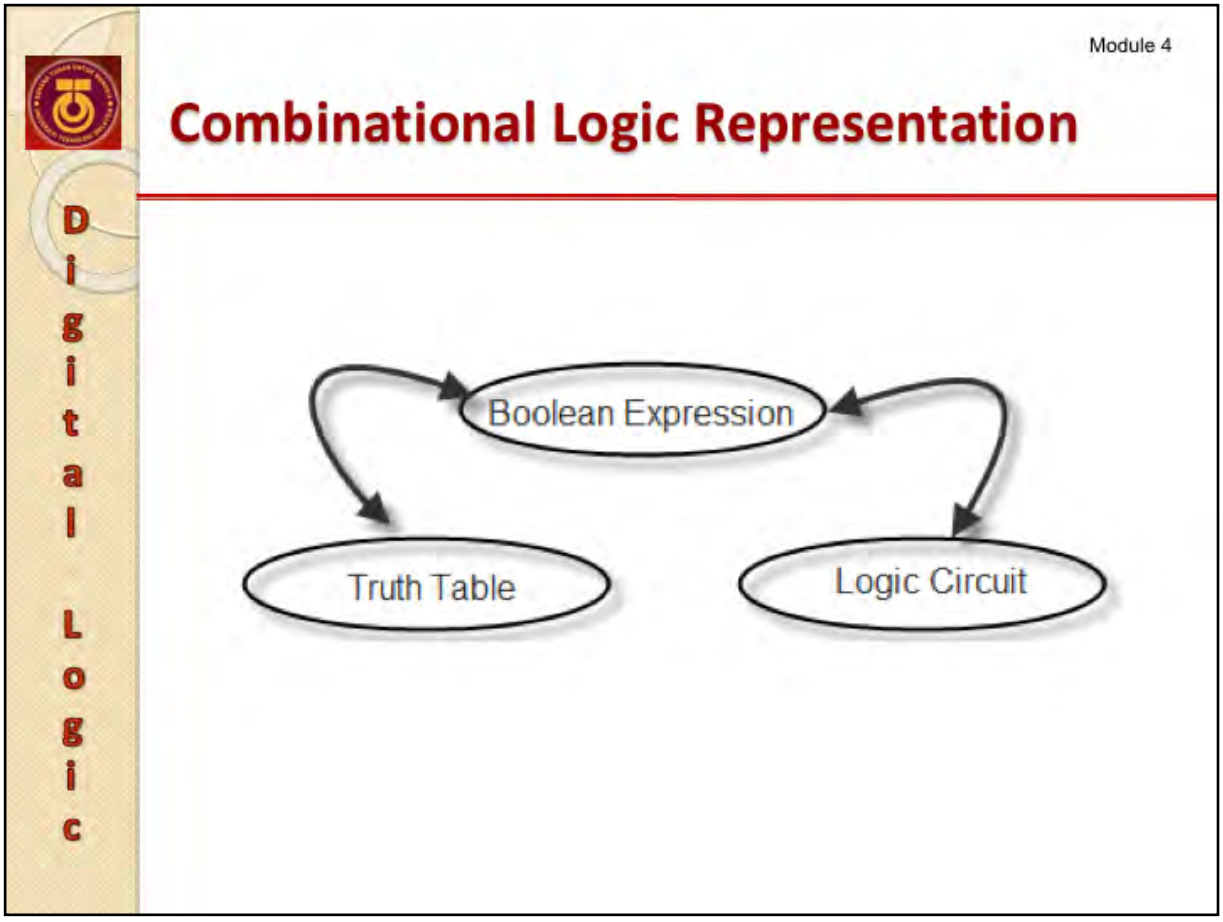
  e.g.

$$\overline{(AB + C)(A + BC)} \qquad \overline{A}(\overline{BC})$$

- by applying DeMorgan's theorem the original expression will be converted to simpler form that can be simplified further by using Boolean rules and laws.

  e.g.

$$(\overline{A} + \overline{B})\overline{C} + \overline{A}(\overline{B} + \overline{C}) = \overline{A}\,\overline{C} + \overline{B}\,\overline{C} + \overline{A}\,\overline{B} + \overline{A}\,\overline{C}$$
$$= \overline{A}\,\overline{C} + \overline{B}\,\overline{C} + \overline{A}\,\overline{B} \qquad \text{, by applying rule } A + A = A$$



- A combinational circuit can be represented as
  1) A Boolean expression

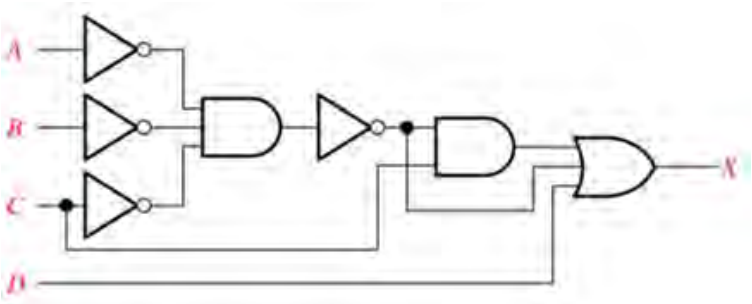
     e.g. $\overline{(AB+C)(A+BC)}$

  2) A Truth Table

     e.g.

A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

  3) A Schematic Diagram (Logic Circuit)



- It is important to be able to convert between different representation

From Boolean Expression to Logic Circuit

Digital Logic

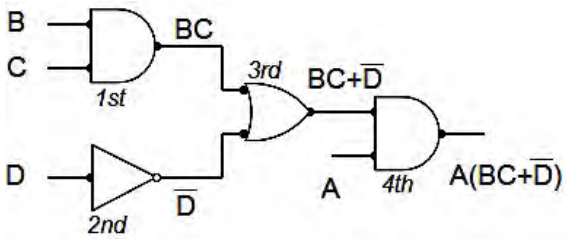
- Treat a Boolean expression as if you try to solve a normal mathematical equation
- Just like a normal mathematic we have the precedence of operation
  - start with ( ) then multiplication (AND) then addition (OR)
  - Therefore to draw the circuit following the same sequence

Conversion from Boolean Expression to Logic Circuit

- **Step 1**: split the equation into individual terms.

  e.g. $X = A(BC + \overline{D}) = A.(BC + \overline{D})$

  → There are two term : term 1 is A, term 2 is $(BC + \overline{D})$ which consist of another 2 term which is term 2a for BC and term 2b which is $\overline{D}$
- **Step 2**: For each term determine the operator between the variables which will determine the gate required:
  - ✓ term 2a will require **AND** gate
  - ✓ term 2b will require **NOT** gate
  - ✓ combination of term 2a and 2b will require **OR** gate
  - ✓ combination of term 1 and term 2 will require **AND** gate
- **Step 3**: Determine number of variable for each term, this will determine the number of input for each gate:
  - ✓ term 2a has 2 variables , it will require **2 input** AND gate
  - ✓ term 2b has a single variable, it will require **single input** NOT gate
  - ✓ combination of term 2a and 2b has 2 terms, it will require **2 input** OR gate
  - ✓ combination of term 1 and term 2 has 2 terms, it will require **2 input** AND gate
- **Step 4**: Draw the circuit with the highest precedence first then followed by the circuit with the lower precedence.



.. Boolean expression to Logic Circuit

Digital Logic

Example: For Boolean expression $X = AB + CDE$

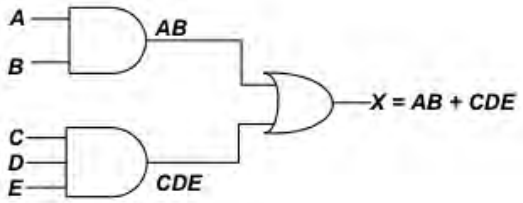
- We see that it is made of 2 terms:
  1. AB
  2. CDE

  Which are then OR-ed

$$X = \underbrace{AB}_{\text{AND}} + \underbrace{CDE}_{\text{AND}}$$

OR

- Both terms 1 and 2 are AND-terms
  - These are done before the OR
  - Just like math, AND has a higher precedence than OR
- To implement this Boolean expression, we need:
  - one 2-input AND-gate for AB
  - one 3-input AND-gate for CDE
  - one 2-input OR-gate to OR the two term of AB + CDE

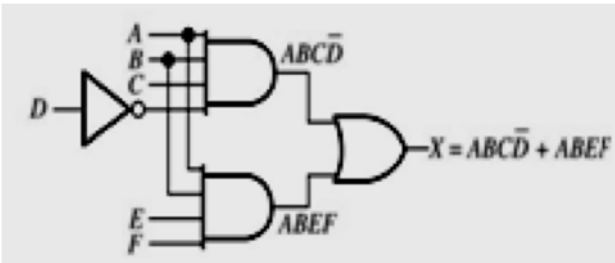


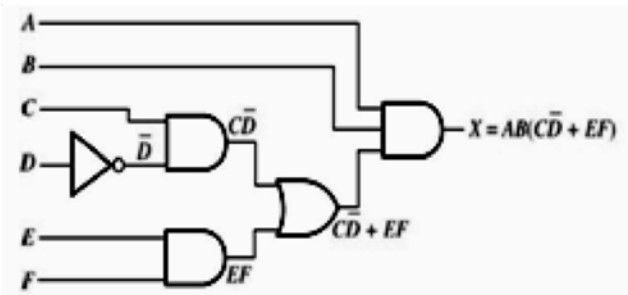
Example: Draw a logic circuit for the following Boolean expression

$$X = AB\left(C\overline{D} + EF\right) \quad = \quad ABC\overline{D} + ABEF$$

$X = AB(C\overline{D} + EF)$

INV	D	1
AND	$C\overline{D}$	2
	EF	3
OR	2 + 3	4
AND	AB(4)	5

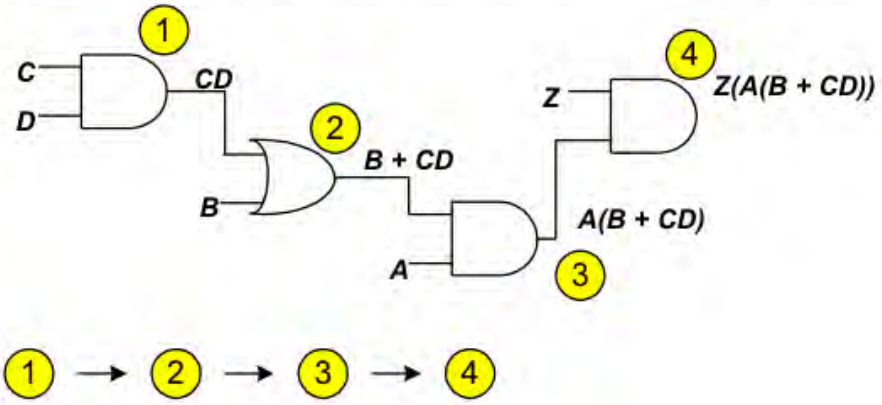




Logic Circuit to Boolean Expression

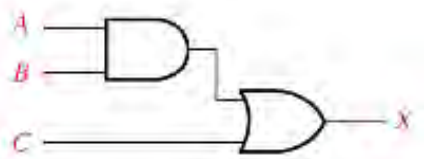
Digital Logic

- To derive the Boolean expression for a given circuit, follow **left-to-right** rule.
  - Begin from the left-most inputs and work towards the right .

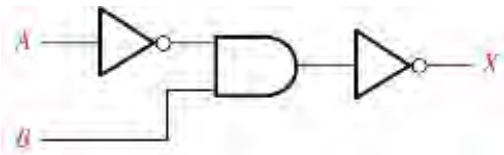


DIY

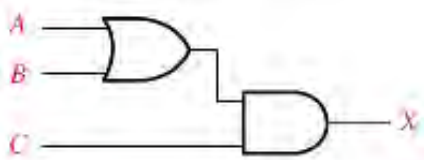
Derive the Boolean expression from the following circuits:



(a)



(b)



(c)

Digital Logic

Boolean Expression to Truth *Table*

- A truth table shows the output for all possible input values.
- From a Boolean expression, a truth table can be developed.
  - If x = number of input variables, then number of possible combinations of input, $n = 2^x$
  - E.g.
    - A(B + CD) → x = 4; $n = 2^4 = 16$

Sometimes the number of variables is not obvious (or unknown)

e.g.: A + CD -- seem like 3 variable but actually 4 because B is missing

*** in this class if the number of inputs is unknown then we use a variable in a proper sequence , if only A and E in the equation -- eg : A + E, actually there are 5 variables A, B, C, D,and E.*

Why B, C, and D missing ? Because the expression is already simplified.

Example 1 :

$x = 2, n = 4$

INPUT		OUTPUT
A	**B**	**F**
0	0	
0	1	
1	0	
1	1	

Example 2 :

$x = 3, n = 8$

INPUT			OUTPUT
A	**B**	**C**	**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	

Steps in constructing a truth table

Digital Logic

- **Step 1:**
  Identify **x** and **n** from the Boolean expression.
- **Step 2:**
  Find the values of the variables that make the expression equal to 1.(*Hint: use the rules for Boolean addition and multiplication*)
- **Step 3:**
  List in a table
  - All the **n** combinations of 1s and 0s (input)
  - The values of variables from step 2 (output)
  - All the other output values will be 0

Example: A(B + CD)

Step 1: x = 4 ; n = $2^x = 2^4 = 16$

Step 2:

A(B + CD) = 1 • 1 = 1 → A = 1

What makes B + CD = 1?

B + CD = 1 + 0 = 1

B + CD = 0 + 1 = 1

B + CD = 1 + 1 = 1 → B = 1 or 0

What makes CD = 1?

CD = 1 . 1 = 1 → C = 1; D = 1

Therefore, the output of A(B + CD) will be 1 if:

A = 1, B = 1, C = 0/1, D = 1/0

A = 1, B = 0 , C = 1, D = 1

Step 3 : Fill in the table with results from Step 2

INPUTS				OUTPUT
A	B	C	D	A(B + CD)
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

From Truth Table to Logic Circuit

Digital Logic

Step 1: Get the product term from HIGH outputs

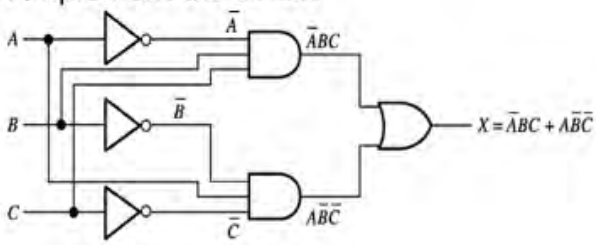
Step 2: From the product term get the expression

- This is done by OR-ing the product terms

$$X = \bar{A}BC + A\bar{B}\bar{C}$$

INPUTS A B C	OUTPUT X	PRODUCT TERM
0 0 0	0	
0 0 1	0	
0 1 0	0	
0 1 1	1	$\bar{A}BC$
1 0 0	1	$A\bar{B}\bar{C}$
1 0 1	0	
1 1 0	0	
1 1 1	0	

Step 3: Implement the circuit

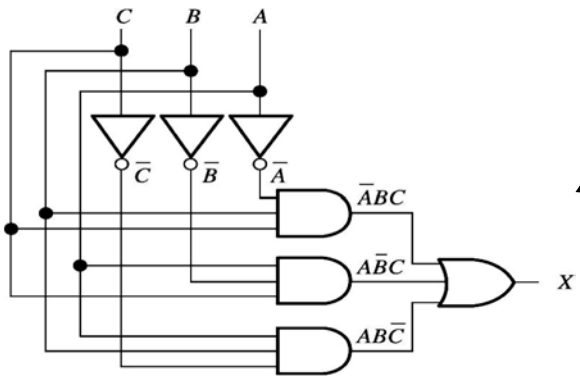


Example: Given the truth table below, produce the logic circuit.

INPUTS A B C	OUTPUT X
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	0

INPUTS A B C	OUTPUT X	PRODUCT TERM
0 0 0	0	
0 0 1	0	
0 1 0	0	
0 1 1	1	$\bar{A}BC$
1 0 0	0	
1 0 1	1	$A\bar{B}C$
1 1 0	1	$AB\bar{C}$
1 1 1	0	

$$X = \bar{A}BC + A\bar{B}C + AB\bar{C}$$



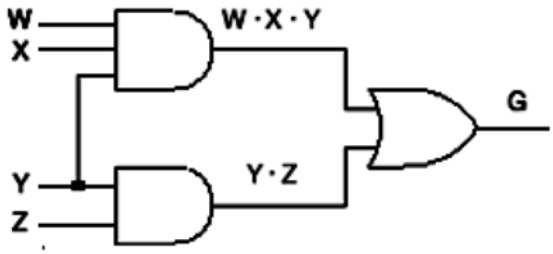
Logic Circuit to Truth Table

Digital Logic

- Requires 2 steps
  1. From circuit derive the Boolean expression of the output
  2. From Boolean expression produce a Truth Table

  Both step has been mentioned

Example: Produce a truth table from the following circuit



Step1: Derive the Boolean expression of the output , G = WXY + YZ

Step 2: From the expression produce the truth table

G will be 1 if either term = 1

→ term 1 which is WXY will only be 1 if W = 1, X = 1 and Y = 1

→ term2 which is YZ will only be 1 if Y =1 and Z = 1

W	X	Y	Z	G
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	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

Simplification using Boolean Algebra

Digital Logic

- Simplification:
  - To reduce an expression to its simplest form
  - To change a form to a more convenient one for efficient implementation
- We use:- basic laws, rules, theorems of Boolean algebra
- Practice makes perfect

Why do we need to simplify?

1.To reduce the expression, so that the implementation will produce a simplest circuit.

2.The simplest circuit contribute to cost reduction.

What do we mean by “reducing the expression”?

1.Reduce the number of terms.

- In general, as a result, it will reduce the inputs to the subsequent level(s).

2.Reduce the number of literals.

- In general, as a result, it will reduce the inputs to the first level.

Digital Logic

Simplify this expression

$$AB + A(B + C) + B(B + C)$$

Step 1: Apply distributive law to $A(B+C)+B(B+C)$

$$AB + AB + AC + BB + BC$$

Step 2: Apply rule 7 ($BB = B$)

$$AB + AB + AC + B + BC$$

Step 3: Apply rule 5 ($AB+AB = AB$)

$$AB + AC + B + BC$$

Step 4: Apply rule 10 ($B + BC = B$)

$$AB + AC + B$$

Step 5: Apply 10 ($B + AB = B$)

$$AC + B$$

Digital Logic

Simplify this expression

$$\overline{A}BC + A\overline{B}\overline{C} + \overline{A}\overline{B}\overline{C} + A\overline{B}C + ABC$$

Step 1: Factor $\overline{A}BC + ABC$ term

$$BC(\overline{A} + A) + A\overline{B}\overline{C} + \overline{A}\overline{B}\overline{C} + A\overline{B}C$$

Step 2: Apply rule 6 to $(\overline{A} + A)$ term, and factor term $(A\overline{B}\overline{C} + A\overline{B}C)$

$$BC.1 + A\overline{B}(\overline{C} + C) + \overline{A}\overline{B}\overline{C}$$

Step 3: Apply rule 4 to *(BC.1)* term, and rule 6 to $(\overline{C} + C)$ term

$$BC + A\overline{B}.1 + \overline{A}\overline{B}\overline{C}$$

Step 4: Apply rule 4 to $(A\overline{B}.1)$ term

$$BC + A\overline{B} + \overline{A}\,\overline{B}\,\overline{C}$$

Step 5: Factor the $A\overline{B} + \overline{A}\,\overline{B}\,\overline{C}$

$$BC + \overline{B}(A + \overline{A}\,\overline{C})$$

Step 6: Apply rule 11 to $\overline{B}(A + \overline{A}\,\overline{C})$

$$BC + \overline{B}(A + \overline{C})$$

Step 7: Use distributive and commutative laws to get the following expression

$$BC + A\overline{B} + \overline{B}\,\overline{C}$$

Standard Form of Boolean Expressions

Digital Logic

- Boolean expression can be converted into one of 2 standard forms:
  - **The sum-of-products (SOP) form**
  - **The product-of-sums (POS) form**
- Standardization makes the evaluation, simplification, and implementation of Boolean expressions more systematic and easier.
- **Product term** = a term with the product (Boolean multiplication) of literals
- **Sum term** = a term with the sum (Boolean addition) of literals

Boolean in SOP form can be represented as

1. Boolean expression , e.g. $X = \overline{A}B\overline{C} + A\overline{B}C + ABC$
2. By using a sigma notation , e.g. X = $\sum_{ABC}$ (2,5,7)

   Where 2 → 010 → $\overline{A}B\overline{C}$ and 5 → 101 → $A\overline{B}C$

Boolean in POS form can be represented as

1. Boolean expression , e.g. $X = (A+B+\overline{C})(\overline{A}+B+C)(\overline{A}+\overline{B}+C)$
2. By using a PI notation , e.g. X = $\prod_{ABC}$ (1, 4, 6)

Determine the terms that 1, 4 and 6 represent?

Sum-of-Products (SOP) Form

Digital Logic

- **SOP** = when 2 or more product terms are summed
- Example: $AB_{P1} + ABC_{P2}$

  $ABC_{P1} + CDE_{P2} + BCD_{P3}$
- SOP can also contain a single variable term

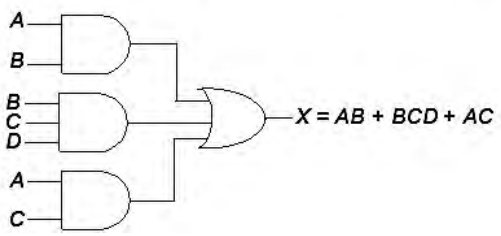
  $ABC_{P1} + C_{P2}$
- In SOP a single overbar cannot extend over more than 1 variable, but more than 1 variable can have an overbar.

  $\overline{A}\,\overline{B}\,\overline{C}$ ☑ $\overline{ABC}$ ☒

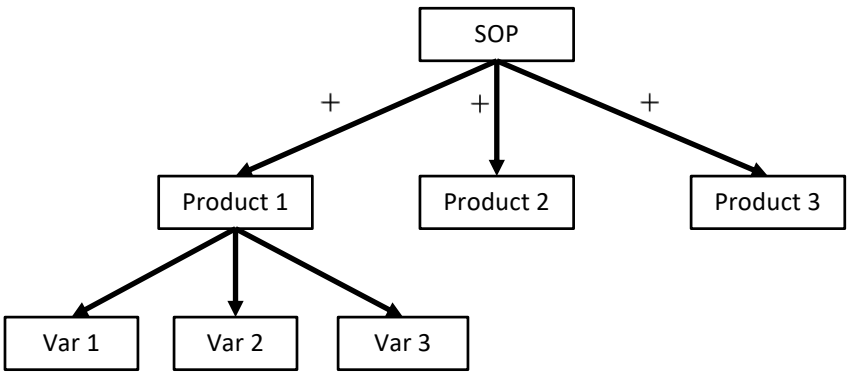
$\overline{A}\,\overline{B}\,\overline{C}$ ☑ This term is valid because it is a product term which is $\overline{A} \cdot \overline{B} \cdot \overline{C}$

$\overline{ABC}$ ☒ This term is invalid because it is not a product term, after applying a DeMorgan's theorem it will change to $\overline{A} + \overline{B} + \overline{C}$

Implementation of the SOP expression AB + BCD + AC



Binary Representation of a Standard Product Term



... Sum-of-Products (SOP) Form

Digital Logic

- A logic expression can be changed to SOP form using Boolean algebra techniques.

$$A(B+CD) = AB + ACD$$

$$AB + B(CD+EF) = AB + BCD + BEF$$

- Standard SOP form is when all the variables appear in each product term in the expression.

$$A\overline{B}CD + \overline{A}\overline{B}C\overline{D} + AB\overline{C}\overline{D}$$

- To convert product terms to standard SOP
  - Multiply each of the nonstandard term with the missing term using Boolean algebra rule 6 ($A + \overline{A} = 1$).
  - Repeat until all variables appear in each product term.

- Convert this Boolean expression to standard SOP form:

$$A\overline{B}C + \overline{A}\overline{B} + AB\overline{C}D$$

- Variables = A, B, C, D.
- What is missing?
  - Term 1: missing D or D'
  - Term 2: missing (C/C') and (D/D')
- Complete these terms by applying Boolean rule 6

$$\text{Term 1: } A\overline{B}C = A\overline{B}C(D+\overline{D}) = A\overline{B}CD + A\overline{B}C\overline{D}$$

$$\text{Term 2: } \overline{A}\overline{B} = \overline{A}\overline{B}C + \overline{A}\overline{B}\overline{C}$$

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

- Now we have

$$A\overline{B}C + \overline{A}\overline{B} + AB\overline{C}D$$

$$= A\overline{B}CD + A\overline{B}C\overline{D} + \overline{A}\overline{B}CD + \overline{A}\overline{B}C\overline{D} + \overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}\overline{C}\overline{D} + AB\overline{C}D$$

Example of a Standard Sum of Product

Digital Logic

Determine the binary value for which the following standard SOP expression is equal to 1:

$$ABCD + A\bar{B}\bar{C}D + \bar{A}\bar{B}\bar{C}\bar{D}$$

The term ABCD is equal to 1 when A = 1, B = 1, C = 1, and D = 1

$$ABCD = 1.1.1.1 = 1$$

The term $A\bar{B}\bar{C}D$ is equal to 1 when A = 1, B = 0, C = 0, and D = 1

$$A\bar{B}\bar{C}D = 1.\bar{0}.\bar{0}.1 = 1.1.1.1 = 1$$

The term $\bar{A}\bar{B}\bar{C}\bar{D}$ is equal to 1 when A = 0, B = 0, C = 0, and D = 0

$$\bar{A}\bar{B}\bar{C}\bar{D} = \bar{0}.\bar{0}.\bar{0}.\bar{0} = 1.1.1.1 = 1$$

The SOP expression equals 1 when any or all of the three product terms is 1.

From the standard SOP → $ABCD + A\bar{B}\bar{C}D + \bar{A}\bar{B}\bar{C}\bar{D}$
it is easy to generate the truth table.

INPUT				OUTPUT
A	B	C	D	F
0	**0**	**0**	**0**	**1**
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	**0**	**0**	**1**	**1**
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	**1**	**1**	**1**	**1**

Product-of-Sums (POS) Form

Digital Logic

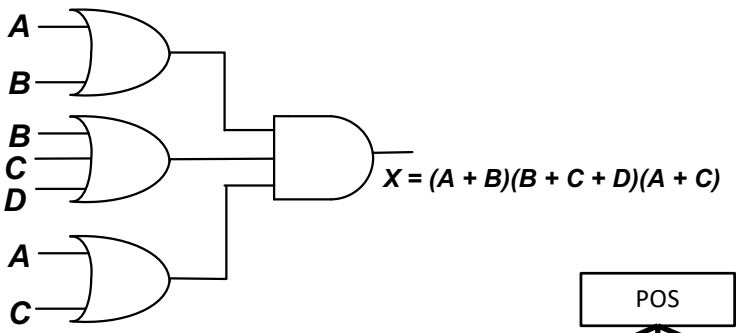
- **POS** = when 2 or more sum terms are multiplied.
  - $(A+B)_{S1}(A+B+C)_{S2}$
  - $(A+B+C)_{S1}(C+D+E)_{S2}(B+C+D)_{S3}$
- Like SOP, POS
  - can also contain a single variable term
  - a single overbar cannot extend over more than 1 variable, but more than 1 variable can have an overbar.

$\overline{A}+\overline{B}+\overline{C}$ ☑ $\overline{A+B+C}$ ☒

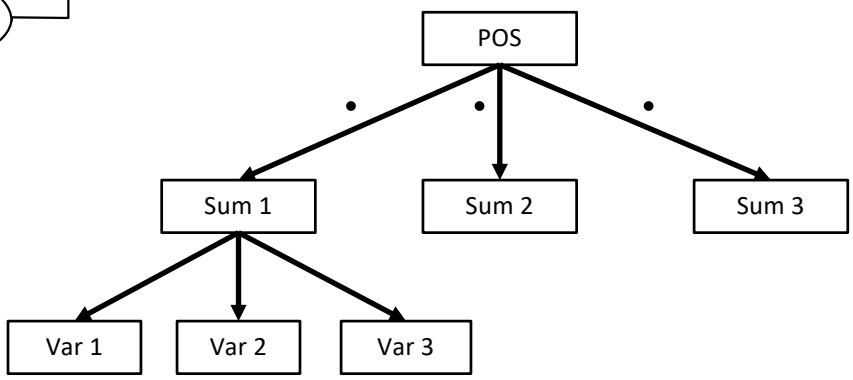
$\overline{A}+\overline{B}+\overline{C}$ ☑ This term is valid because it is a **sum** term

$\overline{A+B+C}$ ☒ This term is invalid because it is not a **sum** term, after applying a DeMorgan's theorem it will change to $\overline{A}\cdot\overline{B}\cdot\overline{C}$

Implementation of the POS expression (A + B)(B + C + D)(A + C)



Binary Representation of a Standard Product of Sum



... Product-of-Sums (POS) Form

Digital Logic

- Standard POS form = where all the variables in the domain appear in each sum term in the expression.
- To convert product terms to standard POS
  - Multiply each of the nonstandard term with the missing term using Boolean algebra rule 8: $A \bullet \overline{A} = 0$
  - Apply **rule 12** : $A + BC = (A + B)(A + C)$
  - Repeat until all variables appear in each sum term.

$$A + BC = (A + B)(A + C)$$

- Convert this Boolean expression to standard POS form

$$(A+\overline{B}+C)(\overline{B}+C+\overline{D})(A+\overline{B}+\overline{C}+D)$$

- **Variable** = A, B, C, D.
- What is missing?
  - Term 1: missing D or D'
  - Term 2: missing A or A'
- Apply rules 8 and 12
- Term1

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

- Term 2

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

- Now we have

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

Example of a Standard Product of Sum

Digital Logic

Determine the binary value for which the following standard POS expression is equal to 0:

$$(A+B+C+D)(A+\overline{B}+\overline{C}+D)(\overline{A}+\overline{B}+\overline{C}+\overline{D})$$

$(A+B+C+D) = 0+0+0+0 = 0$; A=0, B=0, C=0 and D=0.

$(A+\overline{B}+\overline{C}+D) = 0+\overline{1}+\overline{1}+0 = 0$; A=0, B=1, C=1 and D=0.

$(\overline{A}+\overline{B}+\overline{C}+\overline{D}) = \overline{1}+\overline{1}+\overline{1}+\overline{1} = 0$; A=1, B=1, C=1 and D=1.

The POS expression equals 0 when all of the terms is 0.

From the standard POS → $(A+B+C+D)\,(A+\overline{B}+\overline{C}+D)\,(\overline{A}+\overline{B}+\overline{C}+\overline{D})$
It is easy to generate truth table

INPUT				OUTPUT
A	**B**	**C**	**D**	**F**
0	**0**	**0**	**0**	**0**
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	**0**	**0**	**1**	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	**1**	**1**	**1**	**0**

Converting Standard SOP to Standard POS

Digital Logic

- Binary values in a standard SOP expression are not present in the equivalent standard POS expression.

	Standard SOP	Standard POS
Binary values (x)	If x is here	Then x is not here
	If x is not here	Then x is here

Conversion steps:

Step 1:
Evaluate each product term in the SOP expression → i.e. determine the binary numbers of the product terms.

Step 2:
Determine all the binary numbers not included in Step 1.

Step 3:
Write equivalent sum term for each binary number from Step 2 and express in POS form.

*** Using a similar procedure, to go from POS to SOP.*

E.g. Convert this SOP exp. to an equivalent POS exp.

$$\bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C + ABC$$

- Variables = A, B, C = 3. So, $2^3 = 8$ possible combinations.

$$\bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C + ABC$$
$$000 \quad 010 \quad 011 \quad 101 \quad 111$$

- The SOP have 5 of 8, so POS have the other 3 (001, 100, 110) → These 3 make sum term = 0

$$(A + B + \bar{C})(\bar{A} + B + C)(\bar{A} + \bar{B} + C)$$
$$0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0$$

Converting Boolean Expressions to Truth Tables

Digital Logic

- **Step 1**:
  Determine input variables and possible combinations of input values.
- **Step 2**:
  Convert expression to Standard SOP or POS.
- **Step 3**:
  Find the binary values that make the product = 1 for SOP or sum = 0 for POS.
- **Step 4**:
  The remaining combination will be
  - Equal to 0 for SOP
  - Equal to 1 for POS

  Fill in the truth table

From SOP to truth table:

- E.g. Produce a truth table for the standard SOP expression

$$\bar{A}\bar{B}C + A\bar{B}\bar{C} + ABC$$

- Step 1: Variable = A, B, C ; Combinations = $2^3 = 8$
- Step 2: The expression is in a standard SOP
- Step 3: What binary value makes the product term = 1?

$$\bar{A}\bar{B}C + A\bar{B}\bar{C} + ABC$$

$\bar{A}\bar{B}C$
0 0 1 = 1 1 1 = 1

$A\bar{B}\bar{C}$
1 0 0 = 1 1 1 = 1

ABC
1 1 1 = 1 1 1 = 1

- Step 4: Fill the truth table

INPUTS A	B	C	OUTPUT X	PRODUCT TERM
0	0	0	0	
0	0	1	1	$\bar{A}\bar{B}C$
0	1	0	0	
0	1	1	0	
1	0	0	1	$A\bar{B}\bar{C}$
1	0	1	0	
1	1	0	0	
1	1	1	1	ABC

Digital Logic

POS to Truth Table

- Produce truth table for the standard POS expression

$$(A+B+C)(A+\overline{B}+C)(A+\overline{B}+\overline{C})(\overline{A}+B+\overline{C})(\overline{A}+\overline{B}+C)$$

- Step 1: Variable = A, B, C; Combinations = $2^3 = 8$
- Step 2: The expression is in a standard POS
- Step 3: What binary value makes the sum term = 0?

$$(A+B+C)(A+\overline{B}+C)(A+\overline{B}+\overline{C})(\overline{A}+B+\overline{C})(\overline{A}+\overline{B}+C)$$

0 0 0 0 1 0 0 1 1 1 0 1 1 1 0

$\underbrace{0\ 0\ 0}_{0}$ $\underbrace{0\ 0\ 0}_{0}$ $\underbrace{0\ 0\ 0}_{0}$ $\underbrace{0\ 0\ 0}_{0}$ $\underbrace{0\ 0\ 0}_{0}$

Step 4: Fill the truth table

INPUTS A	B	C	OUTPUT X	SUM TERM
0	0	0	0	$(A+B+C)$
0	0	1	1	
0	1	0	0	$(A+\overline{B}+C)$
0	1	1	0	$(A+\overline{B}+\overline{C})$
1	0	0	1	
1	0	1	0	$(\overline{A}+B+\overline{C})$
1	1	0	0	$(\overline{A}+\overline{B}+C)$
1	1	1	1	

DIY:

Produce the truth table for the following expression:

a) $A\overline{B}+BC+\overline{A}D$

b) $(A+\overline{B})(B+C)(\overline{A}+D)$

Determining Standard Expressions from a Truth Table

Digital Logic

Step 1: Find the output =1 for SOP or output=0 for POS

Step 2: Replace each binary number with its corresponding variable.

- **SOP** : if input =1→ write the variable or if input = 0 write the complement of the variable
- e.g. **1010** ⟶ $A\bar{B}C\bar{D}$
- **POS** : if input =0 → write the variable or if input = 1 write the complement of the variable
- e.g. **1001** ⟶ $\bar{A} + B + C + \bar{D}$

Step 3: Complete the expression by

- SOP: OR all the terms
- POS: AND all the terms

Example of Determining Standard Expressions from a Truth Table

INPUTS A	B	C	OUTPUT X	PRODUCT TERM
0	0	0	0	
0	0	1	1	$\bar{A}\bar{B}C$
0	1	0	0	
0	1	1	0	
1	0	0	1	$A\bar{B}\bar{C}$
1	0	1	0	
1	1	0	0	
1	1	1	1	ABC

$$\bar{A}\,\bar{B}\,C + A\,\bar{B}\,\bar{C} + A\,B\,C$$

Step1 Step2

Step3

INPUTS A	B	C	OUTPUT X	SUM TERM
0	0	0	0	$(A + B + C)$
0	0	1	1	
0	1	0	0	$(A + \bar{B} + C)$
0	1	1	0	$(A + \bar{B} + \bar{C})$
1	0	0	1	
1	0	1	0	$(\bar{A} + B + \bar{C})$
1	1	0	0	$(\bar{A} + \bar{B} + C)$
1	1	1	1	

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

Karnaugh Map (K-Map)

Digital Logic

- K-Map is similar to the truth table, but it **presents all of the possible values of input and output**.
- This is shown in an **array of cells**.
- K-Maps can be used for expressions with **2, 3, 4 or 5** variables.
- The number of cells in a K-Map = total number of possible input variable combinations → $3 = 2^3 = 8$
- Adjacent cells can only differ by only one variable value
  - E.g. Cell 010 is adjacent to 0<u>0</u>0, 01<u>1</u> and <u>1</u>10
- Physically, cells that share their walls are adjacent
- In a K-map with 4-variable or more,
  - top-most & bottom-most cells are adjacent
  - Left- most & right most cells are adjacent

Example of 3 and 4 variables K-Map of SOP

AB \ C	0	1
00		
01		
11		
10		

AB \ C	0	1
00	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}C$
01	$\bar{A}B\bar{C}$	$\bar{A}BC$
11	$AB\bar{C}$	ABC
10	$A\bar{B}\bar{C}$	$A\bar{B}C$

3-Variable Karnaugh Map

AB \ CD	00	01	11	10
00				
01				
11				
10				

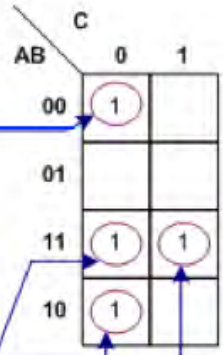
AB \ CD	00	01	11	10
00	$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{A}\bar{B}\bar{C}D$	$\bar{A}\bar{B}CD$	$\bar{A}\bar{B}C\bar{D}$
01	$\bar{A}B\bar{C}\bar{D}$	$\bar{A}B\bar{C}D$	$\bar{A}BCD$	$\bar{A}BC\bar{D}$
11	$AB\bar{C}\bar{D}$	$AB\bar{C}D$	$ABCD$	$ABC\bar{D}$
10	$A\bar{B}\bar{C}\bar{D}$	$A\bar{B}\bar{C}D$	$A\bar{B}CD$	$A\bar{B}C\bar{D}$

4-Variable Karnaugh Map

Mapping truth table to K-Map

Digital Logic

INPUTS A	B	C	OUTPUTS X
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

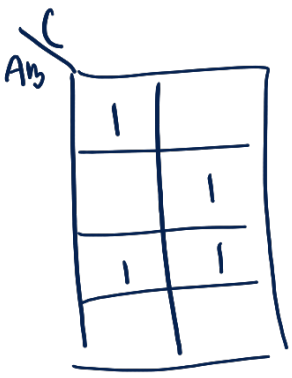


DIY: Map the truth table to K-Map

INPUTS A	B	C	OUTPUTS X
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

SOP

1. Grouping the 1's
2. Determine product term for each group
3. Summing the resulting product terms.



$\bar{A}\bar{B}\bar{C}$

A	B	C
0	0	0

BC

A	B	C
0	1	1
1	1	1

AB

A	B	C
1	1	0
1	1	1

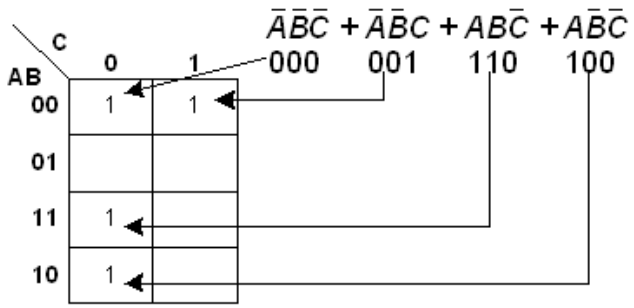
K-Map SOP Minimization

Digital Logic

- K-Map is used to simplify Boolean expressions to their **minimum form**.
- A minimized SOP expression has the fewest possible term with each term having fewest possible variables.
- A minimized SOP expression needs fewer logic gates than standard expression.
- To map an SOP expression to a map:
  - **Step 1:** determine the binary value of each product term
  - **Step 2**: Place a 1 in a cell that have the same value as the product term

Example: Mapping of 3 variables SOP expression

$$\overline{A}\,\overline{B}\,\overline{C}+\overline{A}\,\overline{B}\,C+A\,B\,\overline{C}+A\,\overline{B}\,\overline{C}$$

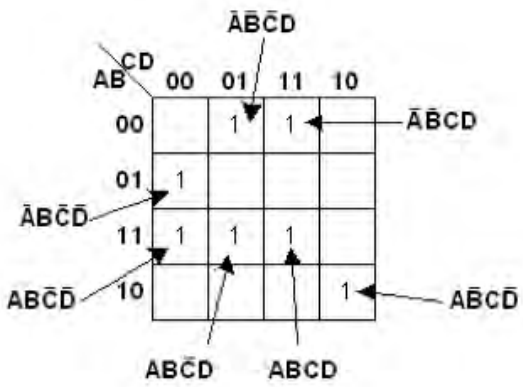


Digital Logic

Example: Mapping of 4 variables SOP expression

$$\bar{A}\bar{B}CD + \bar{A}B\bar{C}\bar{D} + AB\bar{C}D + AB\bar{C}\bar{D} + ABCD + \bar{A}\bar{B}\bar{C}D + A\bar{B}C\bar{D}$$

0011 0100 1101 1100 1111 0001 1010



Example: Mapping of a Non-Standard SOP expression

- To use K-maps, expressions must be in standard form.
- For expressions that are not standard, it must be converted to a standard form.
- Recall: $AB + ABC$
  - $AB(1) \rightarrow AB(C + C') \rightarrow ABC + ABC'$
  - *So: ABC + ABC' + ABC*
- Map the following expression on a K-map:

$\bar{A}$ +	$A\bar{B}$ +	$AB\bar{C}$
000	*100*	*110*
001	*101*	
010		
011		

AB \ C	0	1
00	1	1
01	1	1
11	1	
10	1	1

K-Map Simplification of SOP Expressions

Digital Logic

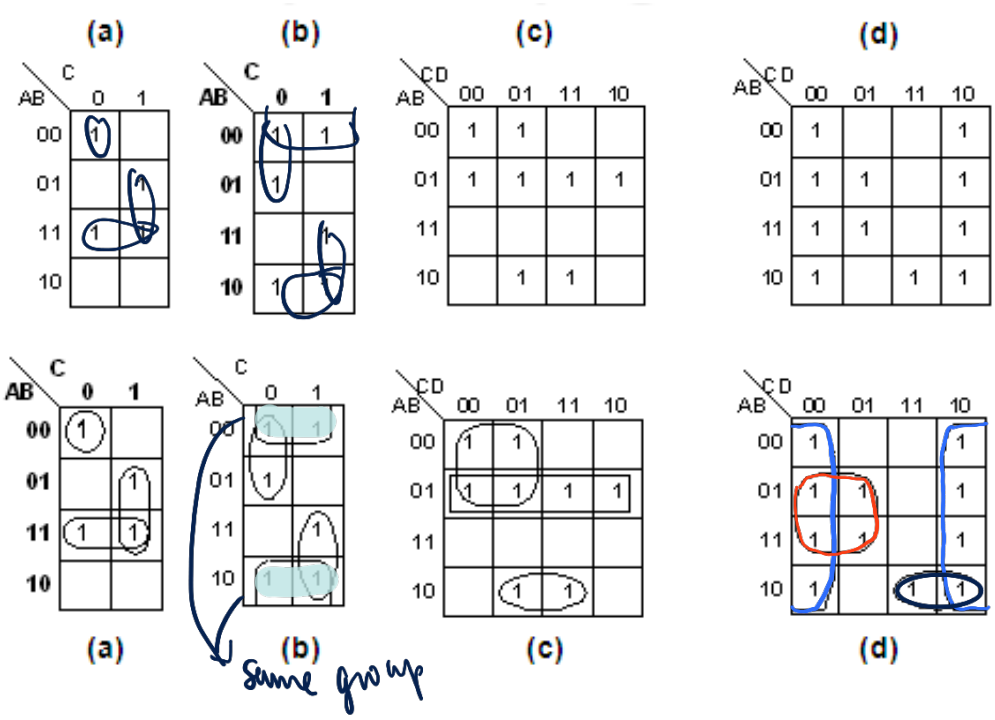
Step1: Grouping the 1s

- Group must contain cells in 2^x combination (i.e. 1, 2, 4, 8, 16)
- Each cell must be adjacent to at least 1 other cell in the group, but <u>all</u> cells in a group need not be adjacent
- Try to have the **biggest possible group of 1s**
- Each 1 must be in at least one group.
- The 1s already in a group may be included in another group so long as the overlapping group includes non-common 1s

Step2: Determine product term for each group

Step3: Summing the resulting product terms

Examples : Grouping the 1's



Determining the Minimum SOP Expression from K-map

Digital Logic

- Within each group made, only choose the variables that occur in one form : complemented or un-complemented. This is called *contradictory variables*.
- Determine the minimum product term for each group.
  - For a 3-variable map
    - A 1-cell group → 3-variable product term
    - A 2-cell group → 2-variable product term
    - A 4-cell group → 1-variable term
    - An 8-cell group → expression value = 1

For a 4-variable map:

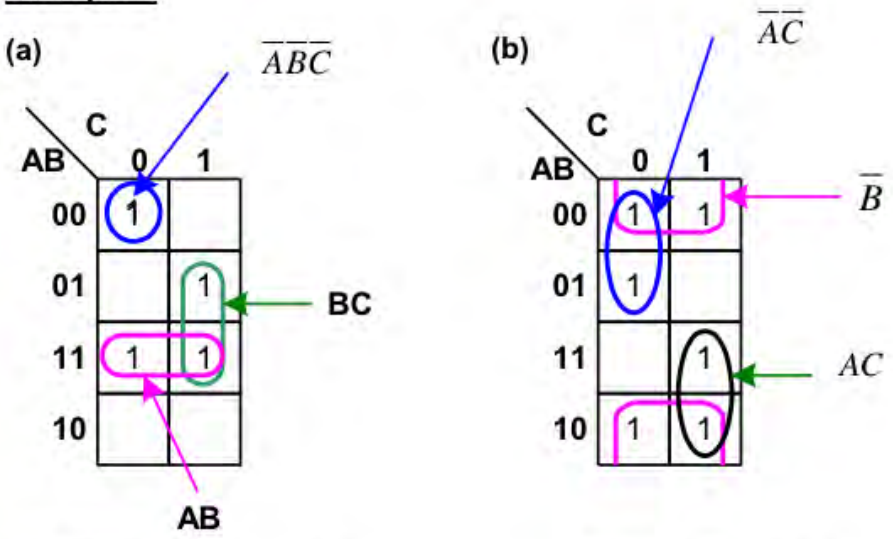
A 1-cell group → 4-variable product term
A 2-cell group → 3-variable product term
A 4-cell group → 2-variable term
An 8-cell group → 1-variable term
A 16-cell group → expression value = 1

When the minimum product terms are derived, sum them to form the minimum SOP.

Step 2 & Step 3

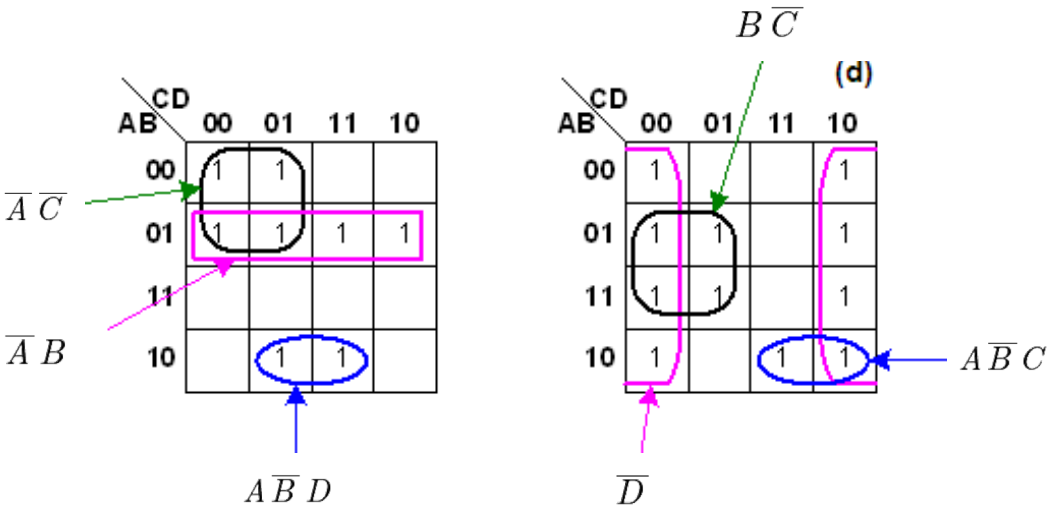
Digital Logic

Example 1



$$\overline{A}\,\overline{B}\,\overline{C} + BC + AB \qquad \overline{A}\,\overline{C} + \overline{B} + AC$$

Example 2



$$\overline{A}\,\overline{C} + \overline{A}\,B + A\,\overline{B}\,D \qquad \overline{D} + B\,\overline{C} + A\,\overline{B}\,C$$

Minimization of Boolean Expression using K-Map

Digital Logic

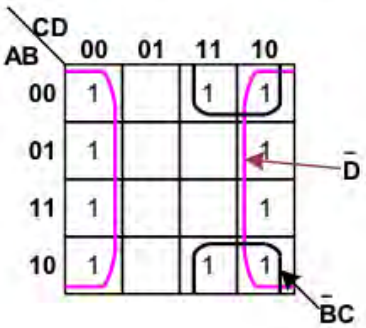
- Original expression:

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

- Change it to standard SOP expression

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

- Map it and get the minimized SOP expression



Minimized expression

$\bar{B}C + \bar{D}$

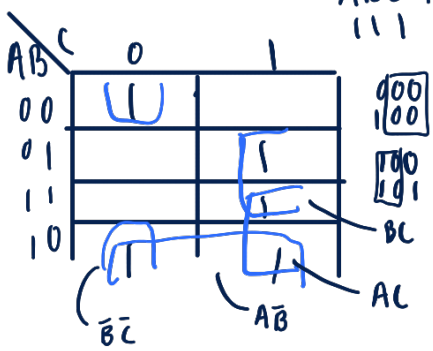
DIY: Minimized the following expression using K-Map

a) $BC + A\bar{B} + \bar{A}\bar{B}\bar{C}$

$BC(A+\bar{A}) + A\bar{B}(C+\bar{C}) + \bar{A}\bar{B}\bar{C}$

$ABC + \bar{A}BC + A\bar{B}C + A\bar{B}\bar{C} + \bar{A}\bar{B}\bar{C}$

111 011 101 100 000



$\therefore \bar{B}\bar{C} + A\bar{B} + AC + BC$

b) $AB + A(B + C) + B(B + C)$

$AB + AB + AC + BB + BC$

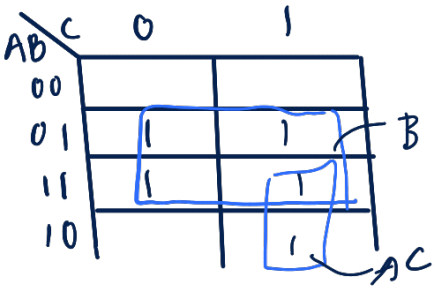
$AB + AC + B + BC$

$AB + AC + B$

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

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

111 110 101 011 010



$B + AC$

'Don't Care' conditions

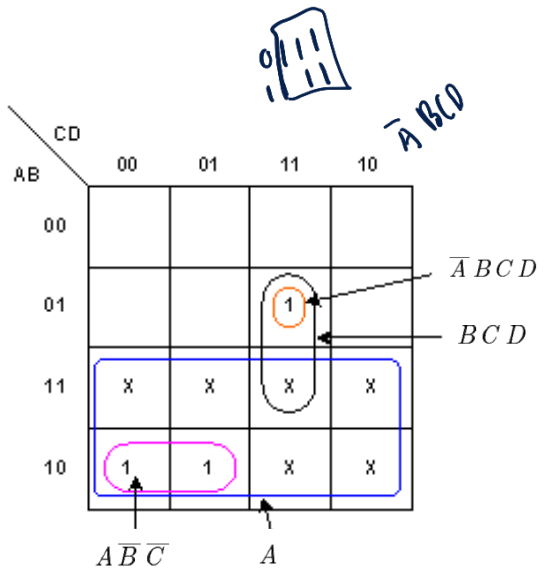
Digital Logic

- Sometime there are certain combinations of input variable that is not allowed.
  - Eg. 6 invalid combinations in BCD code (1010, 1011, 1100, 1101, 1110, 1111)
- These states will never occur in an application using BCD code, hence they can be treated as "***don't care***" terms.
- "*don't care*" terms may be assigned either a 1 or a 0; it doesn't matter as it will never occur.
- "*don't care*" terms can be used with K-maps to produce better results.
  - Used as part of a group → the larger the group, the simpler the resulting term will be.
  - When grouping 1s, the Xs can be treated as 1 to make the group larger.
  - Xs is treated as 0s if they cannot be used to advantage.

Example:

	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	0
	0	1	0	0	0
	0	1	0	1	0
	0	1	1	0	0
	0	1	1	1	1
	1	0	0	0	1
	1	0	0	1	1
Dec 10	1	0	1	0	X
11	1	0	1	1	X
12	1	1	0	0	X
13	1	1	0	1	X
14	1	1	1	0	X
15	1	1	1	1	X

$d(10,11,12,13,14,15)$



Without "don't cares" $Y = A\overline{B}\,\overline{C} + \overline{A}BCD$

With "don't cares" $Y = A + BCD$

K-Map Minimization: a note to remember

Digital Logic

- Making more grouping than necessary
  - More grouping will only produce more terms not a minimized expression, in a circuit this mean more gate need to be use

x \ y	0	0	1	1
1	1	1	1	1

z

x \ y	0	0	1	1
1	1	1	1	1

z

- Not making a larger grouping when possible
  - Larger grouping will produce a less variables in a term which mean less input pin to a gate

x \ y	0	0	1	1
1	1	1	1	1

z

x \ y	0	0	1	1
1	1	1	1	1

z

- Only include don't care (X) in a group if it can minimized an expression and ignore the other don't care (X) if all the 1's already in a group.

w \ x \ y	0	0	1	1
1	1	0	0	1
1	1	1	X	0
0	0	X	1	1
1	1	0	0	X

z

- During minimization it is possible to get more than one solution.

x \ y	0	1	0	1
0	0	1	1	1

z

$y'z + yz' + xz$

x \ y	0	1	0	1
0	0	1	1	1

z

$y'z + yz' + xy$

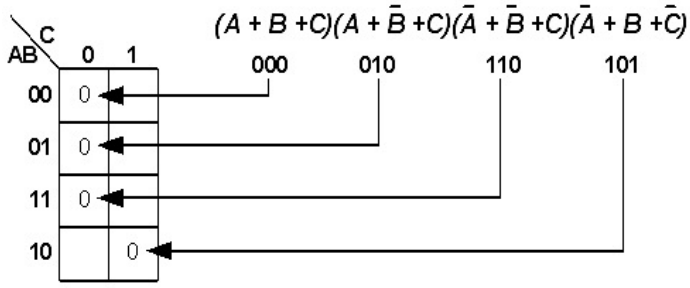
K-Map POS Minimization

Digital Logic

- For POS expression in standard form, a 0 is put in the K-map for each sum term.
- The methods are similar to SOP minimization, except 0 is used.

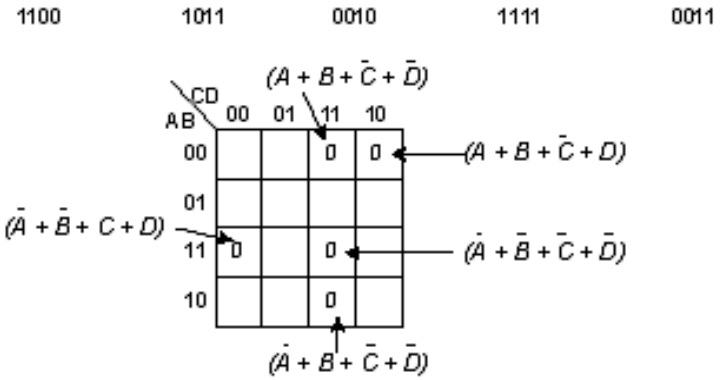
- To map a Standard POS expression:
  - **Step 1**: Determine the binary value of each sum term (i.e. that makes the sum term = 0)
  - **Step 2**: Check result and place a 0 on the corresponding cell in K-map

Example:



Try mapping the following Standard POS expression:

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



K-Map Simplification of POS Expressions

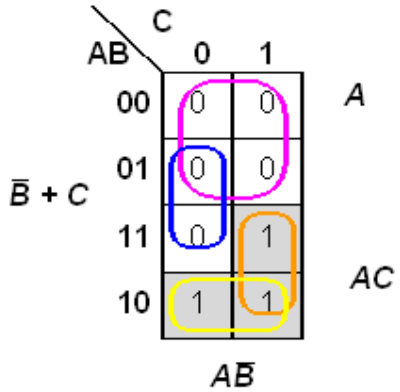
Digital Logic

- The process is basically the same as with SOP expressions:
  - Group 0s instead of 1s.
  - The rules of grouping 0s are the same as those for 1s
  - Expression must be in Standard POS form.

Example:

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

0+0+0 0+0+1 0+1+0 0+1+1 1+1+0



Minimum POS expression = $A(\bar{B} + C)$
The grouping of 1s yields an SOP expression that is equivalent to the grouping of 0s

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

Minimize this POS expression using a K-map

Digital Logic

$(B + C + D)(A + B + \overline{C} + D)(\overline{A} + B + C + \overline{D})(A + \overline{B} + C + D)(\overline{A} + \overline{B} + C + D)$

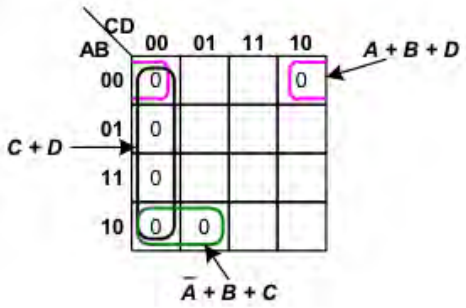
- Change it to standard POS form

$$(A+B+C+D)(\overline{A}+B+C+D)(A+B+\overline{C}+D)(\overline{A}+B+C+\overline{D})(A+\overline{B}+C+D)(\overline{A}+\overline{B}+C+D)$$

- Get the binary values

(0000)(1000)(0010)(1001)(0100)(1100)

- Map it, then group the 0s



- Minimum POS expression

$(C + D)(\overline{A} + B + C)(A + B + D)$

DIY: Minimize the following POS expression using K-Map

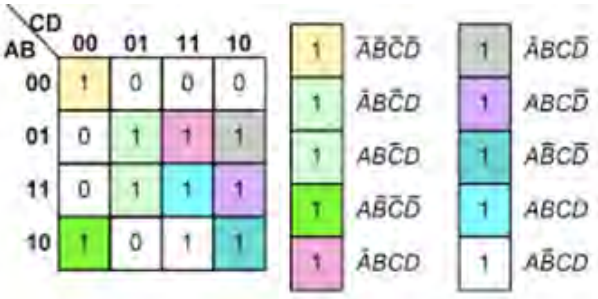
$$(A+\overline{B}+C)\,(\overline{B}+C+\overline{D})\,(A+\overline{B}+\overline{C}+D)$$

Converting between POS and SOP using K-Map

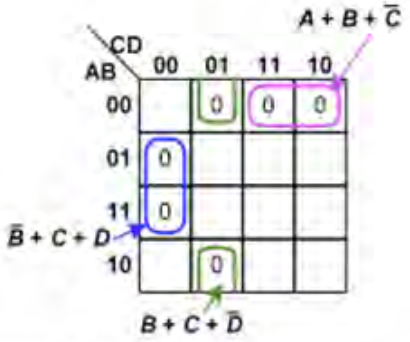
Digital Logic

- A mapped SOP expression can be converted to an equivalent POS expression.
- This is a good way to compare which can be implemented using fewer gates.
- Given a minimum POS map, the 1s will yield a standard SOP expression.
- This SOP expression can then be minimized by grouping the 1s.

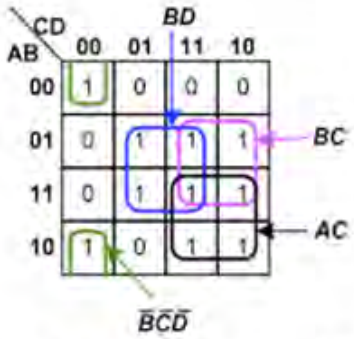
Examples: Converting between POS and SOP using K-Map



Standard SOP: $\bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}\bar{D} + AB\bar{C}\bar{D} + A\bar{B}\bar{C}\bar{D} + \bar{A}BCD + \bar{A}BC\bar{D} + ABC\bar{D} + A\bar{B}C\bar{D} + ABCD + A\bar{B}CD$



Minimum POS:-
$(A + B + \bar{C})(\bar{B} + C + D)(B + C + \bar{D})$



Minimum SOP: $AC + BC + BD + \bar{B}\bar{C}\bar{D}$

Simplify a Combinational Logic Circuit

Digital Logic

- Simplify a combinational logic circuit will result in lesser gates used
- How to do this?

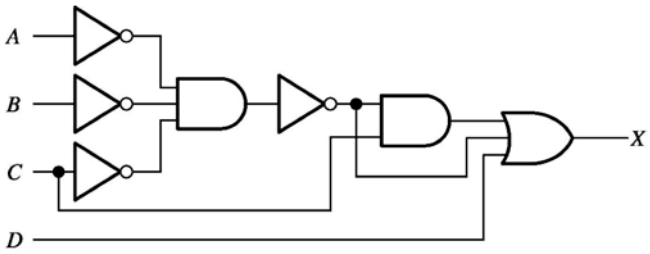
Step 1 : Determine the input and output variables of each gates in the circuit

Step 2 : Get the final output expression

Step 3 : Apply De Morgan's theorem and Boolean algebra or K-map.

Example: Given the logic circuit below, simplify using Boolean and DeMorgan's theorem

Step1: input variables = A, B, C, D
output variables = X



Step 2: Get the final output expression → $X = (\overline{\overline{A}\,\overline{B}\,\overline{C}})C + \overline{\overline{A}\,\overline{B}\,\overline{C}} + D$

Step 3: Apply De Morgan's theorem and Boolean Algebra

$$(\overline{\overline{ABC}})C + \overline{\overline{ABC}} + D$$

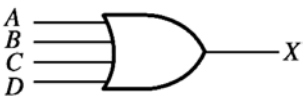
$$= (\overline{A} + \overline{B} + \overline{C})C + (\overline{A} + \overline{B} + \overline{C}) + D \qquad \overline{AB} = \overline{A} + \overline{B}$$

$$= AC + BC + CC + A + B + C + D \qquad \overline{\overline{A}} = A$$

$$= A + AC + B + BC + C + D \qquad C.C = C \qquad C + C = C$$

$$= A + B + C + D \qquad A + AC = A \qquad B + BC = B$$

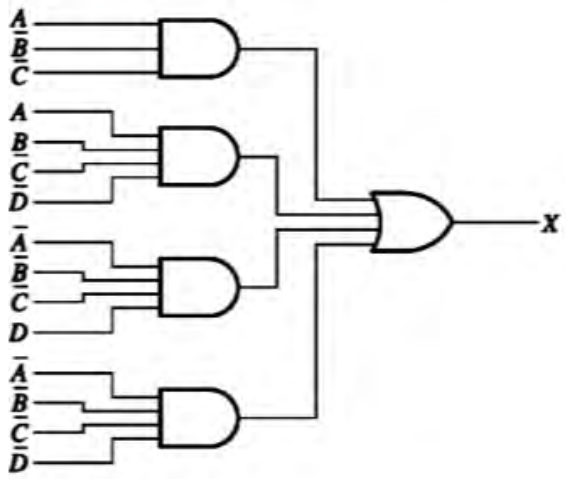
After simplification:



.. Simplify a Combinational Logic Circuit

Digital Logic

Simplify the combinational logic circuit using K-Map



Step 1 Input variable = A, B, C, D, Output variable = X

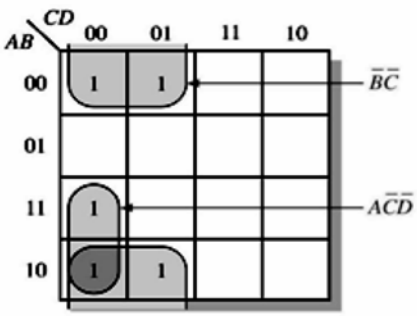
Step 2 Output Expression of the circuit is

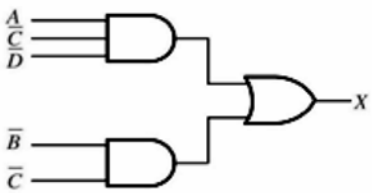
$$X = A\overline{B}\overline{C} + AB\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}\overline{D}$$

Change it to standard SOP form:

$$X = A\overline{B}\overline{C}D + A\overline{B}\overline{C}\overline{D} + AB\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}\overline{C}\overline{D}$$

Step 3 Use K-map to simplify the circuit





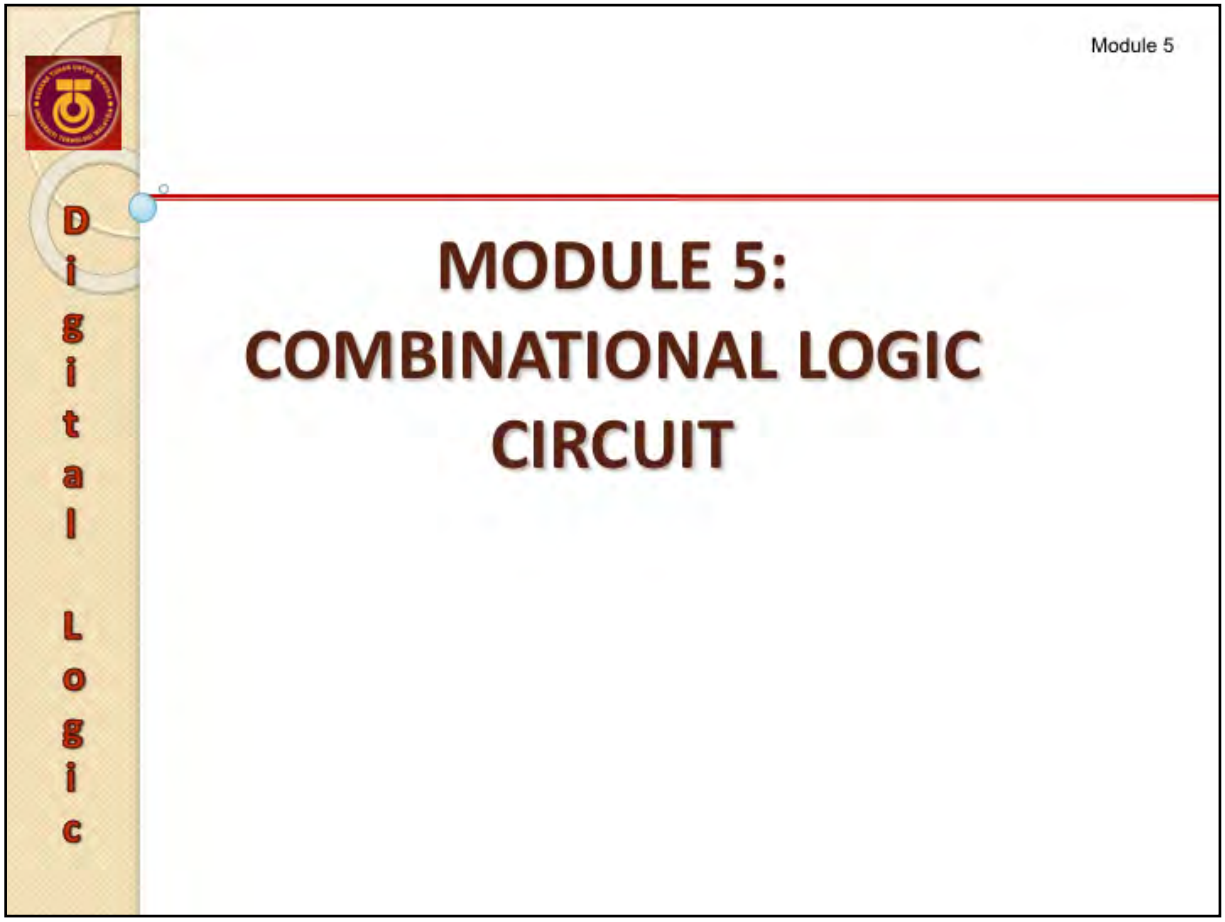
DIY

Use K-map or De Morgan's Theorem and Boolean Algebra to reduce the circuit.

X = ABC + AB + AC

Digital Logic

End of Module 4



Objective:

1. To introduce AND-OR and AND-OR-Invert Logic
2. To illustrate the conversion between different representation of digital circuit
3. To introduce the universality of gate
4. To introduce a dual symbol in a digital circuit
5. To explain the design a combinational circuit

Module 5

Basic Combinational Logic Circuits

Digital Logic

- AND-OR Logic
- AND-OR-Invert Logic
- Exclusive-OR (XOR)Logic
- Exclusive-NOR (XNOR) Logic

- In designing a circuit, we can make the output to become 0 (low) or 1 (high)
  - If we want the output to be active high, then the final circuit will have a SOP expression. This kind of circuit can easily be implemented by **AND-OR Logic**
  - To get an active low output, the output of the circuit with an active high output (AND-OR Logic) has to be inverted to logic 0 by adding the inverter, therefore the circuit becomes **AND-OR-Invert Logic.**

- Some circuit is simpler if not in a form of SOP or POS, this is true for a circuit that produces a checker board pattern on the K- Map

cd \ ab	00	01	11	10
00		1		1
01	1		1	
11		1		1
10	1		1	

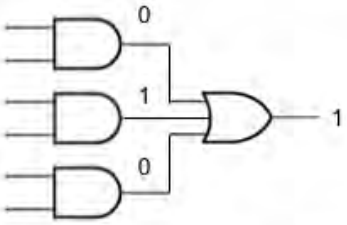
$$X = A \oplus B \oplus C \oplus D$$

- This kind of circuit can be implemented efficiently by using an **XOR** or **XNOR** logic

AND-OR Logic

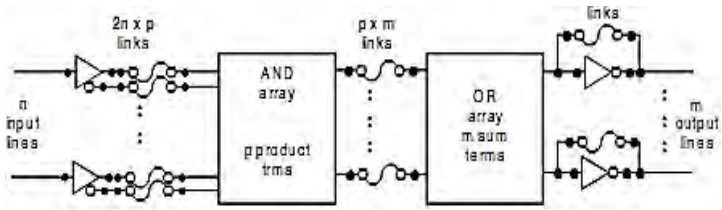
Digital Logic

- A circuit consisting of any number of AND gates followed by an OR gate.
- Example: SOP expressions (e.g. AB + CD)
- If **any** of the AND gates output are HIGH, the output of the OR gate is HIGH



- If **all** of the AND gates outputs are LOW, the output in the OR gate is LOW

Programmable Logic Array (PLA) is an integrated circuit that has an array of AND-OR logic. The connection between component is through a link, a link can be intact (connected) or it can be blown (no connection)



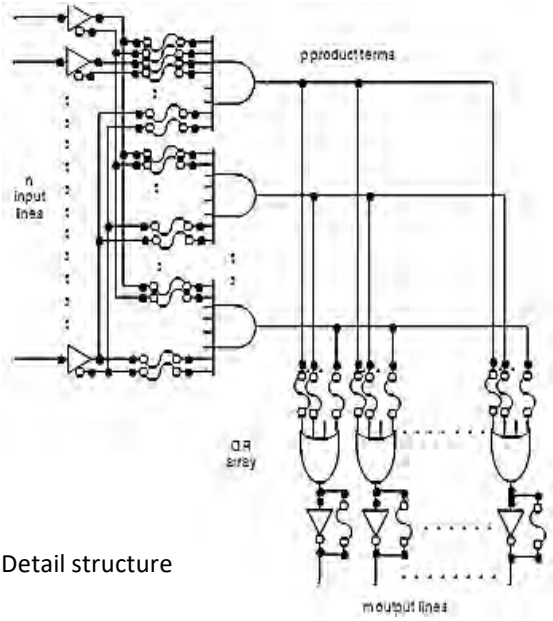
General structure

- The font inverter is to get the complemented form of an input
- The final inverter is whether to have a AND-OR logic or AND-OR-Invert logic
- Example: implement F_1,F_2 and F_3 using AND-OR logic (PLA)

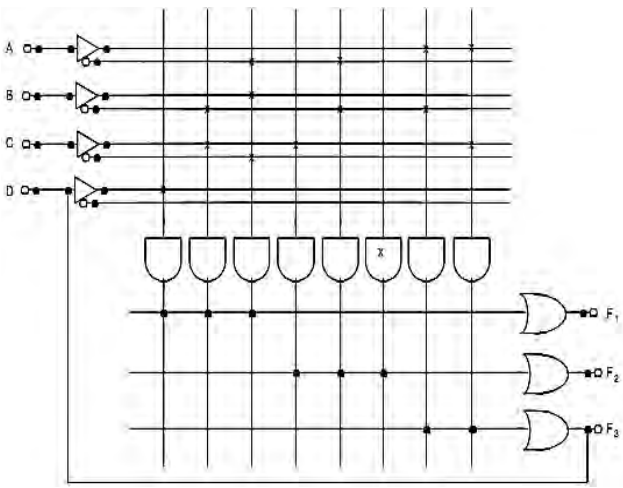
F_1 (A,B,C) = F_3 + B'C + A'BC'

F_2 (A,B,C) = C + A'B'

F_3 = AB' + AC.



Detail structure



Example: AND-OR Logic

Digital Logic

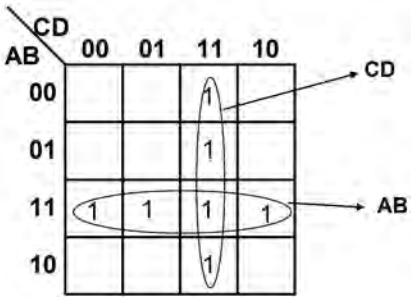
Based on the truth table:
- derive the expression
- simplify using K-Map
- Implement using AND-OR logic

INPUTS						OUTPUT
A	B	C	D	AB	CD	X
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	1	1
0	1	0	0	0	0	0
0	1	0	1	0	0	0
0	1	1	0	0	0	0
0	1	1	1	0	1	1
1	0	0	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	0	0	0
1	0	1	1	0	1	1
1	1	0	0	1	0	1
1	1	0	1	1	0	1
1	1	1	0	1	0	1
1	1	1	1	1	1	1

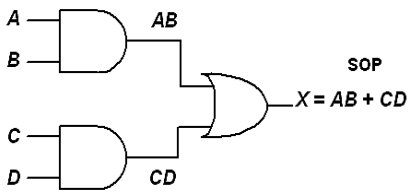
- To have an active high output, determine the SOP by identifying the output is =1
  - ✓ Write the Boolean expression or map the 1's to the K-Map

$$X = \overline{A}\,\overline{B}CD + \overline{A}BCD + A\overline{B}CD + AB\overline{C}\,\overline{D} + AB\overline{C}D + ABC\overline{D} + ABCD$$

  - ✓ Simplify the output expression by using Boolean algebra or K -Map



  - ✓ The simplified expression is X = AB + CD
  - ✓ This expression will produce an AND-OR circuit

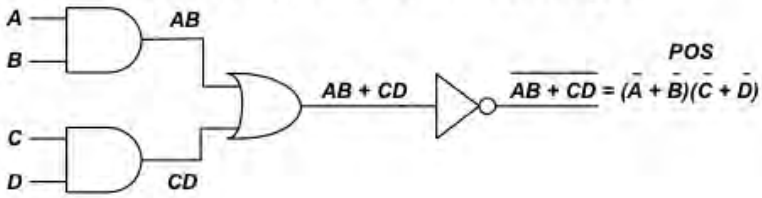


AND-OR-Invert Logic

- It is the complement of AND-OR circuit: **AND** followed by **OR** followed by **NOT** gate
- Example: POS expression $(\overline{A}+\overline{B})(\overline{C}+\overline{D})$

  is actually $\overline{\overline{(\overline{A}+\overline{B})(\overline{C}+\overline{D})}} = \overline{AB+CD}$

  which is the complement of AND-OR circuit, therefore can be implemented using AND-OR-Invert logic.



- If we want active low, put an inverter at the output of SOP circuit, it will produce a POS for $\overline{X}$ (note: *not POS for X*)
- This kind of circuit can be implemented using AND-OR-Inverter (AOI) Logic
- Programmable Logic Device (PLD) using this kind of arrangement to allow the flexibility of producing an active high or an active low output.

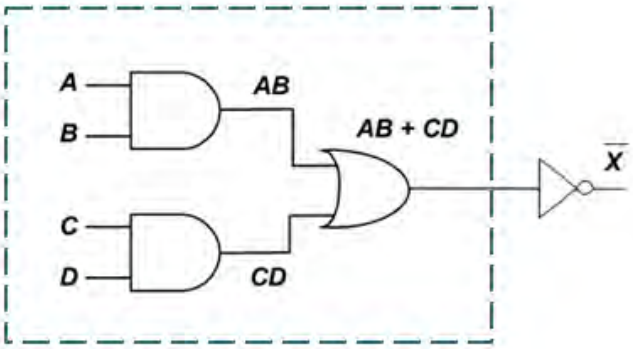
- *AOI gates are particularly advantaged in that the total number of transistors (or gates) is less than if the AND, NOT, and OR functions were implemented separately.*
  - *This results in increased speed, reduced power, smaller area, and potentially lower fabrication cost.*
  - *For example, a 2-1 AOI gate can be constructed with 6 transistors in CMOS compared to 10 transistors using a 2-input NAND gate (4 transistors), an inverter (2 transistors), and a 2-input NOR gate (4 transistors).*

Example: AND-OR-Invert Logic

Digital Logic

Refer to the previous example of AND-OR Logic truth table, solve the problem for an active low output using AND-OR-Invert logic.

To implement an active LOW output, take the complement of X.



DIY

Based on the truth table below, implement the active low output by applying the AND-OR-Invert logic.

sensor inputs

A	B	C	Output
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

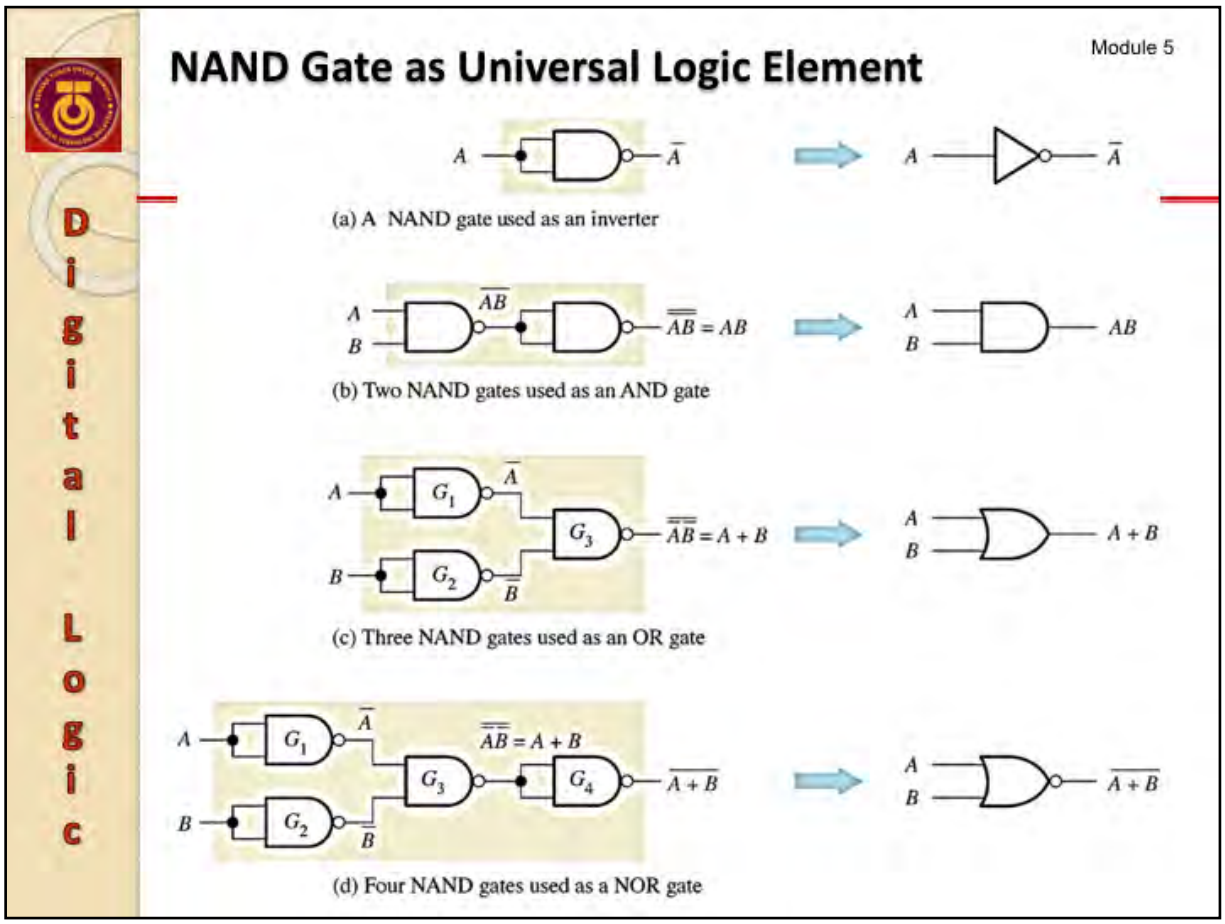
Universal Property of NAND and NOR Gates

Digital Logic

- If a gate can be converted and perform the function of a basic gates, then it can be used to implement any logic circuits. Therefore they are called universal gate.
- There are 2 common universal gates:
  - The NAND Gate
  - The NOR Gate

- It's a good thing to implement the final circuit using only a single type of gate. The advantages are
  - ✓ Have to stock only one type of gates
  - ✓ Buy only one type of gates in a volume so that, it can reduce cost
  - ✓ Can reuse the extra of unused gates to implement other gates

- This can be done by using a universal gate -- NAND or NOR
  - ✓ Usually SOP is implemented using NAND only
  - ✓ and POS implemented as NOR only



- To prove that a gate is universal – it must be able to perform as an AND, OR and Inverter (NOT) gate.
- To prove that NAND can function as an AND, OR and Inverter (NOT) gate
- (a) NAND to NOT:

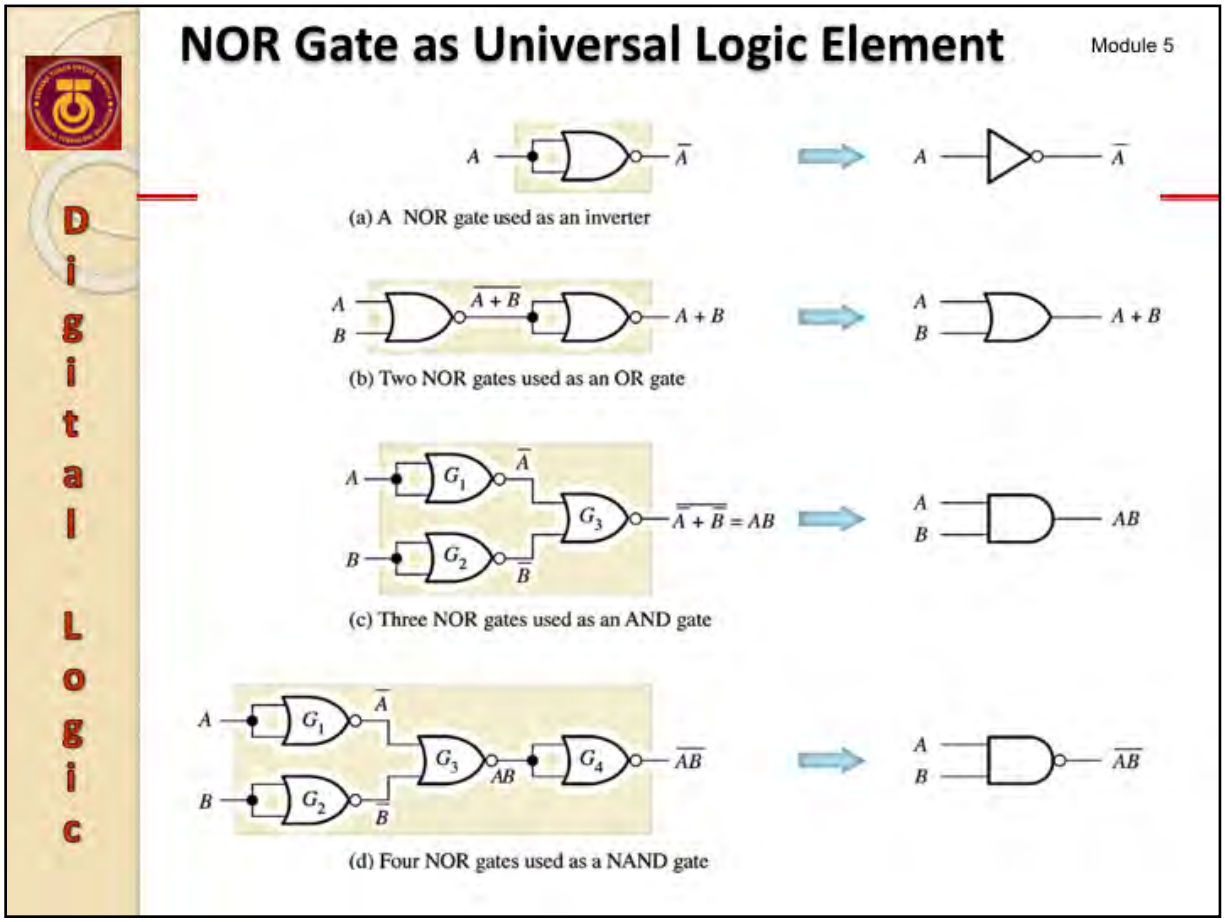
  - tie both input NAND ➔ $\overline{A.A} = \overline{A}$
- (b) NAND to AND:

  - add an inverter at the output of the NAND, it will cancel out the effect of the bubble ➔ $\overline{\overline{AB}} = AB$
- (c) NAND to OR:

  1. from OR put the bubble at all input and output, change the input bubble to an inverter by using NAND gate
  2. change the OR symbol to AND ➔ $\overline{\overline{A}\,\overline{B}} = A + B$

- (d) NAND to NOR:

  1. from OR put the bubble at all input and output, change the input bubble to an inverter by using NAND gate
  2. change the OR symbol to AND ➔ $\overline{\overline{A}\,\overline{B}} = A + B$
  3. tie both input ➔ $\overline{A + B}$



DIY

Question 1:
Use NAND gates to implement each expression:

a) $X = \overline{A} + B$

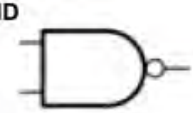
b) $X = A\overline{B}$

Question 2:
Using the same equation as Question (1) implement the corresponding circuits using NOR gates.

Dual Symbol

Digital Logic

NAND



Alternate symbol for NAND – Negative OR

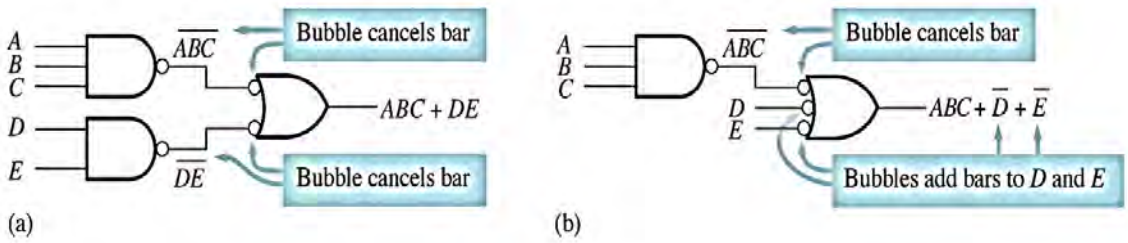
NOR

Alternate symbol for NOR – Negative AND

Converting to an alternate symbol

Step1 : Pull bubble at all input and output
Step 2 : Change the symbol AND to OR, OR to AND
Step 3 : Redraw the symbol, if a bubble connected to another bubble it will cancel out the bubble

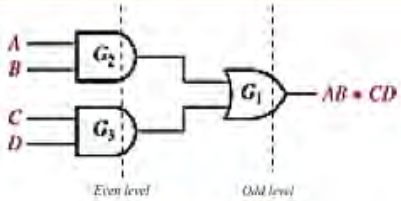
Be careful $\overline{DE} \neq \overline{D} + \overline{E}$



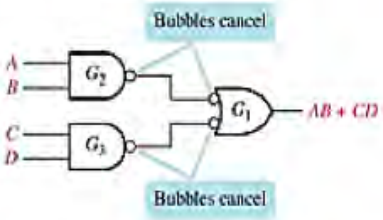
Converting SOP circuit to NAND only

Digital Logic

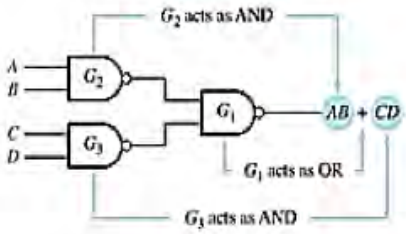
- **Step1:** Put the bubble at the input of the right most OR (Odd level)



- **Step2:** Compensate the effect of putting that bubble, add another bubble at the output of the AND gate



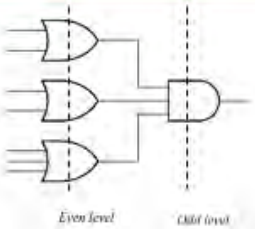
- **Step 3:** Replace the negative OR symbol with a NAND symbol



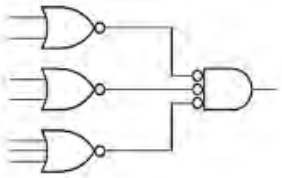
Converting POS circuit to NOR only

Digital Logic

Original POS circuit

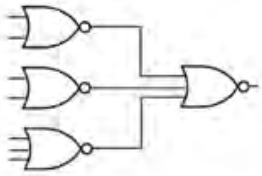


Step1: Put the bubble at the input of the right most AND



Step2: to compensate the effect of putting that bubble, add another bubble at the output of the OR gate

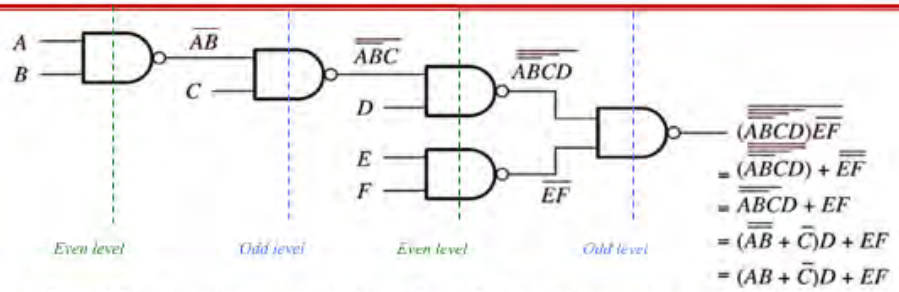
Step 3: Replace the negative OR symbol with a NOR symbol (no need, just to show NOR with the same symbol)



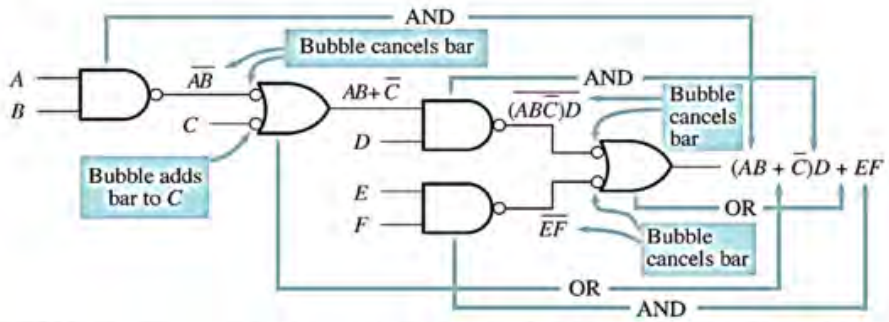
Dual Symbol : NAND only – (I)

- Dual symbol is needed to simplify the "reading" of the schematic
  - ✓ can determine how the circuits works – very fast
- Replace the NAND gate with its dual symbol which is Negative-OR
- Choose which level that you want to replace the alternate symbol -- odd or even
  - ✓ If you choose odd – only replace the odd level with an alternate symbol

Dual Symbol : NAND only – (2)

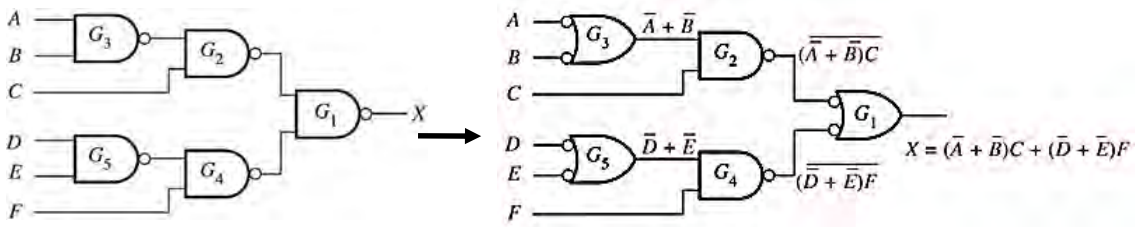


(a) Several Boolean steps are required to arrive at final output expression.

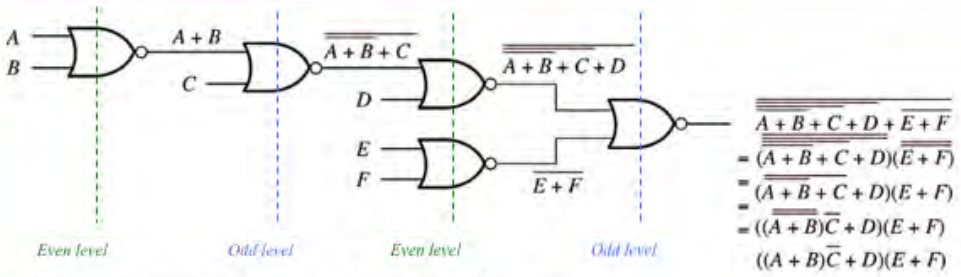


(b) Output expression can be obtained directly from the function of each gate symbol in the diagram

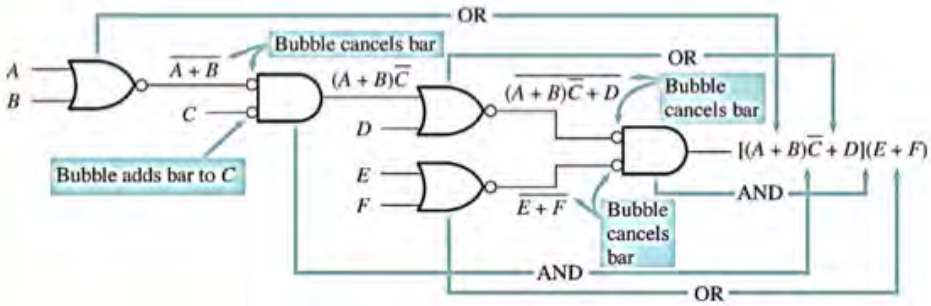
Example : (Odd level)



Dual Symbol : NOR only



(a) Final output expression is obtained after several Boolean steps.

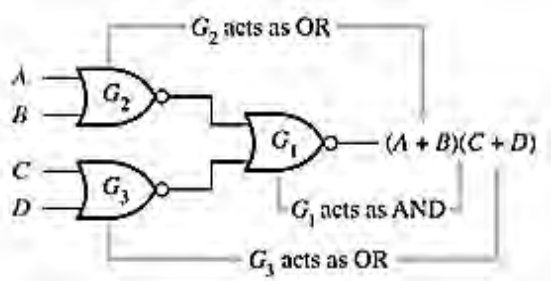


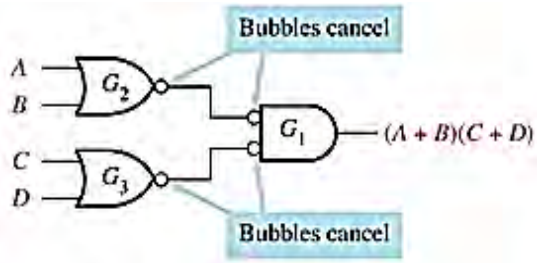
(b) Output expression can be obtained directly from the function of each gate symbol in the diagram.

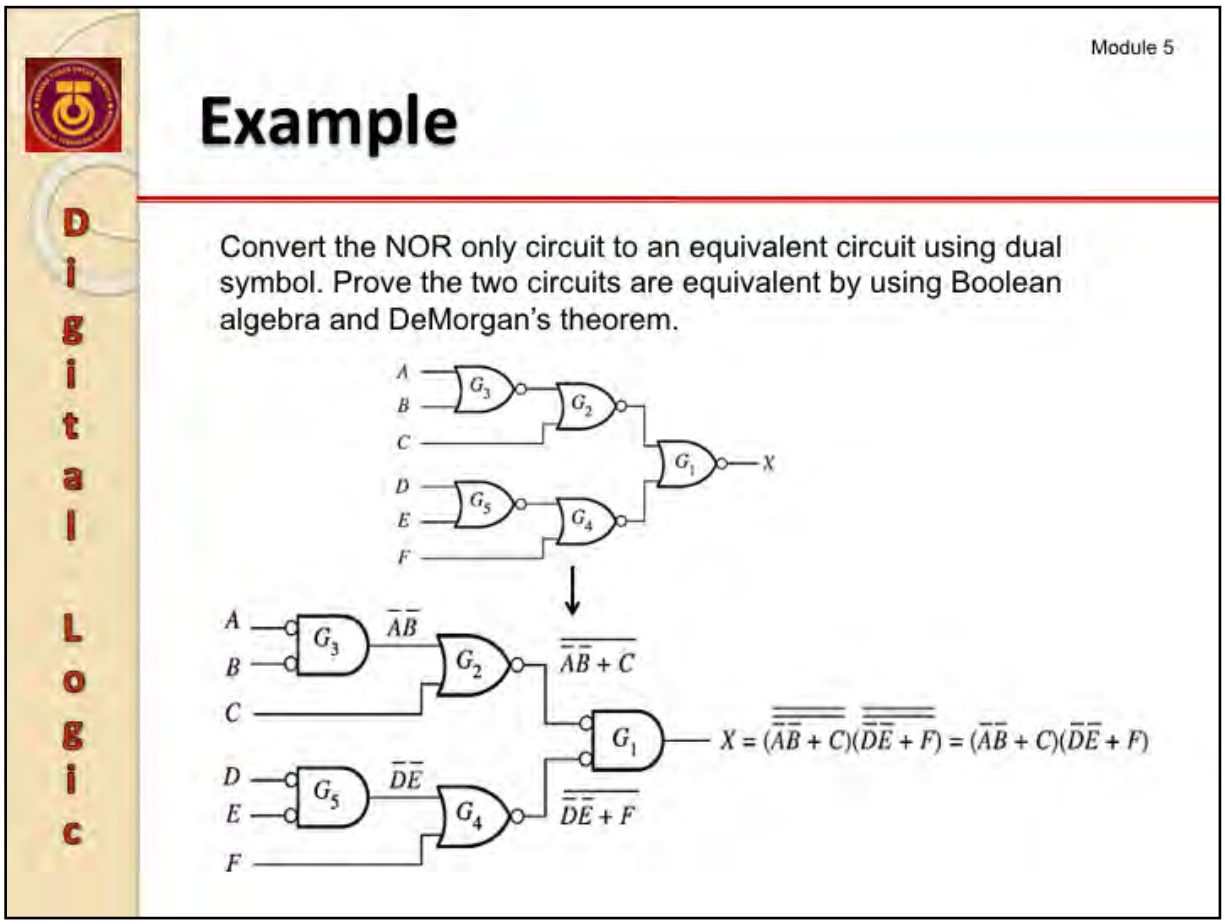
- Replace Odd or even level NOR with its dual symbol which is Negative-AND

Example:

Implement the expression $(A+B)(C+D)$ with NOR logic using appropriate dual symbol.







DIY

Question 1:

Implement the expression using NAND logic and produce the answer using dual symbol.

$$X = \overline{(\overline{A} + \overline{B} + \overline{C})DE}$$

Question 2:

Implement the expression using NOR logic and produce the answer using dual symbol.

$$X = \overline{\overline{A}\,\overline{B}\,\overline{C} + (D + E)}$$

Steps in designing a combinational circuits

Digital Logic

1. Understand the problem to be solved
2. Convert the problem statement to a Boolean expression or truth table, whichever is most natural for the given problem
3. Simplify the output by using Boolean algebra, K- Map or other technique
4. Draw the circuit from the simplified output
5. Convert the simplified circuit to universal gates only
   - SOP to NAND only
   - POS to NOR only
6. Redraw the final circuit using dual symbol

The most critical stage is step 1.

Determine 3 things:

1. How many **inputs** and give it a variable name?
2. How many **outputs** and give it a variable name?
   - If more than one output treat each output separately, design a circuit for each output and finally combine them together
3. What is the **relationship** between output(s) and the inputs?

The problem statement

Digital Logic

- In a certain chemical-processing plant, a liquid chemical is used in a manufacturing process. The chemical is stored in three different tanks. A level sensor in each tank produces a HIGH voltage when the level of chemical in the tank drops below a specified point.
- Design a circuit that monitors the chemical level in each tank and indicates when the level in any two of the tanks drops below the specified point.

Step 1:

Input – 3 tanks – call it A,B,C - if drop below specified point it will input a HIGH (1) else LOW (0).

Output – 1 – call it X - produces HIGH (1) if active.

Relationship – any two tanks drop below specified point (has logic 1) the output active (HIGH).

Step 2:

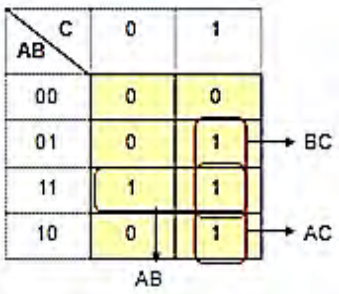
Convert to standard form:

Truth table – the easiest in this case

A	B	C	X
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

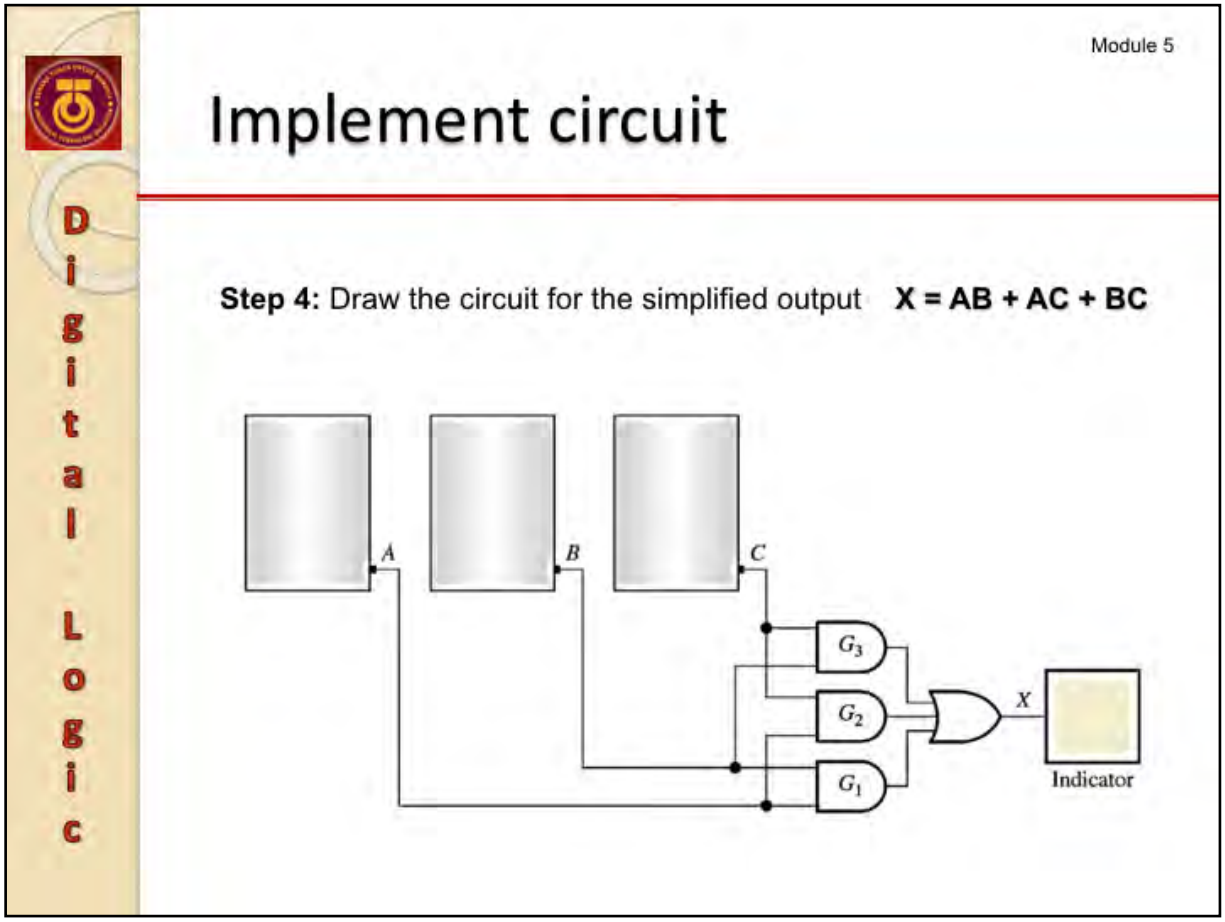
Step 3:

Map to K-Map then simplify



Simplified Equation

X = AB + BC + AC



DIY

Step 5: Draw the circuit using dual symbol.

Step 6: Implement the circuit using universal gates.

Example: Multiple output..

Digital Logic

Design a combinational circuit with 3 inputs A, B and C and 2 outputs X and Y.

- When the binary input is 0, 1, or 2 the binary output is one greater than the input.
- When the binary input is 3, 4, 5, or 6 the binary output is three less than the input.
- The binary input will never be 7.

Step1:

Input – 3 – which is A, B and C

Output – 2 – which is X and Y

Note: *for N number of output, treat each output separately, go through the normal design process, it will produce N different circuits, finally combine all N circuits into one.*

Relationships

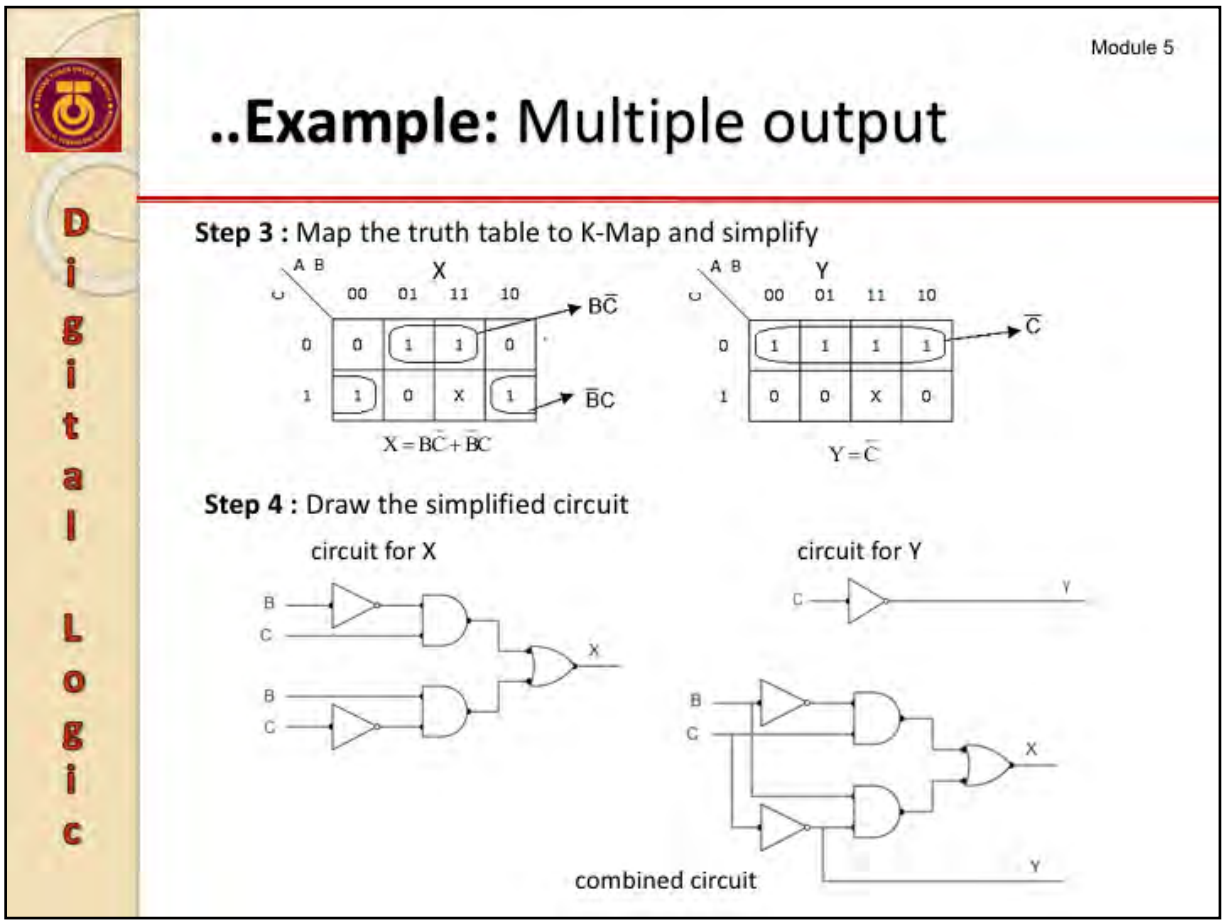
- for 0, 1, 2 the output will be 1, 2, 3
- for 3, 4, 5, 6 the output will be 0, 1, 2, 3
- for 7, it will never exist therefore treat it as don't care

Note: *input A, B, and C will be combined and treated as 3 bits binary number (ABC), X and Y combined together and treated as a binary number (XY)*

Step2:

Use Truth table – the easiest in this case →

A	B	C	X	Y
0	0	0	0	1
0	0	1	1	0
0	1	0	1	1
0	1	1	0	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	1
1	1	1	X	X



DIY

Step 5: Draw the circuit using dual symbol.

Step 6: Implement the circuit using universal gates.

Example: Direct to Boolean expression

Digital Logic

- Design a logic circuit for a car alarm that fulfill the following requirement:
  - The alarm will be triggered if the alarm system is activated and any of the two doors in the trunk are open , or if the vibration sensor is activated and the key is not in the ignition.

Step1:

Input – determine a factor that will trigger an alarm, there are 6

Input	Condition when HIGH
AlarmActivated	Activated
DriverDoorOpen	Open
PassengerDoorOpen	Open
TrunkOpen	Open
Vibration	Vibrate
KeyIgnition	In the ignition

Output – 1 output – signal to trigger an alarm → 0 – alarm off, 1- alarm on, call it X

Relationship:

- Look for the word AND, OR , NOT (complement, invert, not the same) that connect between the input variables.
- *The alarm will go off if the alarm system is activated **and** any of the two doors **or** the trunk are open , **or** if the vibration sensor is activated **and** the key is **not** in the ignition.*

Module 5

..Example: Direct to Boolean expression

Digital Logic

Step2:

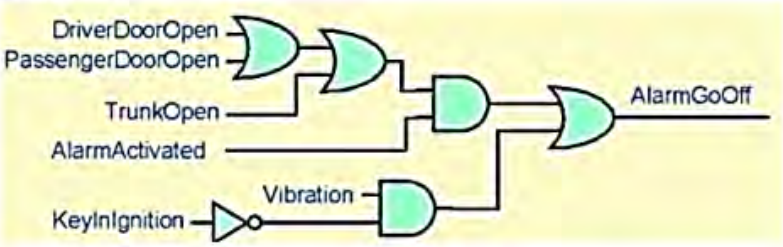
This kind of problem should not be converted to K-Map because it has 6 input and K-Map with variable more than 5 will be difficult to simplify manually.

It is much more easier to convert the problem statement directly to Boolean expression

$$AlarmGoOff = AlarmActivated.(DriverDoorOpen+PassengerDoorOpen+TrunkOpen)+Vibration.\overline{KeyInIgnition}$$

Step3: Simplify the Boolean expression if possible. In this case it is not possible

Step 4: Draw the Circuit



** Note: using only 2 input gate

Converting from problem statement to Boolean expression.

Example 1: "I'll go to lunch if Maisarah goes OR Anis goes, AND Sabri does not go."

Let F represent my going to lunch (1 means I go, 0 I don't go)

Likewise, M for Maisarah going, A for Anis, and S for Sabri

Then **F = (M OR A) AND NOT(S)**

Example 2: An alarm system should triggered if a movement is sensed and the system is set to enabled.

Let Boolean variable.

M represent "movement is sensed,"

E represent "enabled,"

F represent "alarm triggered."

Then an equation is: **F = M AND E**

DIY:

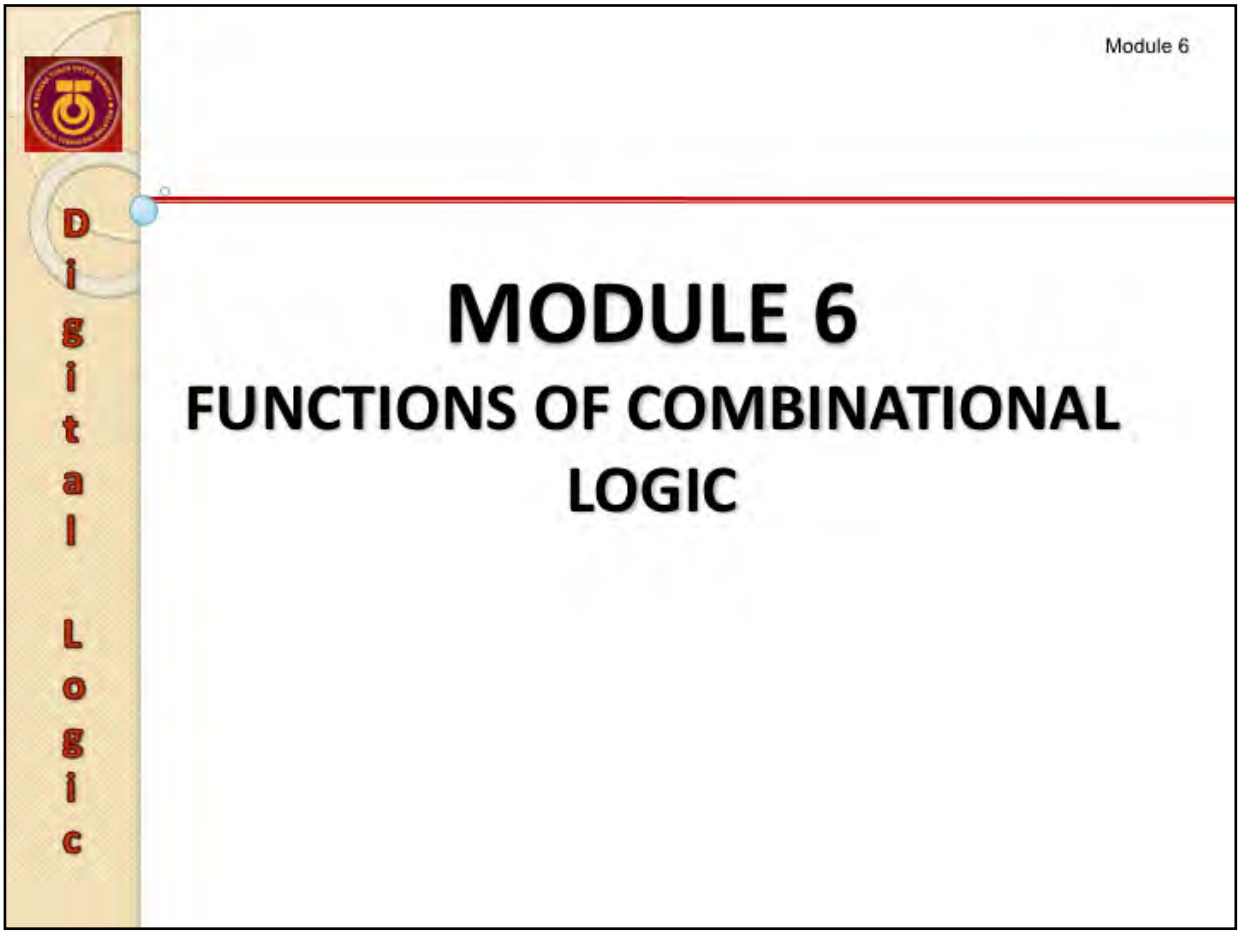
Convert the following statement to Boolean Algebra:

a) I'll come to class on Monday if there is a class on Monday and there is a bus service to lecture hall and I don't wake up late.

b) Two liquids can only be mixed if the room is dark and nobody is in the room or the temperature of the mixture less than 100◦C while there is somebody in the room.

Digital Logic

End of Module 5



Objectives:

1. To introduce a functions of digital circuits.
2. To show the design of the functions.
3. To introduce how to use an MSI chip of the functions.

FUNCTIONS OF COMBINATIONAL LOGIC

Digital Logic

- Basic Adders
- Parallel Binary Adder
- Comparators
- Decoders
- Encoders
- Multiplexers (Data Selector)
- DeMultiplexers (Data Distributor)
- Code Converter
- Parity Generators / Checkers

- Any digital circuit may consist one or more of these functions
- Because these functions are common, it is produced as a single IC
- The IC is a **M**edium **S**cale **I**ntegration (MSI) chip
    - The IC helps to make circuit development faster because there is no need to redesign all the functions from scratch
- It is important to know how to use the IC

Basic Adder

Digital Logic

- Basic adder operation is fundamental to the study of digital system.
- Types of basic adder :
  - Half Adder
  - Full Adder
- Basic rules for binary addition

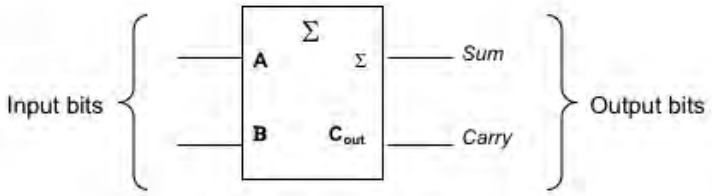
0+0= 0
0+1= 1
1+0= 1
1+1=10

- Adder is a basic circuit in a digital arithmetic operation.
- As mentioned in Module 1, all arithmetic operations can be implemented by using an adder.

Half Adder

Digital Logic

- Accepts two binary digits on its inputs and produces two binary digits on its outputs, a sum bit and a carry bit
- It meant for adding 1-bit number



$$\text{Sum} = A \oplus B$$

- Sum output is 1, if A and B are not equal.

$$C_{out} = AB$$

- Output carry is 1 only when A and B are 1s.

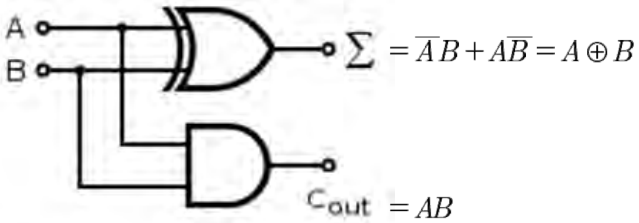
The Design:

A	B	Carry, C_{out}	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

From the truth table:

Sum, $\sum = \overline{A}B + A\overline{B} = A \oplus B$

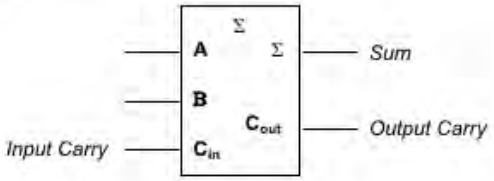
Carry, $C_{out} = AB$



Full Adder..

Digital Logic

- Accepts two input bits and **an input carry** and generates a sum output and an output carry.
- Suitable for adding more than 1 bits by cascading the full adder



- Two or more full adders are connected to form a parallel adders.

```
      1
     11
+    01
   1 00
```

Carry bit from right column

In this case, the carry bit from 2nd column becomes a sum bit

The Design:

Step 1 : Produce a Truth table

Cin	A	B	Cout	Σ
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

Step 2: Map the K-Map

Σ, Cin \ AB	00	01	11	10
0	0	1	0	1
1	1	0	1	0

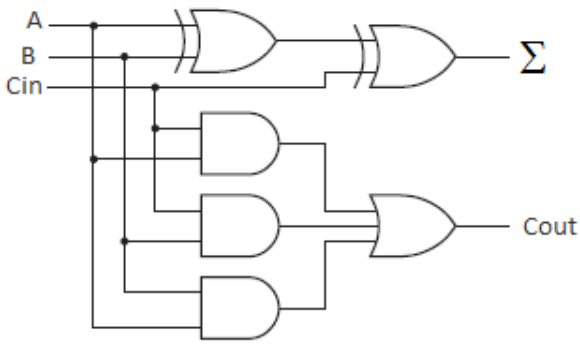
Cout, Cin \ AB	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Step 3: Simplify K-Map & get the equation

$$\Sigma = (A \oplus B) \oplus Cin$$

$$Cout = AB + ACin + BCin$$

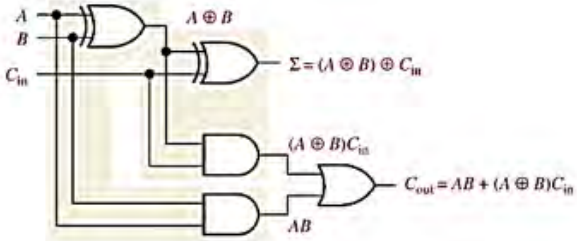
Step 4 : Draw the circuit

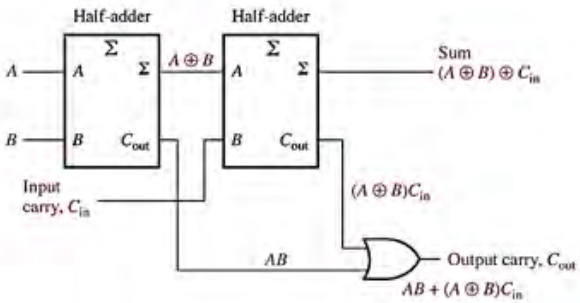


..Full Adder

Digital Logic

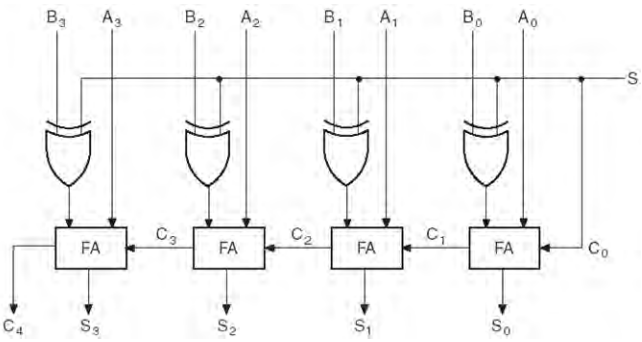
- Full Adder actually can be built by using 2 Half Adder
  - Half Adder is the one shaded



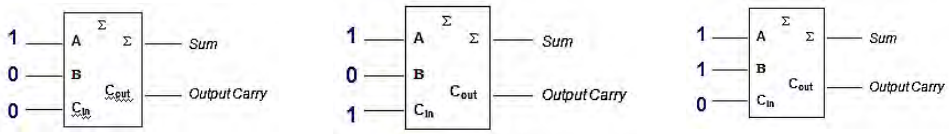


Adder/Subtractor Circuit

- The circuit implement the 2's complement arithmetic
- If S=0 implement **A + B**, no changes to B because B⊕S = B⊕0 = B, therefore full adder will do A+B
- If S =1 implement **A - B**, -B will be the 2's complement of B because B⊕1 = $\overline{B}$ which is the 1's complement of B. To get the 2's complement of B we have to add 1 by connecting C_0 to S so that C_0 = S = 1, therefore full adder will do ***A + 2's complement of B*** which is **A-B**



DIY: For each of the three full adders, determine the outputs for the inputs shown



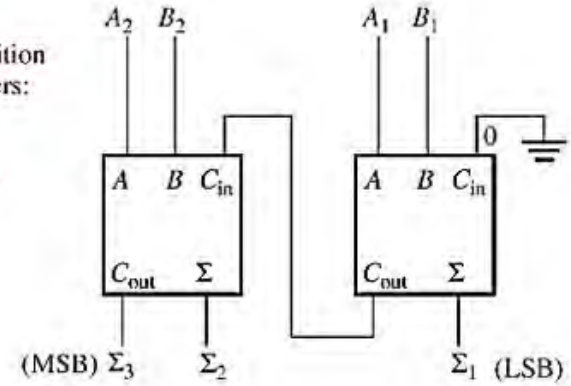
Parallel Adder

Digital Logic

- Two or more full adders are connected together to form a parallel adder
  - It is use to add more than 1 bit
  - Example : 2 bit parallel adder

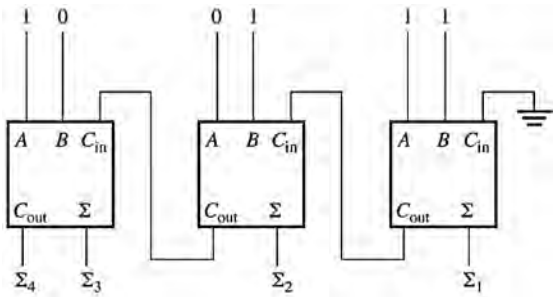
General format, addition of two 2-bit numbers:

$$\begin{array}{r} A_2A_1 \\ +\,B_2B_1 \\ \hline \Sigma_3\Sigma_2\Sigma_1 \end{array}$$



- Addition is done from right to left, therefore the leftmost bit doesn't have a C_{in} , signified by a ground symbol (which has a logic value of 0).
- To add the Least Significant Bit (LSB), either:
  - ✓ use a half adder which doesn't have a C_{in} , or
  - ✓ use a full adder but must make the C_{in} equals to 0.
- All other bits position can only be added by using a full adder because there is a possibility of C_{in} from the previous bit addition.
- From the example in the slide, the addition is for two bit number but the answer is 3 bits number.

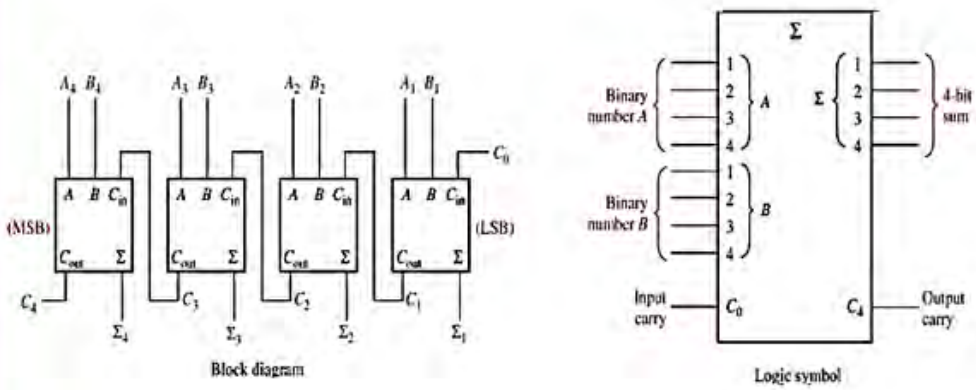
DIY : Determine the sum generated by the 3-bit parallel adder below and show the intermediate carries when the binary numbers 101 and 011 are being added



MSI chip: 4-Bit Parallel Adder

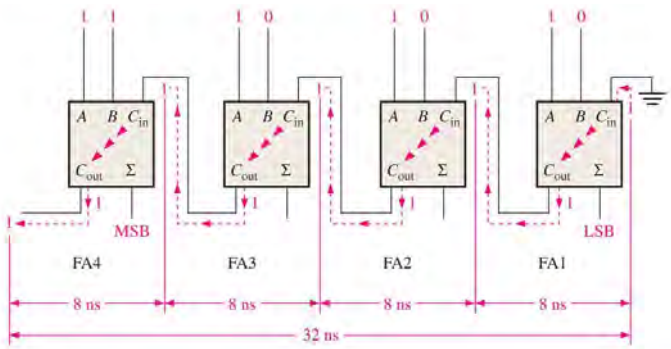
Digital Logic

- 74LS283 is a MSI chip that can do 4 bits addition
- Only the first C_{in} and the last C_{out} exist as a pin on the IC, all other carry pins are connected internally
- Internally, the IC is connected as in the block diagram



Two type of parallel adder, based on how it handle a carry:

1. **Ripple Carry Adder**
   - Carry output of each full-adder is connected to carry in of the next order stage of full adder
   - Carry accumulate from the lowest order bit to the highest order bit
   - Speed of addition limited by the time required for the carries to propagate (or ripple) through all stages
   - Circuit simpler than look-ahead carry adder.



A 4-bit parallel binary ripple carry adder showing "worst-case" carry propagation delays

2. **Look-Ahead Carry Adder**
   - Speed up the addition process by eliminating ripple carry delay
   - It anticipates the output carry of each stage , and based on the inputs, produces the output carry by either **carry generation** or **carry propagation**

Adder: Application example

- A voting system

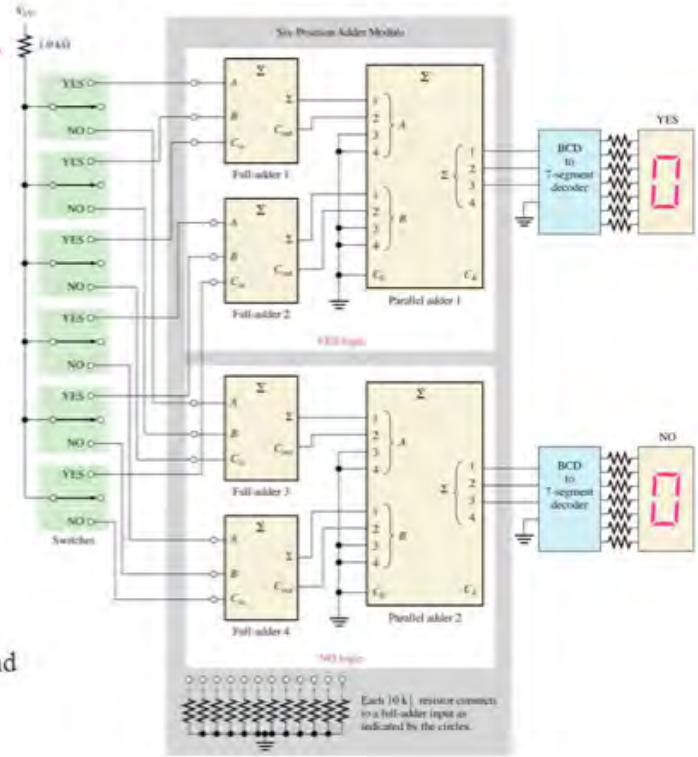
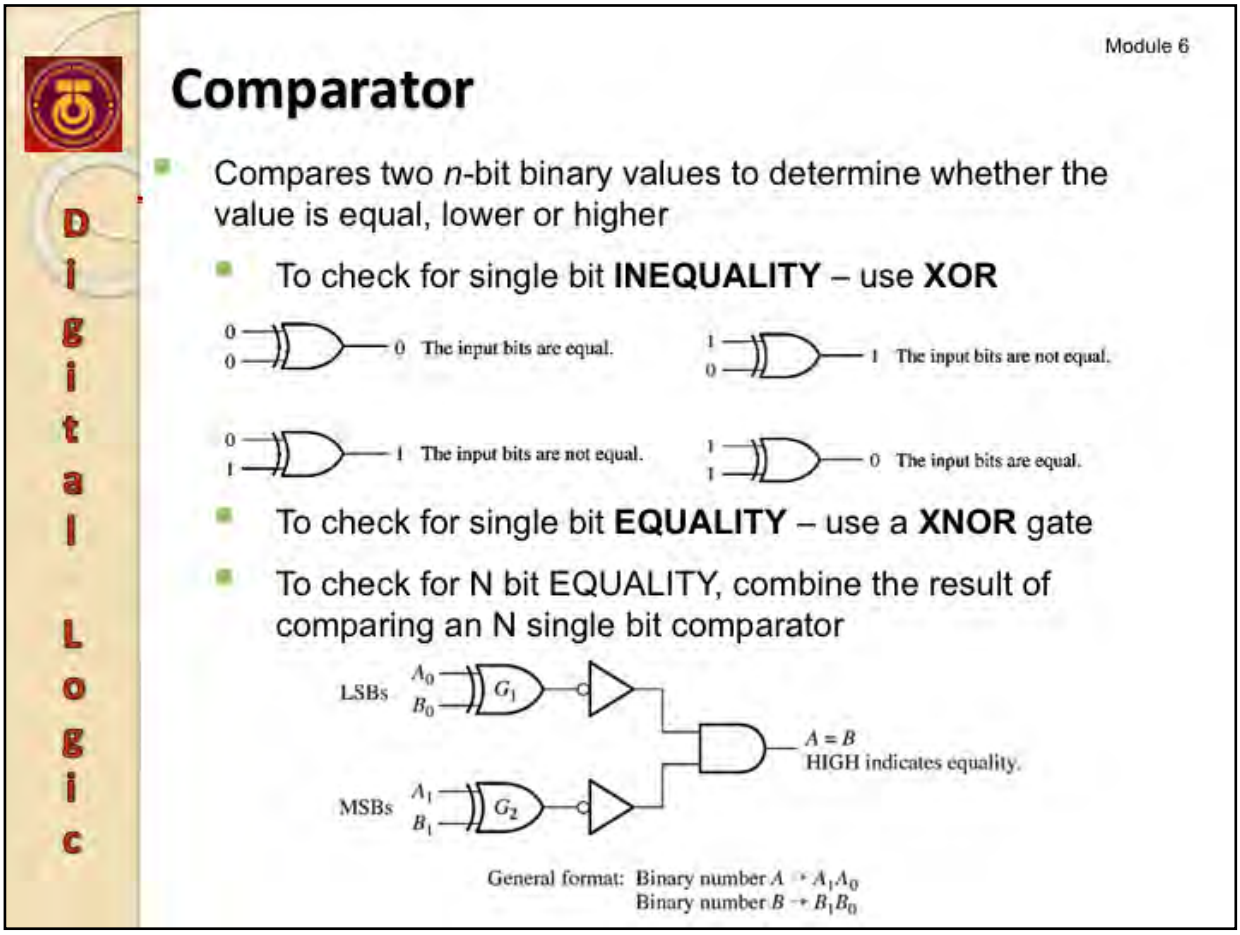


Figure A voting system using full-adders and parallel binary adders.

- The system will determine how many voter vote for YES and NO.
- Each voter press a switch which is an input to the system.
- Each input to the full adder is connected to ground through a 100KΩ resistor, this is to make the input LOW (0) if voter abstain (don't want to choose YES or NO) -- this connection is not shown on the schematic:
    - If a user flips the switch to YES, the input to YES logic will become HIGH (1).
    - If a user flips the switch to NO, the input to NO logic will be HIGH (1).
- Each full adder can add 3 votes, the possible output is 00, 01, 10, 11.
- The output of the full adder is sent to parallel adder, where the output of the two full adder will be added.
- The output was sent to 7 segment display.
- To display the digit, a 7 segment decoder is used.

- **DIY**
    a) If the sequence of voting from top to bottom is Y, N, N, Y, N, N, what will be displayed at the 7 segment display?
    b) Trace the logic value at each input and output of the IC during that voting
    c) What kind of 7 segment display is use in the circuit?
    d) Why one of the input to 7-segment decoder is connected to ground ? Which pin ?



The Design:

Design a single bit comparator that compares X and Y and determine whether $X = Y$, $X < Y$ or $X > Y$

1) The truth table:

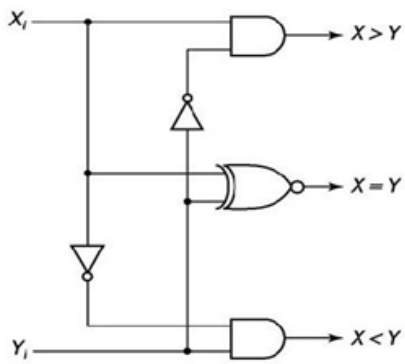
X_i	Y_i	$X > Y$	$X = Y$	$X < Y$
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

2) From truth table, the output equations:

$$X > Y \rightarrow XY$$
$$X = Y \rightarrow \overline{X}\,\overline{Y} + XY = X \odot Y$$
$$X < Y \rightarrow \overline{X}Y$$

3) The circuit:

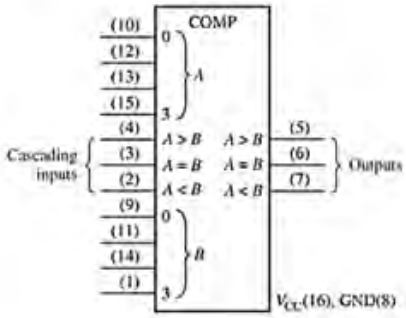
Design a separate circuit for each output and finally combine them.



MSI chip: 4-bit Comparator

Digital Logic

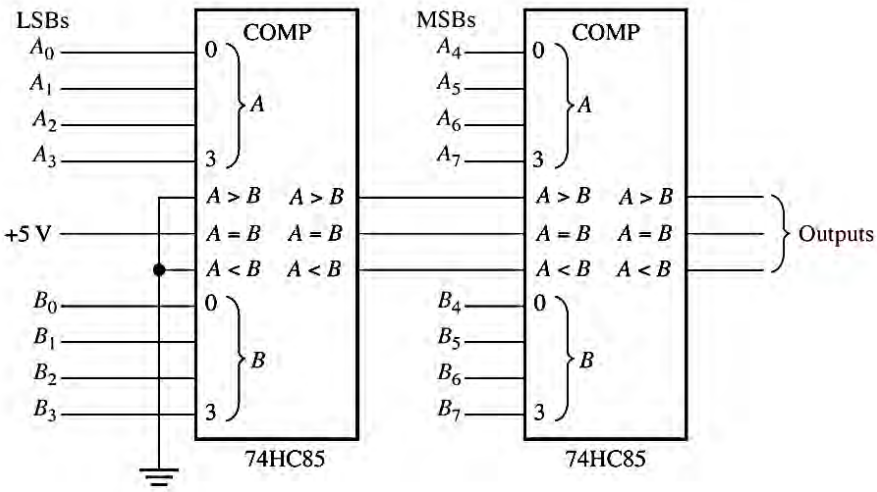
- 74LS85 is an MSI chip that can compare 4 bits value
- It can indicates whether the two number greater, equal or lower than the other
- Its operation: compare the MSB first
  - If A_3= 1 and B_3 = 0 then A > B set without comparing the lower bit
  - If A_3= 0 and B_3 = 1 then A < B set without comparing the lower bit
  - If A_3 = B_3 then A2 and B2 will be compared, until A_0 and B_0, if all bits are equal, only then A= B set
- 74LS85 also has a cascading input, to allow cascading an IC for more bits comparison



Example :

Comparing 8 bits value

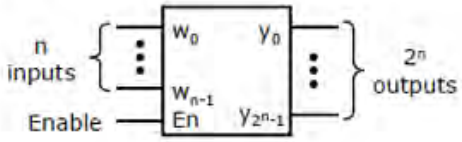
- Each IC can only compare 4 bits, therefore requires 2 ICs in cascaded connection.
- For cascaded connection:
  - ✓ Output of A>B, A=B, A< B must be connected to the corresponding input of the next stage
  - ✓ For lowest order comparator, A=B must be HIGH while A>B and A<B must be connected to LOW



Decoder

Digital Logic

- Function
  - detect the presence of a specified combination of bits (code) on its inputs
  - indicate the presence of that code by a specified output level.
- Decoder has
  - *n* input lines to handle *n* bits
  - from 1 to 2^n output lines to indicate the presence of one or more n-bit combinations.
  - An enable input (En) is used to disable the outputs
    - If En=0, none of the decoder outputs is asserted
    - If En=1, one of the outputs is asserted according to the valuation of the inputs



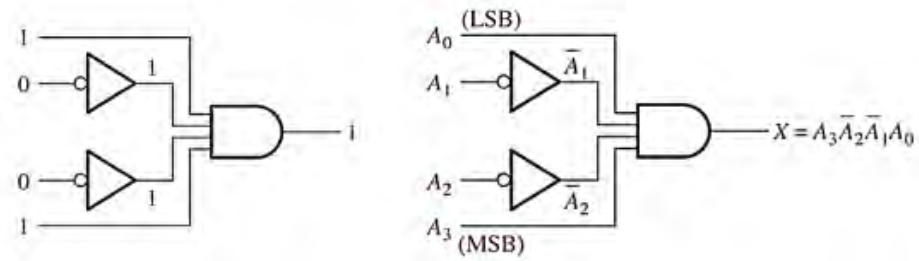
There are several types of decoder. This note only discuss the following types:

- Binary Decoder
- MSI Chip: 4-Bits Decoder
- BCD to Decimal Decoder
- MSI chip: Binary to 7-Segments Decoder
- Combinational Logic using Decoder

Binary Decoder ..

Digital Logic

- A decoder that recognize a binary pattern at its input and activate the output whenever that pattern occurs
  - Use AND if we want the output to be HIGH
  - Use NAND if we want the output to be LOW
- Example : A decoder that produces a logic HIGH if 1001_2 occurs at the input



The Design:

1. Determine what kind of output desired:
   - If HIGH use AND
   - If LOW use NAND
2. Determine the binary pattern at the input:
   - Draw a bubble at the input if input 0
   - Draw a straight horizontal line at the input if input 1

DIY:

a) Design a binary decoder that detect an input of 1011_2 and produces a logic 0 if that logic value exist at the input.

b) An active low device is activated by an output of a decoder. The device will be activated if the decoder detect 110011010_2, design the decoder circuit.

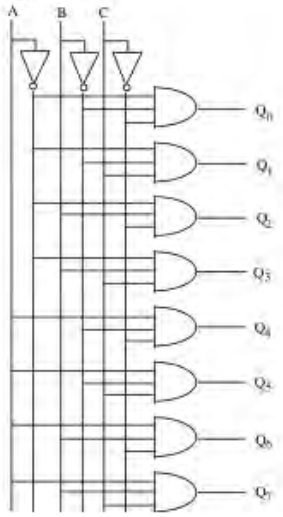
..Binary Decoder

Digital Logic

- Sometimes it is good to have a general purpose decoder that can decode all the possibility of binary pattern at the input
  - Such a decoder has multiple output but its output will only be active if a unique binary pattern exist at the input
  - Eg: 3 bits Decoder

A	B	C	Q_0	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_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

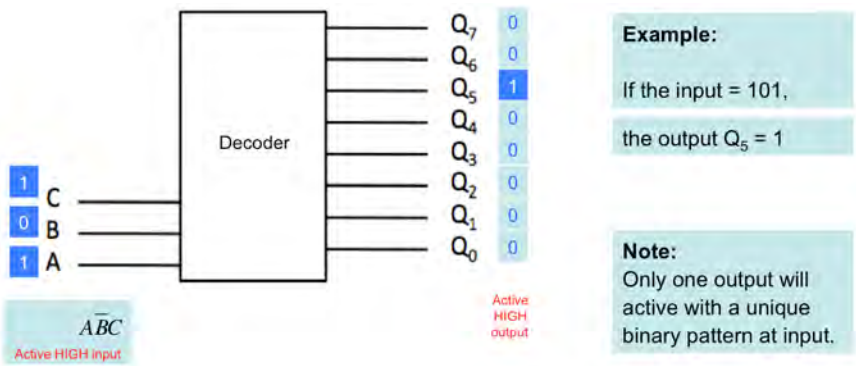
Truth Table for 3 bits decoder



- The design steps is the same as in the previous slide
- AND is used because we want an active high output
- The difference is that all possible input is listed in the truth table, and all the output is also listed
- The each input there will be only one HIGH output
- For an active LOW output, there will be only one LOW for each input
- The equation for each output is determine

$$Q_0 = \overline{A}\,\overline{B}\,\overline{C},\ Q_1 = \overline{A}\,\overline{B}\,C, \ldots .\ Q_6 = A\,B\,\overline{C},\ Q_7 = A\,B\,C$$

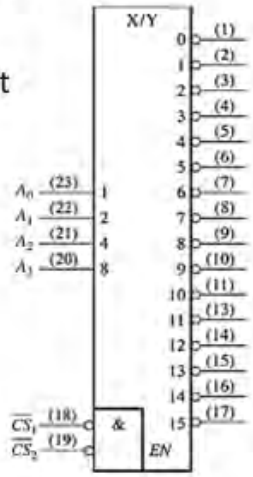
- From the equation the circuit was build. For the above circuit the input is active HIGH and the output is active HIGH
- **Example**: Binary decoder.



MSI Chip: 4-Bits Decoder ..

Digital Logic

- Decoder can be called **n to 2^n** or **1 of 2^n** for the same decoder
- The common MSI decoder chip
  - 2 to 4 (1 of 4), 3 to 8 (1 of 8) and 4 to 16 (1 of 16)
- 74HC154 is a 1 of 16 decoder
  - It has 4 active high input and 16 active low output
  - It also has 2 active low enable (EN) pin
    - This pin is used to select the IC if we use more than one chip
    - To use the IC, this two pin must be connected to logic LOW



The function table -- to descibe the function of the device

		INPUTS				OUTPUTS															
$\overline{E}_0$	$\overline{E}_1$	A_0	A_1	A_2	A_3	$\overline{Y}_0$	$\overline{Y}_1$	$\overline{Y}_2$	$\overline{Y}_3$	$\overline{Y}_4$	$\overline{Y}_5$	$\overline{Y}_6$	$\overline{Y}_7$	$\overline{Y}_8$	$\overline{Y}_9$	$\overline{Y}_{10}$	$\overline{Y}_{11}$	$\overline{Y}_{12}$	$\overline{Y}_{13}$	$\overline{Y}_{14}$	$\overline{Y}_{15}$
H	H	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	L	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	H	X	X	X	X	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	L	L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	H	L	L	L	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	H	L	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H
L	L	H	H	L	L	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H
L	L	L	L	H	L	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H
L	L	H	L	H	L	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H
L	L	L	H	H	L	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H
L	L	H	H	H	L	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H
L	L	L	L	L	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H
L	L	H	L	L	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H
L	L	L	H	L	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H
L	L	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H
L	L	L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H
L	L	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H
L	L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L

Note

1. H = HIGH voltage level
   L = LOW voltage level
   X = don't care

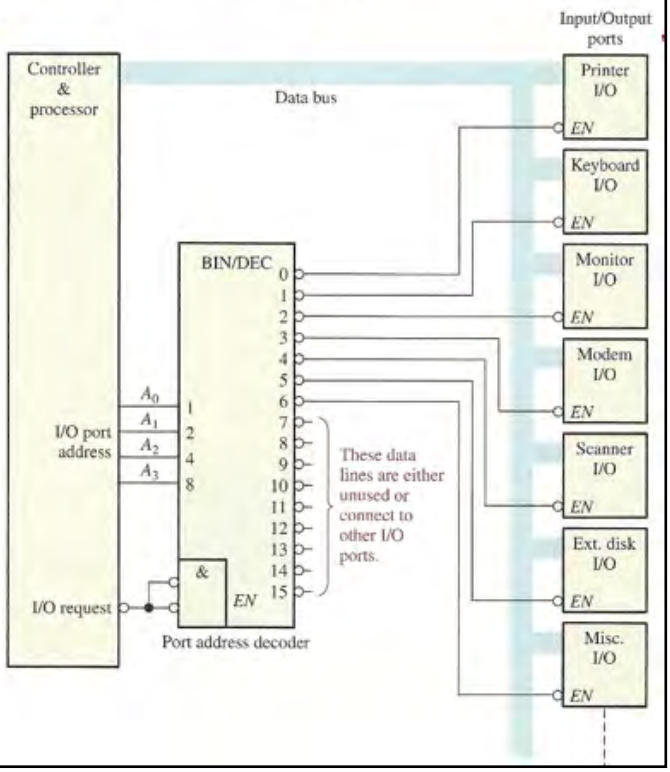
MSI Chip: 4-Bits Decoder ..

Digital Logic

Example:

- **Address Decoder**

  Decoder is use a lot to decode an address of a computer system.



- In a computer system every devices is given an address (like an ID) to identify the devices
- Address must be unique for a device, some devices uses a range of addresses
- Decoder is used to decode the address
- Whenever a computer try to access the I/O device it will issue an I/O request by making that signal LOW
  - this will enable an active low enable pin of a decoder
- Next the address of an I/O device will be sent by the computer as $A_3A_2A_1A_0$
- This address will be decoded and activate one of the output (becomes LOW)
  - If $A_3A_2A_1A_0 = 0110$, then output 6 will be LOW, all other output are HIGH
  - The LOW output will enable an active LOW device at address 6

DIY

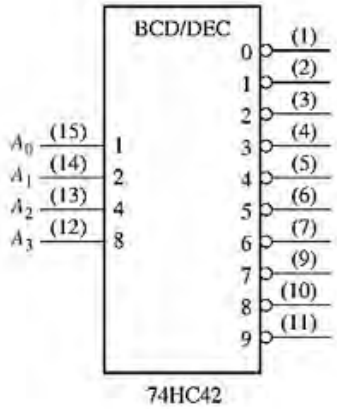
a) Given $A_3A_2A_1A_0 = 0011$, which device will be enabled?

b) Why only one device enabled at any one time?

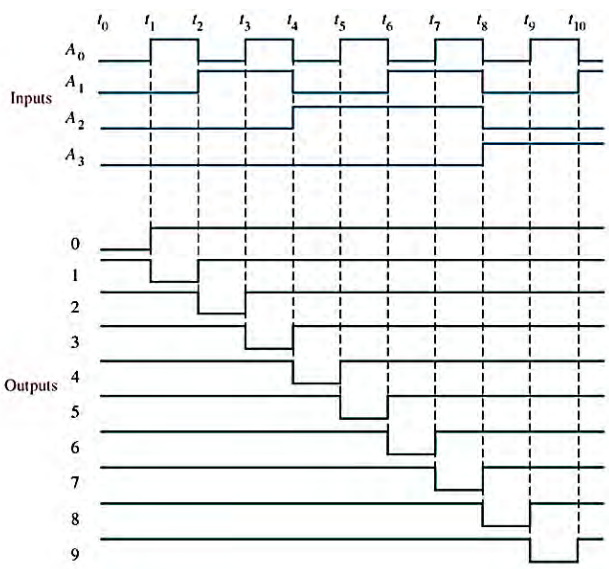
BCD to Decimal Decoder

Digital Logic

- Convert from BCD to decimal
  - Frequently referred as 1 of 10 decoder
  - The same as binary decoder except only 10 decoding gates required
    - An active LOW output MSI Chip is 74HC42



The timing diagram of 74HC42, It shows that only one output active (LOW) for a BCD input



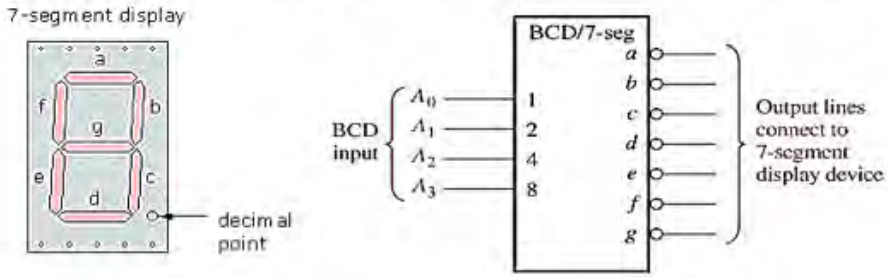
The decoding function , where each output has a unique function

Decimal digit	A_3	A_2	A_1	A_0	Decoding Function
0	0	0	0	0	$\bar{A}_3\bar{A}_2\bar{A}_1\bar{A}_0$
1	0	0	0	1	$\bar{A}_3\bar{A}_2\bar{A}_1A_0$
2	0	0	1	0	$\bar{A}_3\bar{A}_2A_1\bar{A}_0$
3	0	0	1	1	$\bar{A}_3\bar{A}_2A_1A_0$
4	0	1	0	0	$A_2\bar{A}_1\bar{A}_0$
5	0	1	0	1	$A_2\bar{A}_1A_0$
6	0	1	1	0	$A_2\bar{A}_1A_1\bar{A}_0$
7	0	1	1	1	$\bar{A}_3A_2A_1A_0$
8	1	0	0	0	$A_2\bar{A}_1\bar{A}_0$
9	1	0	0	1	$A_2\bar{A}_1\bar{A}_0$

MSI chip: Binary to 7-Segments Decoder

Digital Logic

- Receive a BCD code at the input and lit the proper LED of the 7-segment display based on the BCD value
- The display shows the decimal numbers 0-9
- The individual segments making up a 7-segment display are identified by letters, from *a* to *g*
  - 2 type of 7-segment display
    - Common anode – 0 at it input will lit the segment therefore require a decoder with an active LOW output
    - Common cathode- 1 at it input will lit the segment therefore require a decoder with an active HIGH output



- Truth table for an active HIGH output 7-segment decoder
  - Can drive a common cathode 7-segment display

BCD Inputs				Segment Outputs							Display
D	C	B	A	a	b	c	d	e	f	g	
0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	0	1	1	0	0	0	0	1
0	0	1	0	1	1	0	1	1	0	1	2
0	0	1	1	1	1	1	1	0	0	1	3
0	1	0	0	0	1	1	0	0	1	1	4
0	1	0	1	1	0	1	1	0	1	1	5
0	1	1	0	1	0	1	1	1	1	1	6
0	1	1	1	1	1	1	0	0	0	0	7
1	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	1	1	1	1	0	1	1	9

DIY

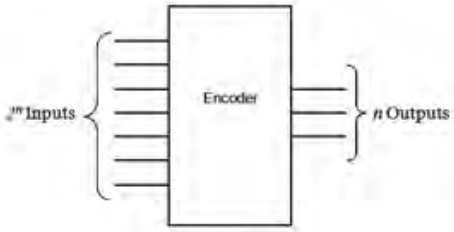
a) What will be displayed if the input to the decoder greater than 9?

b) Produce a truth table for an active LOW 7-segment decoder

Encoder

Digital Logic

- Encoder does the opposite of the decoder
  - E.g: convert Decimal to BCD
- A ***binary encoder encodes information (data) from*** 2*n* *inputs into an n-bit code (output)*
  - Exactly one of the inputs should have a value of one
  - The outputs represent the binary number that identifies which input is equal to 1
- Encoders reduce the number of bits needed to represent given information



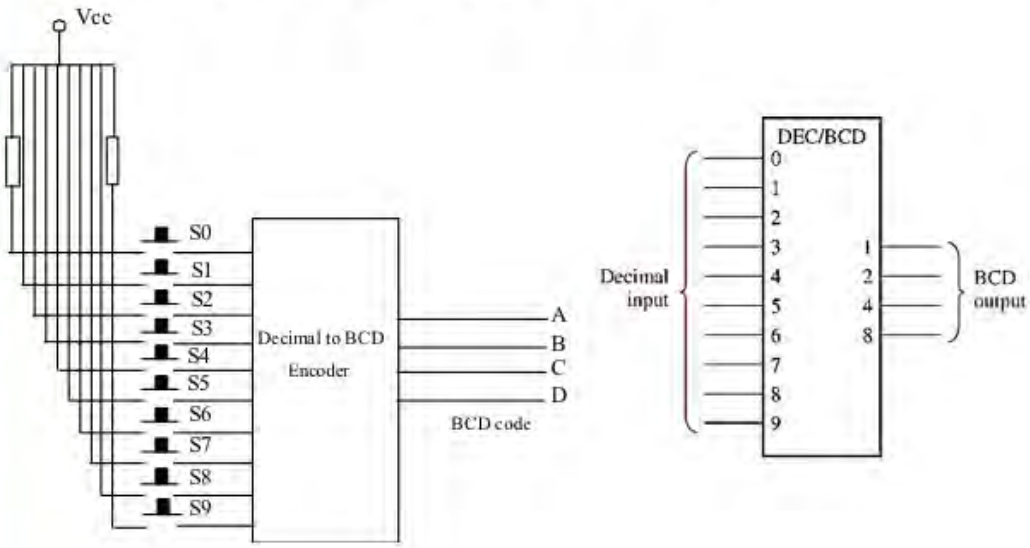
In this note we will discuss the following encoders

- Decimal to BCD Encoder
- Priority Encoder

Decimal-to-BCD Encoder ..

Digital Logic

- 10 input (Decimal) and 4 output (BCD)
  - Also called 10 line to 4 line encoder
  - Convert a decimal value to a BCD code



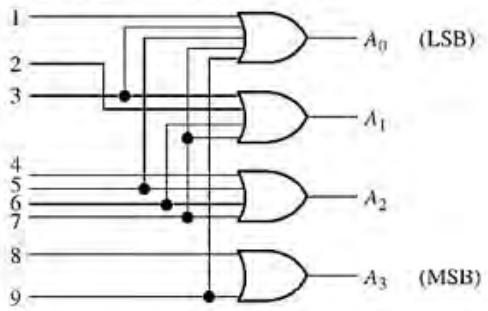
There are 10 input at the output but only one active at a time and each decimal value will be converted to a BCD as shown in the truth table

S0	S1	S2	S3	S4	S5	S6	S7	S8	S9	A_3	A_2	A_1	A_0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	1
0	0	0	0	0	0	1	0	0	0	0	1	1	0
0	0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	1

.. Decimal-to-BCD Encoder

Digital Logic

- From truth table
  - A_0 HIGH if switch 1,3,5,7,9 HIGH
  - A_1 HIGH if switch 2,3,6,7 HIGH
  - A_2 HIGH if switch 4,5,6,7 HIGH
  - A_3 HIGH if switch 8,9 HIGH
    - Therefore OR all the input to make the corresponding output HIGH
- A 0-digit input is not needed because the BCD outputs are all LOW when there are no HIGH inputs



DIY

Suppose, HIGH are applied to input 2 and 9 of the circuit.

a) What are the states of the output lines?

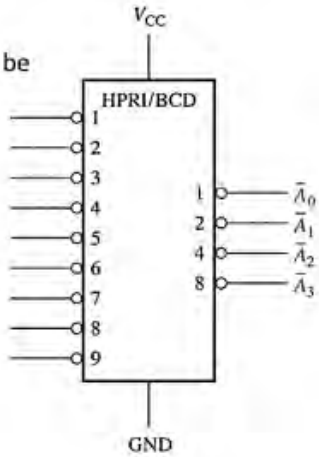
b) Does this represent a valid BCD code?

c) What is the restriction on the encoder logic?

MSI chip: Priority Encoder

Digital Logic

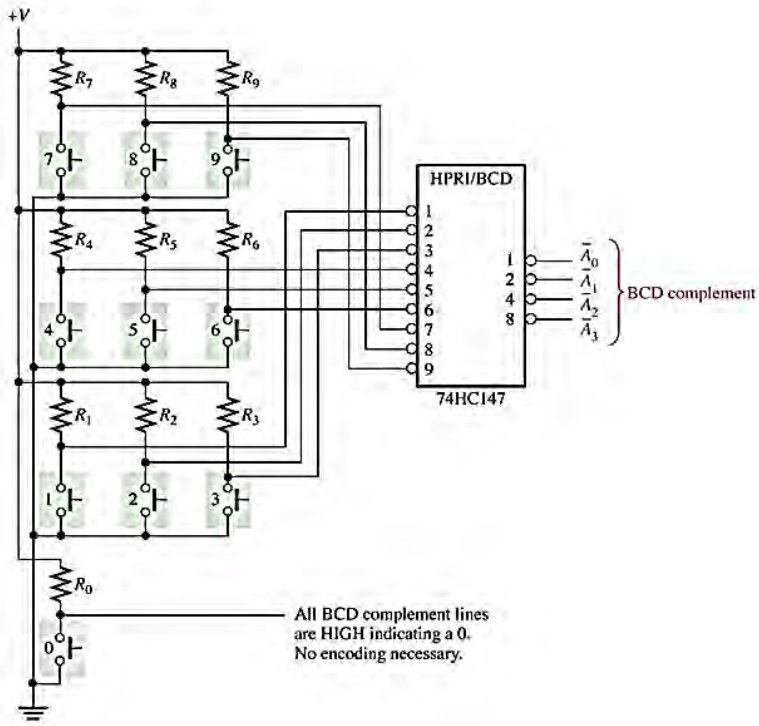
- IC 74HC147 perform the same encoding function but with additional feature
  - Produce BCD output corresponding to the higher decimal value at the input and ignore the lower value
    - HPRI – highest value input has priority
  - Eg:
    - if pin 9 and 5 LOW at the same time, only pin 9 will be recognize because it has a higher priority
    - The output will be 0110 for 9 (because of active low at the output)



An Application:

Keyboard Encoder

- 10 keys at the input , from 0 .. 9
- If no key is pressed , input to the encoder HIGH and its output all HIGH (1111) which is 0 as active LOW
- If a key is pressed, e.g. key 4 then pin 4 input will be LOW.
  - This will cause the output to be 1011 (4 as active LOW)



Multiplexers (Data Selector)

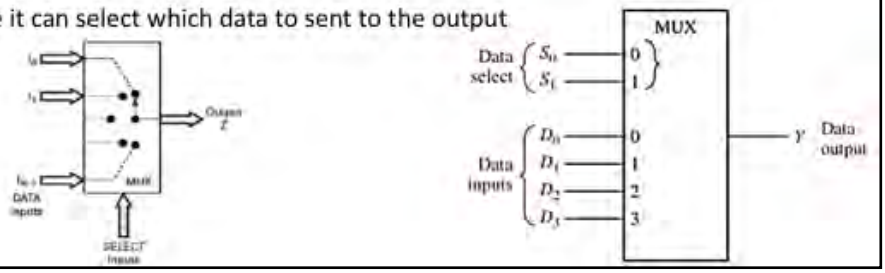
Digital Logic

- A multiplexer (MUX) is a device that allows digital information from several sources to be routed onto a single line for transmission over that line to a common destination.
  - The basic multiplexer has several data-input lines and a single output line, ***Many to One***
  - It also has data-select inputs, which permit digital data on any one of the inputs to be switched to the output line.
    - Select line control which input routed to output

$$\text{number of select bits} = \frac{\log(\text{number of input})}{\log 2}$$

- MUX are also known as data selector

Because it can select which data to sent to the output



The Design:

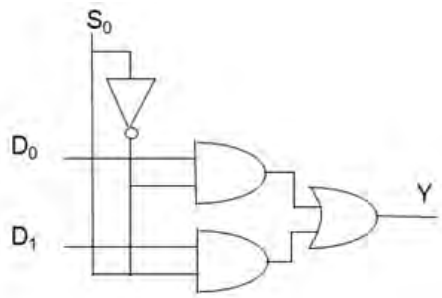
When S_0 =0, Y = D_0 and when S_0 = 1 , Y = D_1

- Truth table is produced from the specification.
- But truth table can be very long, a more compact form is on the right side table

S_0	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

S_0	Data Output
0	$Y=D_0$
1	$Y=D_1$

The circuit equation: $Y = \overline{S_0}D_0 + S_1D_1$

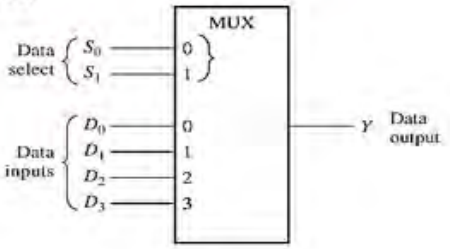


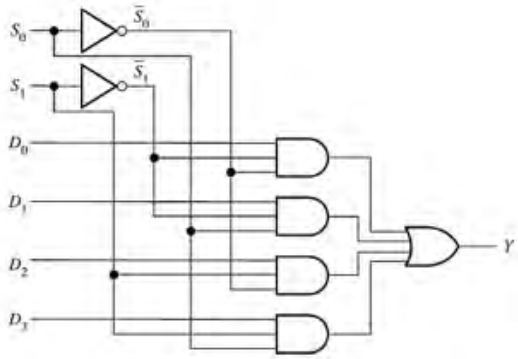
Multiplexer 1-of-4

Digital Logic

- Able to route 4 inputs to 1 output
  - Requires 2 bits data selector

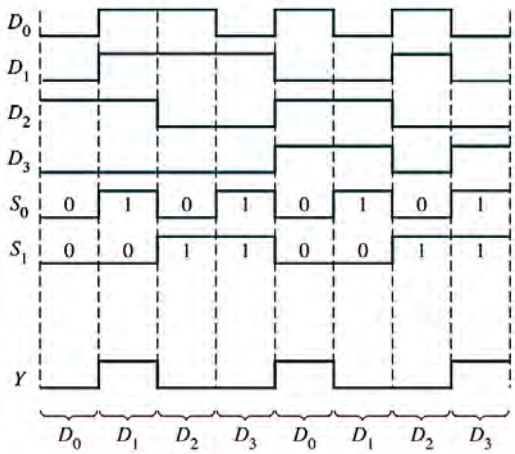
Data-select input		Data output Y
S_1	S_0	
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3





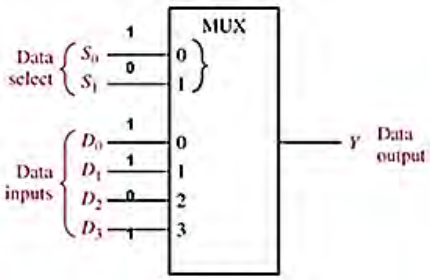
Timing Diagram to show the working of 1 of 4 Multiplexer

- The output will depend on the value of selector
- S_1 is the MSB and S_0 the LSB



DIY:

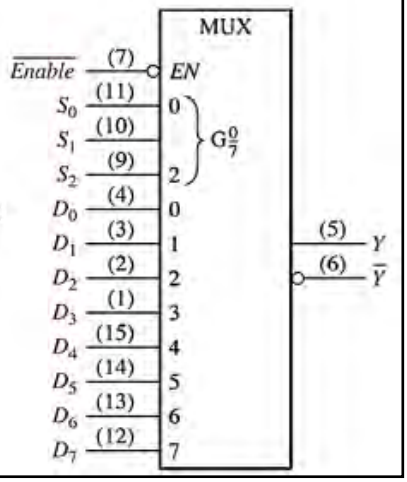
1. Determine which input will be routed to the output?
2. Determine the logic value at the output.



MSI Chip: Multiplexer

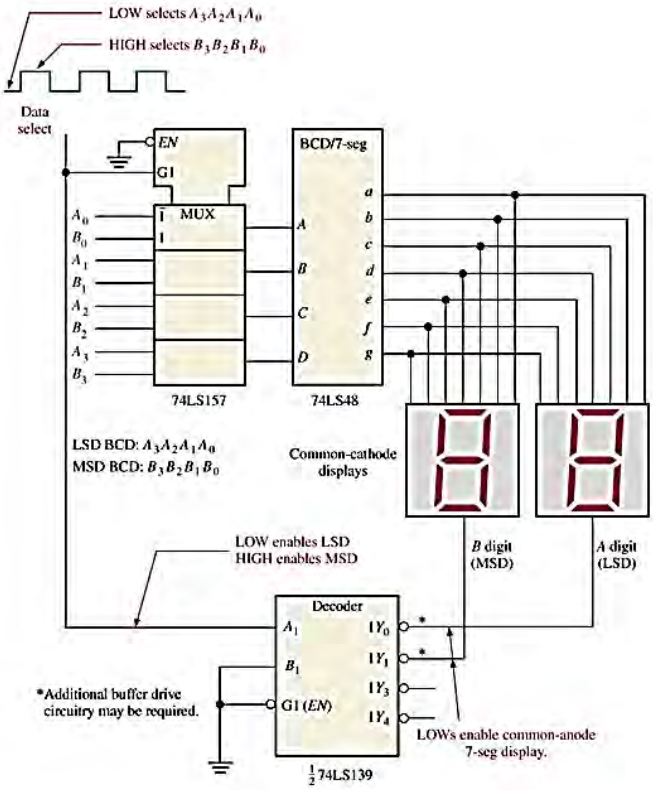
Digital Logic

- Besides Data input and selector pin , IC usually has an enable pin
  - For IC 74LS151
    - A 1 of 8 MUX
    - The enable pin is active LOW, therefore logic 0 must be applied to this pin to make it work
      - If logic HIGH applied to this pin the output will be High Impedance (Hi-Z), ie the output pin disconnected from the output
        -- the output is neither 0 nor 1
    - It has an inverted and a non inverted output



Application:

- One application is to alternately switch on a display by using a MUX
- To lit the LED 2 sets 4 bit data which is A3..A0 and B3 .. B0 need to be sent to 7 segment decoder
- Which set send to the decoder will be determine by data select pin , if LOW set A will be send and if HIGH set B will be send
  - ✓ By alternately turn on the display it will be seen as if those two display controlled simultaneously
- Seven segment is a common cathode therefore need a logic 0 to display a number
  - ✓ That logic 0 is controlled by a the output of the decoder, if $1Y_0$ =0 A digit will be displayed, else if $1Y_1$ =0 then B-digit will be displayed
  - ✓ Which output will be 0 determine by the input A1 which is connected to data select



Example

Digital Logic

Implement the function F (A,B,C,D) = Σm(1,3,4,11,12,13,14,15) using a multiplexer

4 inputs variables therefore use MUX with 3 selector pin (8x1 MUX)

Connect A, B and C to selector pin where A is the MSB.

A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	

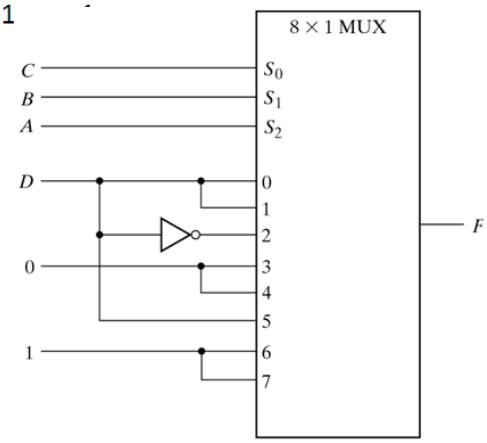
- Produce a truth table and determine the output logic for F
- Draw a line to make a group based on the value of ABC =0,1,2 .. 7
- For each group relate the value of F and D
  - If equal then F=D
  - if complement then F=D'
  - if F always 0 then F =0
  - if F always 1 then F=1
- Connect the value of F from each group to pin of the MUX that has the corresponding value of ABC
  - E.g. ABC=101, F=D then connect D to pin 5 (which is 101)

- By using the other method produce the following table. Take note the value of each table read as ABCD
  - The value is different from the previous example because here we connect the LSB to the input pin of MUX while in the previous example MSB is connected to the input of the MUX

	A'B'C'	A'B'C	A'BC'	A'BC	AB'C'	AB'C	ABC'	ABC
	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
D'	0	2	4	6	8	10	12	14
D	1	3	5	7	9	11	13	15
	D	D	D'	0	0	D	1	

- Finally connect the circuit

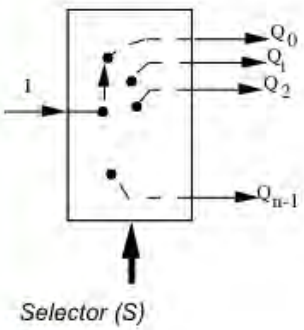
  → Doesn't matter which method is use the final answer will be the same



Demultiplexer (DEMUX)

Digital Logic

- A DEMUX is basically reverse the multiplexing function.
- It takes digital information from one line and distributes it to a given number of output lines, **one to many.**
- For this reason, DEMUX is also known as a data distributor.
- Decoder can also be used as DEMUX.



S_1 S_0	Q_3 Q_2 Q_1 Q_0
0 0	0 0 0 1
0 1	0 0 1 0
1 0	0 1 0 0
1 1	1 0 0 0

The Design

- Truth Table

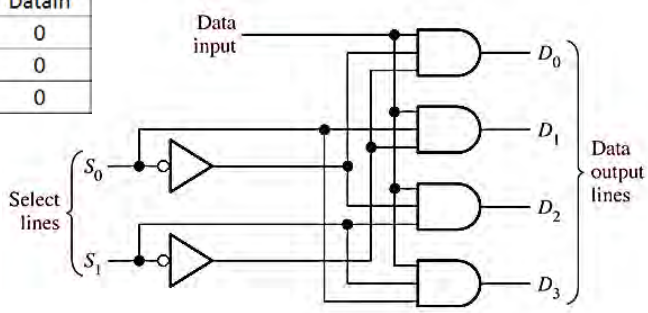
S_1	S_0	DataIn	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0
0	0	1	0	0	0	1
0	1	0	0	0	0	0
0	1	1	0	0	1	0
1	0	0	0	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	0	0
1	1	1	1	0	0	0

$$D_3 = S_1\, S_0\, \text{DataIn}$$
$$D_2 = S_1\, \overline{S_0}\, \text{DataIn}$$
$$D_1 = \overline{S_1}\, S_0\, \text{DataIn}$$
$$D_0 = \overline{S_1}\, \overline{S_0}\, \text{DataIn}$$

- A more compact Table:

S_1	S_0	D_3	D_2	D_1	D_0
0	0	0	0	0	DataIn
0	1	0	0	DataIn	0
1	0	0	DataIn	0	0
1	1	DataIn	0	0	0

- The circuit:

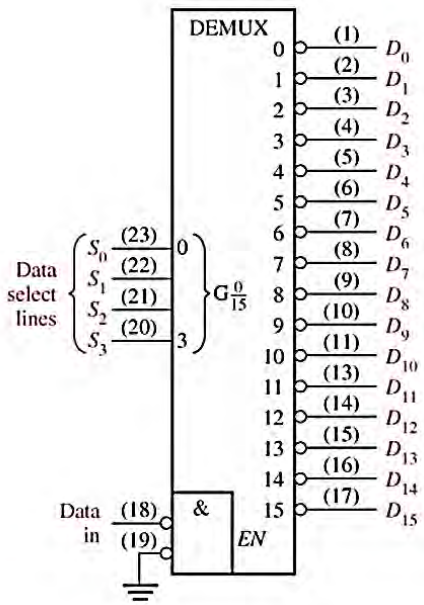


MSI Chip: Decoder as DeMUX

Digital Logic

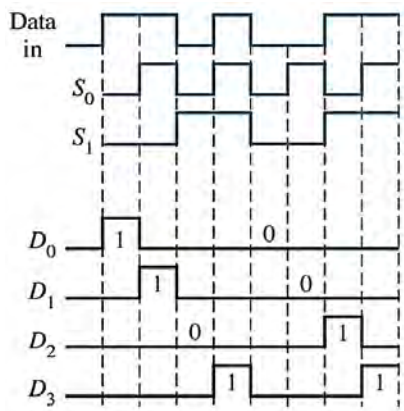
- A Decoder can be use as a DeMUX
  - Decoder 74HC154 can become a DeMUX when
    - Input of a decoder becomes a data select line
    - DataIn is connected to one of the enable pin of the IC
      - The other enable pin permanently connected to ground because it is an active LOW
      - If DataIn = 1, chip disable so that all output = 1 (because active low)
      - If DataIn = 0, chip will be enable,the corresponding output of the data selector will be 0

The circuit connection of a decoder from demux



S_1 S_0	Q_3 Q_2 Q_1 Q_0
0 0	0 0 0 1
0 1	0 0 1 0
1 0	0 1 0 0
1 1	1 0 0 0

The timing diagram to show the working of a 1-to-4 demux or as a 2-to-4 decoder



Code Converter

Digital Logic

- The purpose of ***code converter circuits is to*** convert from one type of code to another type of code
  - For example:
    - A 3-to-8 decoder converts from a binary number to a one-hot encoding at the output
    - A 8-to-3 encoder performs the opposite
- Many different types of code converter circuits can be constructed
  - One common example a a BCD-to7-segment decoder as explained in a decoder section

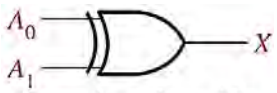
Module 6

Parity Generators / Checkers

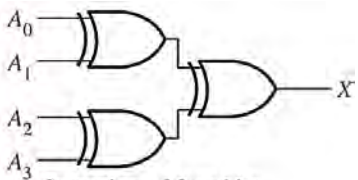
Digital Logic

- Errors can occur as digital codes are being transferred from one point to another within a digital system.
  - The errors take the form of undesired changes in the bits that make up the coded information; that is, a 1 can change to a 0, or a 0 to a 1.
  - When an error occurs undetected, it can cause serious problem
- Basic Parity Logic
  - The sum (disregarding carries) of an even number of 1s is always 0, and the sum of an odd number of 1s is always 1.
    - Therefore, to determine if a given code has even parity or odd parity, all bits in the code are summed.
  - A parity bit indicates if the number of 1s in a code is even or odd for the purpose of error detection.

- To determine whether no of 1's or 0's odd or even
  - Use Modulo 2 summation
  - Device that can do Modulo 2 is XOR
  - If number of input bits
    - EVEN (A_1A_0 = 00 or A_1A_0 = 11) then X =0
    - ODD (A_1A_0 = 01 or A_1A_0 = 10) then X = 1



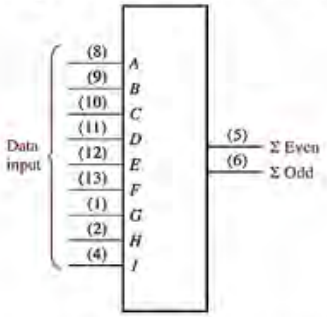
Summing of two bits



Summing of four bits

MSI Chip: Parity Generators / Checkers

- 74LS280 is a 9 bits parity generator/checker
  - Can check odd or even parity on a 9 bits code (8 data bit + 1parity bit)
  - Can generate parity bit for 9 bits data



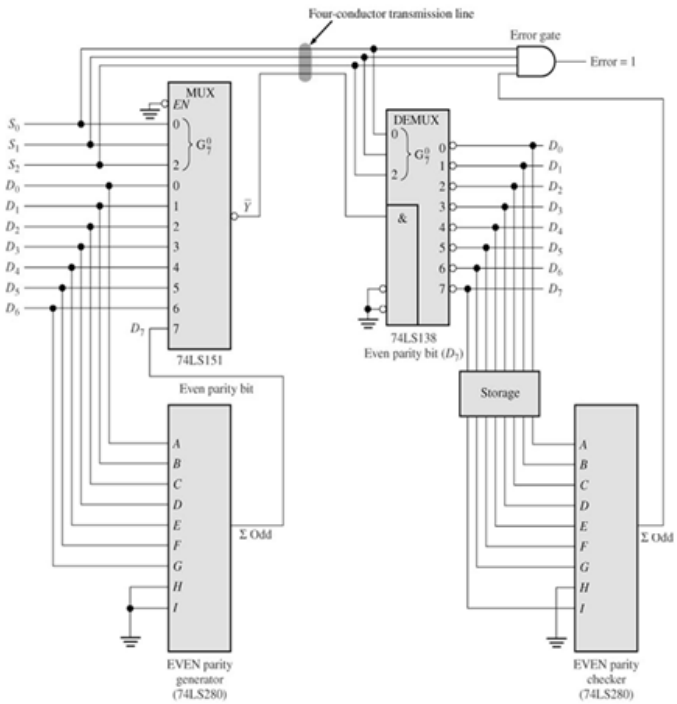
Number of Inputs A–I That Are HIGH	Outputs Σ Even	Outputs Σ Odd
0, 2, 4, 6, 8	H	L
1, 3, 5, 7, 9	L	H

- **Parity Checker** – when this device is used as an parity checker, the number of input bits always be even, and when a parity error occurs, the Σ Even output goes LOW and the ∑ Odd output goes HIGH.
- **Parity Generator** – If this device is used as an EVEN parity generator, the parity bit is taken at the ∑ Odd output because this output is a 0 if there is an even number of input bits and it is a 1 if there is an odd number.

Application

Simple data transmission system with parity.

- 7 bits data with parity bit at MSB transmitted to destination over a single line
- Those data multiplexed to the destination one bit at a time
- The lower 7 bits send to EVEN parity generator, that will insert 1 if the data number of 1's is odd, otherwise it will insert a 0
- At the receiving side, data and parity received is stored one bit at a time
- EVEN parity checker will check the parity of data received, A..G is data while I is parity input
- If no of 1's even, parity checker will output a 0
- The output send to error gate an ANDed with selector signal, which is 111 at the end of transmission
- The output of an Error gate will indicate whether an error occurs



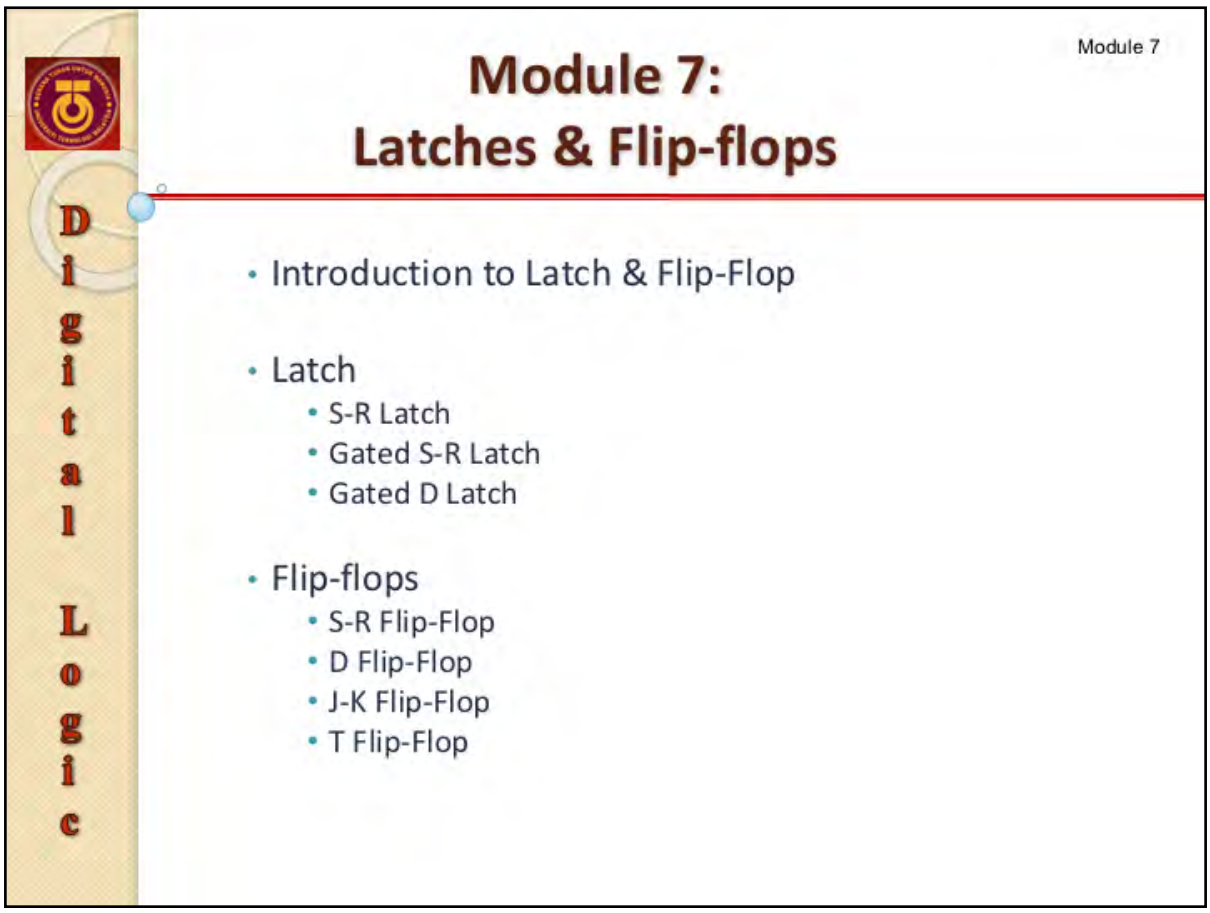
Digital Logic

End of Module 6

Digital Logic

MODULE 7

LATCHES AND FLIP-FLOPS

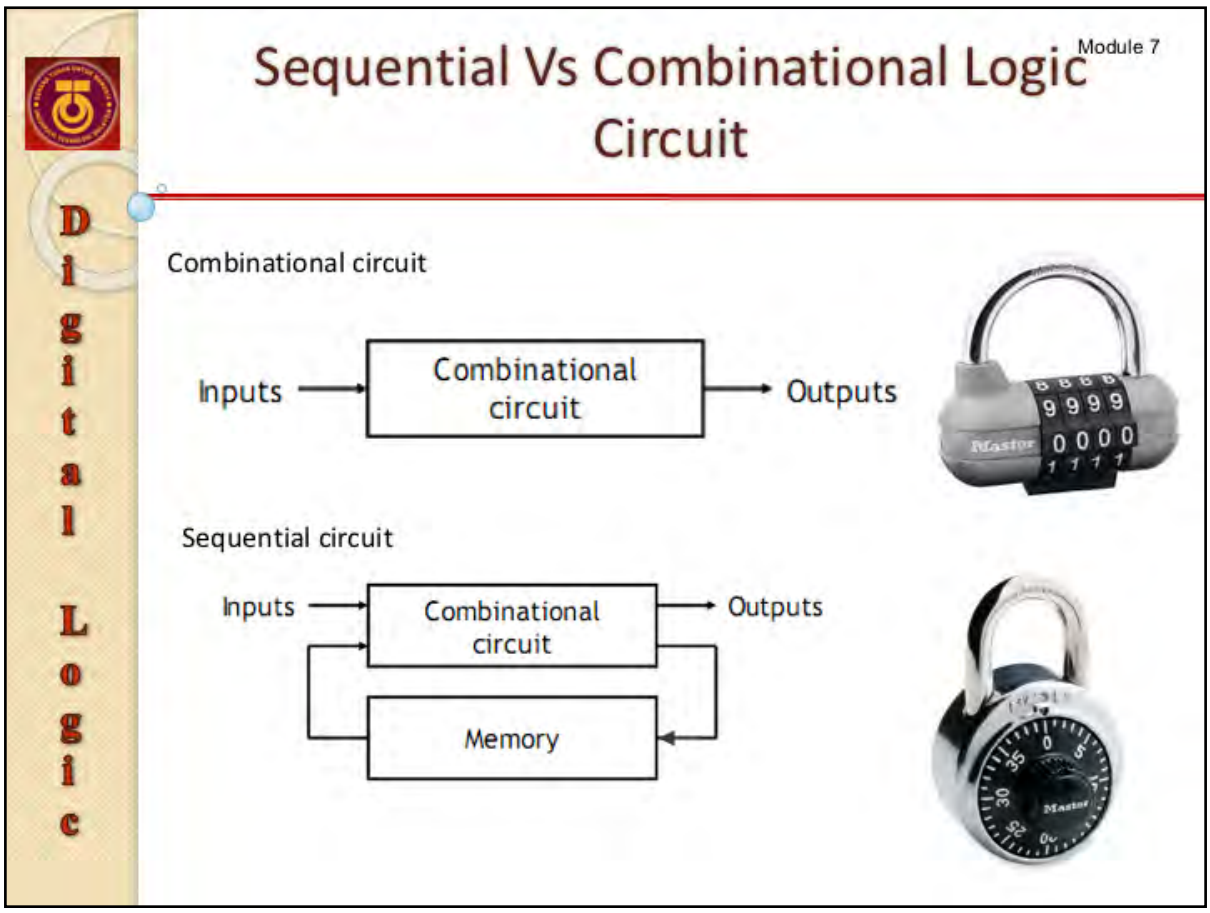


Objectives of this module are:

1 To introduce the various types of Latches and Flip-Flop.

2 To explain the characteristic of Latches and Flip-Flop.

3 To illustrate the timing diagram associated with each Latches and Flip-Flop operations.



Combinational Logic Circuit:

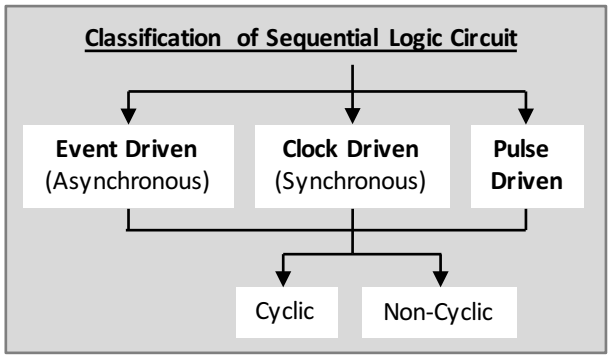
- the output depend on the input
- it has no memory element, therefore it cannot memorize the previous output

Sequential Logic Circuit:

- it has memory element
- the output depend on the input, as well as the previous output from the feedback elements.

Sequential Logic circuits can be divided into 3 main categories:

1. **Clock Driven** - Synchronous Circuits that are synchronized to a specific clock signal.
2. **Event Driven** - Asynchronous Circuits that react or change state when an external event occurs.
3. **Pulse Driven** - Which is a Combination of Synchronous and Asynchronous.



Sequential logic circuits that return back to their original state once reset, i.e. circuits with loops or feedback paths are said to be "Cyclic" in nature.

Module 7

Introduction: Latch

Digital Logic

- **Latch** is a type of temporary storage device
- It has two stable states (bi-stable)
- It is level sensitive, or level-triggered.
  - outputs are dependent on the voltage level applied (HIGH or LOW), not on any signal transition.

- Type of Latch :
  - i) S-R
  - ii) Gated S-R
  - iii) Gated D
  - **S means *Set*, R means *Reset***

INFO :

A ***bi-stable*** state is one with *two-stable* output states.

- A circuit incorporating latches has state; its output may depend not only on its current input, but also on its previous inputs. Such a circuit is described as sequential logic.

- Latch and flip-flop are a type of temporary storage device whereby they can store one bit of information.

- Latches and flip-flops are bi-stable (that has two stable states i.e. either LOW or HIGH state).

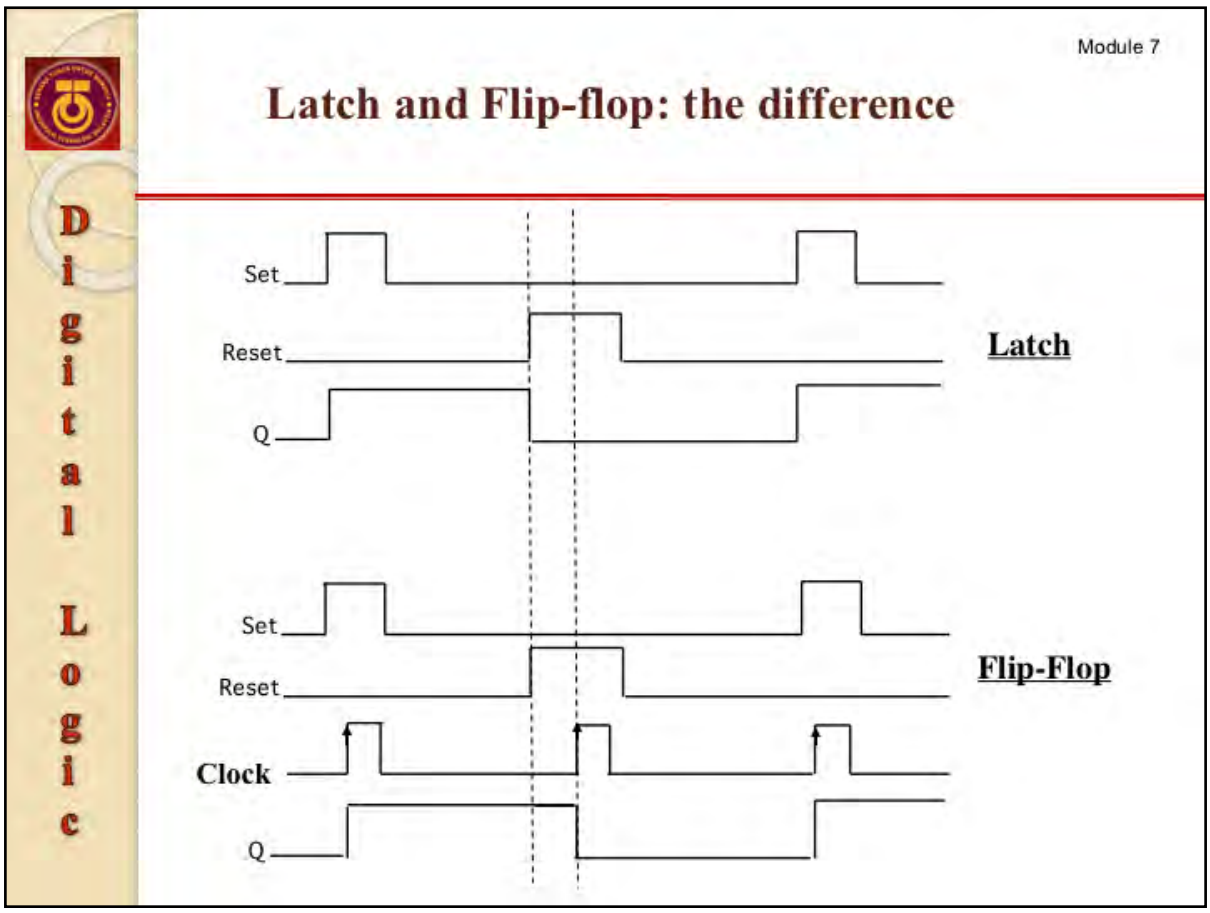
- The main difference between latches and flip-flops is the method used for changing their state.

Introduction: Flip-flop (FF)

Digital Logic

- **FF** are synchronous bi-stable storage devices capable of storing one bit, where the output state only changes at a specified point on a triggering input called the Clock (CLK).
- FF are edge-triggered, means that the output changes on the transition of a clock signal
  - This may either be a LOW-to-HIGH (rising edge) or a HIGH-to-LOW (falling edge) transition.
- Type of FF :
  - i) S-R
  - ii) D
  - iii) J-K
  - iv) T

INFO :
Rising edge = positive edge
Falling edge = negative edge



Explanation:

- The output of latch, Q is determined only by the input Set and Reset. If Set is HIGH and Reset is LOW, Q will be HIGH. If Set is LOW and Reset is HIGH, Q will be LOW.

- As for the Flip-flop, another input is added, which is the Clock. The purpose of the Clock is for synchronization. We can see that when Set is LOW, Reset is HIGH, the output Q is still in HIGH state until the next occurrence of the clock positive edge.

- The difference between a latch and a flip-flop is the way it makes the output change state
    - For latch: any changes to the input will have a direct impact to the output
    - For Flip-flop : changes at the input will have an impact only at the clock edge

*** Please note : the explanation is different for a gated latch, that we will cover later.*

Latches

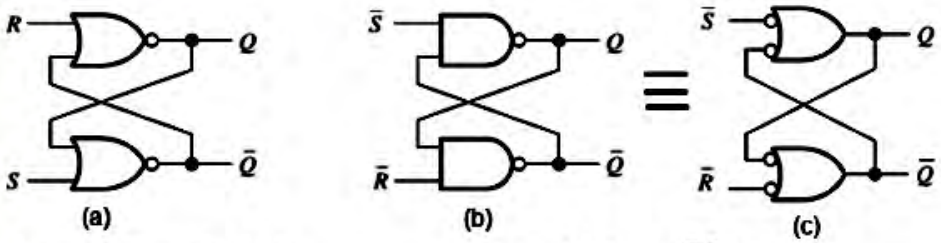
Digital Logic

S-R Latch
Gated S-R Latch
Gated D Latch

S-R Latch

Digital Logic

- The output of each gate is connected to an input of the other gate
- This produces a regenerative feedback (cross-coupled)
- Can be build using
  - NOR - active HIGH
  - NAND – active LOW



Active-HIGH input S-R latch

Active-LOW input $\bar{S}$-$\bar{R}$ latch

When using static gates as building blocks, the most fundamental latch is the simple SR latch, where S and R stands for SET and RESET. It can be constructed from a pair of cross-coupled NOR (Not OR) logic gates. The stored bit is present on the output marked Q.

Circuit (a) Active-HIGH input S-R latch :

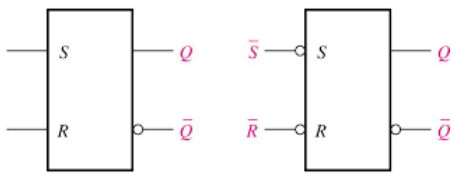
- use NOR gate
- Input R and S
- Output Q and Q'

Circuit (b) Active-LOW input S'-R' latch :

- use NAND gate
- Input S' and R'
- Output Q and Q'

Circuit (c) is equivalent to circuit (b), the only difference is symbol used in the schematic. The symbol in (c) is the dual symbol of NAND gate. This symbol can represent the active LOW input better.

Symbol for :



(a) Active-HIGH input S-R latch

(b) Active-LOW input $\bar{S}$-$\bar{R}$ latch

SR Truth-Tables (Active HIGH)

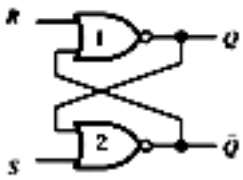
Digital Logic

Active-HIGH

INPUTS		OUTPUTS		COMMENTS
S	R	Q	$\overline{Q}$	
0	0	NC	NC	No change. Latch remains in present state
0	1	0	1	Latch RESET
1	0	1	0	Latch SET
1	1	0	0	Invalid condition

- The Q and Q' outputs are supposed to be in **opposite** states.
- When S and R are both equal to **0**, the Latch outputs "***Not Change***" in their prior states.
- **Q=1** and Q'=0 is defined as ***Set*** (by making S=1 and R=0)
- Q=0 and **Q'=1** is conversely defined as ***Reset*** (by making S=0 and R=1)
- If Q and Q' happen to be forced to the same state (both 0 or both 1), that state is referred to as ***invalid.***

Active High S-R Latch:



We use **NOR Gate, which has the characteristic of an output of 0, if any of the input is 1**.

→ From the circuit : $Q = \overline{R + \overline{Q}}$

$\overline{Q} = \overline{S + Q}$

State 1: S = 1, R = 0. Because S = 1, the output of Gate 2 which is Q', must be 0.

- Output from Q' fed back to Gate 1 which make the input both 0. Therefore the output Q = 1.
- This state is called **SET**

State 2: S = 0, R = 1. Because R = 1, the output of Gate 1 which is Q, must be 0.

- Output from Q fed back to Gate 2 which make the input both 1. Therefore the output Q = 0.
- This state is called **RESET**

State 3: S = 0, R = 0.

Assume initial state Q = 1, Q' = 0.
Q = (R+Q')' and Q' = (S+Q)',
therefore Q = 1 when R = 0 and Q' = 0,
Q' = 0 when S = 0 and Q = 1

Assume initial state Q = 0, Q' = 1.
Q = (R+Q')' and Q' = (S+Q)'
therefore Q = 0 when R = 0 and Q' = 1
Q' = 1 when S = 0 and Q = 0

- we can conclude that there is **no change** in the output
- This state is called **No Change** or **Hold**

State 4: S = 1, R = 1. Assume initial state Q = 1, Q' = 0.
Q = (R+Q')' and Q' = (S+Q)', when R = 1 and Q' = 0 then Q will be 0 and S = 1 and Q = 1 then Q' = 0

- This state is not allowed because Q' must be the complement of Q
- This state is called the **invalid state**

SR Truth-Tables (Active LOW)

Digital Logic

Active-LOW

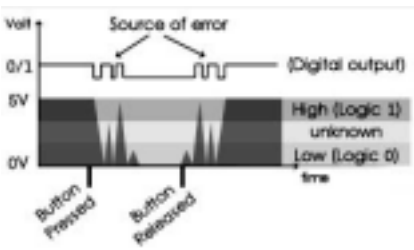
INPUTS		OUTPUTS		COMMENTS
$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$	
0	0	1	1	Invalid condition
0	1	1	0	Latch SET
1	0	0	1	Latch RESET
1	1	NC	NC	No change. Latch remains in present state

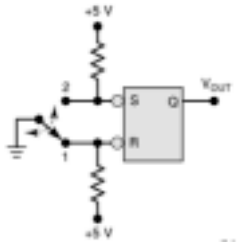
- The Q and Q' outputs are supposed to be in **opposite** states.
- **Q=1** and Q'=1 is defined as Invalid condition (by making S'=0 and R'=0)
- **Q=1** and Q'=0 is defined as Set condition (by making S'=0 and R'=1)
- Q=0 and **Q'=1** is conversely defined as ***Reset*** (by making S'=1 and R'=0)
- When S' and R' are both equal to **1**, the Latch outputs "***Not Change***" from their prior states.

Example of an application:

→ Switch debouncer

Switch is a mechanical device, although when press it makes an electrical contact, actually it makes multiple "contact and release" before the permanent contact. The effect is that we will get a multiple input of 0 and 1, where it should be only one 0 or 1.





To overcome the problem, use an active low SR latch. S' and R' input connected to +5V, this will provide a logic HIGH to the input. A switch is connected to ground that provides a logic LOW.

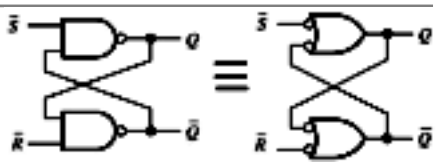
Let say a switch at position 1, it will make R' = 0 and S' = 1, latch will reset and Q becomes 0.

When the switch move to position 2 but at a position 1 and 2 (not in contact with 2 yet) S' = R'= 1 and the state is HOLD and the output stays at 0, even though the switch bounce and there is the possibility of 0 and 1 , it wont affect the output because R'= 0 will make Q = 0 and a R'= 1 will hold the previous value which is 0.

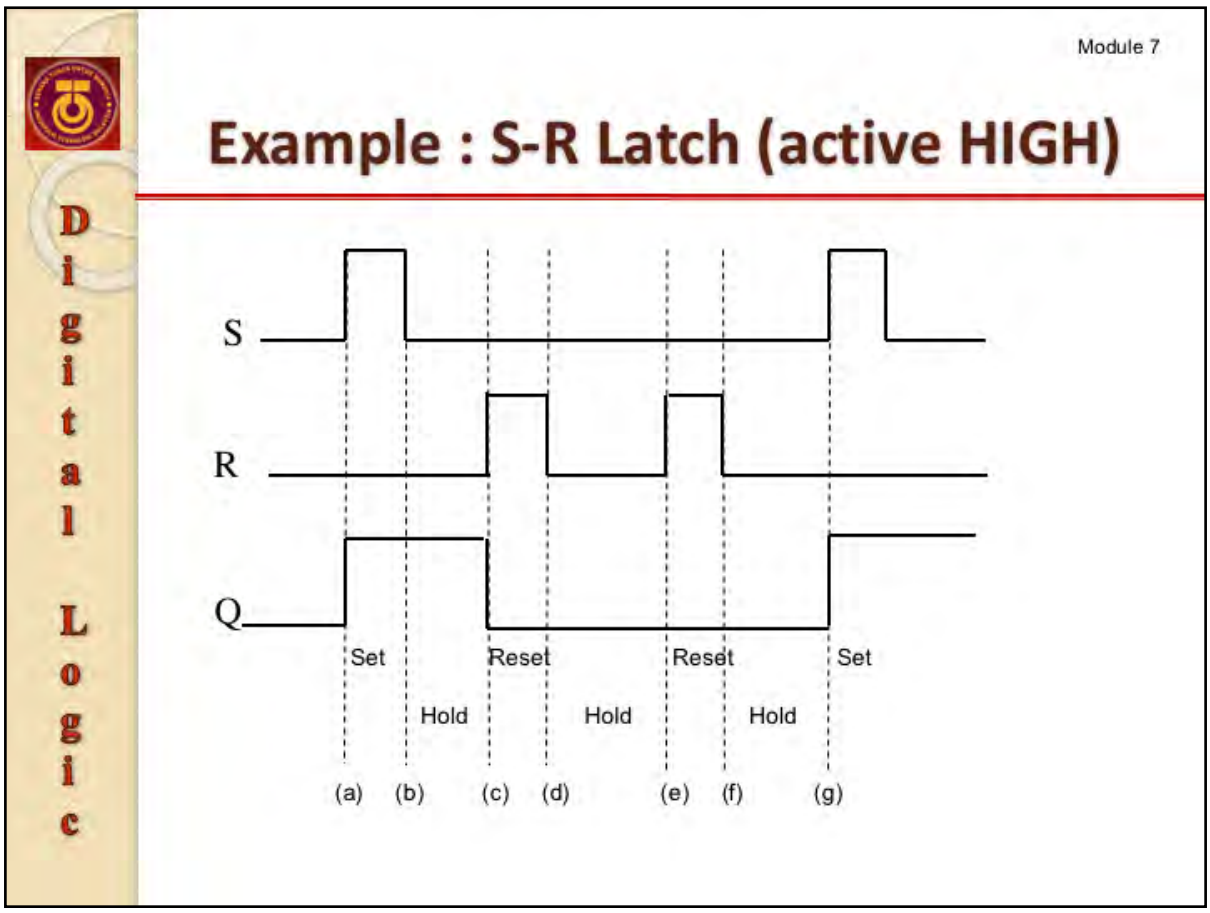
The same argument apply to position 2, when the switch make a contact , S' = 0 and Q will be 1. Even though it bounce , S' will have a value of 1 and 0, but Q will not change because S'= 0 will make Q-1 and S'=1 will be a HOLD state because R' = 1.

DIY:

An active LOW SR Latch can be build by using NAND gate. Explain the operation of the circuit using the same argument as the explanation of an active HIGH SR latch.



Hint: It will be easy if you use the equivalent circuit (negative OR) on the right side.



State Table for active HIGH SR Latch based on the above waveform

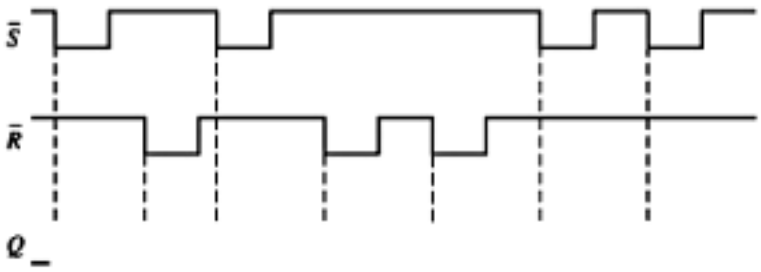
Location	S	R	Q	State
(a)	1	0	1	SET
(b)	0	0	1	HOLD
(c)	0	1	0	RESET
(d)	0	0	0	HOLD
(e)	0	1	0	RESET
(f)	0	0	0	HOLD
(g)	1	0	1	SET

DIY: Do the following:

a) Draw the missing dotted line for every signals change.

b) Draw waveform for Q.

c) Label the output state.



Gated S-R Latch

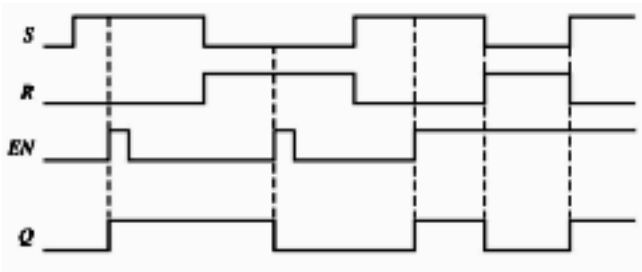
Digital Logic

- S-R latch is *asynchronous*, which means that the output changes very soon after the input S and R change
- Gated S-R has another input, called **enable** (EN), that control when the output should change
- When this input is enabled (EN =1), the output will depend on the value of S and R.
- If EN=0, the SR latch is disable, input S and R doesn't have an effect on the output, the output does not change.
- EN pin has a higher priority than pin S and R
  - SR input will only has an effect to the output if pin EN =1
  - The EN pin can be use to prevent the occurrence of invalid state at the output.

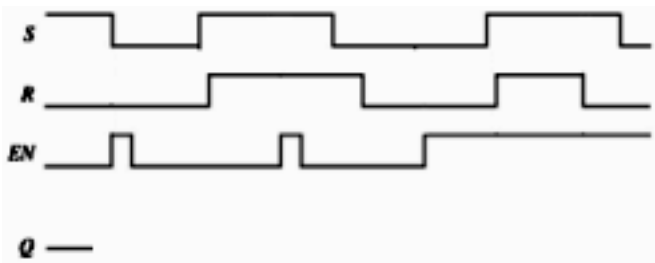
Truth Table for Gated SR Latch operation

EN	S	R	Output	Disable / Enable
0	X	X	No Change	Disable
1	0	0	No Change	Enable
1	0	1	Q = 0	Enable
1	1	0	Q = 1	Enable
1	1	1	Invalid	Enable

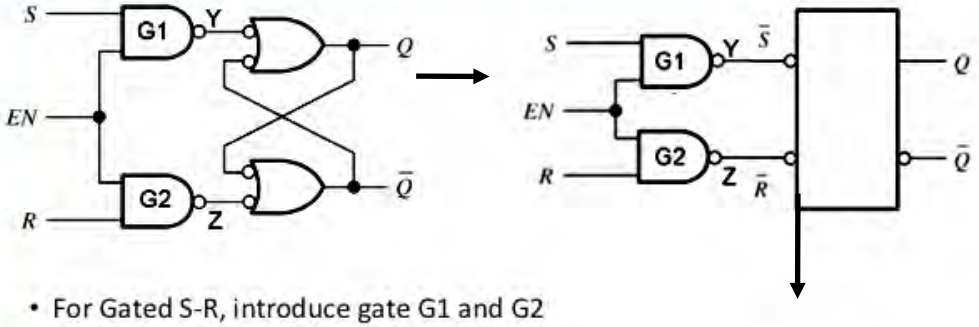
Timing diagram that shows the effect of EN input



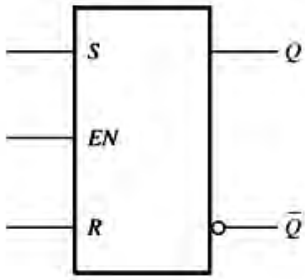
DIY: Draw the output Q waveform.



Gated S-R Latch: Circuit and Symbol



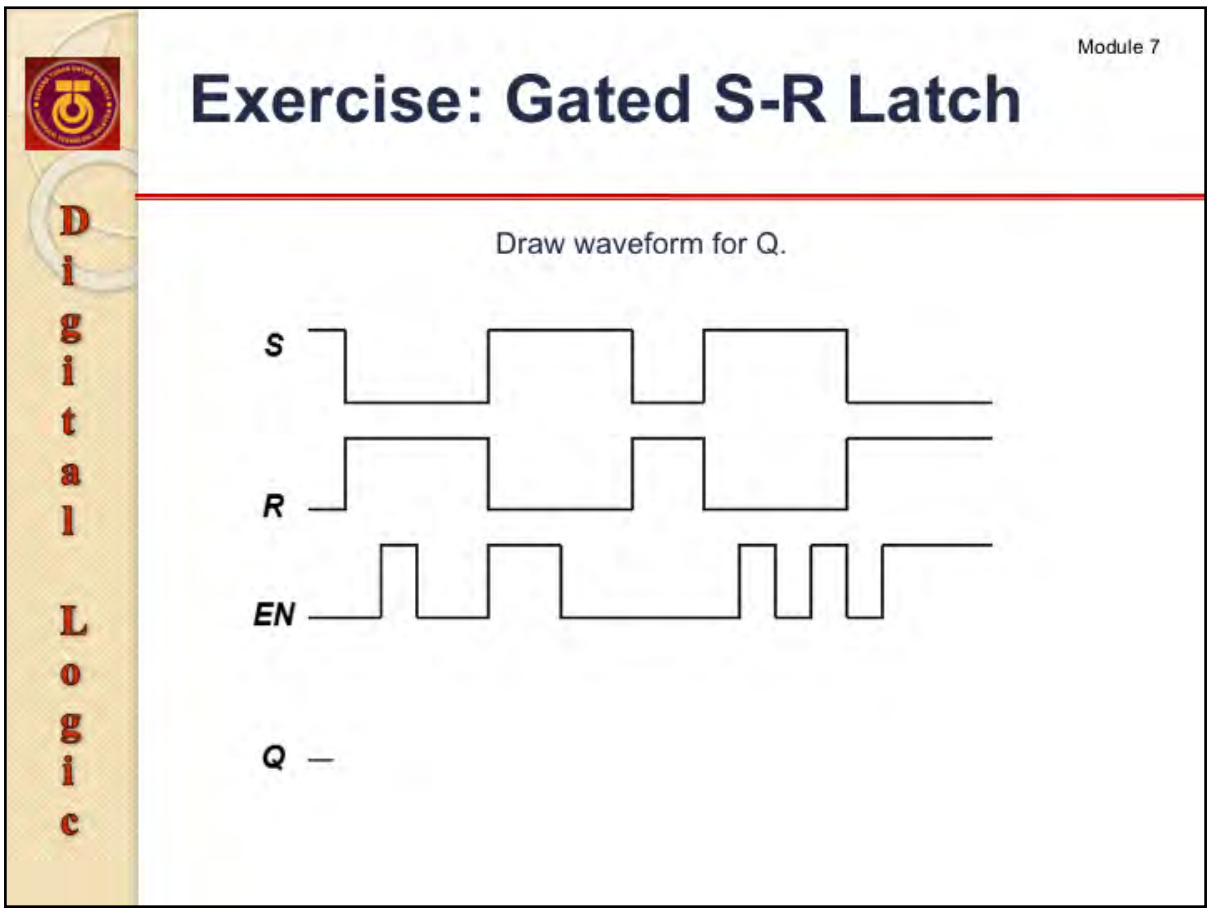
- For Gated S-R, introduce gate G1 and G2
  - behaves as a gate that control whether to allow S to Y and R to Z or to make Y=Z=1.
  - Y = (S.EN)' and Z = (R.EN)'
    - If EN = 0, Y =1 and Z=1
    - If EN=1, Y =S' and Z = R' this will change an active low SR to an active HIGH SR flip-flop



DIY:

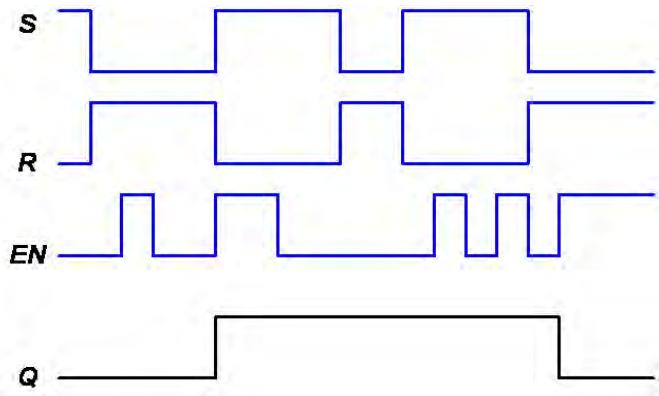
Referring to logic diagram above, fill in the table. Assume initially Q = 0 and Q' = 1

EN	S	R	Y	Z	Q
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			



DIY:

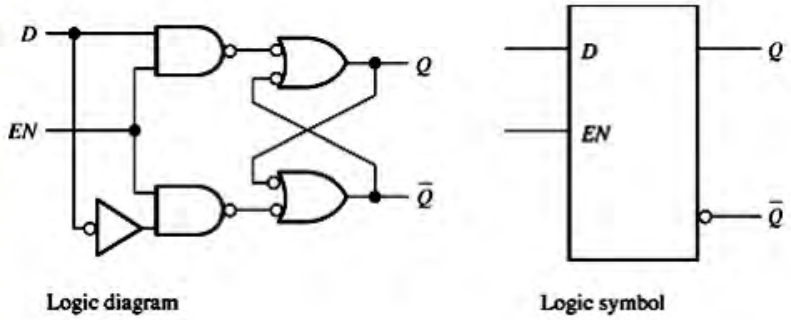
Add the vertical line for EN waveform. Deduce the understanding of the Q waveform.



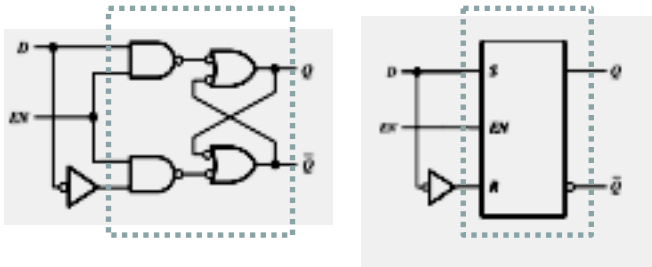
Gated D Latch

Digital Logic

Logic Symbol & Logic Diagram



The Gated SR in the dotted rectangle on the left can be represented by a symbol in the dotted rectangle on the right. From the Gated SR, we can construct the Gated D latch by adding an inverter.



The purpose of the inverter is to make sure that R is the complement of S and R will never be the same as S. So that, we have only two condition at the input of SR.

Condition 1: D = 0, Therefore S = 0, R = 1 which make Q = 0

Condition 2: D = 1, Therefore S = 1, R = 0 which make Q = 1

Just like gated SR, when EN = 0, the output does not change. BUT, if EN = 1, the output will depend on the value of input D.

So, we can conclude that Q = D when EN = 1.

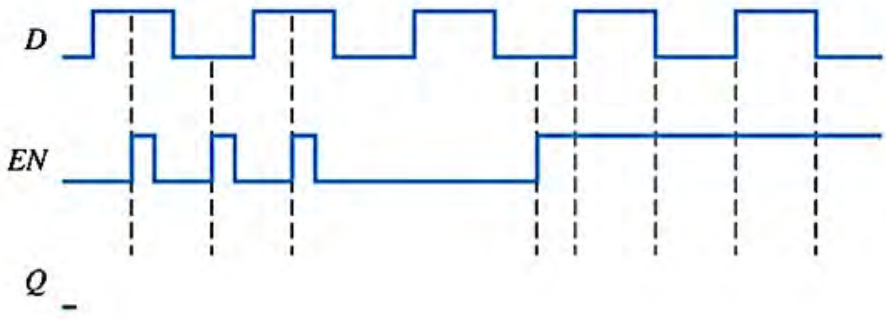
The characteristic of the gated D latch can be represented in the truth table →

EN	D	Output
0	0	No Change
0	1	No Change
1	0	Q = 0
1	1	Q = 1

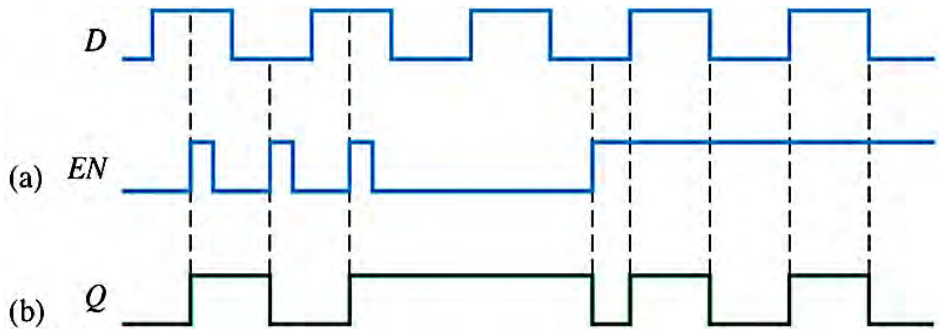
Exercise: Gated D Latch

Digital Logic

Draw waveform for Q.



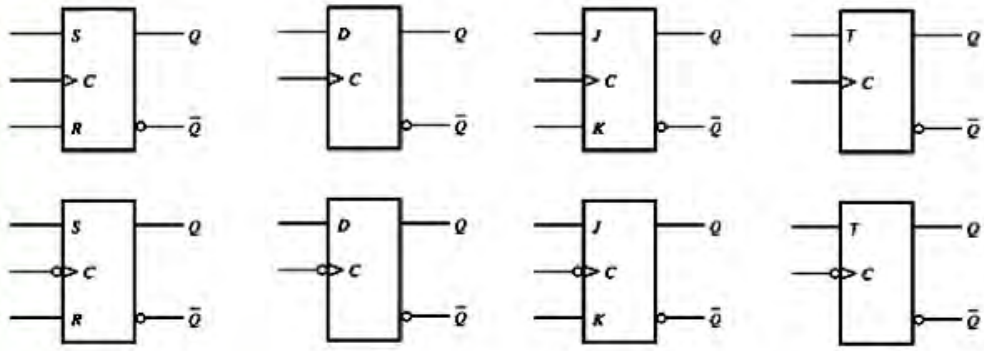
Solution:



Edge-Triggered FF

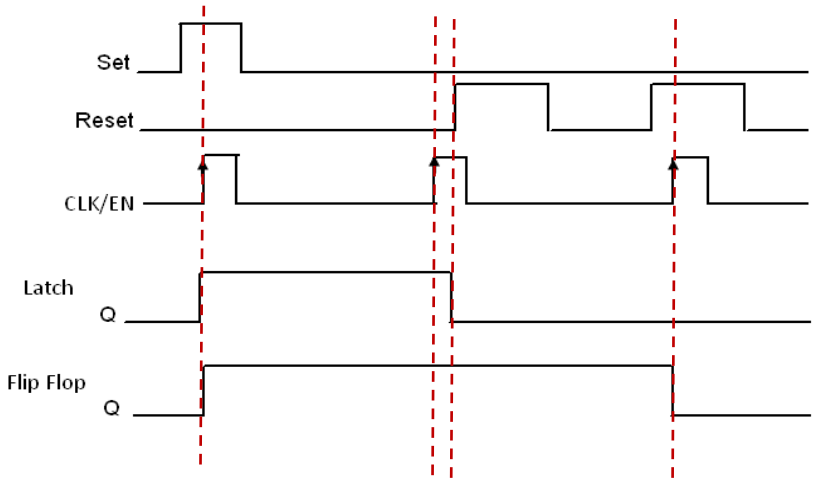
Digital Logic

- Flip-flop is an important device in a sequential circuit



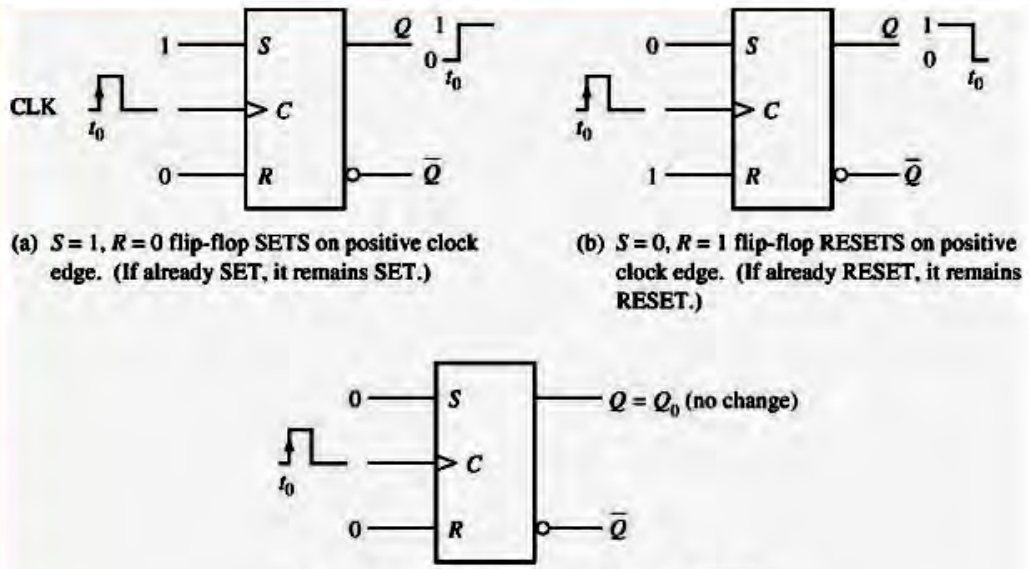
(top: positive edge-triggered; bottom: negative edge-triggered)

- The different between Gated Latch (EN) and Flip-Flop (CLK):
  - Gated Latch is activated by using a level triggered EN (i.e '0' or '1')
  - FF is activated by using edge triggered CLK (i.e 'transition from '0 to 1' or '1 to 0')



- Four type of FFs: SR (Set /Reset) , D (Delay/Data), JK and T (Toggle)
- Triggering of the FF (CLK): +tve edge, –ve edge and Master-slave (obsolete)
- Latch and FF are the basic of sequential circuits.

S-R FF: operation



(a) $S = 1, R = 0$ flip-flop SETS on positive clock edge. (If already SET, it remains SET.)

(b) $S = 0, R = 1$ flip-flop RESETS on positive clock edge. (If already RESET, it remains RESET.)

(c) $S = 0, R = 0$ flip-flop does not change. (If SET, it remains SET; if RESET, it remains RESET.)

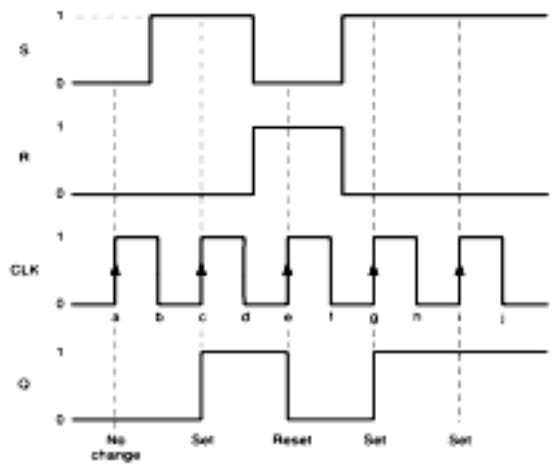
Truth Table for an **active HIGH S-R** flip-flop

S	R	CLK	Output		Comments
			Q	$\overline{Q}$	
0	0	X	Q_0	$\overline{Q_0}$	No Change
0	1	↑	0	1	RESET
1	0	↑	1	0	SET
1	1	↑	?	?	Invalid

TRUTH TABLE

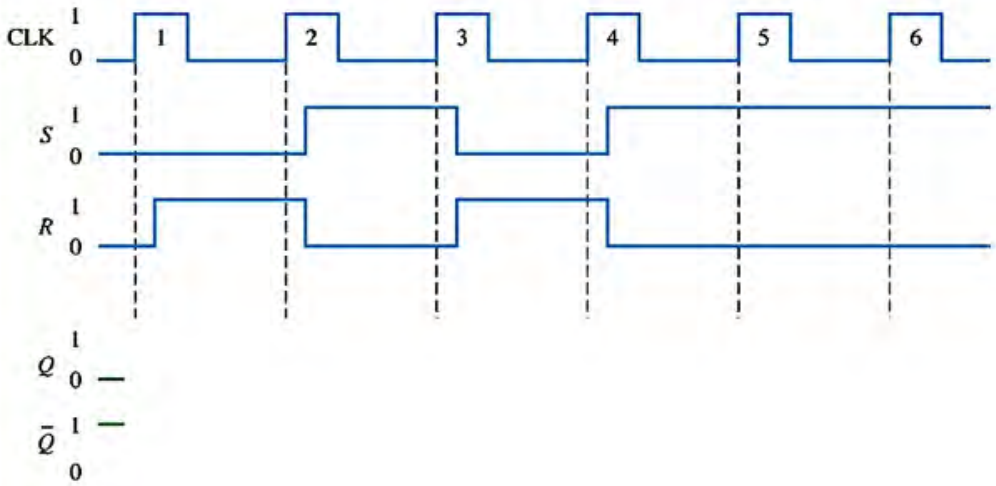
↑ = clock transition LOW to HIGH
X = irrelevant (don't care)
Q_0= output level prior to clock transition

Timing diagram of an **active HIGH S-R** flip-flop

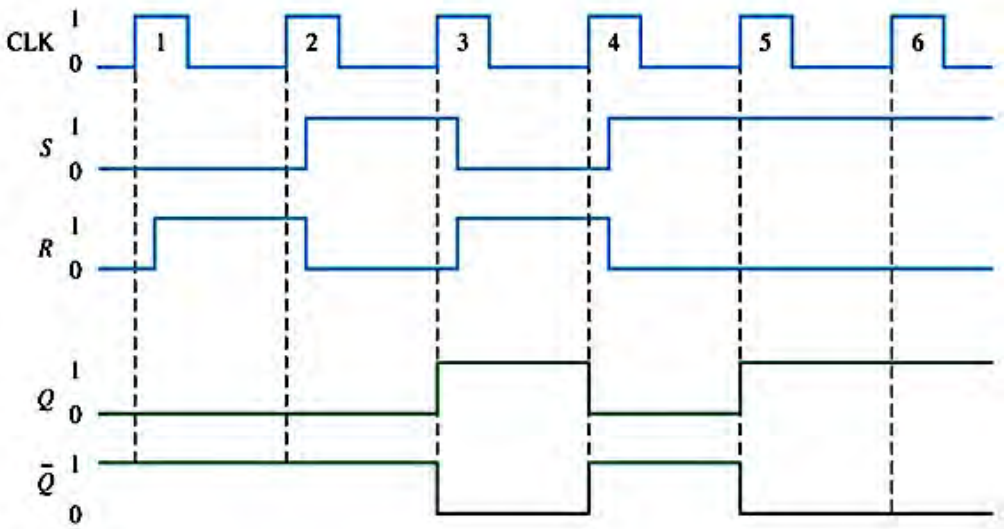


Note : In our class, the logic of input at clock edge is determine by looking at it logic value immediately before the clock edge. What actually happens in the real circuit will depend on the setup and hold time of the flip-flop.

Example : S-R FF



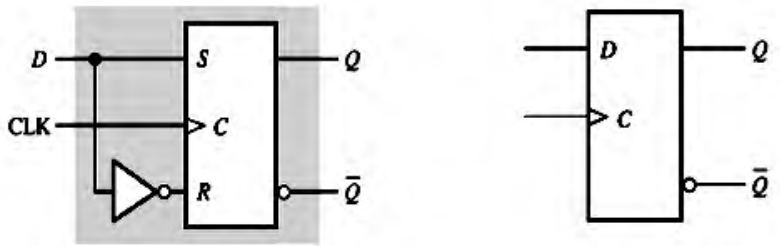
Solution



D Flip-Flop

Digital Logic

- D FF is useful when a single data bit (1 or 0) is to be stored
  - D can mean Data (ability to store data) or Delay (to delay a signal by one clock cycle)
- The addition of an inverter to an S-R FF creates
- basic D FF where a positive edge-triggered type is shown.

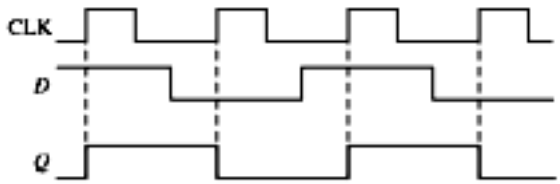


- These flip flops are very useful, as they form the basis for **shift registers** (that we will learn later)
- The advantage of the D flip-flop over the D-type latch is that it "captures" the signal at the moment the clock goes high, and subsequent changes of the data line do not influence Q until the next rising clock edge (for a positive edge D flip-flop)

D Flip-flop truth table:

INPUTS		OUTPUTS		COMMENTS
D	CLK	Q	$\bar{Q}$	
1	↑	1	0	SET (store a 1)
0	↑	0	1	RESET (store 0)

Timing diagram for a positive edge D flip-flop



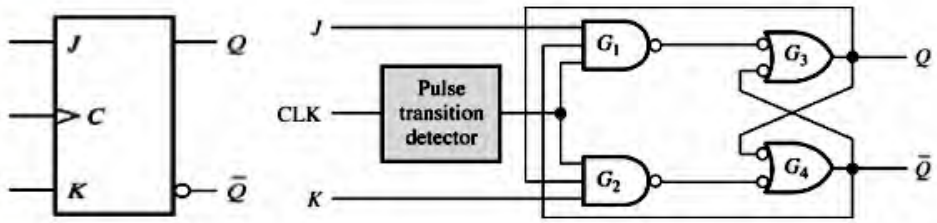
Remember, Q follows D at the active or triggering clock edge

J-K Flip Flop

Digital Logic

- The JK FF is versatile and is a widely used type of FF.
- The difference between J-K and S-R is a J-K has **no invalid state** as SR.

Logic Symbol



The JK flip-flop is a universal flip-flop, because it can be configured to work as an SR flip-flop, a D flip-flop, or a T flip-flop.

The characteristic equation of the JK flip-flop is:

$$Q_{next} = J\overline{Q}_{prev} + \overline{K}Q_{prev}$$

The problem with active HIGH SR is when both of the inputs are HIGH (i.e. invalid state).

In JK flip-flop, J has the same characteristic as active HIGH S and K has the same characteristic as active HIGH R.

However, in JK FF when J = K = 1 and plug in into JK characteristic equation

- ✓ If Q_{prev} =1 then Q_{next} will be 0.
- ✓ If Q_{prev} = 0 , then Q_{next} will be 1

Therefore, there is no **invalid state** BUT we have a new state called **TOGGLE**, where the next output value is NOT the previous output:

$$Q_{next} = \overline{Q}_{prev}$$

Truth Table: J-K FF

Digital Logic

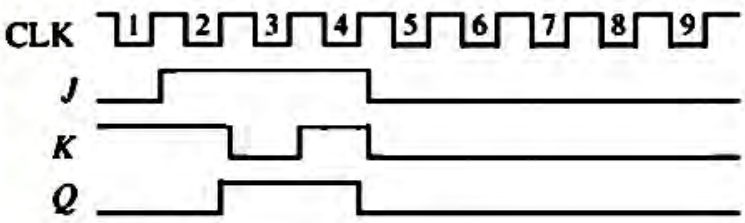
J	K	CLK	Output		Comments
			Q	$\overline{Q}$	
0	0	↑/↓	Q_0	$\overline{Q_0}$	No Change
0	1	↑/↓	0	1	RESET
1	0	↑/↓	1	0	SET
1	1	↑/↓	$\overline{Q_0}$	Q_0	Toggle

TRUTH TABLE

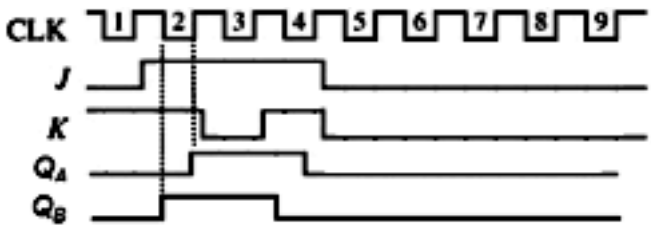
↑ = clock transition LOW to HIGH

Q_0= output level prior to clock transition

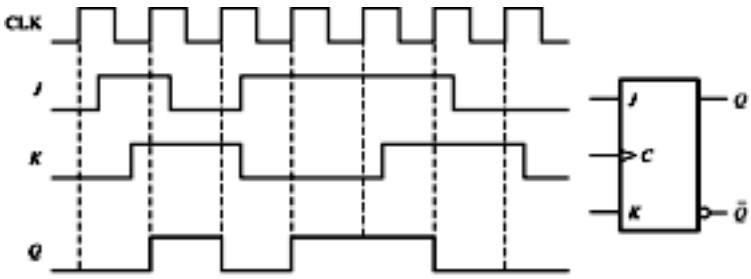
Positive or negative triggered ?



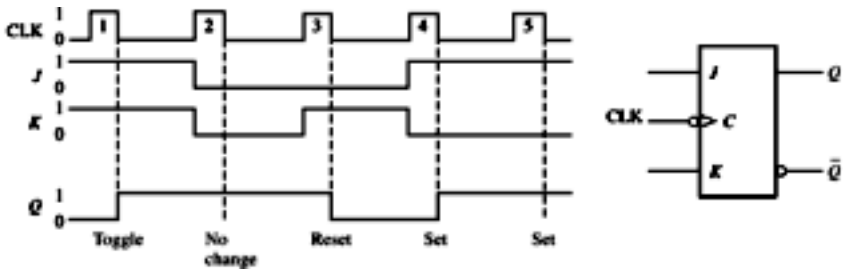
- The truth table above shows the output for positive edged-triggered J-K Flip-Flop.
- The output Q for a negative edged-triggered J-K Flip-Flop is shown in Q_B:



Example 1 : Positive Edge



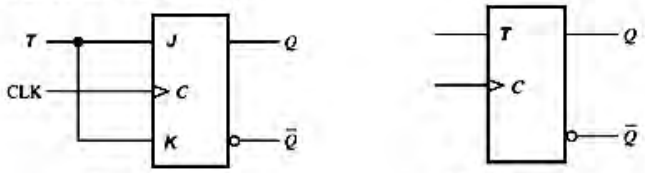
Example 2: Negative Edge



T Flip Flop

Digital Logic

- Also called as Toggle flip flop.
- Can be build from JK by connecting J and K to the same node.
- Therefore it will always make J=K. Thus, there is only 2 output states (i.e. either HOLD or TOGGLE).

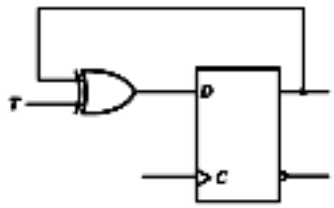


Logic Diagram Logic Symbol

- Truth table for T Flip-flop

T	Q_{prev}	Q_{next}	Comment
0	0	0	Hold state
0	1	1	
1	0	1	Toggle state
1	1	0	

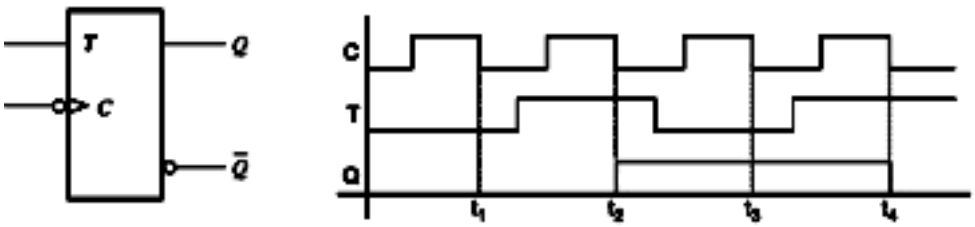
- D can be used to build a T flip-flop as in circuit →



$$D = T \oplus Q = T\overline{Q} + \overline{T}Q$$

- The expression for D input
  - When T = 0 then D = Q, if T = 1, D = Q', it behaves as aT flip-flop.

- Timing diagram of a negative edge T Flip-flop

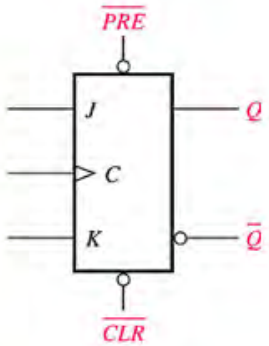


FF Asynchronous Input

Digital Logic

- Synchronous → inputs transferred on the triggering edge of the clock
- Asynchronous → inputs effect FF state independent of the clock
  - Normally labeled (depends on the manufacturer)
    - preset ($\overline{PRE}$) or direct set
    - clear ($\overline{CLR}$) or direct reset

$\overline{PRE}$	$\overline{CLR}$	FF	MODE
0	1	SET	Asynchronous
1	**0**	RESET	Asynchronous
1	1	JK	Synchronous



- Flip-flop input priority:

Priority	FF Input
Highest	$\overline{PRE}$, $\overline{CLR}$ - asynchronous input
Medium	Clock
Lowest	SR, JK,D,T -synchronous input

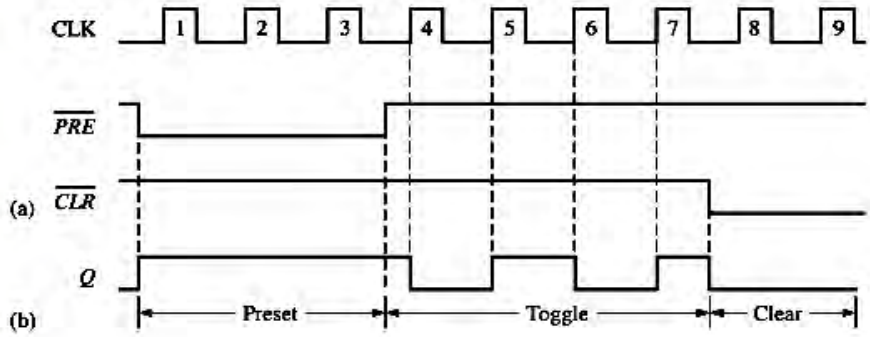
- Flip-flop asynchronous input has the highest priority.
  - Which means that if the asynchronous input active, the output will immediately change regardless the value of synchronous input or clock.

- There are two types of asynchronous input:
  - preset , $\overline{PRE}$ → to set initial value of 1 to the output
  - clear, $\overline{CLR}$ → to reset the output to 0

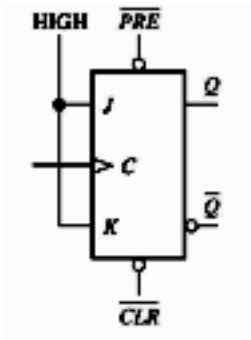
- Usually the asynchronous input are applied momentarily (in a short period of time), because to allow the normal operation of the FF both asynchronous input must be deactivated (HIGH) .

Example: J-K FF

clk pulse	$\overline{PRE}$	$\overline{CLR}$	J	K	FF State	Comment
1,2,3	0		1	1	SET	JK inputs - dont care
4,5,6, 7	1	1	1	1	Toggle	Synchronous mode.
8,9		0	1	1	RESET	JK inputs - dont care



- The timing diagram is for JK Flip Flops connected as T FF, as shown in the figure →

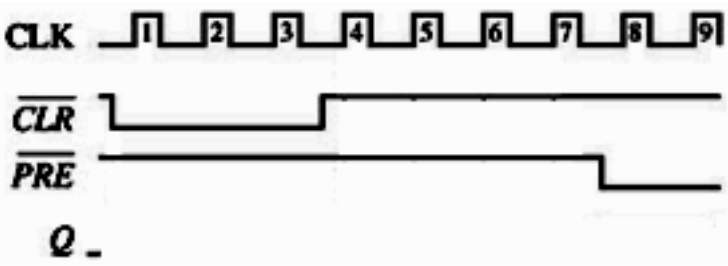


- From the timing diagram, it can clearly illustrates the asynchronous characteristic of the $\overline{PRE}$ and $\overline{CLR}$, where when the asynchronous input change the output will immediately change without waiting for the clock edge.

Tips:

- *If you are required to draw the timing diagram with an asynchronous input, draw output cause by the asynchronous input first. When draw this output ignore the edge of the clock.*
- *After that you can ignore the asynchronous input, draw the effect of a synchronous input and take note that the output will only change at the edge of the clock.*

DIY: Draw the output, Q waveform

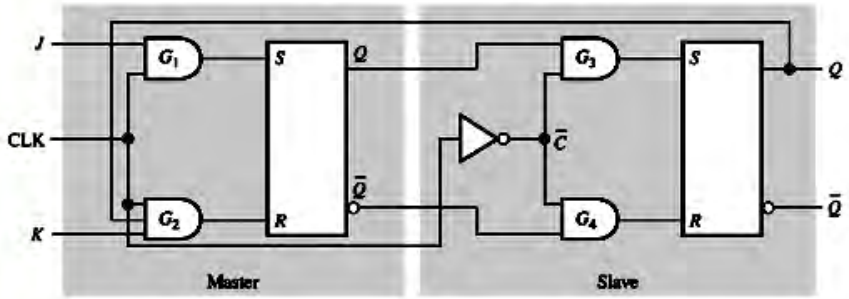


Module 7

Master Slave Flip Flops (JK Example)

Digital Logic

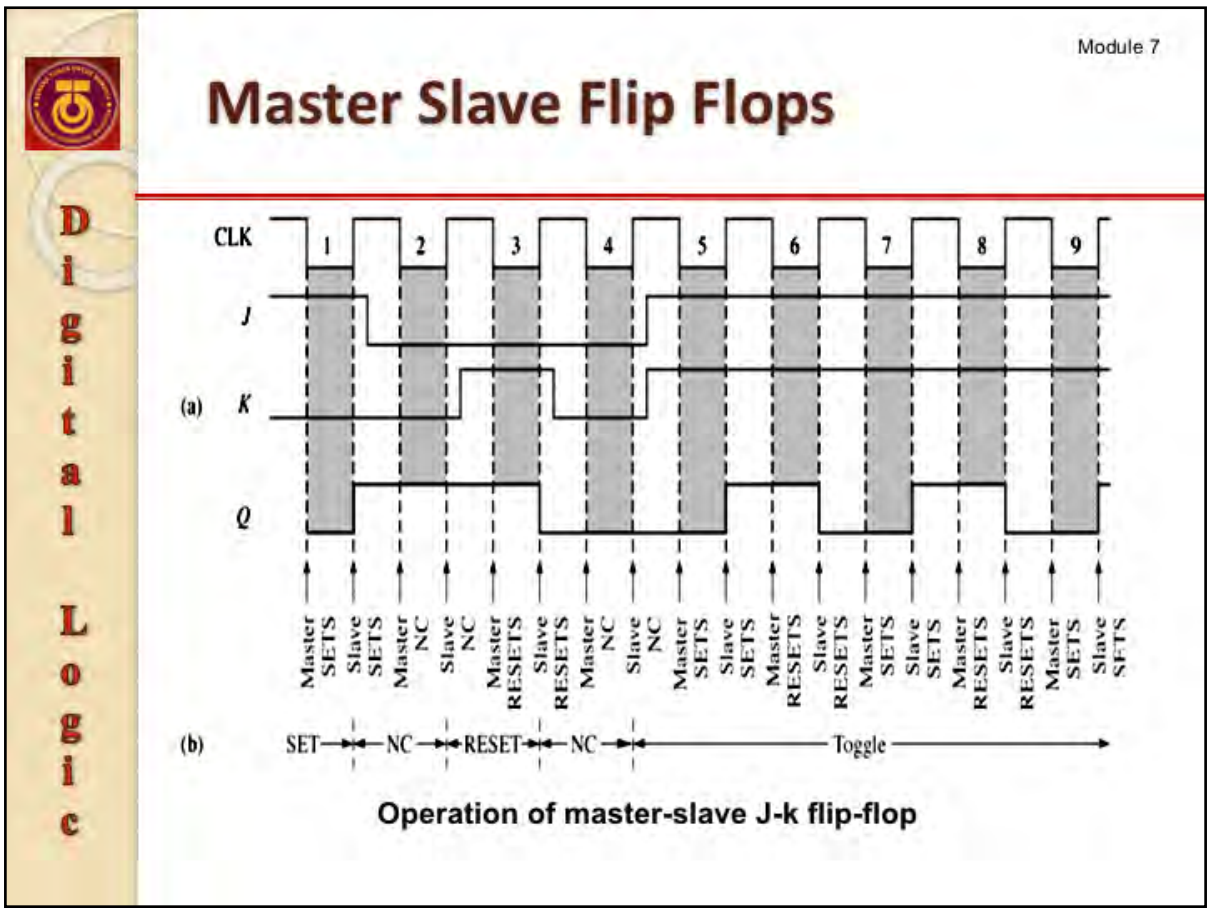
- This type of Flip-flop is already obsolete, the mention here is for the sake of completeness
- You must know it exist, where you might find in the old circuit board.



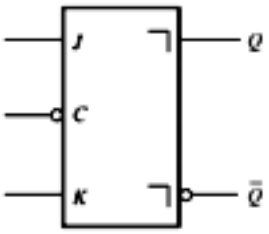
- There are 2 sections called as :
  - **Master section** - *external JK input* followed by a *gated latch*
  - **Slave section** - an *inputs which is the outputs of Master* and the *I inverted clock pulse*

Master-slave flip-flop:

- Another class of flip-flop is the pulse-triggered or master-slave. These flip-flops are constructed from two separate flip-flops. The term pulse-triggered means that data are entered into the flip-flop on the leading edge of the clock pulse, but the output does not reflect the input state until the trailing edge of the clock pulse. This is due to the master flip-flop being rising edge triggered and the slave flip-flop being falling edge triggered as illustrated in the above slide.
- A type of clocked flip-flop consisting of master and slave elements that are clocked on complementary transitions of the clock signal.
- Data is only transferred from the master to the slave, and hence to the output, after the master-device outputs have stabilized.
- This eliminates the possibility of ambiguous outputs, which can occur in single-element flip-flops as a result of propagation delays of the individual logic gates driving the flip-flops.



- The logic symbol for the master-slave flip-flop only indicates the initial inputs to the master and the outputs from the slave



Logic symbol for J-K master-slave flip-flop

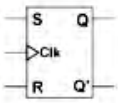
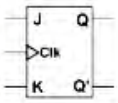
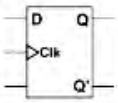
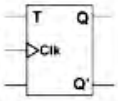
J	K	C	Q	$\bar{Q}$	Operation
0	0	Pulse	Q_0	$\overline{Q_0}$	Hold (no change)
0	1	Pulse	0	1	Reset
1	0	Pulse	1	0	Set
1	1	Pulse	$\overline{Q_0}$	Q_0	Toggle

Truth table for J-K master-slave flip-flop

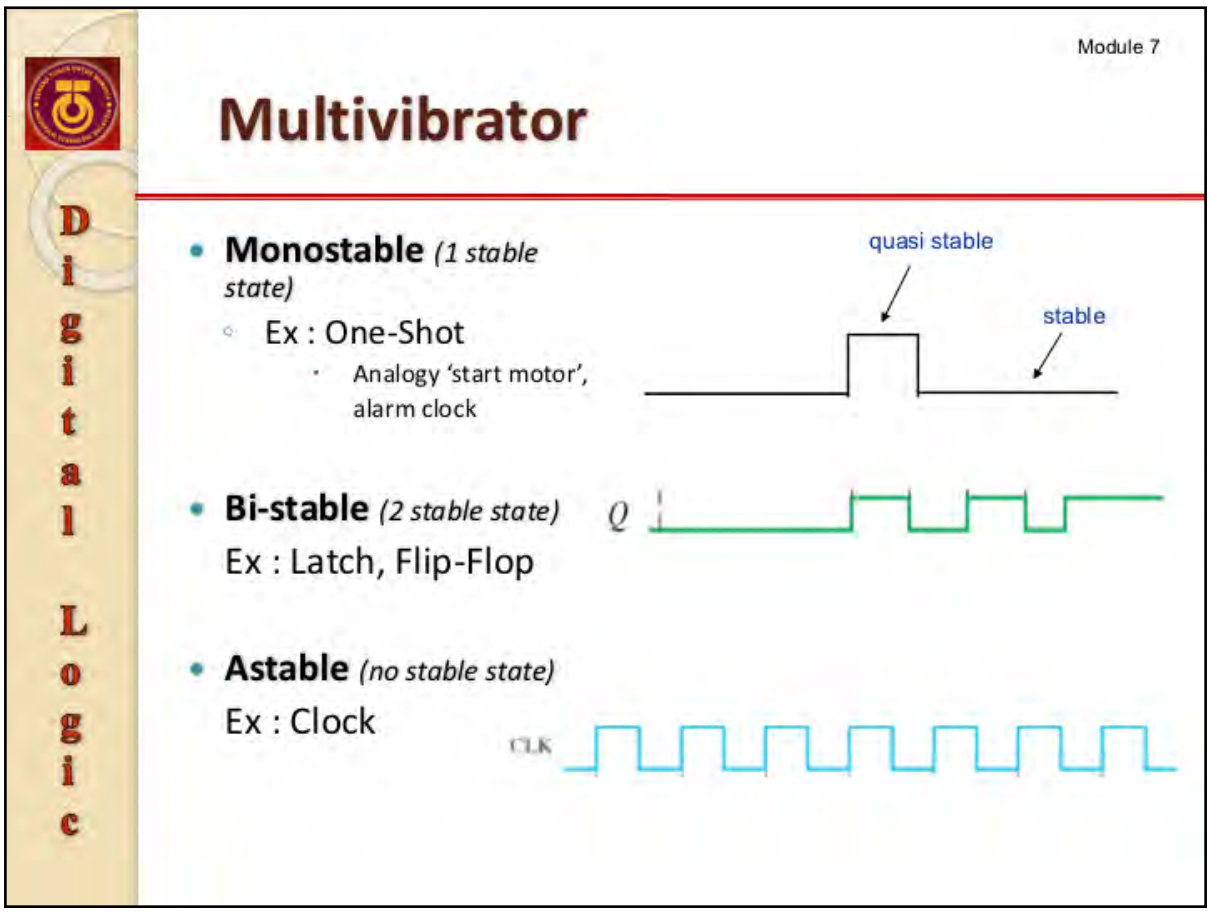
- J-K master slave flip-flop have the same characteristic with the edge triggered J-K , the only difference is how it was triggered and when the input evaluated and the effect appear at the output

Summary of FF characteristic

Digital Logic

FLIP-FLOP NAME	FLIP-FLOP SYMBOL	CHARACTERISTIC TABLE	CHARACTERISTIC EQUATION	EXCITATION TABLE
SR		S R $Q_{(next)}$ 0 0 Q 0 1 0 1 0 1 1 1 ?	$Q_{(next)} = S + R'Q$ $SR = 0$	Q $Q_{(next)}$ S R 0 0 0 X 0 1 1 0 1 0 0 1 1 1 X 0
JK		J K $Q_{(next)}$ 0 0 Q 0 1 0 1 0 1 1 1 Q'	$Q_{(next)} = JQ' + K'Q$	Q $Q_{(next)}$ J K 0 0 0 X 0 1 1 X 1 0 X 1 1 1 X 0
D		D $Q_{(next)}$ 0 0 1 1	$Q_{(next)} = D$	Q $Q_{(next)}$ D 0 0 0 0 1 1 1 0 0 1 1 1
T		T $Q_{(next)}$ 0 Q 1 Q'	$Q_{(next)} = TQ' + T'Q$	Q $Q_{(next)}$ T 0 0 0 0 1 1 1 0 1 1 1 0

- Characteristic table shows what the output will be for a given input
- Characteristic Equation allow us to determine what the output will be for a given input. Q is the previous input, to determine the output sometimes the previous output will need to be assumed.
- Excitation table can be used to determine what should be applied at the input to get certain output. This table will be use when we design a counter in the next module.



- A multivibrator is an electronic circuit used to generate a square wave or pulse waveform.
- The output can have two state
  - Stable state – stays in one state and only change if force by an external source
  - Quasi-stable – change to this state for a specified period of time and return back to the previous state without applying an external force

- There are three types of multivibrator circuit:

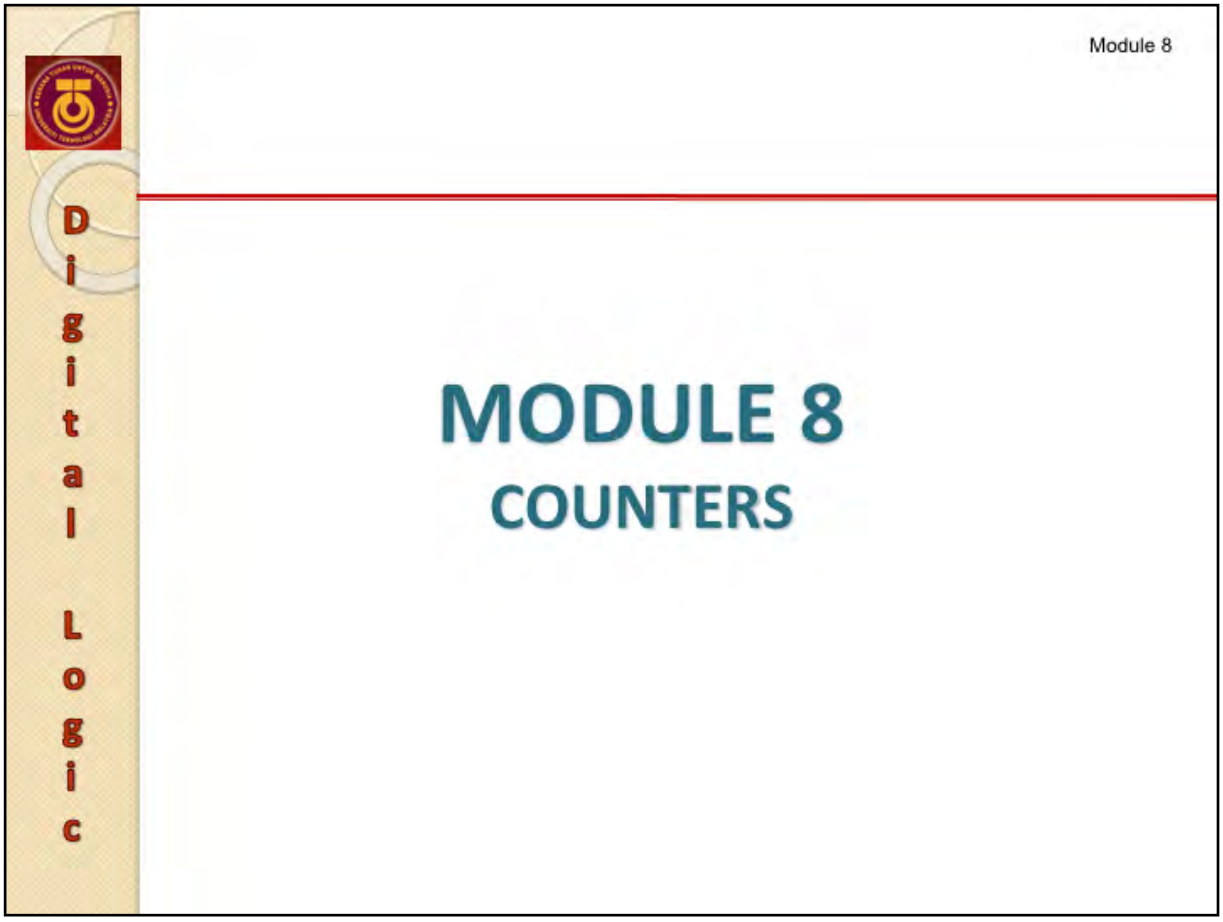
  * **Astable**, in which the circuit is not stable in either state—it continuously oscillates from one state to the other.

  * **Monostable**, in which one of the states is stable, but the other is not—the circuit will flip into the unstable state for a determined period, but will eventually return to the stable state. Such a circuit is useful for creating a timing period of fixed duration in response to some external event. This circuit is also known as a one shot. A common application is in eliminating switch bounce.

  * **Bistable**, in which the circuit will remain in either state indefinitely. The circuit can be flipped from one state to the other by an external event or trigger. Such a circuit is important as the fundamental building block of a register or memory device. This circuit is also known as a flip-flop.

Digital Logic

End of Module 7



The objectives of this module are:

1. To distinguish the different between synchronous and asynchronous counters.
2. To illustrate the designing of counters .
3. To explain the application of counters.
4. To demonstrate the analysis of sequential circuits.

Module Content

Digital Logic

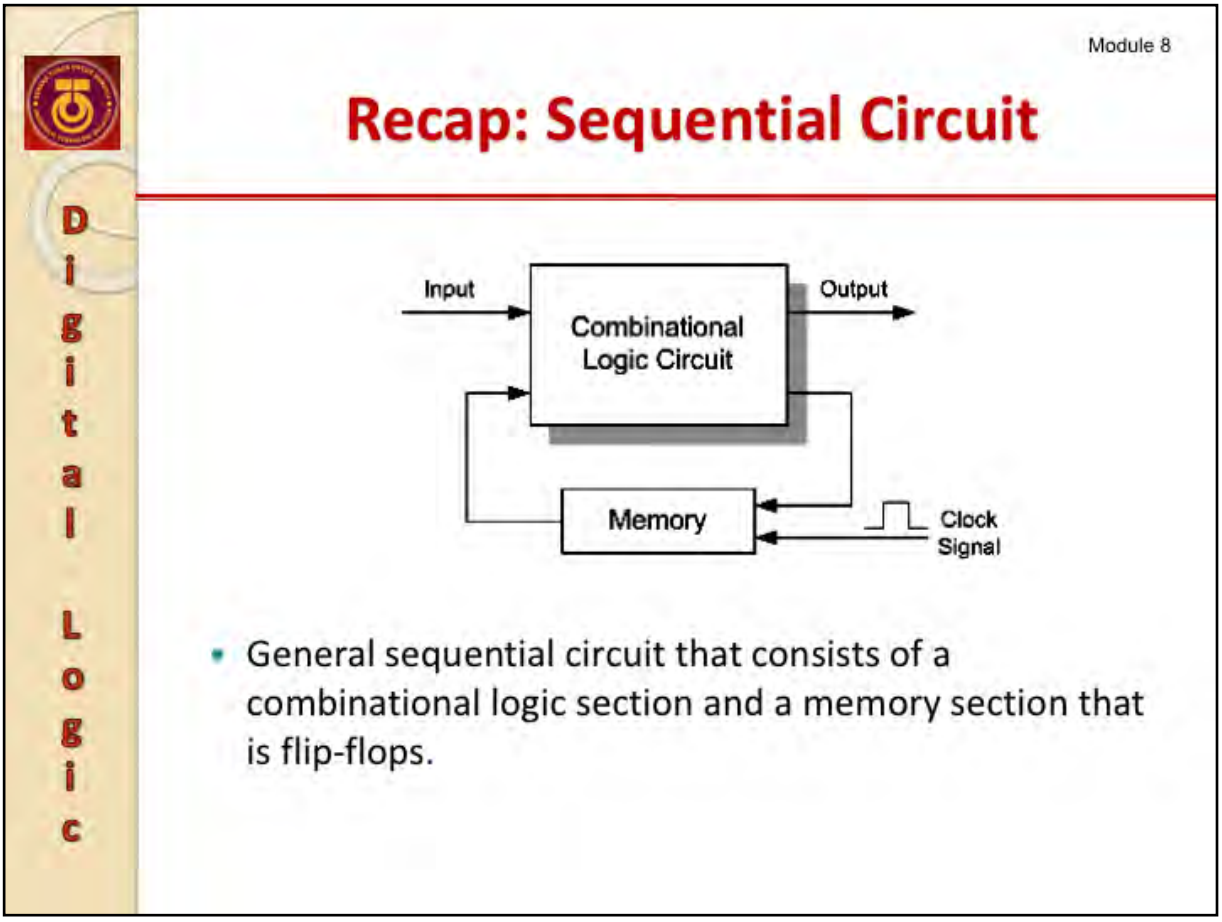
- ASYNCHRONOUS COUNTER
  - 2 bit / 3 bit / 4 bit / BCD Decade
- SYNCHRONOUS COUNTER
  - 2 bit / 3 bit /4 bit / BCD Decade
- UP/DOWN COUNTER
- DESIGN THE SYNCHRONOUS COUNTER
- CASCADED COUNTER
- COUNTER DECODING

There is a variety of counters based on its construction, BUT in general it can be categorized into two categories Asynchronous and Synchronous counter. In both categories the counters can be differentiated by the following criterion:

1. Clock Trigger: Positive edged or Negative edged
2. Counts: Binary, Decade
3. Count Direction: Up, Down, or Up/Down

In the previous lectures, the characteristic of S-R, D, J-K and T flip-flops were discussed. These flip-flops can be connected together to perform certain operations.

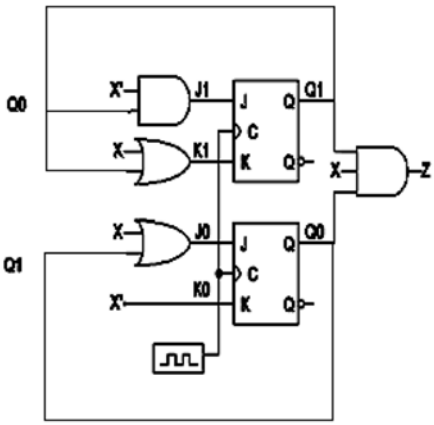
In the following lectures, we will use these flip-flops to construct a variety of counters. In addition we will also learned the method for analysing the different types of counters.

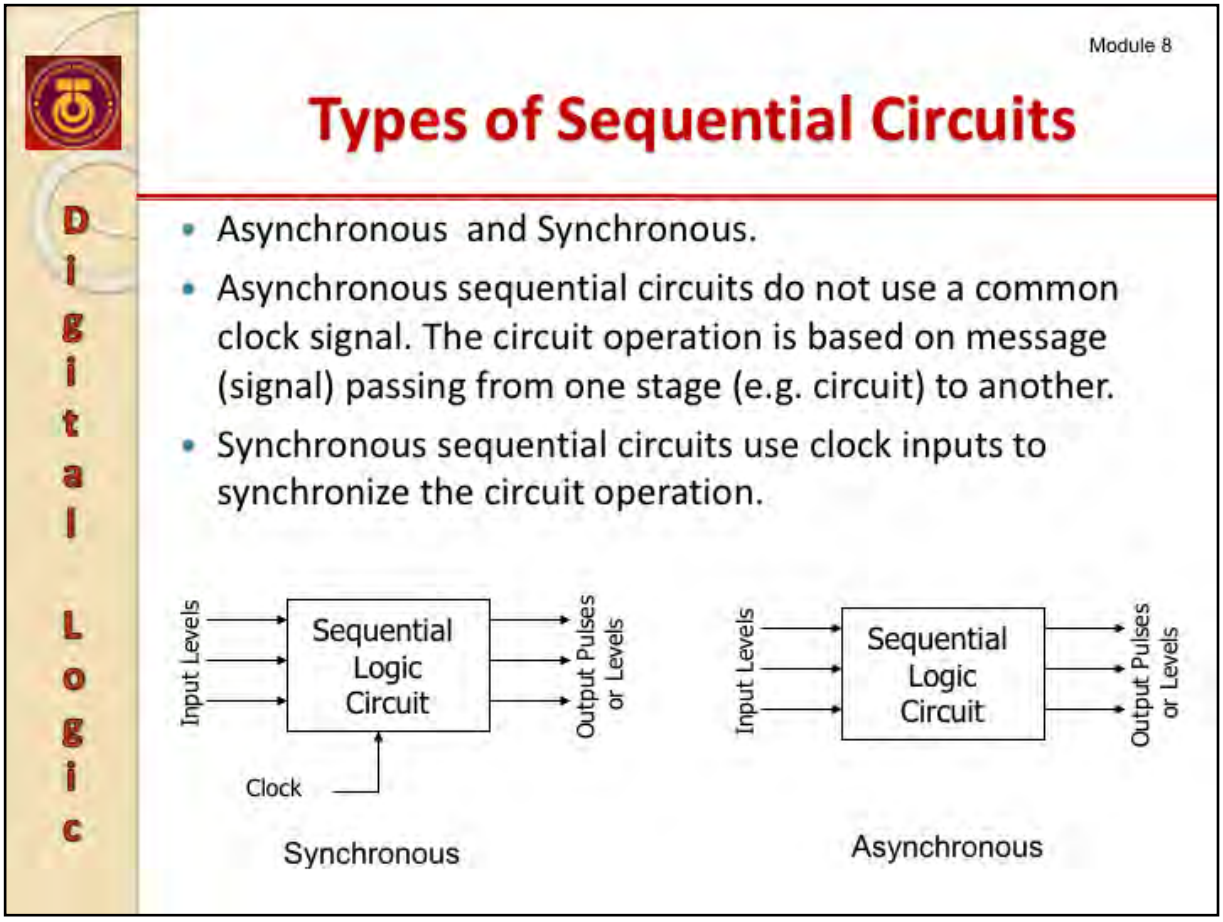


- Memory elements are devices capable of storing binary information.
- Binary information stored at any given time defines the state of the sequential circuit.
- The input and the present state of the memory element determines the output.
- A sequential circuit is specified by a time sequence of inputs, outputs, and internal states.

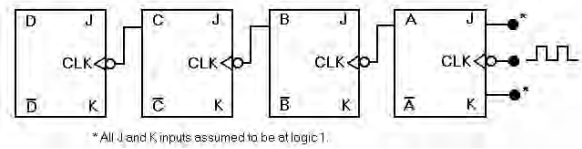
How does sequential circuits look like?

Below is a sequential circuit with two JK flip-flops. There is input X and output Z. The values of the flip-flops (Q1Q0) form the state, or the memory, of the circuit. The flip-flop outputs also go back into the primitive gates on the left. This matches the abstract sequential circuit diagram in the slide.





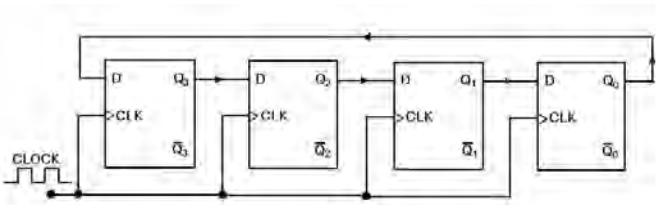
For asynchronous counters , not all flip flops are hooked to the same clock.



Asynchronous Counter

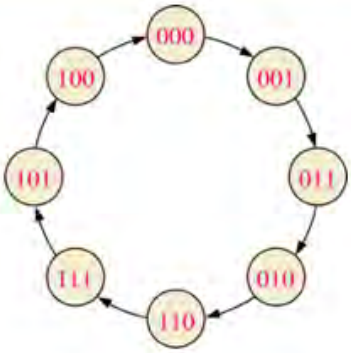
Asynchronous counters, also known as ripple counters, are not clocked by a common pulse and hence every flip-flop in the counter changes at different times. The flip-flops are usually clocked by the output pulse of the preceding flip-flop.

Synchronous counters have a common clock which results in all the flip-flops being triggered simultaneously.



Synchronous Counter

Counters



- Counters are sequential circuit that goes through a prescribed (predefine) sequence of states upon the application of input pulses.
- Sequence of states in a counter may follow a binary count or any other sequence.

- Counters can retain an output state after the input condition which brought about that state has been removed.
- Counter can have more than two states
    - value of a state is expressed as a multi digit binary number
    - the number `1's and `0's are usually derived from the outputs of internal flip-flops that make up the counter.
- The number of states a counter may have is limited only by the amount of electronic hardware that is available.
- The changes from one state to another is controlled by the input pulses (count pulses). It may be clock pulses or originated from an external source and may occur at prescribed intervals of time or at random.

- Counters are generally made up of flip-flops and logic gates.
    - Main types of flip-flops used in designing counter are J-K, T or D flip-flops.

Uses of Counters

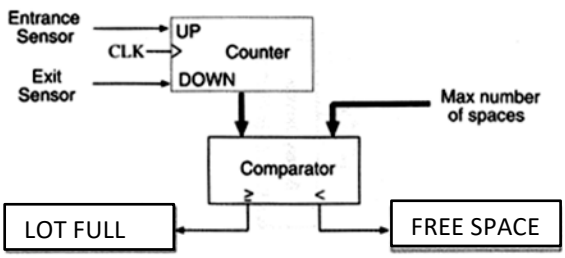
Digital Logic

- To count the number of times that a certain event takes place; the occurrence of event to be counted is represented by the input signal to the counter (see figure below).
- To control a fixed sequence of actions in a digital system
  - e.g gate A must be open before gate B.
- To generate timing signals
  - e.g a bomb can only be detonated 20 second after a button is pressed
- To generate clocks of different frequencies.
  - By dividing the frequency, we can get a different clock frequencies from the same source

An application of a counter:

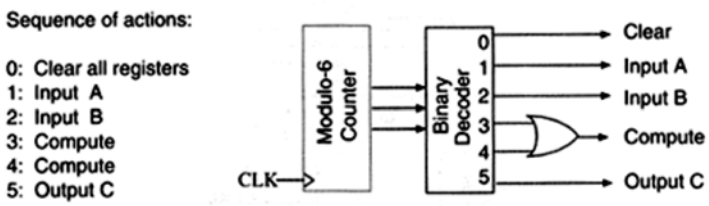
1. **Parking Space**

   There are sensors at the entrance and exit of the parking lot. A car at the entrance will increment the counter and car exit from the parking lot will decrement the counter. The current number of car in the parking lot will be compared to the Max number of spaces that the parking lot has. If number of car less than max number of spaces then FREE SPACE display will lit else LOT FULL will be displayed.



2. **Action sequencer**

   A counter can generate a number 0 to 5. Every positive edge of a clock the number will change to the next number. Every number generated by the counter will be decoded to perform some action. This will allow a sequence of actions to occurs as listed below.



Count Sequence

Digital Logic

- A counter may count up or count down or count up and down depending on the input control.
- Because of limited word length, the count sequence is limited.
  - For an *n*-bit counter, the range of the count is [0, 2^n-1]. The count sequence usually repeats itself.
- When counting up, the count sequence goes in this manner: 0, 1, 2, … 2^n-2, 2^n-1, 0, 1, …etc.
- When counting down the count sequence goes in the same manner: 2^n-1, 2^n-2, … 2, 1, 0, 2^n-1, 2^n-2, … etc.

A counter can be constructed by an asynchronous circuit or by a synchronous circuit. With an asynchronous circuit, all the bits in the count do not all change at the same time. With a synchronous circuit, all the bits in the count change synchronously with the assertion of the clock.

A counter may count up or count down or count up and down depending on the input control.

3-bit Up Counter	3-bit Down Counter
000	000
001	111
010	110
011	101
100	100
101	011
110	010
111	001

The complement of the count sequence counts in reverse direction. If the uncomplemented output counts up, the complemented output counts down. If the uncomplemented output counts down, the complemented output counts up.

3-bit Up Counter	Complement of the Count
000	111
001	110
010	101
011	100
100	011
101	010
110	001
111	000

Characteristic of Counter

Digital Logic

- Counter can be described by the following characteristics:
  - Asynchronous or synchronous
  - Counting sequence
  - Modulus (MOD)
  - Whether the counter repeat the counting sequence

- Asynchronous or Synchronous
  - ✓ Whether the clock come from the same source or not
    - ❑ Synchronous – all clock from the same source
    - ❑ Asynchronous – clock for each flip-flop comes from a different source

- Counting sequence: Up or Down or Up-Down
  - ✓ Up or Down – unidirectional – the count sequence only one way.
  - ✓ Up-Down – bidirectional – the direction of the count can be change during operation.

- Modulus (MOD): the number of state that the counter can have.
  - ✓ e.g. MOD 4 (4 states with the following state 0, 1, 2, 3), MOD 9 (0, 1, 2, … , 8).
  - ✓ the maximum count usually $2^N - 1$, where N is the number of the FF, or for the truncated sequence the max count will be $< 2^N - 1$.

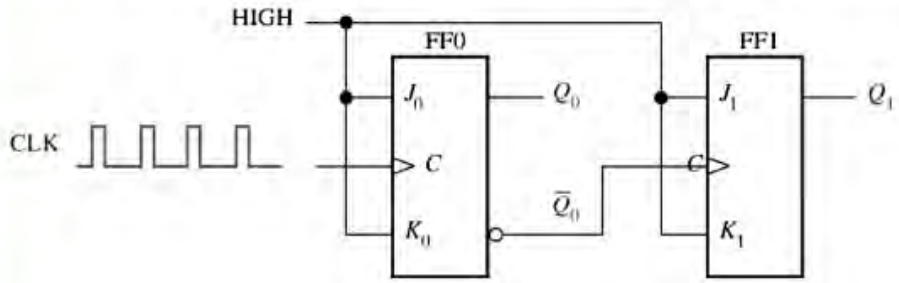
    E.g. MOD 9 (0, 1, 2, …, 8) is truncated sequence.

- Whether the counter repeat the counting sequence:
  - ✓ Recycle – if the counter goes back to the first count after the maximum count
  - ✓ Saturated – the counter repeat the maximum count if count up or repeat the minimum if count down.

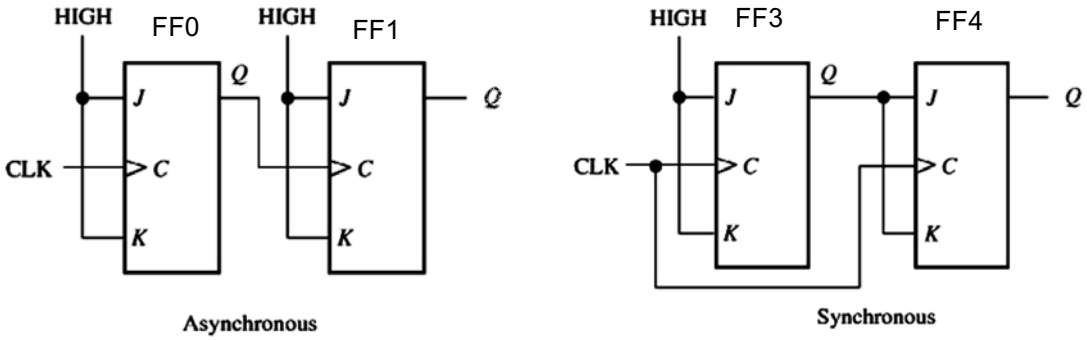
Asynchronous Counter (AC)

Digital Logic

- An AC is one in which the FF within the counter do not change states at exactly the same time because they do not have a common clock pulse.
- The clock input of an AC is always connected only to the LSB FF.
- The clock for the next FF come from the output of the previous FF
- An AC also known as **ripple** counter – message (signal) passing.



Comparison of Asynchronous and synchronous Counter. See the different clock connection!



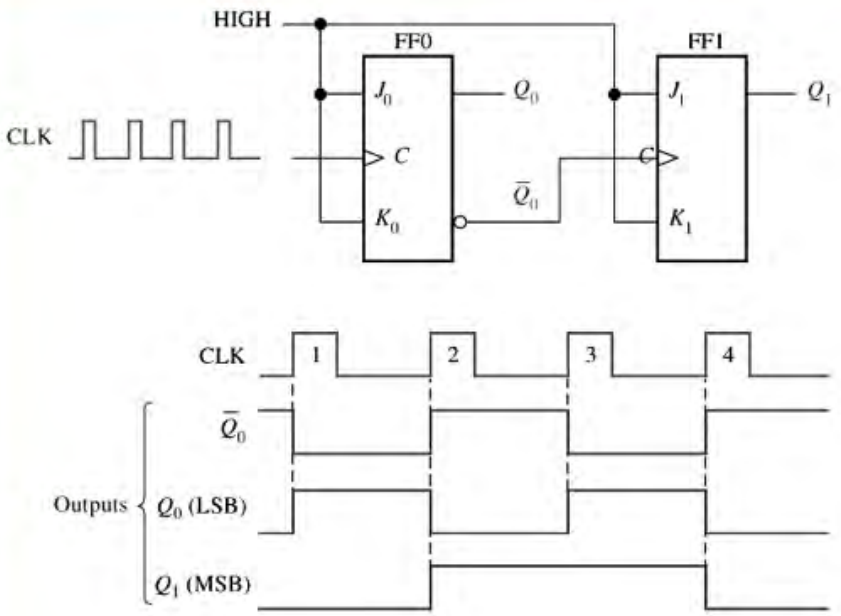
Asynchronous counter operation:

- A two-bit asynchronous counter is shown on the left.
- The external clock is connected to the clock input of the first flip-flop (FF0) only.
- So, FF0 changes state at the positive edge of each clock pulse, but FF1 changes only when triggered by the positive edge of the Q output of FF0.
- Because of the inherent propagation delay through a flip-flop, the transition of the input clock pulse and a transition of the Q output of FF0 can never occur at exactly the same time.
- Therefore, the flip-flops cannot be triggered simultaneously, producing an asynchronous operation.

This asynchronous counter is slow because the cascaded clocking scheme:

- The clock source ripples from stage-to-stage
- The ripple effect is similar to that of a ripple carry adder circuit

Count Up 2-Bit Asynchronous Counter



Notice:

1. Input JK for each FF is tied up together. Therefore J = K = 1 (HIGH). In other word, the JK FF function in a toggle mode that is every clock pulse Q will be complemented.
2. The clock of FF1 come from the complemented output of FF0, $\overline{Q}_0$
3. For FF0, every positive edge of the clock the output , Q_0 will toggle.
4. For FF1, the clock depend on $\overline{Q}_0$ and the output Q_1 will only toggle on the positive edge of $\overline{Q}_0$
5. The output of the circuit counter is read $Q_1 Q_0$

CLOCK PULSE	Q_1 (MSB) (Bit 2)	Q_0 (LSB) (Bit 1)
Initially	0	0
1	0	1
2	1	0
3	1	1
4	0	0

The 2-bit ripple counter circuit above has four different states, each one corresponding to a count value. Similarly, a counter with *n* flip-flops can have 2^n states. The number of states in a counter is known as its mod (modulo) number. Thus a 2-bit counter is a mod-4 counter.

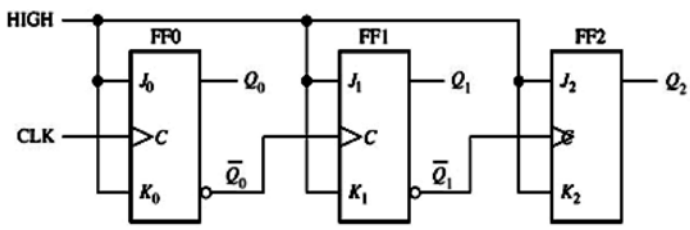
3-Bit Asynchronous Counter

Digital Logic

CLOCK PULSE	$Q_{2(MSB)}$ (Bit 3)	Q_1 (Bit 2)	$Q_{0(LSB)}$ (Bit 1)
Initially	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8	0	0	0

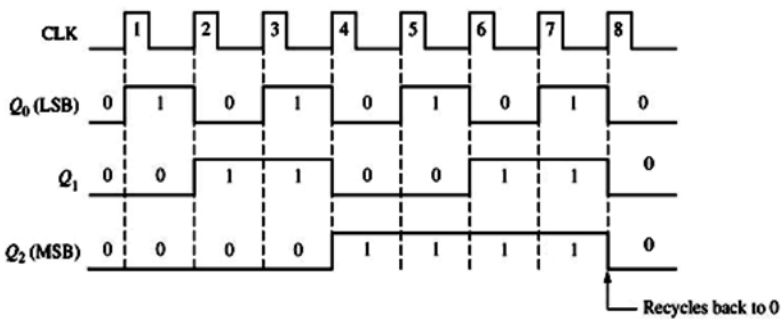
3 bit asynchronous count up counter circuit

- The circuit connection is the same as 2 bit asynchronous counter, for a 3 bit counter just add another flip-flop.



The timing diagram of the 3 bit asynchronous counter.

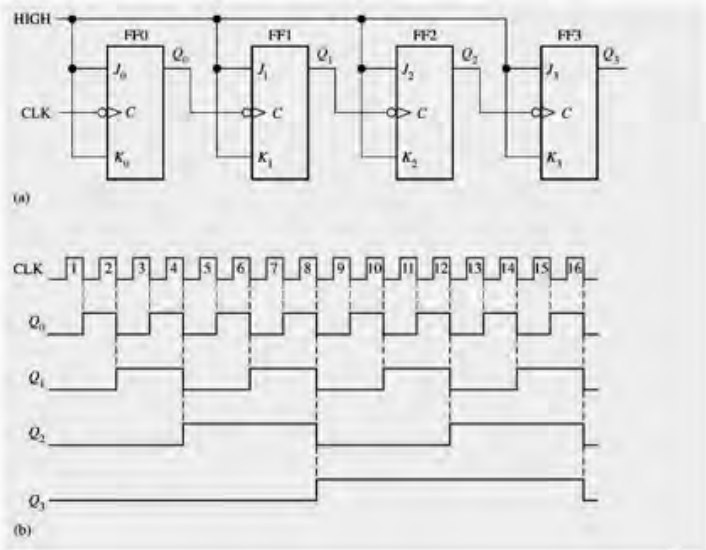
- The count change every positive edge of a clock.
- The counting sequence is 0, 1, 2, 3, 4, 5, 6, 7 and recycle back to 0



4-Bit Asynchronous Counter

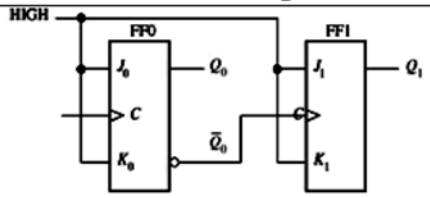
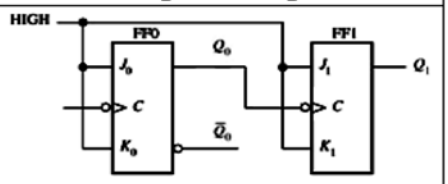
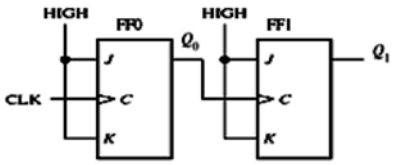
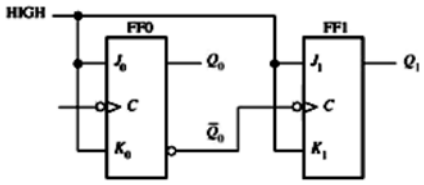
Digital Logic

- The counter is using a negative edge JK FF, but it is still a count up counter because clock is connected to Q.



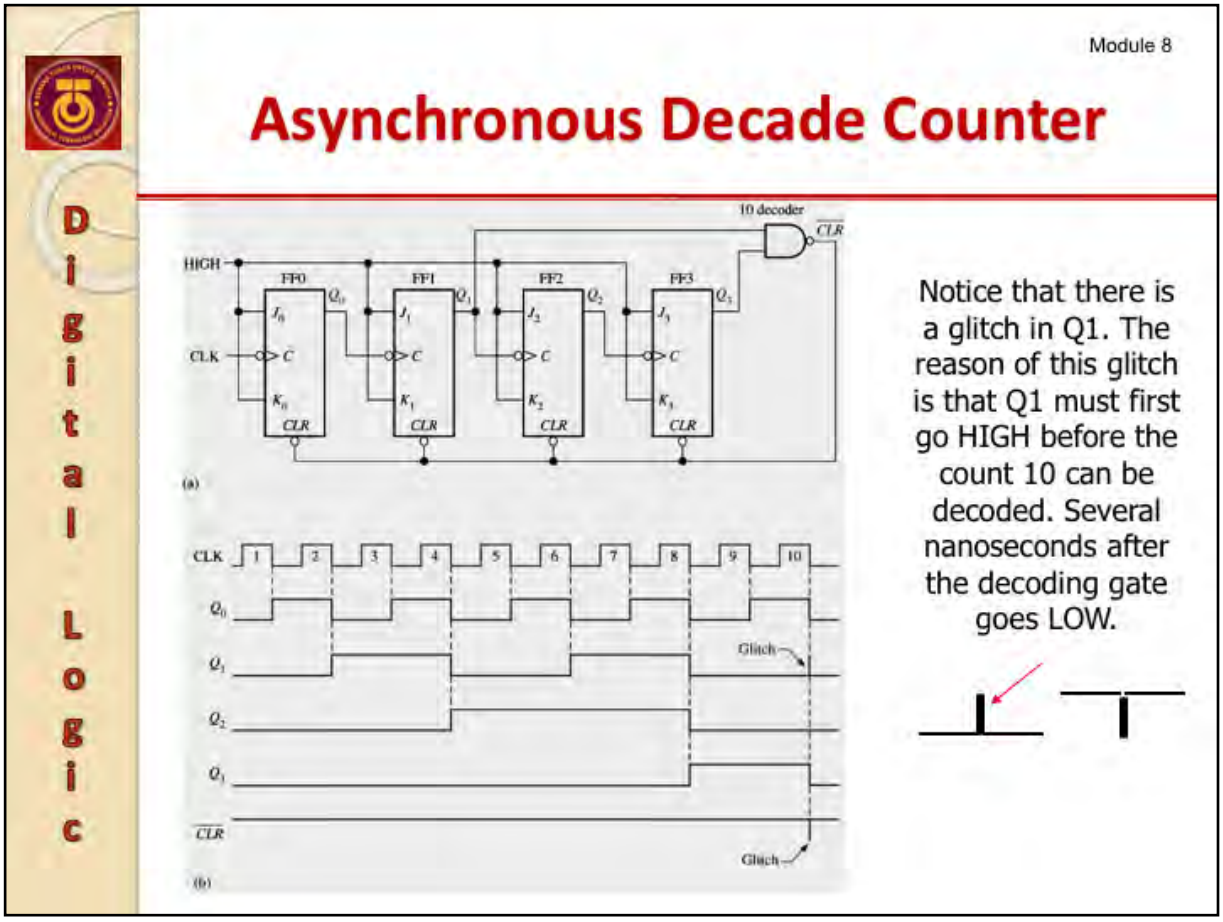
From Floyd

Generalization of an Asynchronous Counter

	Positive Edge	Negative Edge
Count Up		
Count Down		

We can summaries as follows:

1. Basically, the circuit connection for AC is almost the same where the external clock connected to the LSB Flip Flop (FF).
2. Clock for each FF come from the previous FF, except for the first FF (that is FF0)
3. Every FF operates in Toggle mode (i.e. next output is a complement of the previous output).
   - To operate in toggle mod**: for RS:** R = Q' and S = Q, **for JK:** J = K = 1, **for D:** D = Q', **for T:** T = 1
4. The design connection is the same , if we want more bits just add more flip-flops.
5. The difference in connection will be determine whether we want to count UP or DOWN and by using which type of flip-flop – positive edge or negative edge.



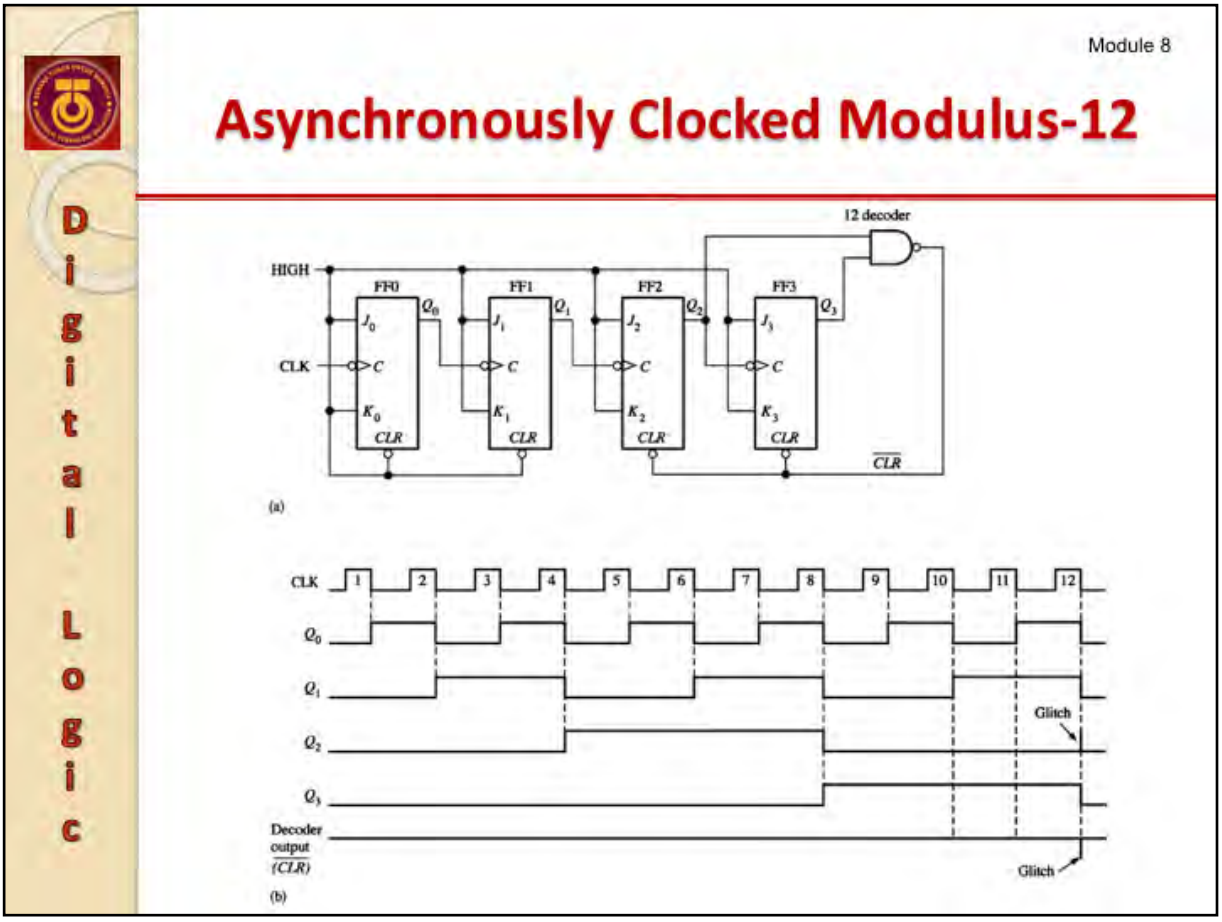
Previously we examined a binary counter that progresses through all of its possible states. Recall that the maximum number of possible states (maximum modulus) of a counter is 2^n, where n is the number of flip-flops in the counter.

Counters can be designed to have a number of states in their sequence that is less than 2^n. The resulting sequence is called a *truncated sequence*. One common modulus for counters with truncated sequences is ten. Such counters are referred to as *decade* counters. Furthermore, a counter with count a sequence of zero (0000) through nine (10001) is referred to as a *BCD decade* counter because its ten state sequence is the BCD code.

To obtain a truncated sequence it is necessary to force the counter to cycle before going through all of its normal states. For example, the BCD decade counter must recycle back to the state 0000 after state 1001. Note that a decade counter will require 4 flip-flops (three flip-flops are insufficient because $2^3 = 8$).

One method that can be used to achieve recycling after the count of nine (1001) is to decode ten (1010) with a NAND gate and connect the output of the NAND gate to the clear inputs of all the flip-flops.

Notice that only Q_1 and Q_3 are connected to the NAND gate inputs. This arrangement is an example of *partial decoding*, in which two unique states (Q_1=1 and Q_3=1) are enough to decode the count of 10. This is because none of the other states (zero through nine) have Q_1 and Q_3 high at the same time. Consequently, when the counter goes into count ten (1010) the decoding gate output goes LOW and asynchronously RESETS all of the flip-flips.



Design of an Asynchronous Counter

Step 1: Determine number of flip-flop required → $\text{Number of flip-flops} = \left\lceil \frac{\log \text{MOD}}{\log 2} \right\rceil$

Step 2: Make a circuit connection

- connect the circuit as ***in Generalization of an Asynchronous Counter***
- The circuit will depend whether count up or down and positive or negative edge triggered flip-flop is used.

Step 1 and 2 is enough for max MOD asynchronous counter, but for a truncated counter an extra step is required

Step 3: Design a decoder to decode the MOD of the truncated sequence

- Use a NAND gate, at the input connect the binary equivalent of the MOD number
  - ✓ E.g. : MOD 12 , need to clear all flip-flop when output =12 ($Q_3Q_2Q_1Q_0$ = 1100)

  - ✓ There is a case where we don't need to decode all the output if there is a unique bit pattern
    - E.g. for MOD 12, need to use only Q_3Q_2 because the counting sequence is 0 to 11 where there is no Q_3Q_2 = 11 in that sequence.

- The output of the NAND gate is connected to all the $\overline{\text{CLR}}$ (CLEAR) input of the flip-flop.

Advantage and Disadvantage of AC (1)

Digital Logic

- Advantage:
  - The design step is simple (easy)
- Disadvantage:
  - Propagated delay accumulate as in ripple binary adder which may cause a missing counting state
    - Especially at high speed (frequency) operation, therefore this kind of counter cannot operates at high frequency.
    - The maximum operating frequency $f_{max} = \frac{1}{Nt_{pd}}$, where N is the number of flip-flop an t_{pd} is the propagation delay of the flip-flop

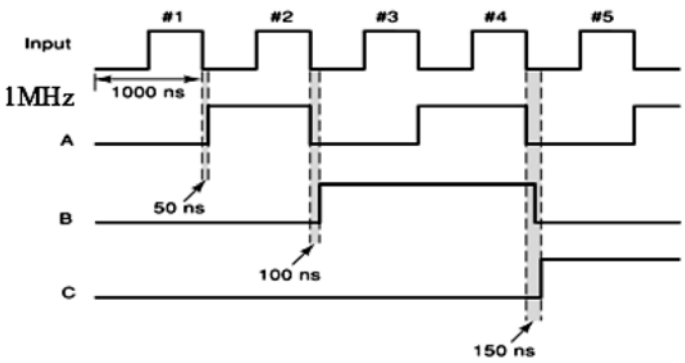
Effect of propagation delay on AC

Assume 3 bits counter using a flip-flop with a propagation delay of 50ns.

Case 1 :

Operating at low frequency of 1 MHz.

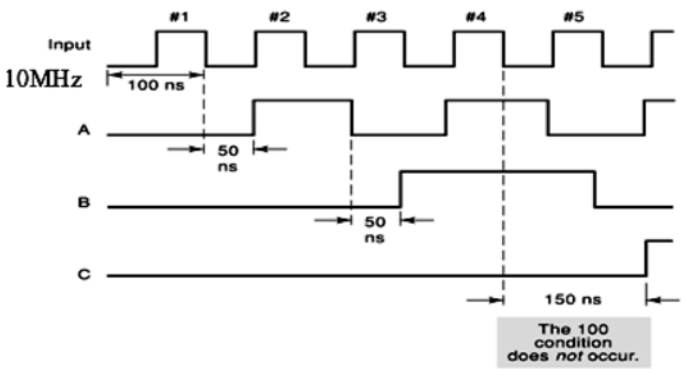
For each stage there is a 50 ns delay and it accumulates up to 150ns at the 3rd stage. The accumulated delay is still lower than the period of the signal which is 1000ns , therefore there is no effect to the counting sequence.



Case 2:

Operating at a higher frequency of 10 MHz.

The clock period is only 100ns. For a 3 stage flip-flop the accumulated delay is the same as before which is 150ns. But, now the accumulated delay is more than the clock period, therefore there is a missing count, 100 doesn't exist!



Advantage and Disadvantage of AC (2)

Digital Logic

Example:

If a propagation delay of a mod 32 asynchronous recycle UP counter using JK FF is 40ns, what is the maximum operating frequency?

Solution: Number of flip - flops $= \left\lceil \frac{\log MOD}{\log 2} \right\rceil$

$$= \frac{\log 32}{\log 2}$$

$$= \frac{\log 2^5}{\log 2}$$

$$= \frac{5 \log 2}{\log 2}$$

$$N = 5$$

$$f_{max} = \frac{1}{Nt_{pd}}$$

$$= \frac{1}{5(40ns)}$$

$$= \frac{1}{5(40 \times 10^{-9} s)}$$

$$f_{max} = 5 \text{ MHz}$$

DIY:

Design a mod 64 Asynchronous recycle UP counter using JK FF. If propagation delay is 30 ms, what is the maximum operating frequency?

Synchronous Counter Design

Digital Logic

- **Step 1**
  Describe the characteristic of the counter that going to be designed
- **Step 2**
  Develop state diagram
- **Step 3**
  Create next state table
- **Step 4**
  Fill in the next state table with flip-flop transition values
- **Step 5**
  By using the present state value, fill in the K-Map with the input of the FF. Use K-Maps to simplify the logic equations
- **Step 6**
  Draw the circuit of the counter and Implement the circuit.

The design of a synchronous sequential circuit starts from a set of specifications and culminates in a logic diagram or a list of Boolean functions from which a logic diagram can be obtained. In contrast to a combinational logic, which is fully specified by a truth table, a sequential circuit requires a state table for its specification. The first step in the design of sequential circuits is to obtain a state table or an equivalence representation, such as a state diagram.

A synchronous sequential circuit is made up of flip-flops and combinational gates. The design of the circuit consists of choosing the flip-flops and then finding the combinational structure which, together with the flip-flops, produces a circuit that fulfils the required specifications. The number of flip-flops is determined from the number of states needed in the circuit.

F/F Excitation Table

Digital Logic

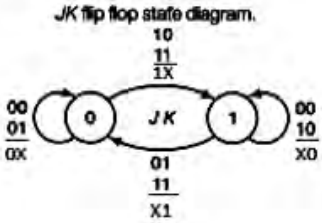
- While designing synchronous counter excitation table is needed as in Step 4.
- Following is an excitation table for JK Flip-Flop

FF State		Present State	Next State
J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

≡

Present State	Next State	FF State	
Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

JK flip flop state diagram.



S-R Excitation Table

FF State		Present State	Next State
S	R	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	Invalid
1	1	1	invalid

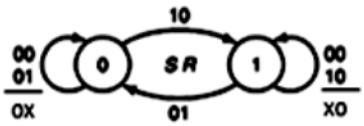
≡

Present State	Next State	FF State	
Q_n	Q_{n+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

S-R Truth Table

S	R	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	-

SR flip flop state diagram.



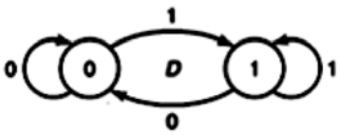
D Excitation Table

FF State	Present State	Next State
D	Q_n	Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

D Truth Table

D	Q_{n+1}
0	0
1	1

D flip flop state diagram..



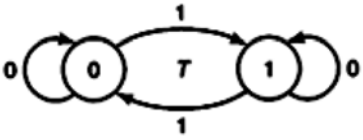
T Excitation Table

FF State	Present State	Next State
T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

T Truth Table

T	Q_{n+1}
0	Q_n
1	$\overline{Q_n}$

T flip flop state diagram.

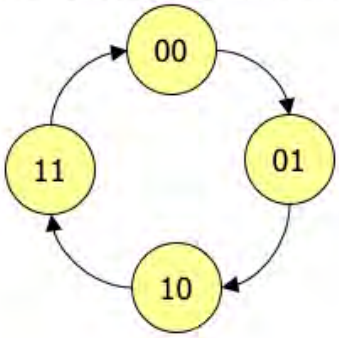


Design 2-Bit Synchronous Counter

Digital Logic

- **Step 1**:
  Design 2-bit synchronous recycle count-up counter using JK FF.

- **Step 2**: Draw state diagram



- **Step 3:** Create next state table

Present State		Next State	
$Q1_n$	$Q0_n$	$Q1_{n+1}$	$Q0_{n+1}$
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

Note on State Table

A state diagram describes circuit functionality, but does not describe circuit implementation

- Translation to a tabular form is necessary
- The state table should contain
  - ✓ All transitions from each present state to each next state for all variations of the input signals
  - ✓ Example: The counter has three states A, B and C. There is one input that control the counting sequence which is W.
  - ✓ If W = 0, the state sequence are A → C → B → A → C → B → A … recycle
  - ✓ If W = 1, the state sequence are A → B → C → A → B → C → A … recycle

Present state	Next state	
	w=0	w=1
A	C	B
B	A	C
C	B	A

… 2-Bit Synchronous Counter

Digital Logic

- **Step 4:** Fill in the **flip-flop transition table**

Present State		Next State		JK Transition			
$Q1_n$	$Q0_n$	$Q1_{n+1}$	$Q0_{n+1}$	J1	K1	J0	K0
0	0	0	1	0	X	1	X
0	1	1	0	1	X	X	1
1	0	1	1	X	0	1	X
1	1	0	0	X	1	X	1

Note:

- While filling in the next state table, please refer to the excitation table.

Present State	Next State	FF State	
Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

- Constructing the next-state table depends on the type of flip-flop (D, T, JK) used for the implementation:
  - ✓ For JK flip-flop every transition we need to determine two value which is J and K.
  - ✓ For D and T every transition we need to determine only a single value for D or T.

… 2-Bit Synchronous Counter

Digital Logic

- **Step 5**: Use K-maps to simplify the logic equations.

Q1 \ Q0	0	1
0	1	X
1	1	X

J0 = 1

Q1 \ Q0	0	1
0	0	1
1	X	X

J1 = Q0

Q1 \ Q0	0	1
0	X	1
1	X	1

K0 = 1

Q1 \ Q0	0	1
0	X	X
1	0	1

K1 = Q0

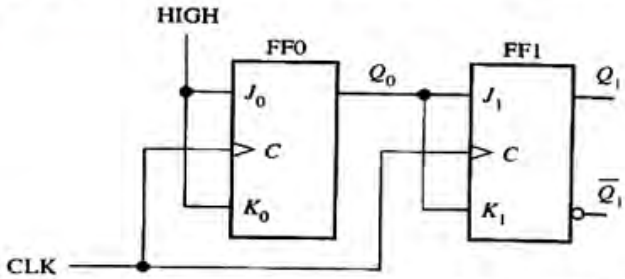
Note:

- K-maps are constructed from the state table to simplify the circuit and determine input equation for the FF.
- While mapping to the K-Map, refer to the Present state column not the Next state column of the state table.
- If the implementation is using a JK flip-flop there will be twice the number of K-Maps compared to using a D or T flip-flop, this is because JK flip-flop has two input while the D and T flip-flop has only a single input.

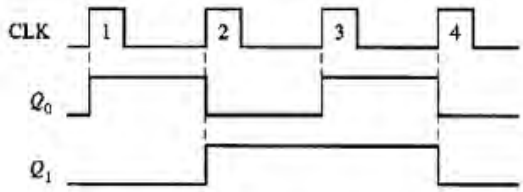
… 2-Bit Synchronous Counter

Digital Logic

- **Step 6**: Draw the circuit of the counter and Implement the circuit.



- The timing diagram of the implemented the circuit that shows the counting sequence from 0, 1, 2, 3, 0 (recycle)



- From the timing diagram, we can read the output of the counter as Q_1Q_0

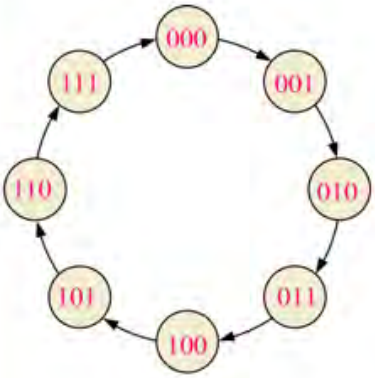
initial value	- 00	which is 0	has 4 unique state (MOD 4)
during Clock 1	- 01	which is 1	
during Clock 2	- 10	which is 2	
during Clock 3	- 11	which is 3	
during Clock 4	- 00	which is 0 -- it repeat again (recycle)	

- Note that
  - ✓ the changes of state occurs at the positive edge of the clock
  - ✓ Q_0 always change/TOGGLE every positive edge of the clock when $J_0 = K_0 = 1$ (HIGH)
  - ✓ Q_1 will only change/TOGGLE if Q_0 before the positive edge is 1 (HIGH), because this will make $J_1 = K_1 = 1$ (HIGH)

Design 3-bit Synchronous Counter

Digital Logic

- State diagram



- Next State table

Note: + mean next state

Present State			Next State		
Q2	Q1	Q0	$Q2_+$	$Q1_+$	$Q0_+$
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

Next State Transition Table

Present State			Next State			JK Transition Table					
Q2	Q1	Q0	$Q2_+$	$Q1_+$	$Q0_+$	J2	K2	J1	K1	J0	K0
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	1	1	1	X	0	X	0	1	X
1	1	1	0	0	0	X	1	X	1	X	1

… 3-bit Synchronous Counter

Digital Logic

- Create K-map to determine the Boolean expression

Q2 \ Q1Q0	00	01	11	10
0	0	0	1	0
1	X	X	X	X

J2 = Q1Q0

Q2 \ Q1Q0	00	01	11	10
0	0	1	X	X
1	0	1	X	X

J1 =

Q2 \ Q1Q0	00	01	11	10
0	1	X	X	1
1	1	X	X	1

J0 =

DIY: Fill in the K-map and get the simplified K inputs of each JK FF.

Q2 \ Q1Q0	00	01	11	10
0	X	X	X	X
1	0	0	1	0

K2 =

Q2 \ Q1Q0	00	01	11	10
0	X	X	1	0
1	X	X	1	0

K1 =

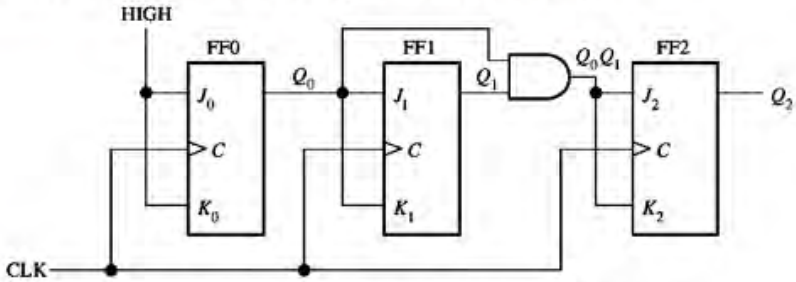
Q2 \ Q1Q0	00	01	11	10
0	X	1	1	X
1	X	1	1	X

K0 =

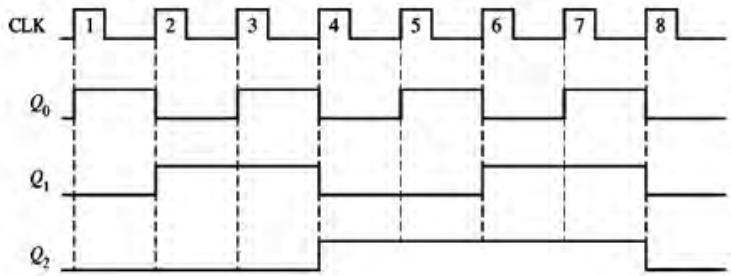
… 3-bit Synchronous Counter

Digital Logic

- The circuit diagram of 3-bit synchronous counter



- The timing diagram of 3-bit synchronous counter



Note:

Actually we can design a synchronous counter by observing its timing diagram, based on the above timing diagram:

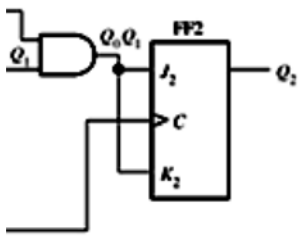
- Q_0 always toggle. To achieve this, we have to operate FF0 in toggle mode by connecting J_0 and K_0 to HIGH
- Q_1 goes to opposite state every time Q_0 is HIGH (at the positive edge of clock CLK2, CLK4,CLK6 and CLK8).
  - ✓ From the observation we can achieve this by connecting Q_0 to the J_1 and K_1 inputs of FF1
- Q_2 changes state when both Q_0 and Q_1 is HIGH. (at the positive edge of clock CLK4 and CLK8).
  - ✓ From the observation we can achieve this by 'AND'ing Q_0 and $Q_1$to the J_2 and K_2 inputs of FF2

FF2	FF1	FF0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

(recycles)

- The most important advantage of synchronous counters is that there is no cumulative time delay because all flip-flops are triggered in parallel. Thus, the maximum operating frequency for this counter will be significantly higher than for the corresponding ripple counter.

- The maximum operating frequency, f_{max} for the above circuit is only limited by the summation of the AND gate and flip-flop propagation delay



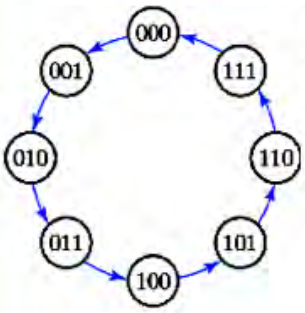
$$f_{max} = \frac{1}{\text{total gate delay} + \text{delay of a single flip-flop}}$$

3-Bit Synchronous Counter

Digital Logic

Design a 3-bit synchronous recycle up counter using T FF.

State Diagram



Next State Table

Present State			Next State			Flip-Flop Inputs		
A_2	A_1	A_0	A_2	A_1	A_0	T_{A2}	T_{A1}	T_{A0}
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

K-map for T_{A0}, T_{A1} and T_{A2}.

A_2 \ A_1A_0	00	01	11	10
0			1	
1			1	

$$T_{A2} = A_1A_0$$

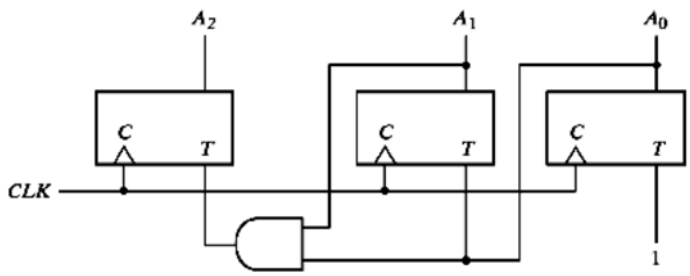
A_2 \ A_1A_0	00	01	11	10
0		1	1	
1		1	1	

$$T_{A1} = A_0$$

A_2 \ A_1A_0	00	01	11	10
0	1	1	1	1
1	1	1	1	1

$$T_{A0} = 1$$

The circuit diagram for 3-bit synchronous counter: T_{A0}, T_{A1} and T_{A2}



In this example, the K-Map and the circuit diagram is different from the previous examples.

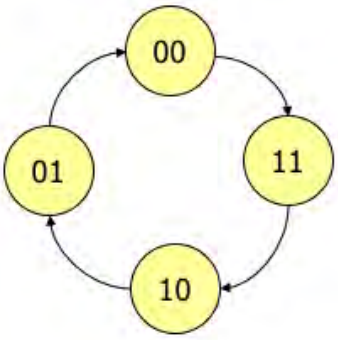
The circuit diagram is drawn in such a way that the MSB is on the left side of the circuit. So that, it can naturally be read with a proper value. (i.e. MSB …..LSB).

2-Bit Down Synchronous Counter

Digital Logic

- **Step 1**:
  To design 2-bit down recycle synchronous counter using D FF.

- **Step 2**: Draw state diagram



- **Step 3**: Create next state table

Present State		Next State	
$Q1_n$	$Q0_n$	$Q1_{n+1}$	$Q0_{n+1}$
0	0	1	1
0	1	0	0
1	0	0	1
1	1	1	0

Step 4: Draw flip-flop transition table. For D FF, D input is the same as the next state values.

Present State		Next State		D Transition	
$Q1_n$	$Q0_n$	$Q1_{n+1}$	$Q0_{n+1}$	D1	D0
0	0	1	1	1	1
0	1	0	0	0	0
1	0	0	1	0	1
1	1	1	0	1	0

Step 5: Use the K-Map to simplify the equation

Q1 \ Q0	0	1
0	1	0
1	0	1

D1 = Q1′Q0′ + Q1Q0

Q1 \ Q0	0	1
0	1	0
1	1	0

D0 = Q0′

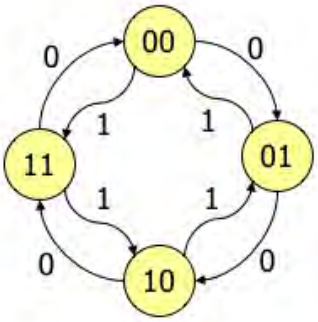
DIY:

Step 6: Draw the circuit

Up-Down Synchronous Counters

Digital Logic

- **Step 1**:
  To design 2-bit up down synchronous counter using D FF based on input mode selector X. When X = 0, it is up counter and down when X = 1.
- **Step 2:** Given the state diagram



- **Step 3:** Create next state table

	Input, X	Present State		Next State	
		$Q1_n$	$Q0_n$	$Q1_{n+1}$	$Q0_{n+1}$
Count up	0	0	0	0	1
	0	0	1	1	0
	0	1	0	1	1
	0	1	1	0	0
Count down	1	0	0	1	1
	1	0	1	0	0
	1	1	0	0	1
	1	1	1	1	0

- Up–down counter is a bidirectional counter that can counts both up and down
  - ✓ The count direction is controlled by an input , as X in the above example
    - If X =1 , the counter will count down
    - If X = 0, the counter will count up
  - ✓ During the operation, the value of X can be changed, and the effect of changing the value will occur at the next clock edge

.. Up-Down Synchronous Counters

Digital Logic

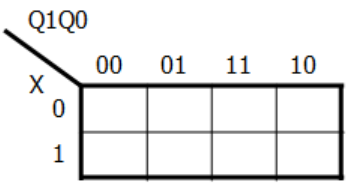
- **Step 4 and 5**:
  Fill in the D1 and D0 by referring to D FF transition table and determine Boolean expression.

Input, X	Present State		Next State		D FF	
	$Q1_n$	$Q0_n$	$Q1_{n+1}$	$Q0_{n+1}$	D1	D0
0	0	0	0	1	0	1
0	0	1	1	0	1	0
0	1	0	1	1	1	1
0	1	1	0	0	0	0
1	0	0	1	1	1	1
1	0	1	0	0	0	0
1	1	0	0	1	0	1
1	1	1	1	0	1	0

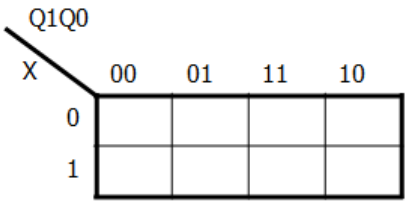
- When mapping to the K-Map , X is one of the input besides the value of Q1and Q0 from the present state column

DIY:

Fill in the K-Map and determine the simplified expression for D1 and D0.



D1 =



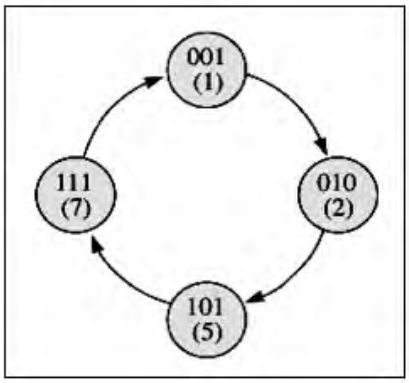
D0 =

- **Step 6**: Draw the circuit diagram for the 2-bit up-down synchronous counter

Counter for Arbitrary Sequences

Digital Logic

- Design the counter base on the given state diagram using T FF.



From the state diagram, the counter is a 3 bit counter.

For a 3 bits counter, the number of states should be 2^3 which is 8.

But, in the state diagram there are only 4 states (called **valid state**).

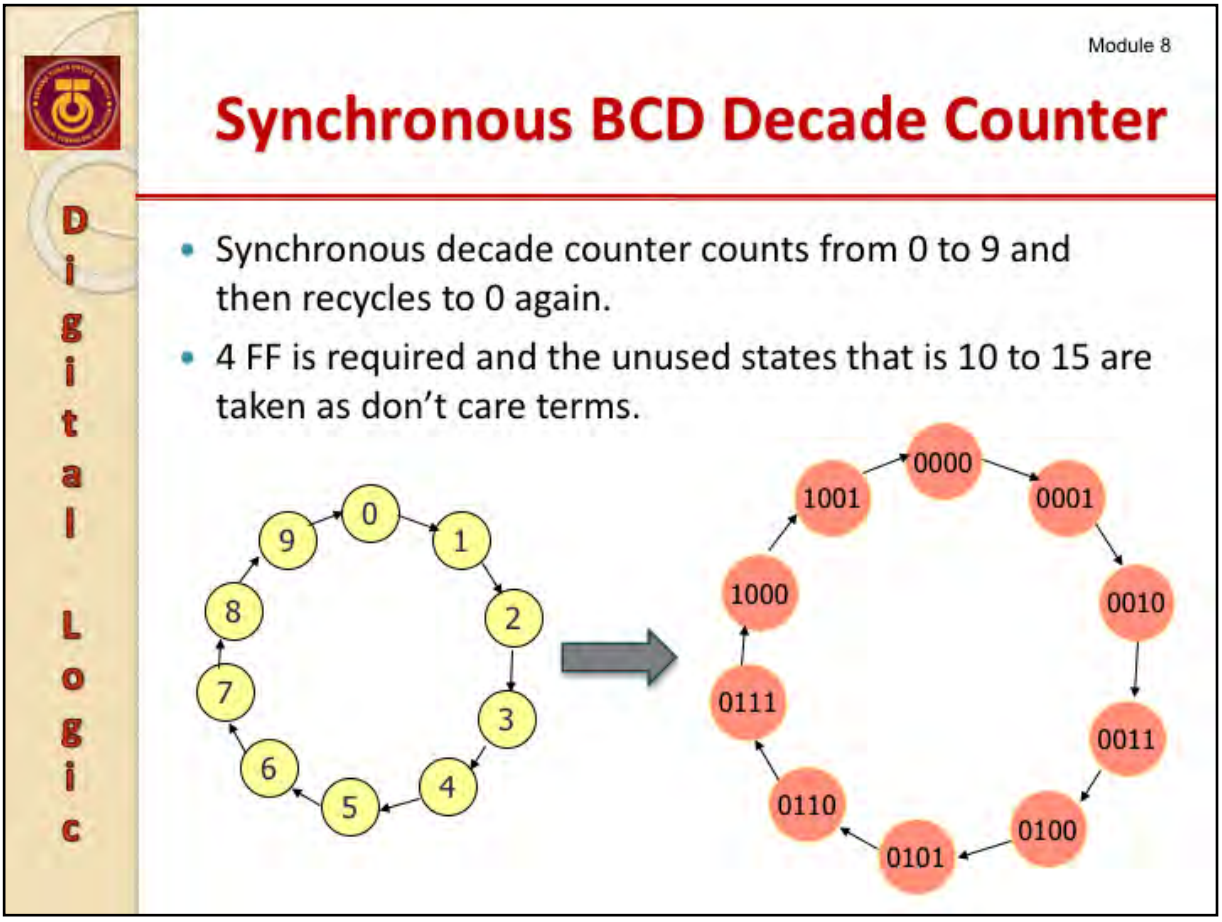
The other 4 states which is 0,3,4 and 6 that does not appear in the state diagram is called the **invalid state**.

Usually we assume an invalid state will never occurs and treat it as don't care, and plotted as X in the next state table as in the first row of the table below. The input to the FF also considered as a don't care.

State table and T FF transition table.

Present State			Next State			T FF		
$Q2_n$	$Q1_n$	$Q0_n$	$Q2_{n+1}$	$Q1_{n+1}$	$Q0_{n+1}$	T2	T1	T0
0	0	0	X	X	X	X	X	X
0	0	1	0	1	0	0	1	1
0	1	0						
0	1	1						
1	0	0						
1	0	1						
1	1	0						
1	1	1						

DIY : Derive the Boolean expression and draw the circuit diagram.



The maximum possible number of states of a counter is 2^n, where n is the number of flip-flops in the counter. Regular binary counters have maximum modulus; they progress through all of their possible states. Counters can also be designed to have a number of states that is less than 2^n . The resulting sequence is called truncated sequence.

A very common modulus for counters with truncated sequences is 10. Counters with 10 states in their sequence is called decade counters.

A BCD counter or decade counter can be constructed from a binary counter by terminating the "ripple-through" counting when the count reaches decimal 9 (binary 1001).

.. Synchronous BCD Decade Counter

Digital Logic

Present State				Next State				T FF Transition			
Q3	Q2	Q1	Q0	$Q3_+$	$Q2_+$	$Q1_+$	$Q0_+$	T3	T2	T1	T0
0	0	0	0	0	0	0	1	0	0	0	
0	0	0	1	0	0	1	0	0	0	1	
0	0	1	0	0	0	1	1	0	0	0	
0	0	1	1	0	1	0	0	0	1	1	
0	1	0	0	0	1	0	1	0	0	0	
0	1	0	1	0	1	1	0	0	0	1	
0	1	1	0	0	1	1	1	0	0	0	
0	1	1	1	1	0	0	0	1	1	1	
1	0	0	0	1	0	0	1	0	0	0	
1	0	0	1	0	0	0	0	1	0	0	
1	0	1	0	X	X	X	X	X	X	X	
1	0	1	1	X	X	X	X	X	X	X	
1	1	0	0	X	X	X	X	X	X	X	
1	1	0	1	X	X	X	X	X	X	X	
1	1	1	0	X	X	X	X	X	X	X	
1	1	1	1	X	X	X	X	X	X	X	

DIY: Fill in column T0 and the K-maps

Q3Q2 \ Q1Q0	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	X	X	X	X
10	0	0	X	X

T3 =

Q3Q2 \ Q1Q0	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	X	X	X	X
10	0	0	X	X

T2 =

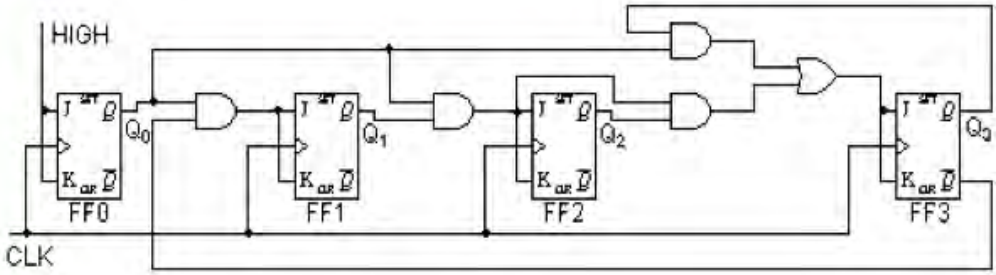
Q3Q2 \ Q1Q0	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	X	X	X	X
10	0	0	X	X

T1 =

Q3Q2 \ Q1Q0	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	X	X	X	X
10	1	1	X	X

T0 =

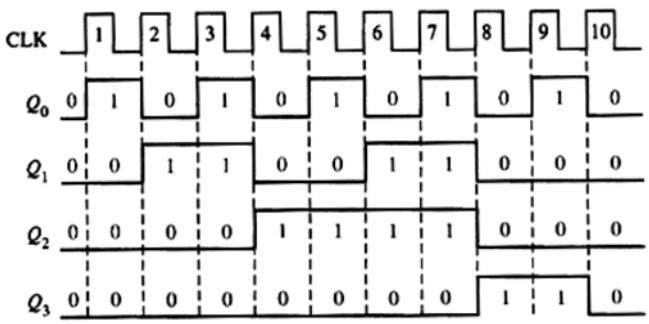
.. Synchronous BCD Decade Counter



- Q_0 toggles on each clock pulse.
- Q_1 changes on the next clock pulse each time Q_0=1 and Q_3=0.
- Q_2 changes on the next clock pulse each time Q_0=Q_1=1.
- Q_3 changes on the next clock pulse each time Q_0=1, Q_1=1 and Q_2=1 (count 7), or then Q_0=1 and Q_3=1 (count 9).

- The circuit above is using a JK flip-flop. But each flip-flop J is connected to K so that J = K which is equivalent to a T flip-flop.
- An exercise from the previous slide ask you to implement the circuit using T flip-flop, Do you get the same answer?

- The timing diagram of the above circuit



Clock Pulse	Q3	Q2	Q1	Q0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

From Floyd

Recap… Modulus

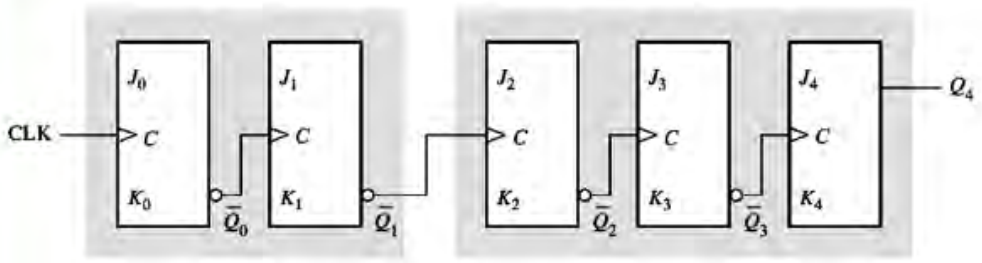
Digital Logic

- The modulus is the number of unique states through which the counter will sequence.
- Maximum possible number of states of counter is 2^n, n is the number of flip-flops in the counter.
  - Example : Modulus 8 = 2^3 (Need 3 flip flops)
- Counter can be designed to have a number of states in their sequence that is less than maximum, 2^n. This is called a truncated sequence.
- One common modulus for counters in truncated sequences is ten (Modulus 10).
  - It is called BCD or decade counters (as explained in previous slides).

Cascaded Counter

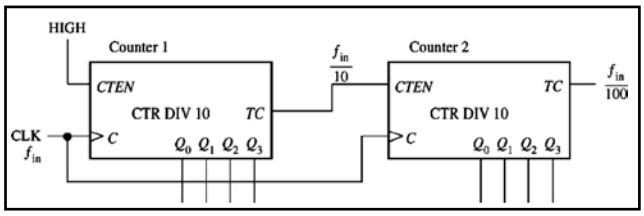
Digital Logic

- Counter can be connected to achieve higher modulus operation.
- Cascading means that the last-stage output of one counter drives the input of the next counter.
- Effect of cascading a counter is that it total MOD is a product of multiplying each MOD of the counter
  - Example: Two counters, modulus-4 and modulus-8 connected in cascade, will produce a MOD 32 (that is 8 x 4) counter
- Example: Two counters, modulus-4 and modulus-8 connected in cascade, can achieve count until 32 CLK (modulus-32 (that is 4 × 8)).



Modulus-4 counter Modulus-8 counter

- Modulus-100 counter can be build by using two cascaded decade (MOD10) counters.
- This counter can be viewed as a frequency divider.
  - ✓ A counter with MOD *N* will divide the frequency by *N*
- For a modulus 100 counter It will divides the input clock frequency by 100.



- Counter can also be used as a frequency divider.

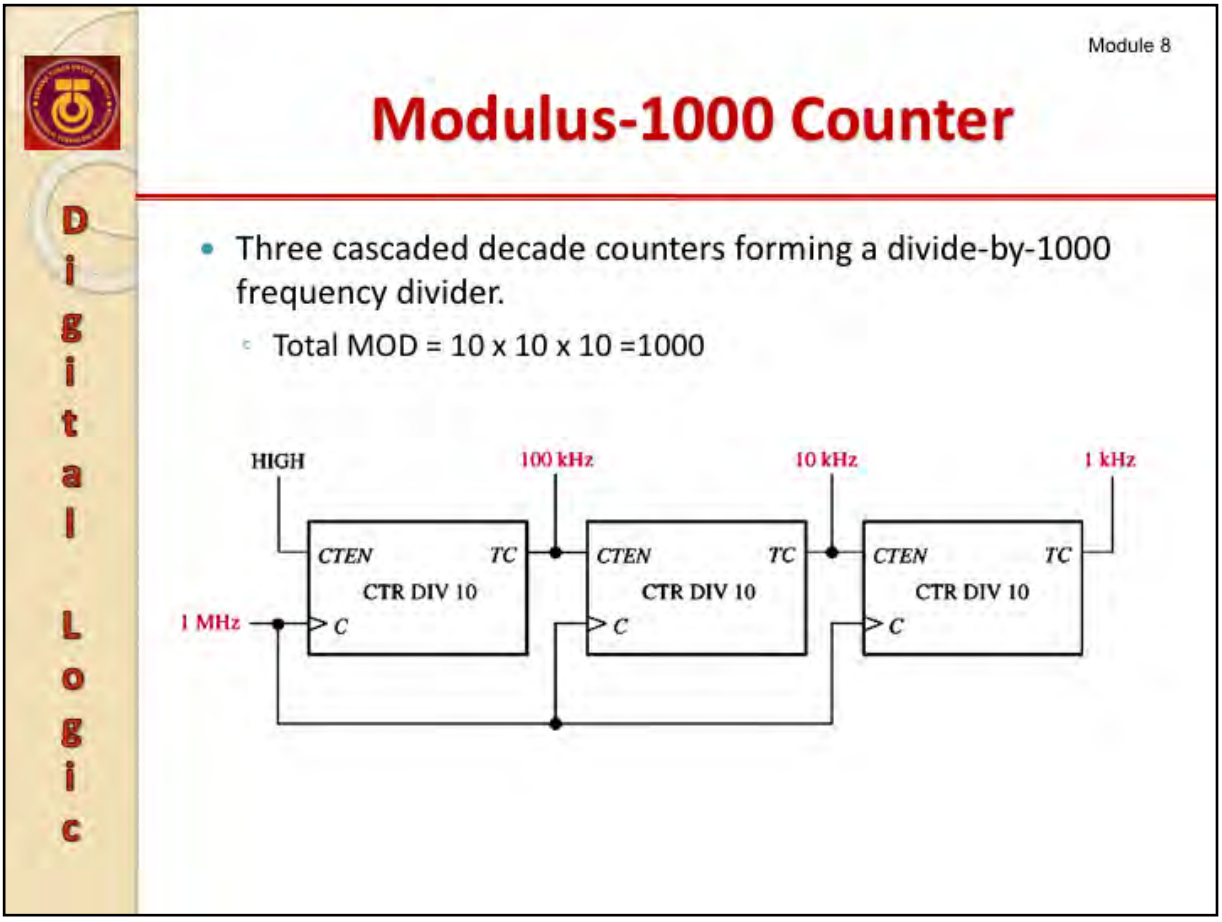
- Counter with MOD *n* will divide the input clock frequency by *n*. From the above figure, Counter 1 and Counter 2 will divide the input clock by 10. If we cascade both counter we will end up with mod 100 counter which will divide the clock (f_{in}) by 100.

- *CTEN = Counter Enable*

  If *CTEN* =1 then the counter will count in a proper sequence i.e. 0, 1, ..., 9 for MOD 10 counter (CTR DIV 10)

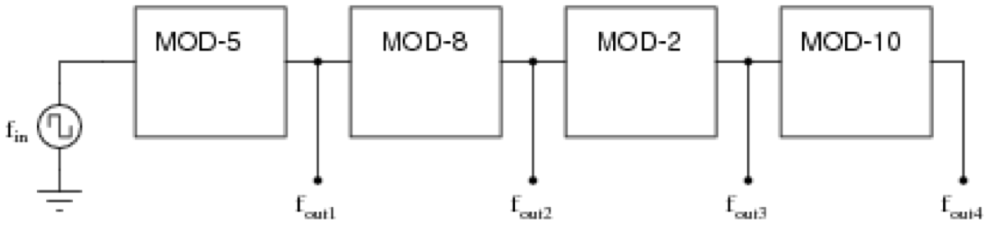
- *TC=Terminal Count*

  When the counter reaches the max count, TC will be equal to 1. This terminal is use to enable Counter 2 so that the next clock cycle will increment the count in Counter 2.



- If given a clock with a frequency of 1 MHz and you wish to obtain 100KHz, 10KHz, and 1KHz, a series of cascaded decade counters can be used as in the above figure
    - Divide the 1 MHz signal by 10, the output is 100KHz. Then divide the 100 KHz signal by10, the output is 10KHz. Further division by 10 gives the 1 KHz frequency.

- When counters are used as frequency dividers, they are often drawn as simple boxes with one input and one output each, like this:



DIY:

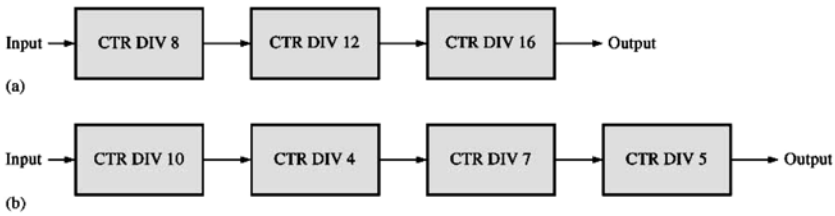
1. Calculate the four output frequencies (f_{out1} through f_{out4}) given an input frequency of 25 kHz:

   Answer: f_{out1} = 5 kHz, f_{out2} = 625 Hz, f_{out3} = 312.5 Hz, f_{out4} = 31.25 Hz

2. Find:

   (a) Overall Modulus of (a)

   (b) Overall Modulus of (b)



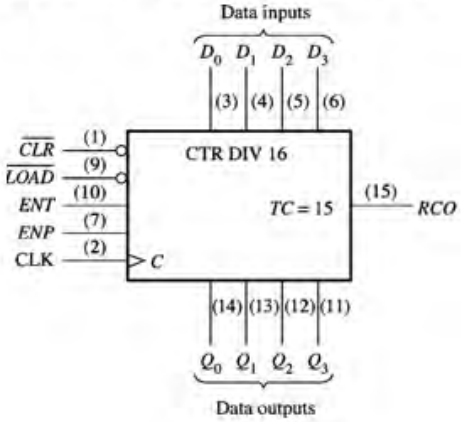
   (c) How many cascaded counter are required to divide a clock frequency by 100,000?

Module 8

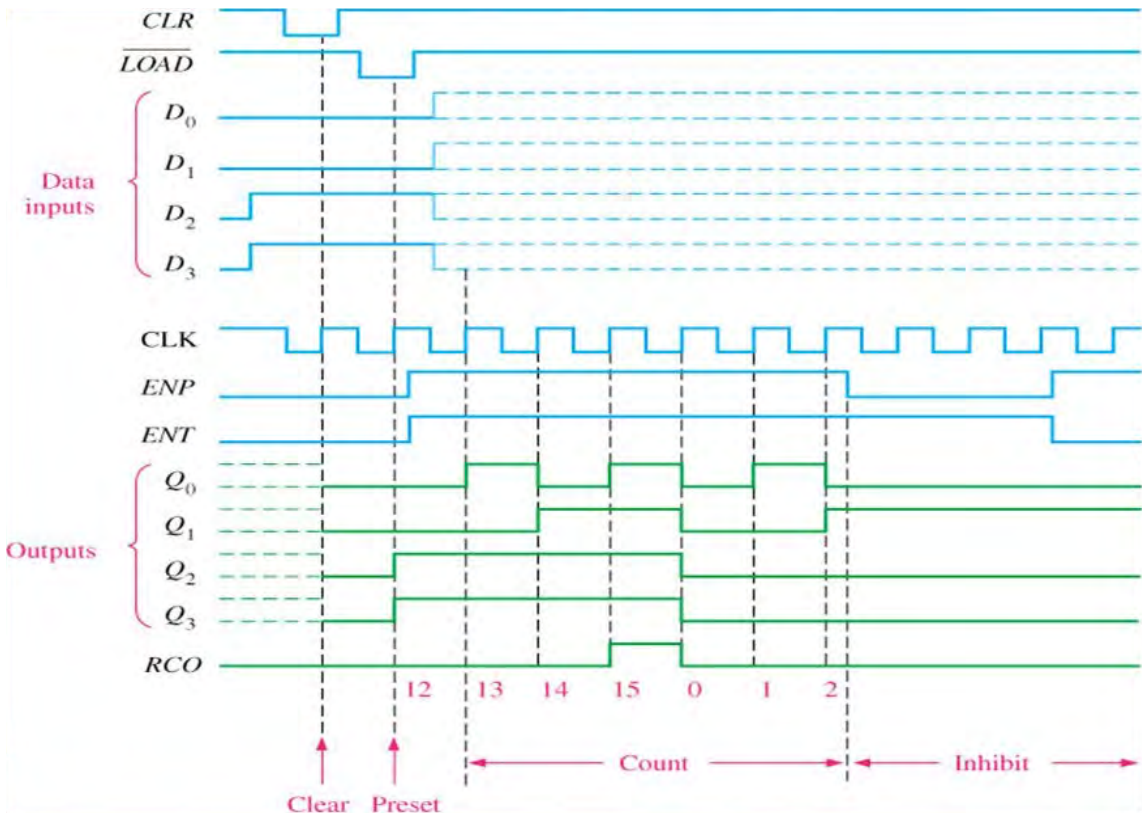
74HC163:
4-Bit Synchronous Binary Counter

Digital Logic

- There is a dedicated counter IC, e.g 74HC163 which is a MOD 16 counter IC
- The starting counting sequence can be change by setting the initial value at $D_3D_2D_1D_0$
  - To load the initial value, LOAD' must be 0



The timing diagram to highlight the function of 74HC163 pins

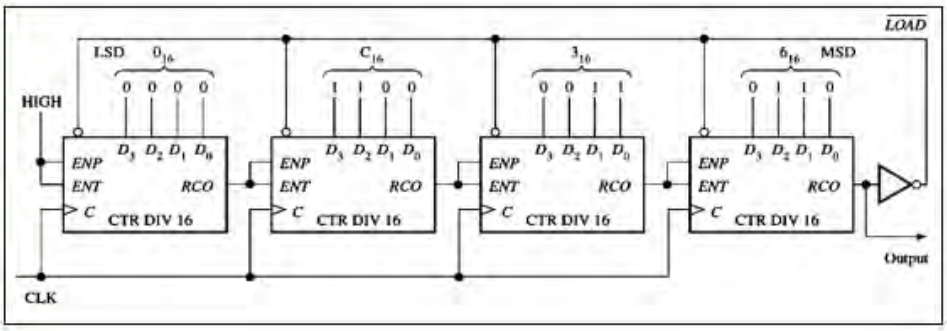


Cascaded Counter with Truncated Sequence

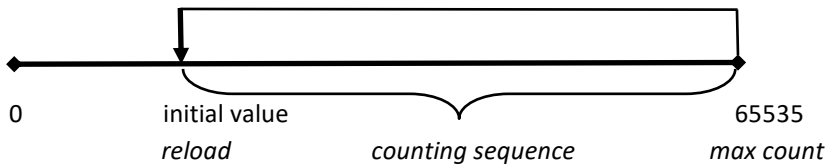
Module 8

Digital Logic

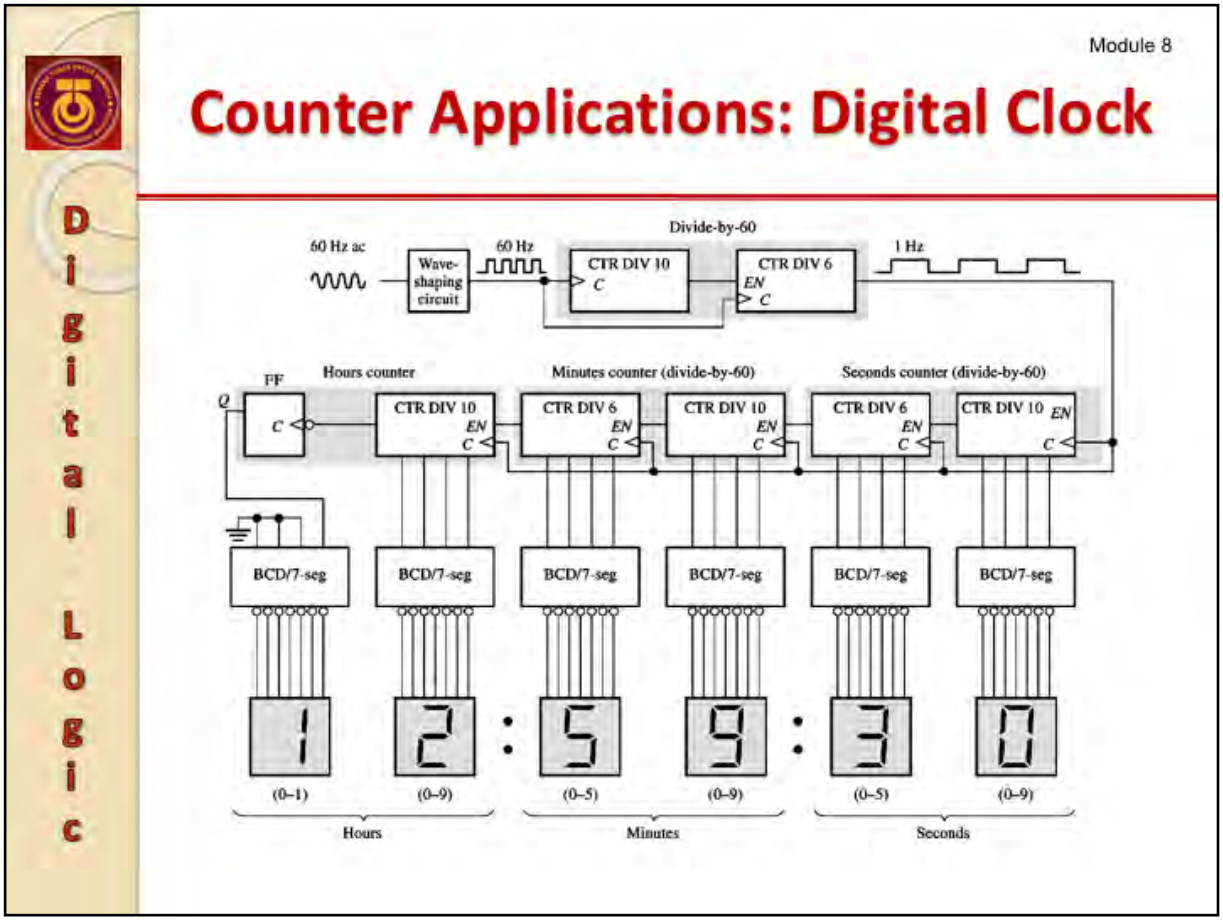
- The cascaded counter below has 16^4 = 65,536 states (full modulus for four cascaded CTR DIV 16).
- If we need a modulus 40,000 counter only. So, how?
  1. Determine the initial value : 65536-40000 = 25536 ($63C0_{16}$) make the counter starts at this count.
  2. Therefore preset the cascaded counter to $63C0_{16}$ by setting the value of $D_3D_2D_1D_0$ as shown below.



- An IC 74HC163 is a 4 bit synchronous binary counter,which is a MOD 16 counter
- This IC can be use to produce a bigger MOD by cascading the IC
  - ✓ If the IC is cascaded as the figure above it will produce a MOD of 16 ×16 × 16 × 16 or 16^4 = MOD 65536.
  - ✓ It counting sequence is 0, 1, 2, 3 …. 65534, 65535
  - ✓ The ENP and ENT is an active HIGH enable pin and must be connected to HIGH to enable the IC.
  - ✓ At the LSB stage , the ENP and ENT always connected to HIGH, the other stage this input is connected to the output RCO from the previous stage.
  - ✓ RCO is a Ripple Carry Out, which will be HIGH if the counter achieve the maximum count, which is 15 for this IC. The output of RCO is use to enable the next stage so that it can increment it count when the previous stage has a maximum count
- All counter is preloaded with an initial value. This initial value will increment to the maximum count, at this point the RCO of the MSB will be 1 and inverted to become 0 and send to an active LOW LOAD' pin. This will reload the initial value to the counter and the counting sequence repeats.



- Previously, the design of a truncated counter is by limiting the maximum count
  - ✓ Example: Decade counter – limit the maximum count to 9 rather than 15
- In the above design, the maximum count is not limited but the minimum count is varied so that the number of state (MOD) changed.



The circuit shows one of the application of a counter in digital clock.

The source frequency of 60Hz is from the wall socket. It is not a square wave , a wave shaping circuit will convert that signal to a 60Hz square wave.

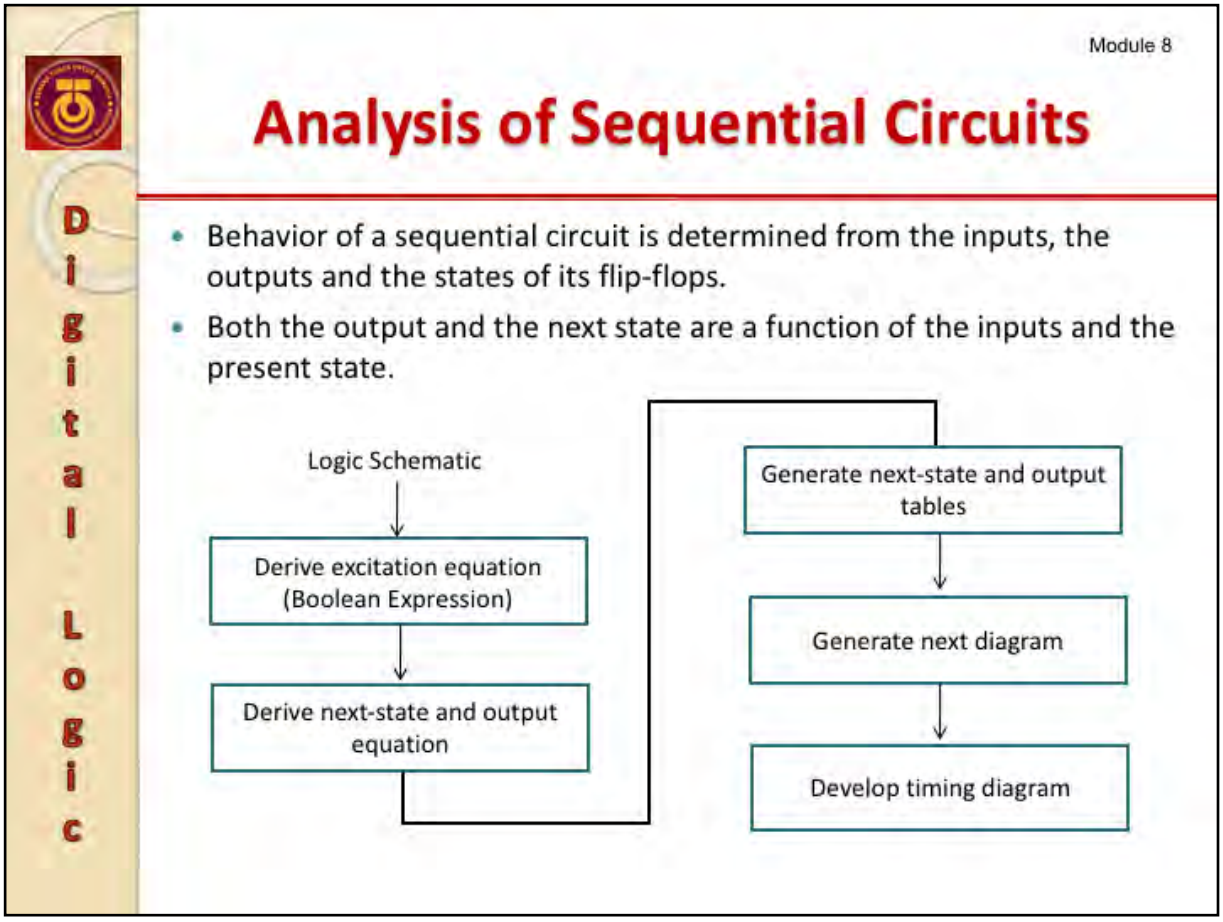
Seconds Counter : The 60Hz square wave is divided by 60 by MOD 60 counter to get a 1Hz (which change every second) square wave – this will be use as a clock to a second counter.

Minutes Counter: To provide a clock for a minute counter the 1 Hz square wave is divided by another cascaded counter with MOD 60 (mod 10 × mod 6).

Hours Counter: The clock frequency of a minute counter is divided by another MOD 60 counter.

The clock division will make sure that the an hour will be incremented by one every 60 minutes and a minutes is incremented after every 60 seconds.

To display the value of the counter a BCD to 7 Segment decoder is used. The decoder will drive a 7-segment display and display the time.



Step 1:
Start with the logic schematic from which we can derive excitation equations for each flip-flop input.

Step 2:
To obtain next-state equations, we insert the excitation equations into the characteristic equations. The output equations can be derived from the schematic.

Flip-flop characteristic equation:

$$\begin{aligned} \text{active HIGH SR}: Q_{next} &= S + \overline{R}Q, SR = 0 \\ JK: Q_{next} &= J\overline{Q} + \overline{K}Q \\ D: Q_{next} &= D \\ T: Q_{next} &= T\overline{Q} + \overline{T}Q \end{aligned}$$

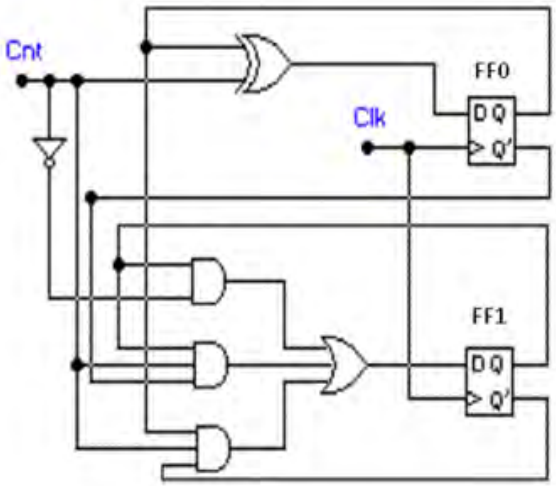
Step 3:
Once we have our output and next-state equations, we can generate the next-state and output tables as well as state diagrams.

Step 4:
When we reach this stage, we use either the table or the state diagram to develop a timing diagram which can be verified through simulation.

Method 1

Digital Logic

- Counter is a sequential circuit. The steps in analyzing the counter can be use to analyze any sequential circuit.
- Analyze and determine the characteristic of the counter:



Step 1: Boolean expressions for the inputs of each flip-flops in the schematic.

$$D0 = Cnt \oplus Q0 = \overline{Cnt} \bullet Q0 + Cnt \bullet \overline{Q0}$$

$$D1 = \overline{Cnt} \bullet Q1 + Cnt \bullet \overline{Q1} \bullet Q0 + Cnt \bullet Q1 \bullet \overline{Q0}$$

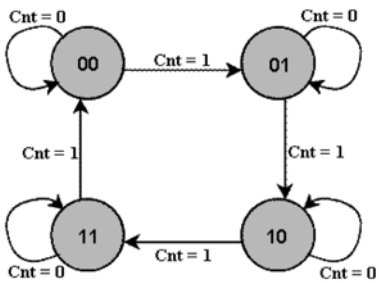
Step 2: Derive the next-state equations by converting these excitation equations into flip-flop characteristic equations. In the case of D flip-flops, $Q_+ = D$.

$$Q0_+ = D0 = Cnt \oplus Q0 = \overline{Cnt} \bullet Q0 + Cnt \bullet \overline{Q0}$$

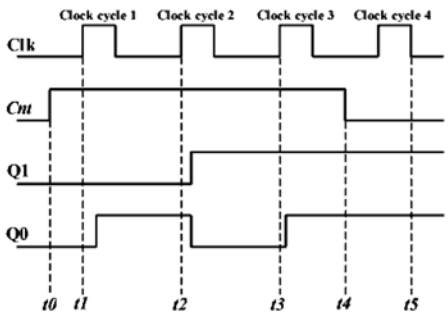
$$Q1_+ = D1 = \overline{Cnt} \bullet Q1 + Cnt \bullet \overline{Q1} \bullet Q0 + Cnt \bullet Q1 \bullet \overline{Q0}$$

Step 3: From the equation, produce a state table and a state diagram

Input, Cnt	Present State		Next State	
	Q1	Q0	$Q1_+ = D1$	$Q0_+ = D0$
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	0



Step 4: Determine the timing diagram



We can conclude:

a) From the counter circuit:

- 2- bit counter because there are two FFs in the design.
- Sycnhronous counter because the FFs have a common clock.

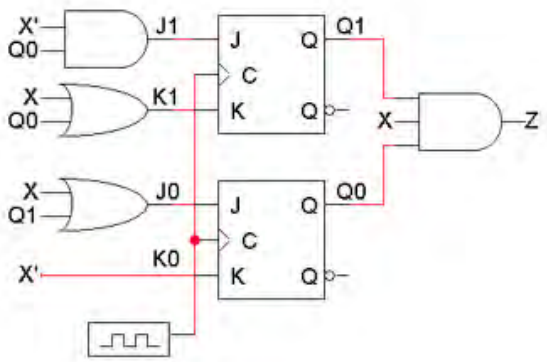
b) From the counter state diagram:

- MOD 4 because has 4 state (i.e. $2^2 = 4$), not a truncated counter.
- Count Up counter when Cnt = 1, and stay in the previous state when Cnt = 0.

Method 2

Digital Logic

- Analyze the following circuit that uses JK flip-flop with one input, *X* and one output, *Z*.



There is one input X, one output Z, and two flip-flops Q_1Q_0.

Step 1: The Boolean expression of each flip-flop and output Z

$J_1 = X' Q_0$ $\quad$ $J_0 = X + Q_1$ $\quad$ $Z = Q_1Q_0X$

$K_1 = X + Q_0$ $\quad$ $K_0 = X'$

Step 2: Produce a state table with JK FF transition using the Boolean expression.

Input, X	Present State		Next State		JK FF Transition				Output, Z
	Q_1	Q_0	Q_{1+}	Q_{0+}	J_1	K_1	J_0	K_0	
0	0	0			0	0	0	1	0
0	0	1			1	1	0	1	0
0	1	0			0	0	1	1	0
0	1	1			1	1	1	1	0
1	0	0			0	1	1	0	0
1	0	1			0	1	1	0	0
1	1	0			0	1	1	0	0
1	1	1			0	1	1	0	1

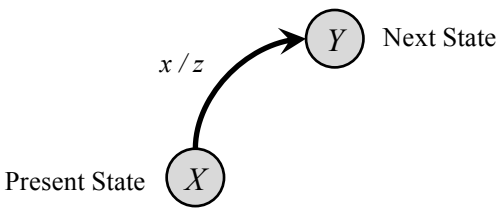
Method 2 (continue)

Digital Logic

Step 3: Complete the state table with next state values using JK excitation table, then draw the state diagram.

Input, X	Present State		Next State		JK FF Transition				Output, Z
	Q_1	Q_0	Q_{1+}	Q_{0+}	J_1	K_1	J_0	K_0	
0	0	0	0	0	0	0	0	1	0
0	0	1	1	0	1	1	0	1	0
0	1	0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0
1	0	1	0	1	0	1	1	0	0
1	1	0	0	1	0	1	1	0	0
1	1	1	0	1	0	1	1	0	1

The drawing of a state diagram for a sequential circuit with an output must have



- A circle to represent a Present state – the state of the sequential circuit at that moment
- A circle to represent a Next state – the state that the sequential circuit going to be
- An arrow that show the changes from the Present state to Next state
  - ✓ for each arrow must write x / z, where
    - ❑ x is the **input** that cause the transition from Present state to the Next state
    - ❑ z is the **output** of the sequential circuit (not the state) when the transition from Present state to Next state occurs

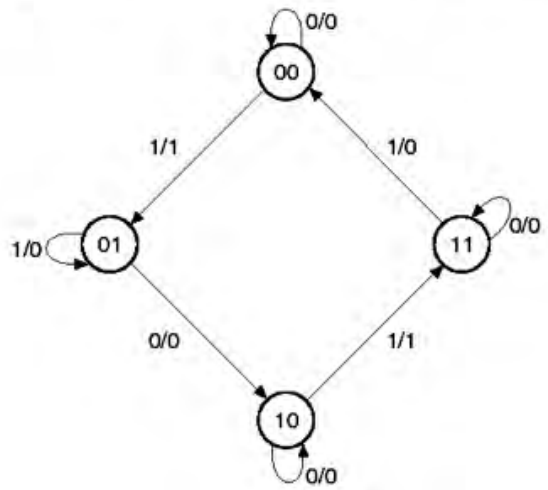
DIY:

Draw the state diagram.

Example: Design

Digital Logic

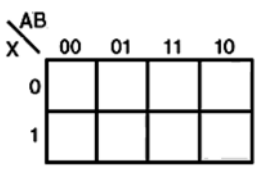
Design a clocked sequential circuit using D flip-flop that operates according to the state diagram given:



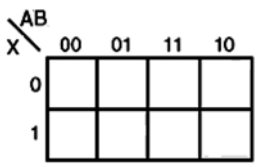
The transition table:

Input	Present State		Next State		Output
X	A	B	A	B	Y
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	0	1	1
1	0	1	0	1	0
1	1	0	1	1	1
1	1	1	0	0	0

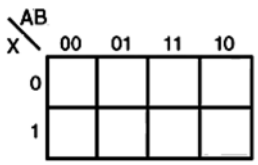
Since the design requires only D flip-flop, then $D_A = A_+$ and $D_B = B_+$.



$D_A =$

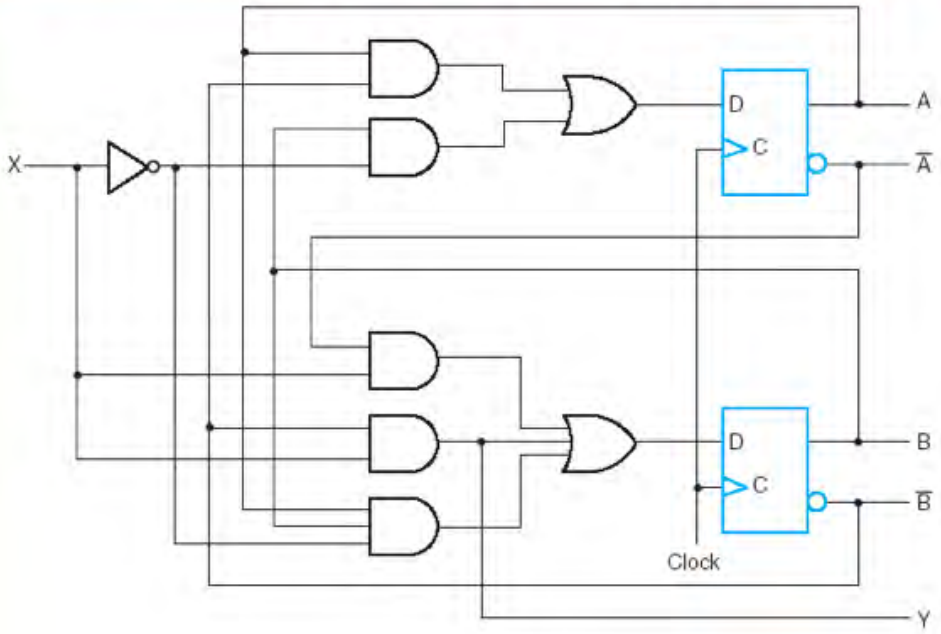


$D_B =$



$Y =$

… Example: Logic Diagram for Sequential Circuit with D Flip-Flops



Digital Logic

End of Module 8

Digital Logic

MODULE 9
SHIFT REGISTERS

Objectives:

1. To introduce the characteristics and function of a shift register
2. To highlight different types of shift register
3. To introduce shift register IC
4. To explain shift register counter

MODULE CONTENT

Digital Logic

Basic Shift Register functions

Shift Register:

- Serial In/Serial Out (SISO)
- Serial In/Parallel Out (SIPO)
- Parallel In/Serial Out (PISO)
- Parallel In/Parallel Out (PIPO)

Bidirectional Shift Registers

Shift Register counters

- Ring Counter
- Johnson Counter

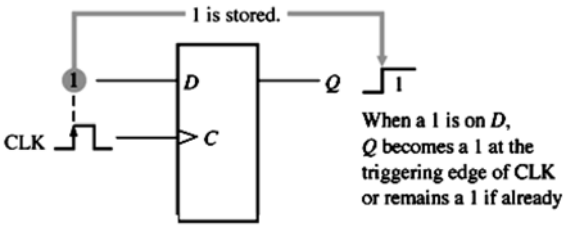
Basic Shift Register (SR) Functions

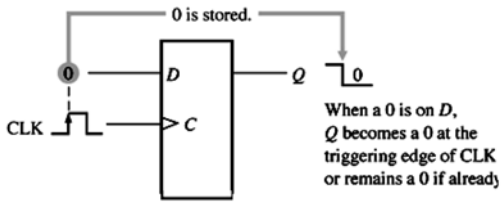
Digital Logic

- SR is a register in which binary data can be stored, and this data can be shifted to the left or right when a signal is applied.
- It consist of arrangements of FFs.
- Digital System : Important in application involving :
  - Data storage
  - Data transfer/movement

Data Storage

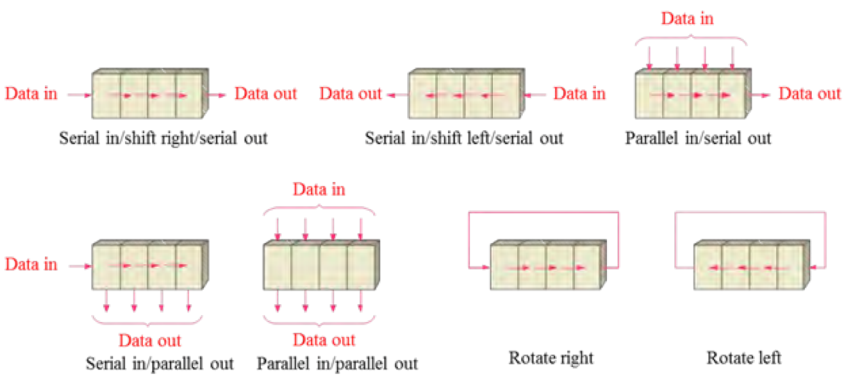
Flip-flop can be use as a storage element. Each flip-flop can store 1 bit data. If more bits required than more flip-flop is used.





Data transfer/movement

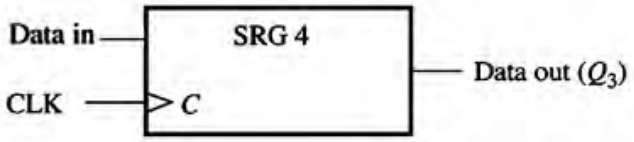
A register that is capable of shifting data one bit at a time is called a shift register. The logical configuration of a serial shift register consists of a chain of flip-flops connected in cascade, with the output of one flip-flop being connected to the input of its neighbor. The operation of the shift register is synchronous; thus each flip-flop is connected to a common clock. Using D flip-flops forms the simplest type of shift-registers. Shifting an N bit shift register will lose all the data after 2N clock cycles. If want to maintain the data use rotate.



Serial In/Serial Out (SISO) ..

Digital Logic

- Accepts data serially, through *Data in* one bit at a time on a single line.
- Every clock pulse one bit of *Data in* will enter the shift register and at the same time one bit will be shifted to Q3
- Output produced in serial form.

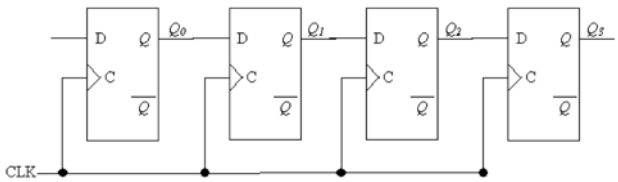


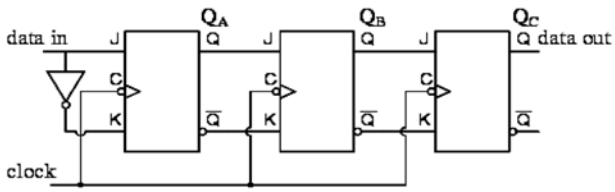
Logic symbol for an 4-bit serial in/serial out shift register

The circuit diagram shows a serial in-serial out shift register implemented using D flip-flops and JK flip-flop configured as D flip-flop.

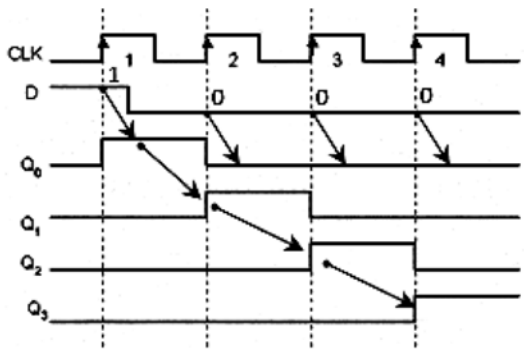
Assuming that the register is initially clear (all 0's) the timing shown the waveform diagram when '1000' is shifted through the register.

Let us now take a closer look at the effect of shifting the binary digit '1' through the register over time. Again, we assume that the register is initially reset and the bit enters the left-most stage.





Serial-in, serial-out shift register using type "JK" storage elements



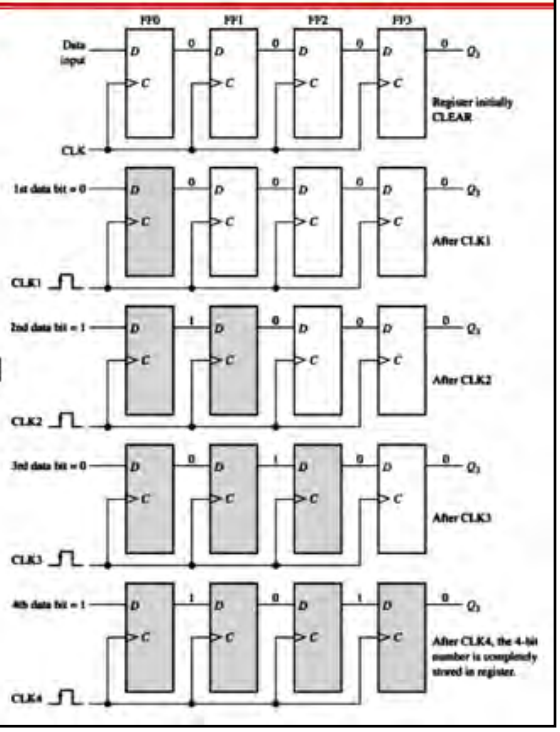
Clock	FF0	FF1	FF2	FF3	Value
Initially	0	0	0	0	0
1	1	0	0	0	8
2	0	1	0	0	4
3	0	0	1	0	2
4	0	0	0	1	1

Note that at each time step from t=1 onwards, the right shift results in a state change which indicates a division by two operation. Similarly, if we operate in reverse chronological order (i.e. from t=4 to t=1), the left shift will result in a multiplication be two.

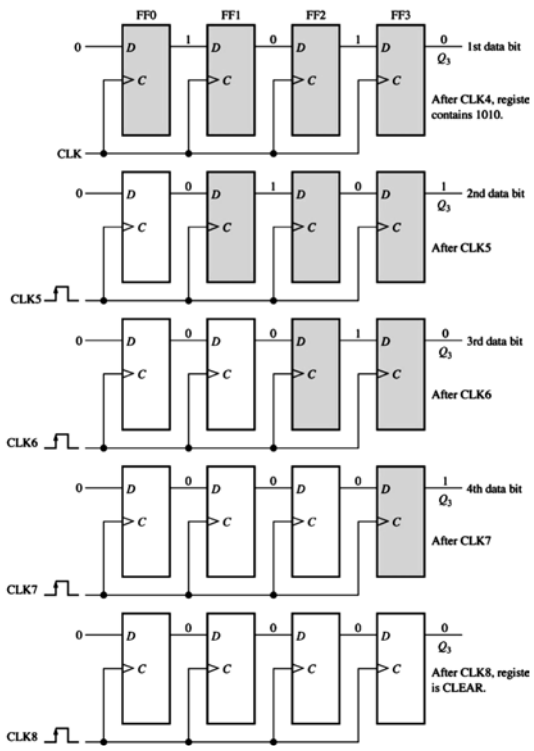
.. Serial In/Serial Out (SISO)

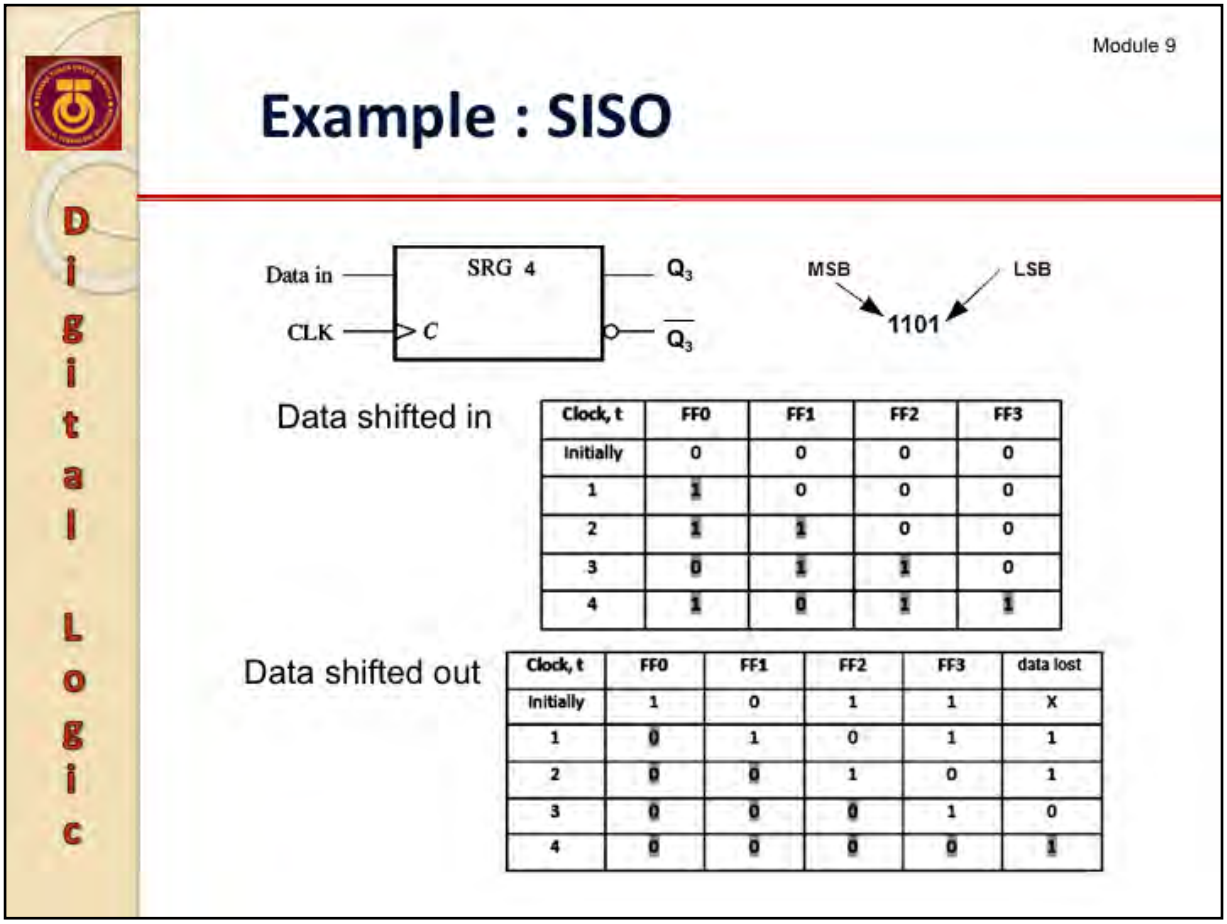
Digital Logic

- Four bits (1010) being entered serially into the register
- Every clock cycle a data is shifted to the right one position
- For n bit SISO it requires n clock cycle to completely entering the shift register
  - i.e. for the first data entered to appear at the output



- Once all data is shifted in , every clock cycle will make the data shifted another one position to the right
- On the 5th clock cycle, the first 0 is shifted out and lost, its position replace by the 2nd data, at the left most flip-flop new data entering the SISO
- This kind of shifting repeat every clock cycle.
- Assuming every time a new data is 0, at the 8th clock cycle all flip-flop in SISO will be replaced by a new data
- Therefore for n bit SISO
  - It requires n clock cycle to shifted in all the data set (n bit)
  - It requires another n clock cycle to shifted out all the data set (n bit)

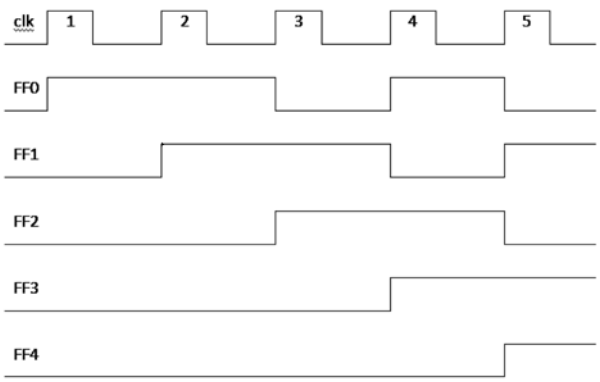




Example: 5-bit Serial In Serial Out (Right Shift Register)

- Data input: 11010
- Initially all FF s are cleared.
- MSB are shifted in first.

Clock	FF0	FF1	FF2	FF3	FF4
Initially	0	0	0	0	0
1	**1**	0	0	0	0
2	1	**1**	0	0	0
3	0	1	**1**	0	0
4	1	0	1	**1**	0
5	0	1	0	1	**1**

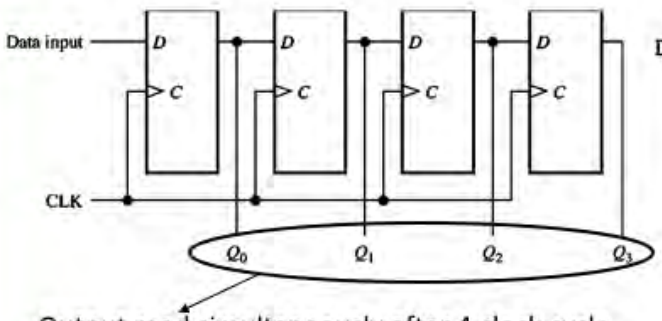


Serial In Parallel Out (SIPO)

Digital Logic

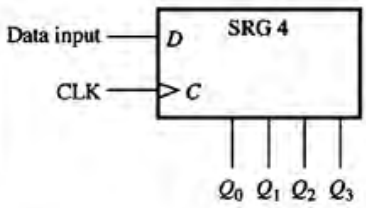
- Data bits are entered serially one bit at a time
- Output is parallel, all bits are available simultaneously but only after N clock cycles for an N bit SIPO
- For SISO , output only at Q_3, for SIPO output $Q_3Q_2Q_1Q_0$

Circuit for 4 bit SIPO

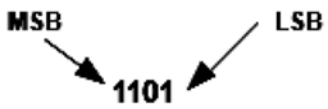


Output read simultaneously after 4 clock cycle

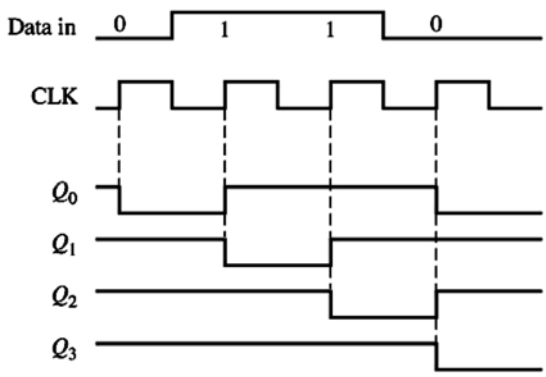
Symbol for 4 bit SIPO



- For a 4 bit SIPO, it can store 4 bit data and shift the data at every clock cycle
- Data entered serially, only after 4 clock cycle the data entered can be read at the output
- Usually MSB data enter the SIPO first (actually which data entered first will depend on the design), after 4 clock cycle it will appear at Q_3



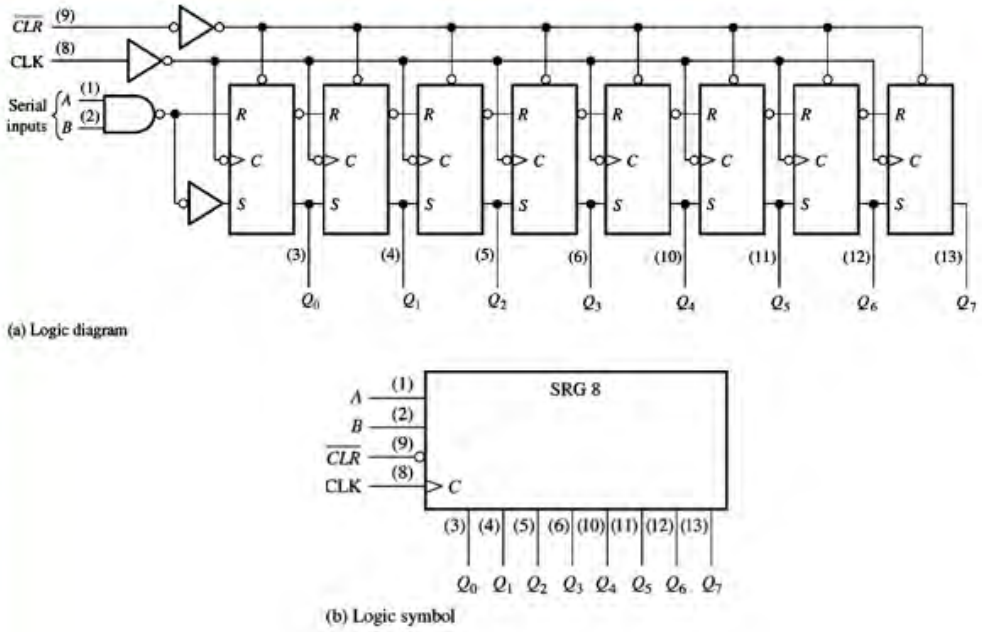
Clock	FF0	FF1	FF2	FF3
Initially	0	0	0	0
1	**1**	0	0	0
2	1	**1**	0	0
3	0	1	**1**	0
4	1	0	1	**1**
	Q_0	Q_1	Q_2	Q_3



DIY:

- Determine what data is entered to SIPO register?
- Circle on the timing diagram the valid output of the SIPO register?

IC 74HC164: 8-Bit Shift Register



(a) Logic diagram

(b) Logic symbol

This is an example of an IC for 8-bit SIPO.

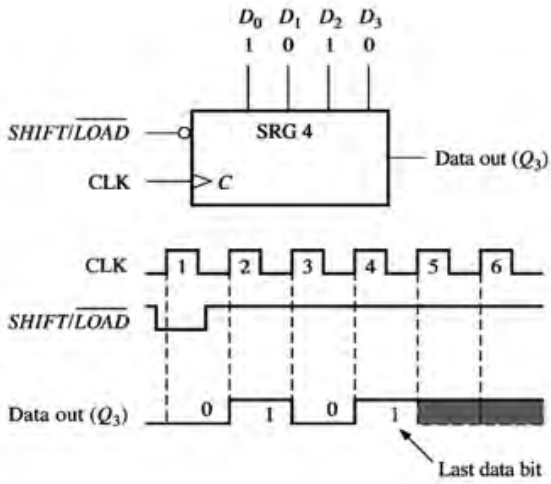
- Serial data can be input at pin A or B.
- CLR' is an active LOW input. If the logic level is 0, it will clear all output to 0000 0000.
- Data entered at pin A or B, will appear at Q_0 after the first clock pulse. The next clock pulse will shift the data to the right.
- The data will appear at Q_7 after the 8-th clock pulse.
- A valid parallel data only appear at all output after 8 clock cycle.

Module 9

Parallel In Serial Out (PISO)

Digital Logic

- Inputs entered simultaneously into respective stages on parallel lines
- Outputs are one bit at a time
- *SHIFT/$\overline{LOAD}$*
  - = 0 → Loads the input values
  - =1 → shifts them out at clock pulse

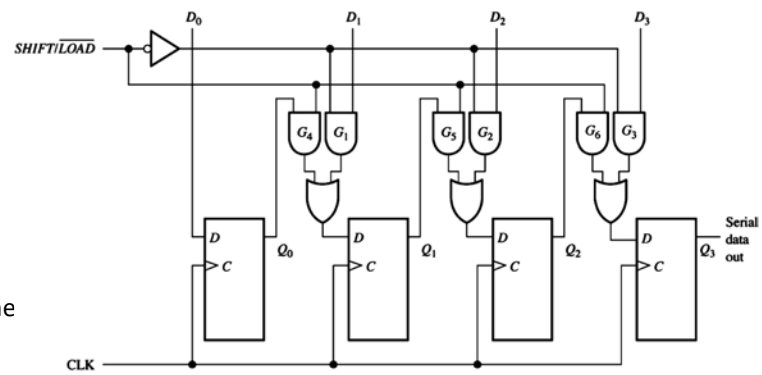


Operation of PISO involves the following 2 steps:

1. **LOAD MODE:**

   a) Input data must be available at parallel input pins: D_3 D_2 D_1 D_0

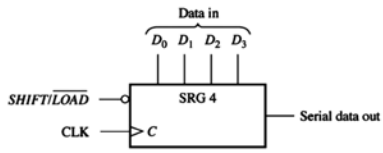
   b) Load the initial parallel value by making $SHIFT/\overline{LOAD} = 0$. This will make the data loaded to the 4 flip-flop inside the shift register at the positive edge of the clock. The MSB of the data will appear at Data out.



(a) Logic diagram

2. **SHIFT MODE:**

   If $SHIFT/\overline{LOAD} = 1$, data will be shifted on every positive edge of the clock cycle. After 4 clock cycle the LSB data will appear at the Data out. At the 5th clock cycle all the 4 bit data set will be lost and replaced by a new set of 4 bit data.



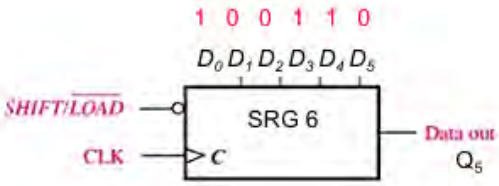
(b) Logic symbol

Example: PISO

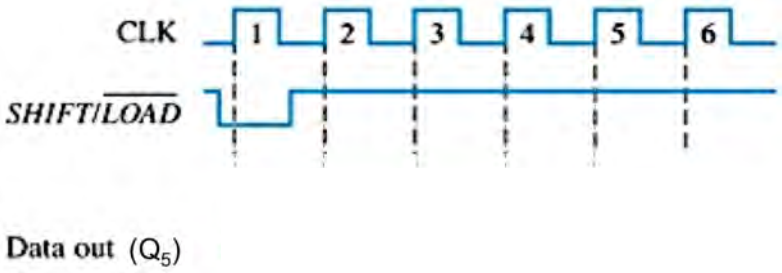
Digital Logic

Complete the timing diagram of 6–bit PISO shift register (SRG 6) for the data input 011001. The register initially all is 0s and assume MSB shifted out first.

a) Draw the shift register logic symbol with the inputs.



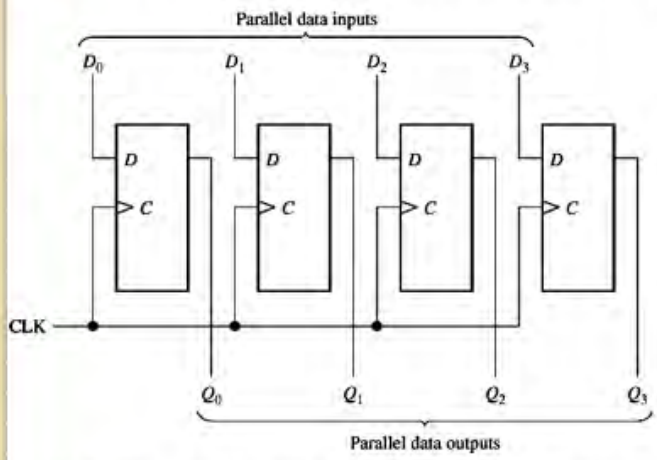
b) Complete the timing diagram.

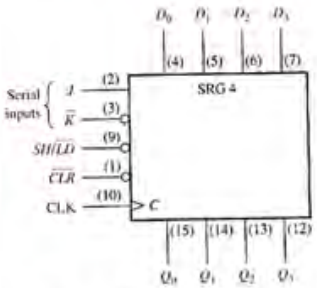


Parallel In Parallel Out (PIPO)

Digital Logic

- Input and output done in parallel.
- Enter all inputs - Bits appear on the parallel outputs.

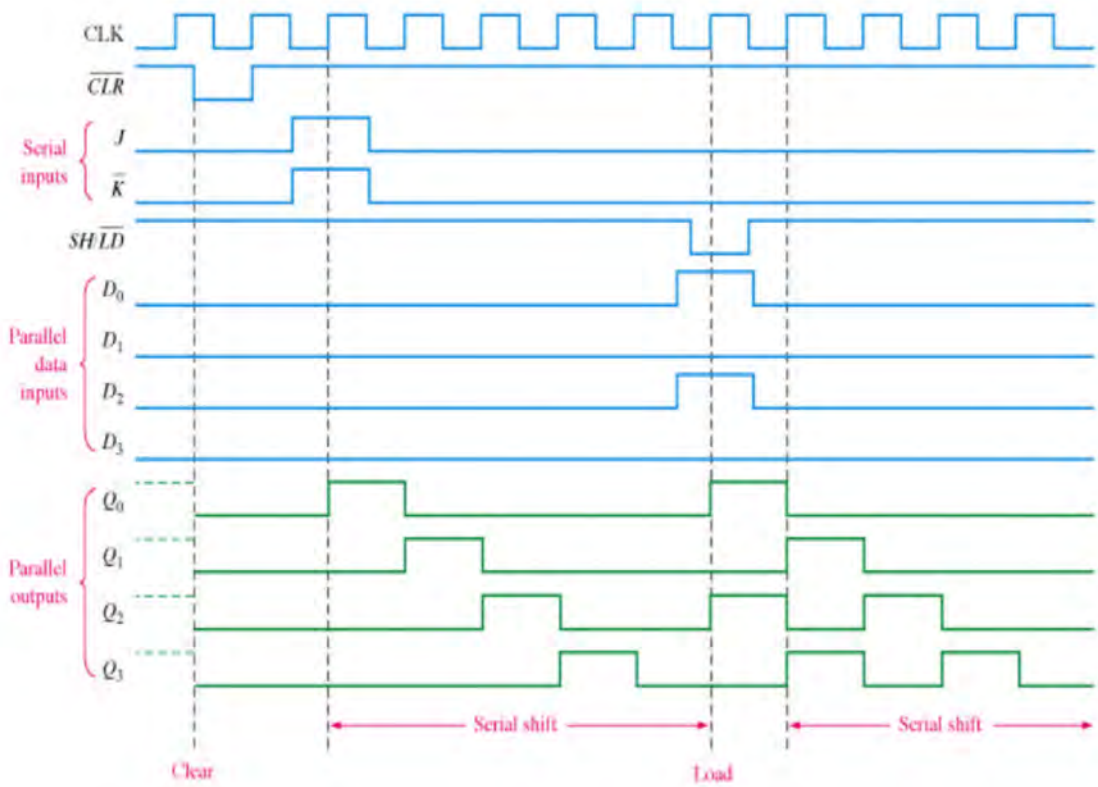




The 74LS195A 4-bit parallel access shift register

- Data input parallel, this data doesn't appear at the output
- The data only appear at the output at the positive edge of the clock.

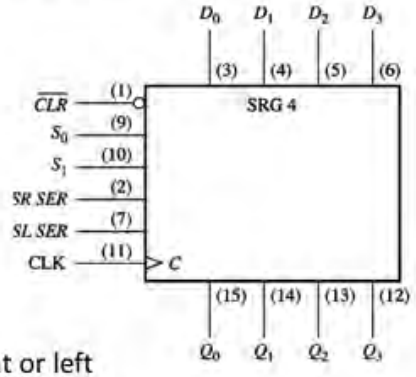
- The 74HC195 can be used for parallel in/parallel out operation, serial in/serial out and serial in/parallel out operations.
- Q3 is the output when it is used for parallel in/serial out operation.



Universal Bidirectional Shift Register

Digital Logic

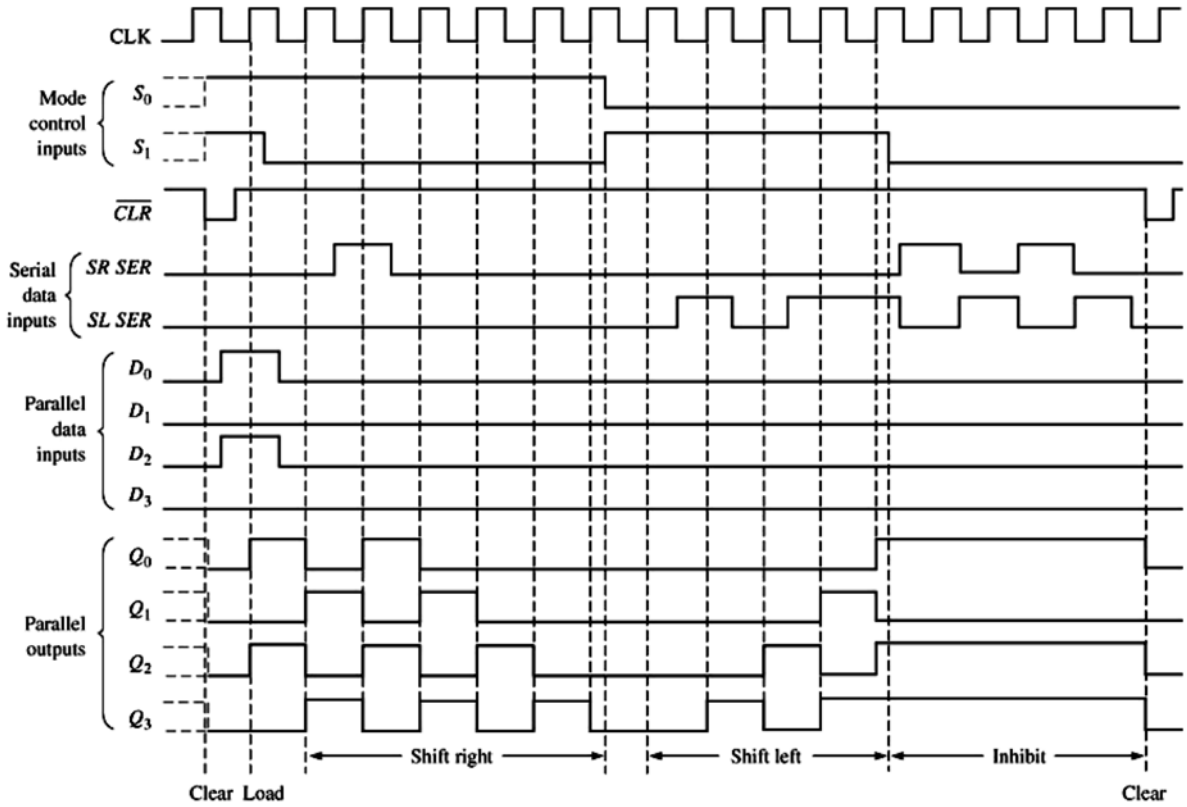
- 74HC194 is a universal bidirectional shift register
  - It has both serial and parallel input and output capability
    - Can implement SISO,SIPO,PISO,PIPO
    - SISO- in: SR SER , out: Q_3 for right shift
      - in: SL SER , out: Q_3 for left shift
    - SIPO – in: either SL SER or SR SER
      out: $Q_3Q_2Q_1Q_0$
    - PISO – in: $D_3D_2D_1D_0$,
      out : either SL SER or SR SER
    - PIPO – in: $D_3D_2D_1D_0$, out: $Q_3Q_2Q_1Q_0$
  - Bidirectional - data can be shifted to right or left



Mode of Operation controlled by S_1S_0

- $S_1S_0 = 00$ - Clock Inhibit , clock has no efect to the output, output don't change
- $S_1S_0 = 01$ - Shift Right, serial data entered from shift right serial input (SR SER)
- $S_1S_0 = 10$ - Shift Left, serial data entered from shift left serial input (SL SER)
- $S_1S_0 = 11$ – Parallel Load, load the parallel data from $D_3D_2D_1D_0$

The timing diagram of a universal bidirectional shift register



Shift Register Counters

Digital Logic

- A shift register with serial output connected back to the serial input to produce special sequences.
- 2 most common types:
  - Ring counter
  - Johnson counter

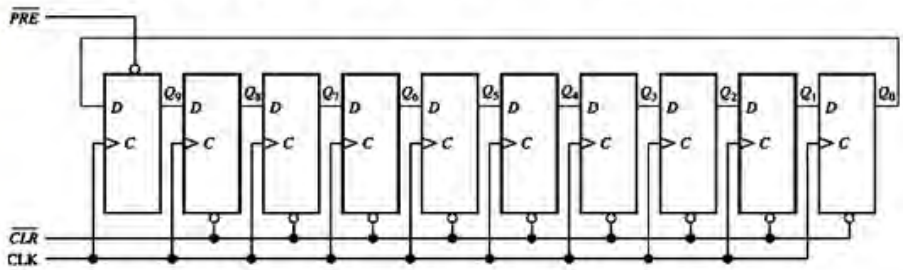
- In some cases, we want a counter that provides individual digit outputs rather than a binary or BCD output.
  - This is usually a case if we want to use a counter to control something rather than counting something
- We can do this by adding a decoder circuit to the binary counter, so that we can detect the state of interest and use it to control something
- However, in many cases it is much simpler to use a different counter structure, that will permit much simpler decoding of individual digit outputs.
  - This is the main reason why we have Ring and Johnson counter.

Module 9

Ring Counter

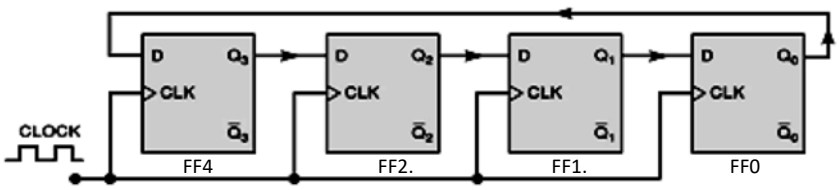
Digital Logic

- Ring Counter is a Serial-In Serial-Out (SISO) shift register but with the final output fed back to the first input



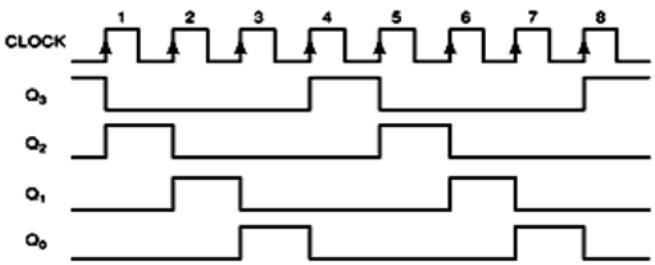
- The first FF (Q_9) is preset to '1' by making PRE' = 0 momentarily, all other flip-flop are cleared (0) by making CLR'= 0 --representing state 0
- The '1' is shifted round the ring, so next clock cycle $Q_8 = 1$, while all other flip-flop output is '0' --represents a state one

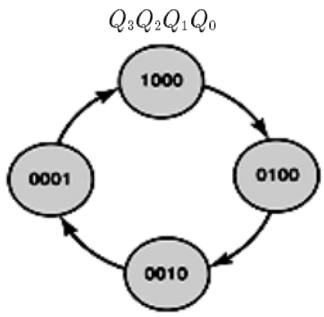
- For n bit ring counter, the number of states it has is n (MOD n)
- For a 4-bit ring counter, there are 4 flip-flops which make it a MOD 4 counter
  - Initially, PRE' = 0 to set $Q_3 = 1$
  - It will recycle after 4 clock cycle



Q_3	Q_2	Q_1	Q_0	CLOCK pulse
1	0	0	0	0
0	1	0	0	1
0	0	1	0	2
0	0	0	1	3
1	0	0	0	4
0	1	0	0	5
0	0	1	0	6
0	0	0	1	7
.	.	.	.	.
.	.	.	.	.

- If only one flip-flop is HIGH while all other LOW, every state will have a unique '1' based on the position of '1'
  - This mean that the '1' can be use to decode any state without the need to use a decoder
  - If we want to decode state 0, get the input from the output of Q_3, whenever $Q_3 = 1$ means the state is state 0
  - To decode state 2, get the input from the output Q_1





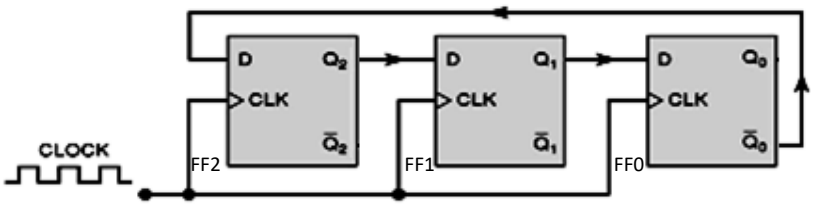
Johnson Counter

Digital Logic

- A Johnson counter also called switchtail ring counter, twisted-ring counter, walking-ring counter, or Moebius counter is a modified ring counter
- In the Johnson counter the last complemented output is fed back in as an input to the first FF
- Can be implemented with any types of FF, but configured as D flip-flop.
- Number of unique states are 2 times the number of bits (FF)
  - 4 bits ➔ 4*2 = 8 states
  - 5 bits ➔ 5*2 = 10 states
- Johnson counter will produce a modulus of 2n; (n = number of stages)

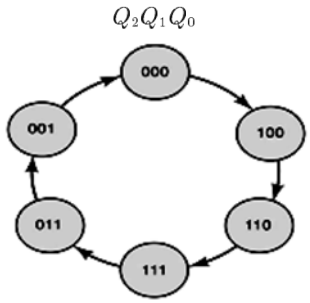
3 bit Johnson counter

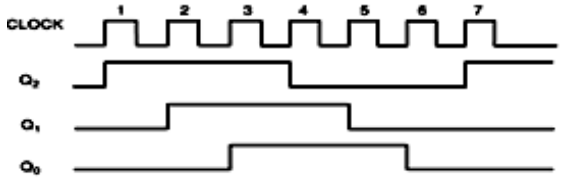
- Circuit: requires 3 D flip-flop, where Q_2' is connected back to the input of the first flip-flop



- The timing diagram and a table showing the operation of Johnson Counter
  - After the 6th clock cycle, the pattern repeat again, therefore 6 unique states (MOD 6)
  - The counting sequence is not binary.

Clock, t	FF2	FF1	FF0	$\bar{Q}_0$
Initially	0	0	0	1
1	1	0	0	1
2	1	1	0	1
3	1	1	1	0
4	0	1	1	0
5	0	0	1	0
6	0	0	0	1
7	1	0	0	1
8	1	1	0	1





Johnson Counter characteristic

Digital Logic

- For n bit flip-flop the MOD is 2n
- For a Johnson counter there always exist 2 bit that are unique compared to the other state
  - This mean that it only requires 2 input decoder to decode a state

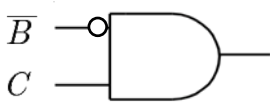
Sequence number	Flip-flop outputs				AND gate required for output
	A	B	C	E	
1	0	0	0	0	$A'E'$
2	1	0	0	0	AB'
3	1	1	0	0	BC'
4	1	1	1	0	CE'
5	1	1	1	1	AE
6	0	1	1	1	$A'B$
7	0	0	1	1	$B'C$
8	0	0	0	1	$C'E$

Count sequence and required decoding

- The a above table is for 4 bit Johnson Counter:
  - For a 4 bit counter there is 8 states
  - Each state have a unique 2 input variable as shown on the rightmost column
  - These unique state simplify the design of a decoder because a decoder with only 2 input is required to decode any state

- For example, to decode state 7 to drive an active HIGH device, we can use the following decoder

Sequence number	Flip-flop outputs				AND gate required for output
	A	B	C	E	
7	0	0	1	1	$B'C$



Comparison of a counter

Digital Logic

- For a given n flip-flop, the binary counter can produce the most state and ring is the worst
- In term of decoding , Ring counter is the best because there is no need for a decoder to decode each state, binary counter is the worst because require a more complex decoder
- Johnson always in the middle when comparing those feature

	MOD for n flip-flop	Decoder Input
Ring	n	no need for a decoder
Johnson	2n	2 input decoder
Binary	2^n	usually > 2 input decoder

Digital Logic

End of Module 9

Tutorials

School of Computing
Faculty of Engineering

UNIVERSITI TEKNOLOGI MALAYSIA

Printed @ 2018

TUTORIAL 1: Digital Logic Overview

1. Fill in the terms for the definition.

Term	Definition
i) Analog	Being continuous or having continuous values.
ii) AND gate	A basic logic operation in which a true (HIGH) output occurs only when all the input conditions are true (HIGH).
iii)	The basic timing signal in digital system.
iv)	Related to digits or discrete quantities.
v)	A sudden change from one level to another, followed after a time, called the pulse width, by a sudden change back to the original level.
vi)	The time interval on the leading edge of a pulse between 10% and 90% of the amplitude

2. Find the duty cycle of a digital waveform if the period is twice the pulse width.

3. Name the device that is use for

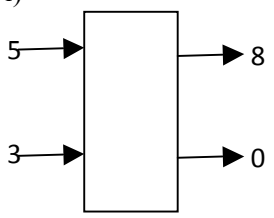
   i. converting a binary number to 7-segment display format.

   ii. data storage.

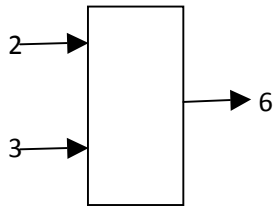
4. A basic 2-input logic circuit has a HIGH on one input and a LOW on the other input, and the output is LOW. Identify the circuit.

5. Name the logic function of each of the block below based on your observation of the inputs and outputs.

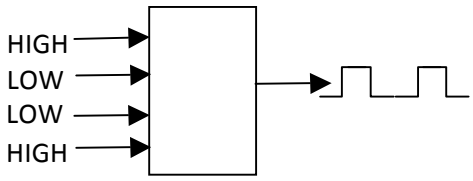
i)



ii)



iii)



6. A pulse waveform with a frequency of 10 kHz is applied to the input of a counter. During 100ms, how many pulses are counted?

7. A periodic digital waveform has a pulse width of 25 µs and a period of 150 µs. Determine the frequency and the duty cycle.

8. i) List 3 main advantages of a digital system compared to an analog system.

ii) An _____ is required to convert an analog signal to a digital signal, and a system that consist of both analog and digital circuits is called a _____ system

iii) The smallest unit in a digital systems is called _____

iv) Determine wheter the following is an analog or a digital quantities, circle the right answer

1) A person weight ***Analog / Digital***
2) Number of cars at the parking lot ***Analog / Digital***
3) Storage capacity of a memory ***Analog / Digital***
4) Tyre pressure ***Analog / Digital***

9 i) Determine the frequency of a waveform in Fig 1 if T is 10ms.

Fig 1

ii) Draw a digital waveform to represent the following digital value 1000101110 (left value first),if the pulse width is 1µs, determine the duration of each bit before it change to a new bit.

10. How many times the digital logic level changed in 1 second if the signal is a square wave with a frequency of 1MHz.

11. Draw a square wave with 25% duty cycle and clearly label the positive edge and the trailing edge.

12. List the suitable logical function for the following problems

i) sending multiple input to a destination using a single cable : _____

ii) converting a key press on a keypad to a BCD code : _____

iii) determine the number of visitors to an expo : _____

iv) determine whether a car exceed the speed limit : _____

v) routing a different packet for a designated destination : _____

vi) memorize characters type on a keyboard : _____

13. What is the difference between a fixed function IC compared to programmable IC

14. Determine which gate has the following property, assume FALSE = 0 and TRUE = 1

a. Output is opposite of the input
b. If both input FALSE then the output will be FALSE
c. If one of the input FALSE the output will be FALSE

TUTORIAL 2: Number Systems and Codes

1. Convert the binary numbers to its decimal equivalent.
   a. 10110_2
   b. 10010101_2
   c. 1101011_2
   d. 100100001001_2
   e. 100101100_2
2. Convert the decimal numbers to its binary equivalent using **repetitive division method**.
   a. 77_{10}
   b. 96_{10}
   c. 205_{10}
   d. 1040_{10}
   e. 3216_{10}
3. Convert the decimal numbers in Question (2) to its binary equivalent using **weighted summation** method.
4. Convert the decimal numbers to its binary equivalent (to four radix point)
   a. 1305.375_{10}
   b. 111.33_{10}
   c. 301.12_{10}
   d. 164.875_{10}
   e. 1000.01_{10}
5. Convert the hexadecimal numbers to its decimal equivalent.
   a. 743_{16}
   b. 2000_{16}
   c. $7FF_{16}$
   d. $ABCD_{16}$
   e. 165_{16}
6. Convert the decimal numbers to its hexadecimal equivalent.
   a. 59_{10}
   b. 1024_{10}
   c. 2313_{10}
   d. $65{,}536_{10}$
   e. 919_{10}

7. Convert the octal numbers to its decimal equivalent.

   a. 56_8

   b. 467_8

   c. 1000_8

   d. 2341_8

   e. 31456_8

8. Convert the decimal numbers to its octal equivalent.

   a. 59_{10}

   b. 1024_{10}

   c. 2313_{10}

   d. 65536_{10}

   e. 919_{10}

9. When a large decimal number is to be converted to binary, it is **sometimes** easier to convert it to hex and then from hex to binary. Try this procedure for 3216_{10} and compare it with the procedure used in Question (2).

10. Convert the binary numbers in Question (1) to

    a. hexadecimal

    b. octal

11. Convert the binary numbers to its equivalent hexadecimal and octal values.

    a. 10111010100.101

    b. 1000001101111.01

12. Encode the decimal numbers in BCD.

    a. 77_{10}

    b. 96_{10}

    c. 205_{10}

    d. 1040_{10}

    e. 3216_{10}

13. The following numbers are in BCD. Convert them to decimal.

    a. 100101110101010010

    b. 010101010101

    c. 0111011101110101

14. Which of the following numbers are valid BCD values?

    a. 0110101100110001

    b. 1001110000001000

    c. 00000000000

15. What is the largest BCD-encoded decimal value that can be represented in three bytes?

16. For the question below please refer to the ASCII table.

    a. What is the most significant nibble of the ASCII code for letter E?

    b. Represent the statement "X = 3 × Y" in ASCII code. Attach each character values with even parity bit.

    c. The following bytes (shown in hex) represent a person's name as it would be stored in the computer's memory. Determine the name of each person.

        i. 42 72 61 64 20 50 69 74 74

        ii. 41 6E 67 65 6C 69 6E 61

17. Calculate the lower and upper bound of signed number for 7-bit number system using the representation of

    a. sign and magnitude

    b. 1's complement

    c. 2's complement

18. Calculate the binary signed values in the representation format of (i) sign and magnitude, (ii) 1's complement and 2's complement using 8-bit number system.

    a. $+ 55_{10}$

    b. $+ 127_{10}$

    b. $- 87_{10}$

    c. $- 128_{10}$

19. Given a number system specification: **size of a number is 6 bit, including the sign bit AND signed numbers using 2's complement**

    Calculate and show your working for the arithmetic operations below.

    a. 18 + 3

    b. -18 + 3

    c. 18 – 3

    d. -18 - 3

TUTORIAL 3: Logic Gates Overview

1. For a 2-input NOR gate functioning as a negative-AND gate, output X is HIGH if both inputs A and B are HIGH.

   TRUE / FALSE

2. A two-input XNOR gate will produce a HIGH output when both inputs are equal.

   TRUE / FALSE

3. A NOR gate with inverters at the inputs has the same logic function as an AND gate.

   TRUE / FALSE

4. A 2-input NAND gate and a 2-input NOR gate produces the same output when both inputs are HIGH.

   TRUE / FALSE

5. The _____ gate produces a HIGH output when all inputs are LOW.

   a. NOR

   b. NAND

   c. XOR

   d. AND

6. A 2-input logic gate **X** produces a HIGH output when input A is LOW and input B is HIGH. Which of the following is **NOT** logic gate **X**?

   a. OR

   b. NOR

   c. NAND

   d. XOR

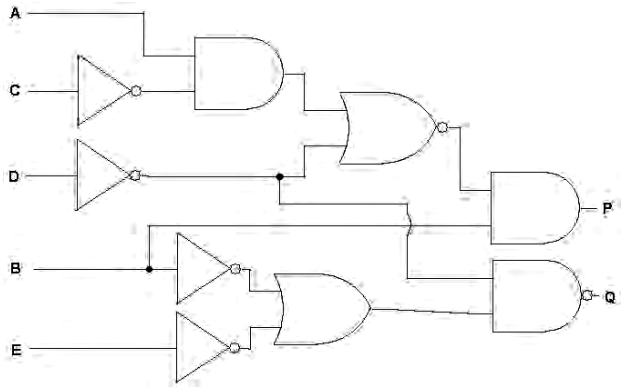
7. Complete the following questions:

   i) Draw the logic symbol of an XOR gate.

   ii) Give appropriate labels to the inputs and output.

   iii) Write its truth table.

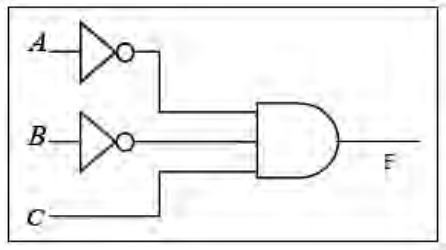
8. Write the Boolean expressions of output P and Q.

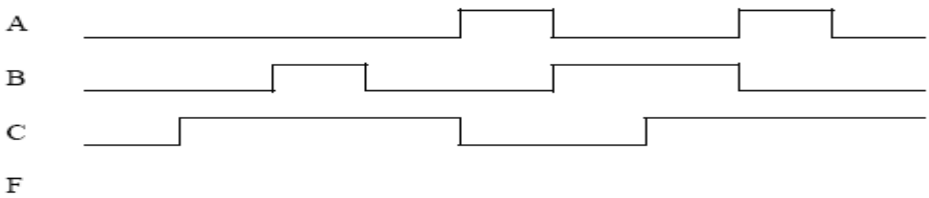


P = ?

Q = ?

9. Complete the timing diagram based on the given input for the following logic diagram.





10. Draw symbol for the following gates:

i) 2-input AND gate

ii) 2-input OR gate

iii) Inverter

iv) 2-input NAND gate

v) 2-input NOR gate

11. Given an AND gate with 3 inputs, what should the input values be to get an output of 1 (HIGH)?

12. Given an OR gate with 3 inputs, what should the input values be to get an output of 1 (HIGH)?

13. Fill in the table below, follow the example given.

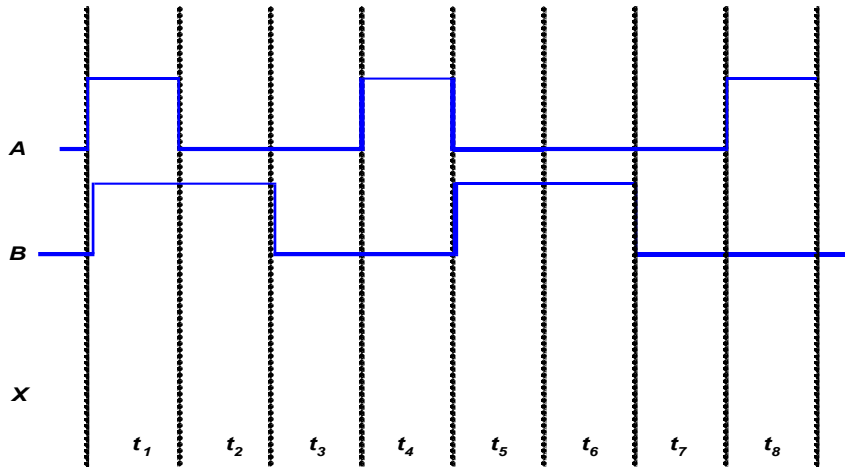
Gate	**Input**			**Output**
	A	**B**	**C**	
AND	1	0	1	
OR	0	1	0	
AND	1	1	1	
OR	0	0	0	X = A + B + C = 0 + 0 + 0 = 0
NOR	1	1	1	
AND	1	0	0	

14. Given the input waveform(s) below, show the appropriate output waveform, ***X***, with a timing diagram.

    i) XNOR

    ii) XOR

    iii) NAND



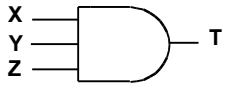
15. Identify the following devices according to logic function:

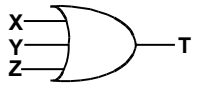
i) 74LS04

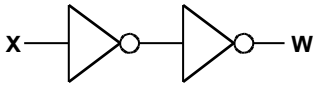
ii) 74ALS10

iii) 74HC00

16. Given the logic gates below, write the logic expression for it.

i) 

ii) 

iii) 

TUTORIAL 4: Boolean Algebra and Logic Simplification

1. Using any logic gates, draw the logic diagram of the given function. Do NOT simplify the function

$$X = \overline{A} + B(D + \overline{E}\,\overline{F})(\overline{A + C})$$

2. Directly apply DeMorgan's law to the following expressions. You do NOT have to simplify the expressions.

   i) $Y = (\overline{A + \overline{B} + C}) + \overline{\overline{C}\,D\,\overline{E}}$

   ii) $Y = \overline{P + \overline{Q}\,(\overline{R + S\,P})}$

3. Simplify the function using Boolean Algebra

$$Y = A\,B\,\overline{C} + A\,\overline{B} + A\,B\,C + \overline{A}\,\overline{B}$$

4. Develop a truth table for the following expression. From the truth table derive a standard product-of-sums (POS) expression.

$$f = (A + \overline{B})(A + C)(A + B + \overline{C})$$

5. Use a Karnaugh map to reduce the expression to a minimum sum-of-products (SOP) form

$$f = \overline{A}\,B\,(C\,\overline{D} + C\,D) + A\,C\,D$$

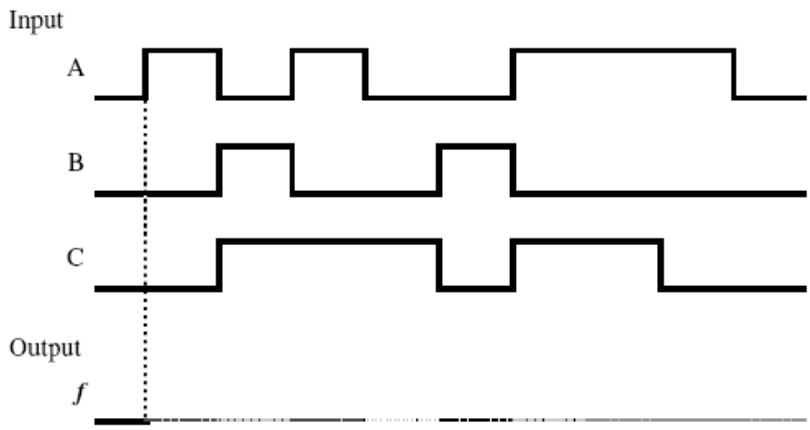
6. For the truth table given below,

A	B	C	f
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

   i) Express the output function, f in Standard Sum-of-Products (SOP) form.

   ii) Express the output function, f in Standard Product-of-Sums (POS) form.

   iii) If the following waveforms are applied to the inputs, A, B, and C of the logic circuit, draw the output waveform for the function, f.



7. Below is the truth table of a three-input XOR gate.

A	B	C	Y
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

i) Express Y in standard sum-of-product (SOP) variable form.
ii) Express Y in standard product-of-sum (POS) variable form

8. Construct the truth table for

$$Y = A\overline{BC} + \overline{AB}\,C + \overline{AC}\,B$$

9. Using any logic gates, draw the logic diagram of

$$P = \overline{\overline{(X + Z)} + \overline{Y}(W + \overline{XZ})}$$

10. Using Karnaugh Map,find the minimum SOP expression for the following function

i)

$$Y = \overline{P}\,\overline{Q}\,\overline{R}\,\overline{S} + \overline{P}\,\overline{Q}\,\overline{R}\,S + \overline{P}\,\overline{Q}\,R\,\overline{S} + \overline{P}Q\,\overline{R}\,\overline{S} + P\,\overline{Q}\,\overline{R}\,\overline{S} + \overline{P}\overline{Q}\,R\,\overline{S} + P\,Q\,\overline{R}\,\overline{S}$$

ii)

$$G(w, x, y, z) = \sum m(1, 3, 14, 15) + d(0, 2, 6, 8, 13)$$

11. Implement $Y = \overline{a}(b + \overline{d}(\overline{\overline{ac} + e}))$ using any type of logic devices

12. Using Karnaugh Map, find the minimum SOP of the given m-notation

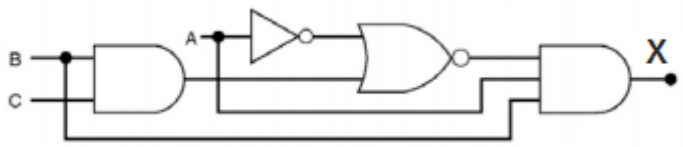
$$F(p, q, r) = \sum m(0,1,3,4,6)$$

13. Prove that

$$x\,y + x\,\overline{y} + \overline{x}\,\overline{y} = x + \overline{y}$$

TUTORIAL 5: Combinational Logic Circuit

1. Draw the logic circuit for the following using only **2-input AND gates**, **2-input OR gates** and **inverters.**
   a. $(A\overline{B}C + D\overline{E})$
   b. $(D\overline{E}F + A\overline{B})C$
   c. $(\overline{A} + \overline{B} + \overline{C})(\overline{D} + \overline{E} + \overline{F})$
   d. $A\overline{B}(\overline{C} + \overline{D})$
   e. $(\overline{A}\overline{B} + C + D)\overline{E}F$
2. Convert the circuits in Question 1a, 1b,and 1d, using only **NAND** gates.
3. Convert the circuits in Question 1c and 1e, using only **NOR** gates.
4. Convert the following circuit to NAND only then prove that the converted circuit is the same as the original circuit.



5. Referring to truth table below, draw the circuit to implement the function using
   i) AND-OR
   ii) AND-OR-Invert

Inputs			Output
A	**B**	**C**	**X**
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

DESIGN EXERCISE

For each of the given problems, do the following:

a. Determine the input, output and the relationship between output and the input
b. Create the truth table or Boolean expression
c. From the truth table or Boolean expression, gets the simplified expression using a Karnaugh map(s) or Boolean Algebra.
d. Draw a logic circuit for the simplified expression using only 2-input AND gates, 2-input OR gates and inverters.
e. Re-draw the simplified logic circuit using only NAND gates for POS and NOR only for SOP.
f. Re-draw the simplified logic circuit using dual symbol

1. **Car Safety Alarm**
   - Design a car safety alarm considering four inputs
     - Door closed (D)
     - Key in (K)
     - Seat pressure (S)
     - Seat belt closed (B)
   - The alarm (A) should sound if
     - The key is in and the door is not closed, or
     - The door is closed and the key is in and the driver is in the seat and the seat belt is not closed

2. **A Majority High**
   - Design a circuit that will give a HIGH output (F), when the majority of the 3 inputs are HIGH

3. **Elevator Door Control**
   - Design a logic circuit that controls an elevator door in a three-story building, considering the following inputs
     - Elevator moving (M)
       - M = 1 : elevator is moving
       - M = 0 : elevator has stopped
     - F1, F2, F3 :- floor indicator
       - HIGH when elevator is positioned at the level of the particular floor
       - F2 = 1 : elevator at level with floor 2. F1 = F3 = 0.
   - The circuit output is the OPEN signal.
     - OPEN = 1 : elevator door will open.

4. **Sound the Horn**
   - A manufacturing plant needs to have a horn sound to signal quitting time. The horn should be activated when either of the following conditions is met:
     - It is after 5 p.m. and all machines are shut down.
     - It is Friday, the production run for the day is complete, and all the machines are shut down.
   - Design a logic circuit that will control the horn. (Hint: use four logic input variables to represent various conditions; for example, input A will be HIGH only when the time of day is 5 p.m. or later.)

5. **BCD Counter**
   - A BCD counter that produces 4 bits output, representing the BCD code for the number of pulses that have been applied to the counter input. For example, after 4 pulses have occurred, the counter outputs are *DCBA* = $0100_2 = 4_{10}$. The counter resets to zero on the tenth pulse and starts counting over again. In other words, the DBCA outputs will be never represents a number greater than $1001_2 = 9_{10}$.
   - Design the logic circuit that produces a HIGH output whenever the count output is 2, 3 or 9

6. **Conditional Output**

   The circuit has 4 inputs, labelled as ABCD, where A is the MSB and D is the LSB. The inputs represent a number in a 2's complement form. You are required to design a circuit that has the following characteristic

   a) When B = C = 0, the output will be LOW
   b) Other than condition a), when the input value is **positive odd** or **negative even**, the output will be HIGH.
      Positive odd is positive numbers that are odd and negative even is negative numbers that are even.
   c) The condition where all input 1's does not exist in the system

7. **Count numbers of 1's at the input**
   Input to the system is a binary number that can have a value of 0 to 7. Count the number of 1 at the input and display it at as a two bit binary number. For example if the input is 5 it will be input as 101, and the number 1's at the input are 2 therefore the output is 10 (two in binary).

TUTORIAL 6: Functions of Combinational Logic

1. What are the **equations** to retrieve the sum (**Σ**) and the carry output (**C_{out}**) of a full adder?
2. How **many full adders** are needed to complete these additions?
   a. 101 and 010
   b. 1100 and 0101
   c. 111 and 001
   d. 1010 and 1101
   e. 010 and 011
   f. 11001 and 10101

3. For Question 3 (Show your workings):
   a. Draw the block diagram of parallel adders for Question 3
   b. Get *every instance* of sum (**Σ**) and carry output (**C_{out}**) for each full adder.
   c. Get the results.

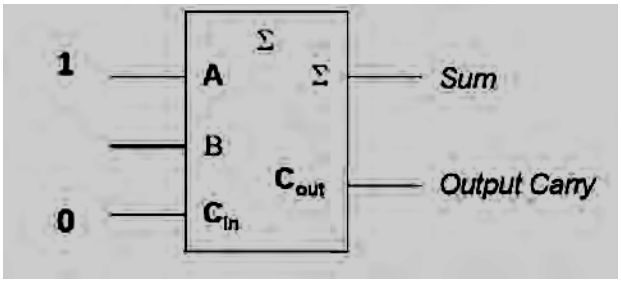
4. Use the parallel adder truth table (Table 1) to find the sum and output carry for the addition of the following two 4 bit number if the input carry (C_{n-1}) is 0. Show the diagram.

$$X_4 X_3 X_2 X_1 = 1101 \qquad Y_4 Y_3 Y_2 Y_1 = 0101$$

Table 1

X_n	Y_n	C_{n-1}	C_n	$\sum_n$
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

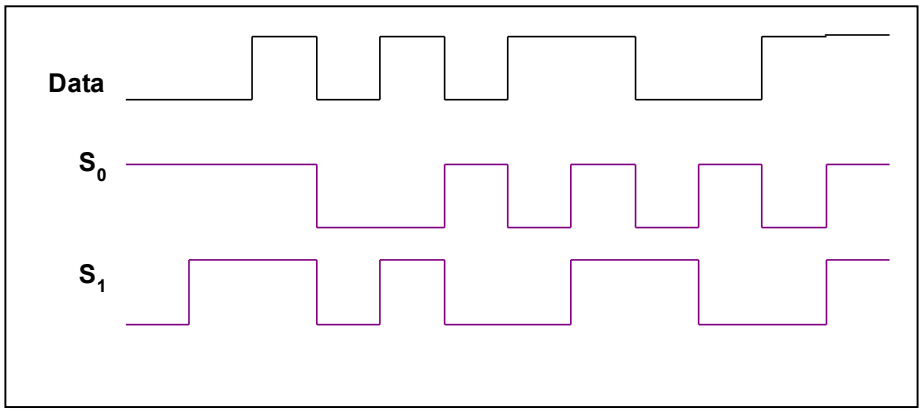
5. Determine the outputs of **Sum** and **Output Carry** for the inputs given for Full Adder as shown below.



6. Implement the function f(A,B,C) = Σ(0, 2, 4, 7) using a 4 x 1 multiplexer.

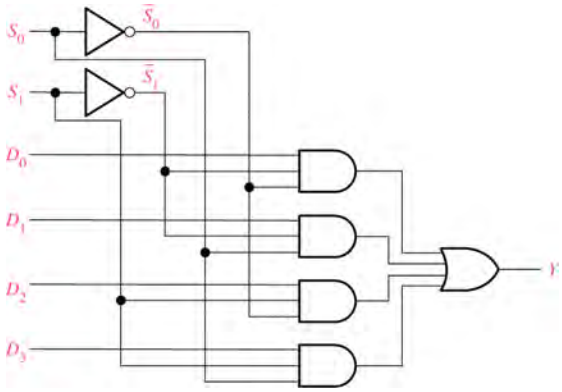
7. Convert the following:
   a. 98 in BCD to binary
   b. 01110101 in BCD to binary

8. A DEMUX has 4 outputs D0, D1, D2 and D3. Given the information below, draw the appropriate waveforms for the outputs.



9. a) What type of multiplexer shown in a figure below.

   b) Develop the truth table for multiplexer in figure below.



10. What is the difference between **decoder** and **multiplexer**.

11. Design :

    a. BCD-to-decimal decoder using the minimal number of 2-input AND gates

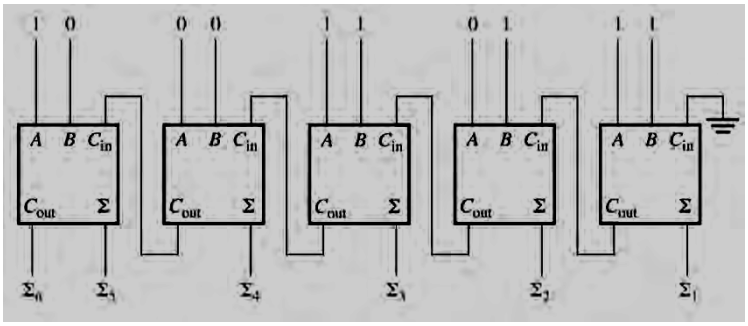
    b. Repeat, using two 2-to-4-line decoders and a few interconnecting AND gates

12. Using a full-adder, determine the logic state (1 or 0) at each gate output for the following inputs:
    (a) $A = 1, B = 1, C_{in} = 1$
    (b) $A = 0, B = 1, C_{in} = 1$
    (c) $A = 0, B = 1, C_{in} = 0$

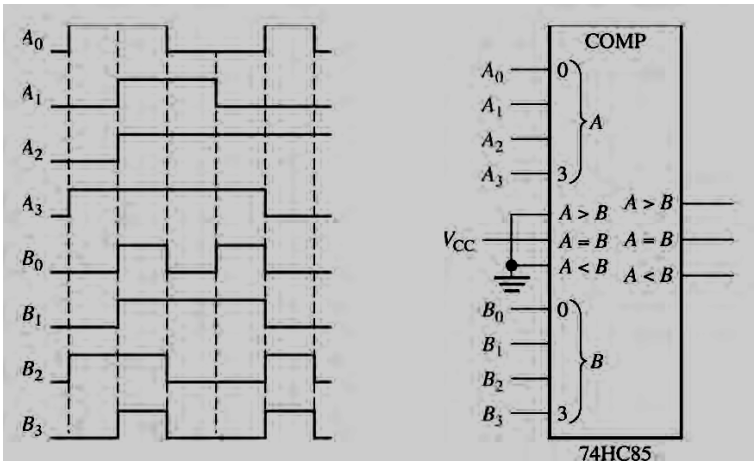
13. What the full-adder inputs that will produce each of the following outputs:
    (a) $\Sigma = 0, C_{out} = 0$
    (b) $\Sigma = 1, C_{out} = 0$
    (c) $\Sigma = 1, C_{out} = 1$
    (d) $\Sigma = 0, C_{out} = 1$

14. Determine the outputs of a full-adder for each of the following inputs:
    (a) $A = 1, B = 0, C_{in} = 0$
    (b) $A = 0, B = 0, C_{in} = 0$
    (c) $A = 0, B = 1, C_{in} = 1$
    (d) $A = 1, B = 1, C_{in} = 1$

15. For the parallel-adder, determine the complete sum by analysis of the logical operation of the circuit. Verify your result by longhand addition of the two input numbers.



16. For the 4-bit comparator, plot each output waveform for the inputs shown. The outputs are active-HIGH.



17. Show the decoding logic for each of the following codes if an active-HIGH (1) output is required:

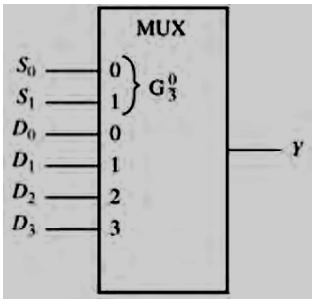
(a) 1101
(b) 1000
(c) 11011
(d) 11100
(e) 101010
(f) 111110
(g) 000101
(h) 1110110

18. For the decimal-to-BCD encoder logic, assume that the 9 input and the 3 input are both HIGH. What is the output code? Is it a valid BCD code?

19. Convert the following decimal numbers to BCD and then to binary.

    (a) 2
    (b) 8
    (c) 13
    (d) 26
    (e) 33

20. For the multiplexer given, determine the output for the following input states: $D_0 = 0$, $D_1 = 1$, $D_2 = 1$, $D_3 = 0$, $S_0 = 1$, $S_1 = 0$.



21. Draw a logic circuit for 2-to-1 multiplexer (MUX) using gates.

22. (a) Show how two 2-to-1 MUX (with no added gates) could be connected to form a 3-to-1 MUX. Input selection should be as follows:
    If $AB = 00$, select I_0
    If $AB = 00$, select I_1
    If $AB = 1$ – (B is a don't care), select I_2

    (b) Show how two 4-to-1 and one 2-to-1 MUX could be connected to form an 8-to-1 MUX with three control inputs.

    (c) Show how four 2-to-1 and one 4-to-1 MUX could be connected to form an 8-to-1 MUX with three control inputs.

23. Design a circuit which will either subtract X form Y or Y from X, depending on the value of A. If $A = 1$, the output should be X - Y, and if $A = 0$, the output should be Y – X. Use a 4 bit subtracter and two 4-bit 2-to-1 MUX.

24. Realize a full adder using 3-to-8 line decoder and

    (a) two OR gates
    (b) two NOR gates

TUTORIAL 7 : Latches and Flip-Flops

A) Objective Question

i. What ***advantage*** does a J-K flip-flop have over an S-R FF?
   a. It has fewer gates.
   b. It has only one output.
   c. It has no invalid states.
   d. It does not require a clock input.

ii. If both inputs of an S-R flip-flop are ***low***, what will happen when the clock goes high?
   a. An invalid state will exist.
   b. No change will occur in the output.
   c. The output will toggle.
   d. The output will reset.

iii. The ***asynchronous inputs*** are normally labelled _____ and _____, and are normally active _____ inputs.
   a. PRE, CLR, low
   b. ON, OFF, high
   c. START, STOP, low
   d. SET, RESET, high

iv. When ***both inputs*** of a J-K pulse-triggered FF are high, and the clock cycles, the output will
   a. be invalid.
   b. not change.
   c. remain unchanged.
   d. toggle.

v. Flip-flops are normally used for all of the following applications, ***except***
   a. counting.
   b. logic gates.
   c. frequency division.
   d. data storage.

vi. When a flip-flop is used to ***divide*** the clock frequency,

a. the J and K inputs are connected to ground.

b. the J and K inputs are connected to Vcc.

c. the reset is tied to the clock.

d. all the inputs are connected to the preset.

vii. The ***invalid state*** of an S-R latch occurs when

a. S = 1, R = 0

b. S = 0, R = 1

c. S = 1, R = 1

d. S = 0, R = 0

viii. Which of the following best describes the action of ***pulse-triggered FF's***?

a. The clock and R-S inputs must be pulse shaped.

b. The data is entered on the leading edge of the clock, and transferred out on the trailing edge of the clock.

c. A pulse on the clock transfers data from input to output.

d. The synchronous inputs must be pulsed.

ix. Like the latch, the ***flip-flop*** belongs to a category of logic circuits known as

a. Monostable multivibrators

b. Bistable multivibrators

c. Astable multivibrators

d. One-shots

B) Subjective Question

1) Answer the following questions:

a. List three types of **latches**.

b. What is the Q output of a D latch when **EN = 1 and D = 1**?

c. How does a J-K flip-flop **differ** from an S-R flip-flop in its basic operation?

2) For a **gated S-R active high latch**,

a. Determine the **Q output** for the input shown in ***Figure 1***. Show them in proper relation to enable (EN) input. Assume that Q starts LOW.

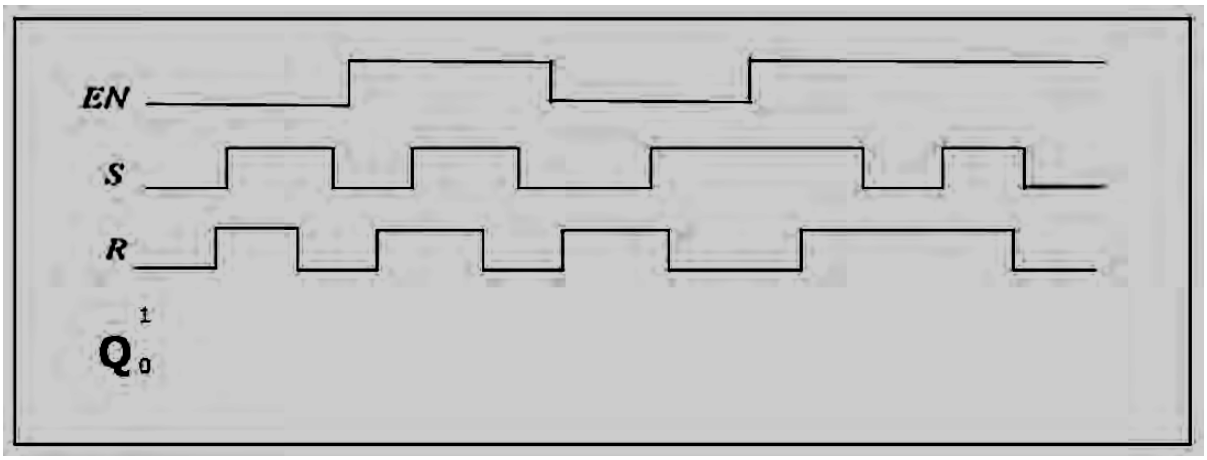


Figure 1

b) Determine the **Q waveform** relative to the clock if the signals shown in ***Figure 2*** are applied to the inputs of the J-K flip-flop. Assume that Q is initially LOW.

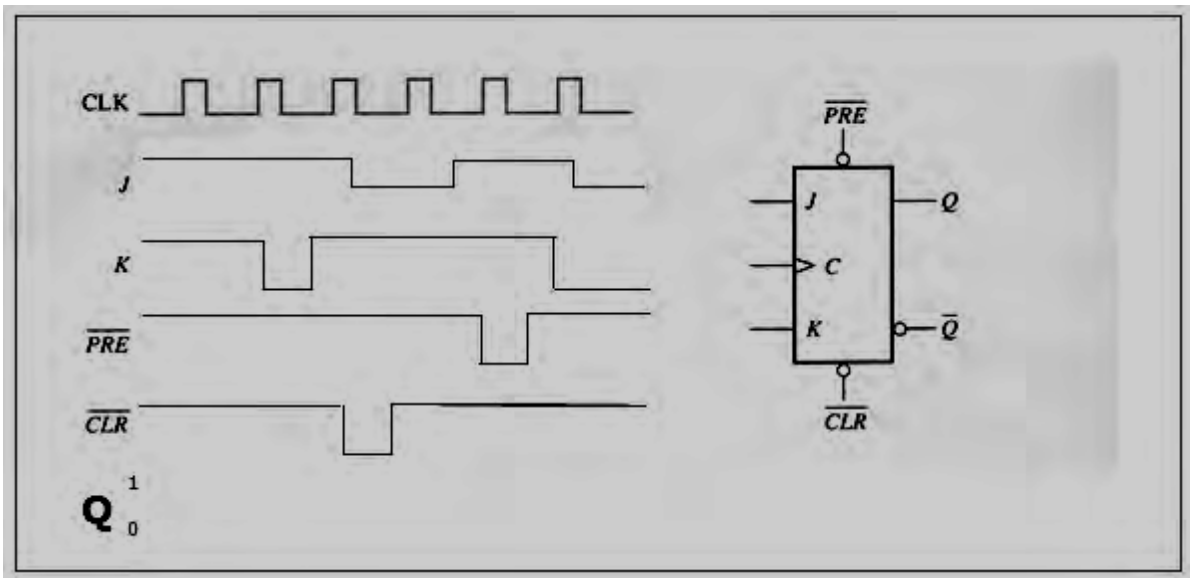


Figure 2

TUTORIAL 8: Counters

I. Answer True or False for the following statement.

1. Sequential circuits can be synchronous and asynchronous. True/ False

2. A synchronous sequential circuit changes its states at discrete instants of time. True/ False

3. Asynchronous sequential circuits can have state transitions at discrete instants of time. True/ False

4. Synchronous sequential circuits are also known as clocked sequential circuits. True/ False

5. A transition of a clock from 0 to 1 is called the falling edge. True/ False

6. The clock period is the time when the clock signal is equal to 1. True/ False

7. The memory used in synchronous sequential circuits are flip-flops. True/ False

II. Design and Analysis Problems

1. Design a 4-bit synchronous binary counter using T flip-flops by following the procedure:
   a. Create the state table.
   b. Identify the input of T flip-flops.
   c. Draw K-maps to identify the Boolean expression for the T flip-flops.
   d. Draw the circuit diagram.

2. Design the 3-bit Gray code counter based on the state diagram of Figure 1 using JK flip-flops.

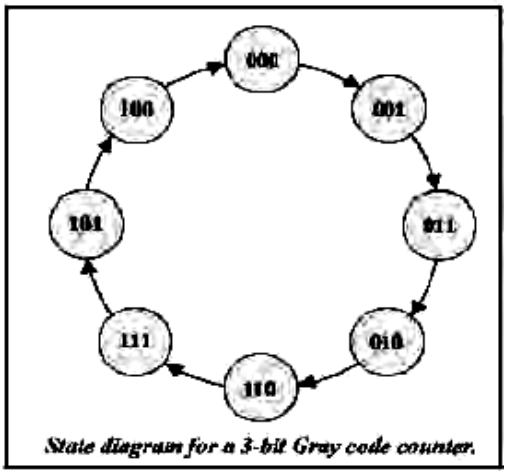


Figure 1

3. Design a 3-bit up/down counter of Gray code sequence as shown in Figure 2 using D flip-flops.

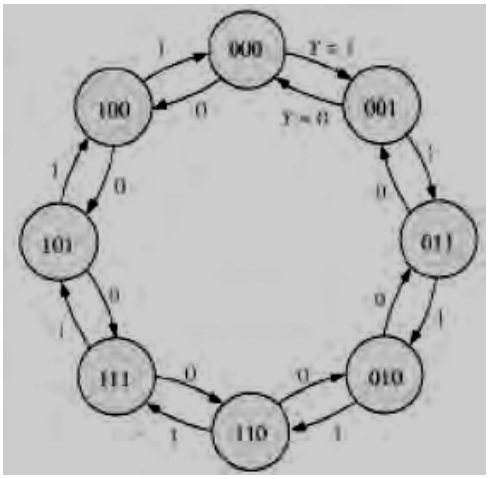


Figure 2

4. Design a synchronous counter of the sequence in Figure 3 using T flip-flop.

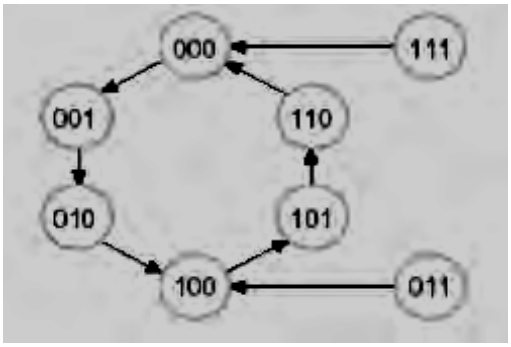


Figure 3

5. Design a 3-bit counter which counts in the sequence:

000, 011, 010, 110, 111, 101, 100, (repeat) 000, …..

a. Use D flip-flops
b. Use T flip-flops
c. Use JK flip-flops

6. An M-N flip-flop works as follows:

If MN = 00, the next state of the flip-flop is 0.

If MN = 01, the next state of the flip-flop is the same as the present state.

If MN = 10, the next state of the flip-flop is the complement of the present state.

If MN = 11, the next state of the flip-flop is 1.

a. Complete Table 1 (use don't care when possible).

Table 1: Excitation Table for MN Flip-Flop

Present State	Next State	M	N
0	0		
0	1		
1	0		
1	1		

b. Using MN flip-flops, design a counter which counts in the following sequence:

000, 001, 011, 111, 101, 100, (repeat) 000, …

7. Use JK flip-flops to design a 3-bit synchronous up/down counter that starts at 001 and cycles through Prime numbers only (i.e. numbers divisible by only themselves or 1). Assume that M=0 counts down, M=1 counts up.

8. Design a synchronous counter to count backwards from 9 down to 0 using JK flip-flops. (Procedure: determine the number of flip-flops required, sketch the state transition table, use K-maps to simplify each J,K signal, sketch the circuit).
Note: Ensure that if an unused state is encountered the counter will return to state 9 on receipt of the next clock pulse.

9. Answer True or False based on the state diagram in Figure 5.

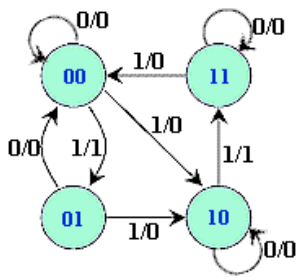


Figure 5

a. When the circuit is in state 00, the label 1/0 means that the circuit will go to the next state 10.

b. If the present state is 01, and the input is 0, the next state would be 10.

c. If the circuit is presently in state 11, it will remain in its present state 11 if the input is 0 and the output is 0.

10. Consider a sequential circuit shown in Figure 4. It has one input *X*, one output *Z* and two state variables Q1Q2 (thus having four possible present states 00, 01, 10, 11). Based on the circuit, derive:

a. the Boolean expression for D1, D2 and *Z*.

b. the state table and FF transition table.

c. the state diagram.

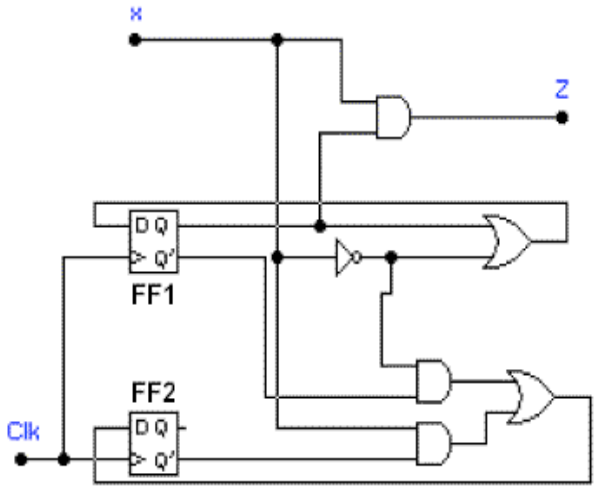


Figure 4

11. Derive the next state, the output table and the state diagram for the sequential circuit shown in

a. Figure 5 b. Figure 6

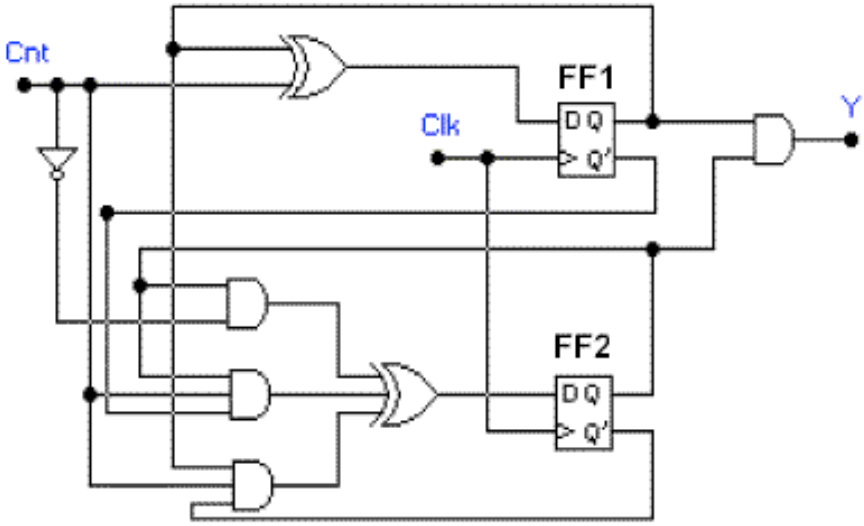


Figure 5

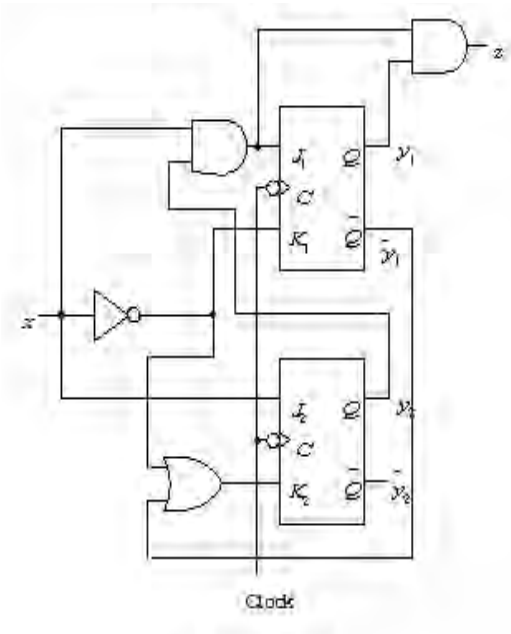


Figure 6

TUTORIAL 9: Shift Registers

1. a) What are the two properties of a shift register?
   b) For a 5-bit SISO with a clock of 1MHz, how long does it takes to transfer a data entered at the input to the output?

2. Referring to IC 74HC164, answer the following questions
   a) Input A and B of the IC held HIGH and its Q_7 output is connected to the pin $\overline{CLK}$ via an inverter. Determine the output sequence of the IC for every clock cycle.
   b) An 8-bit digital word is applied to a 74HC164 8-bit SIPO shift register. The word is to be applied to three different circuits after a time delays of 3µs, 6µs and 8µs. Show the connection to the IC to satisfy the requirement.

3. Initially an 8-bit SISO shift register is loaded with the data word 11101001. The data word 156_{10} is then entered serially from right to left. After 7 clock pulses the circuit is disabled. Write down a table that shows the bits stored after 7 clock pulses.

4. An 8-bit binary counter, a 6-bit ring counter and a 10-bit Johnson counter are connected in cascade. A clock frequency of 10MHz is applied to the input of the first counter. Calculate the frequency of the output waveform.

5. a) Describe the operation of a shift register in Figure 1.

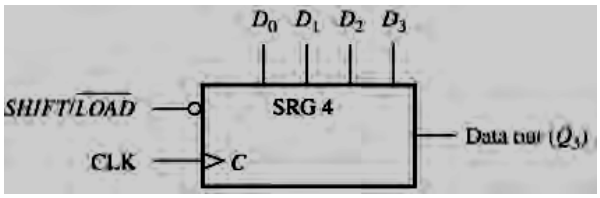


Figure 1

   b) If a nibble data 1101 is to be loaded and shifted by the circuit,
      i) Determine the logic value at the inputs.
      ii) Draw a timing diagram to show the operation of the shift register.

6. a) Which shift-register counter requires the most flip-flops for a given MOD number?
   b) Which shift-register counter requires the most decoding circuitry?

7. a) Design a 3-bit ring counter using D flip-flop.
   b) How to set the most significant output of the 3-bit ring counter that you design?
   c) By using table, show the output of the ring counter for 6 clock cycles.
   d) Draw the state diagram of the designed ring counter and determine its MOD.

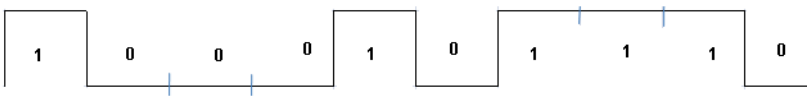
8. Redo all Questions 7 but change the circuit to 5-bits Johnson counter.

ANSWERS TO SELECTED QUESTIONS

Tutorial 1

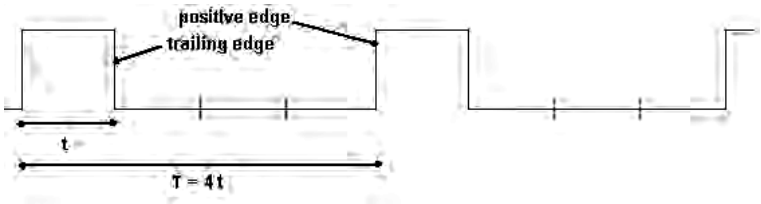
1. i) Analog ii) AND gate iii) clock iv) digital v) pulse vi) risetime
2. 50%
3. i) 7 segment driver ii) memory
4. AND or NOR or XNOR gate
5. i) Adder ii) Multiplier iii) Multiplexer
6. 1000
7. Frequency = 6.67 KHz, duty cycle = 16.7 %
8. i) a) programmable
   b) ease of storage
   c) ease of fabrication on IC
   ii) Analog to Digital Converter (ADC), hybrid
   iii) bit
   iv) 1) Analog 2) Digital 3) Digital 4) Analog

9. i) Cannot be determine because it is a non periodic (aperiodic) signal

   ii)
   

   iii) 1µs

10. A squarewave has a high and low for each cycle and it is a periodic signal. Therefore the
    $Period, T = \frac{1}{f} = \frac{1}{1Mhz} = \frac{1}{10^6} = 1\mu s$
    Within a cycle there are 2 signal change low to high and high to low.
    For 1 second, $no\ of\ cycles = \frac{1}{1\mu s} = 1 \times 10^6$
    Therefore no of signal change = 2 x (1×10^6) = 2 million times.

11.


12. i) Multiplexer
    ii) Encoder
    iii) Counter
    iv) comparator
    v) DeMultiplexer
    vi) Memory
13. For fix function IC the function of the IC cannot be changed while for Programmable IC ,its function can be change by reprogramming the IC
14. a. NOT
    b. OR , AND
    c. AND

Tutorial 2

1b. 149	1d. 2313	
2a. 1001101	2c. 11001101	2e. 110010010000
4a. 10100011001.0110	4c. 100101101.0001	4e. 1111101000.0000
5b. 8192	5c. 43981	
6a. 3B	6c. 909	6e. 397
7b. 311	7d. 1249	
8b. 2000	8d. 200000	

10a. 16, 95, 6B, 909, 12C

11a. 5D4.A, 2724.2

12c. 0010 000 0101	12e. 0011 0010 0001 0110

13b. 555

14b. Invalid	14c. Valid

17c. -64 to + 63

18b. **0**111 1111, **0**111 1111, **0**111 1111

18d. Out of range for sign & magnitude representation, out of range for 1's complement, -128 for 2's complement

Tutorial 3

1. FALSE
2. TRUE
3. TRUE
4. TRUE
5. AND
6. NOR
7. (a) XOR Symbol and its labels

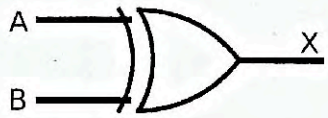


Figure 1: XOR Symbol and its labels

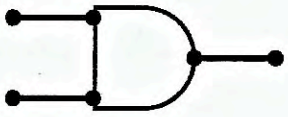
(b) Figure 1 truth table.

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

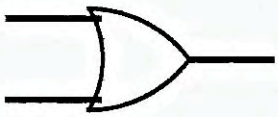
8. $P = \overline{A\overline{C} + \overline{D}} \times B, \quad Q = \overline{\overline{B} + \overline{E} \times \overline{D}}$

9. F = 010100001

10. (a) AND Gate

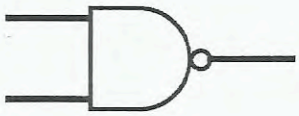


(b) OR Gate

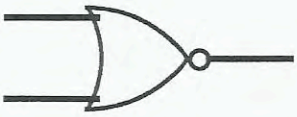


(c) NOT Gate

(d) NAND Gate



(e) NOR Gate



11. All inputs are 1 (HIGH).

12. At least one input is 1 (HIGH)

13.

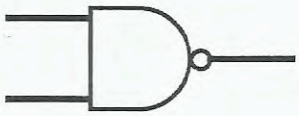
GATE	INPUT A	B	C	OUTPUT
AND	1	0	1	0
OR	0	1	0	1
AND	1	1	1	1
OR	0	0	0	0
NOR	1	1	1	0
AND	1	0	0	0

14. (a) XNOR, X = 10100010
    (b) XOR, X = 01011101
    (c) NAND, X = 01111111

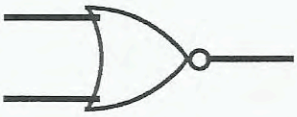
15. (a) Hex Inverter
    (b) Triple 3-input NAND
    (c) Quad 2-input NAND

16. (a) $T = X.Y.Z$
    (b) $T = X + Y + Z$
    (c) $W = \overline{\overline{X}} = X$

(d) NAND Gate



(e) NOR Gate



11. All inputs are 1 (HIGH).

12. At least one input is 1 (HIGH)

13.

GATE	INPUT A	INPUT B	INPUT C	OUTPUT
AND	1	0	1	0
OR	0	1	0	1
AND	1	1	1	1
OR	0	0	0	0
NOR	1	1	1	0
AND	1	0	0	0

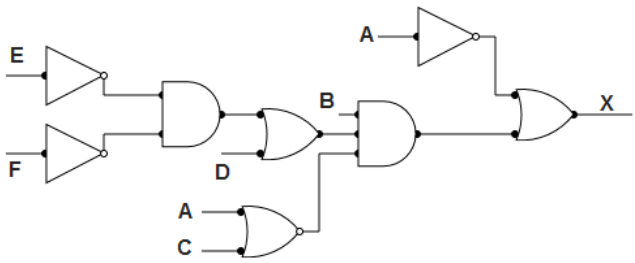
14. (a) XNOR, X = 10100010
    (b) XOR, X = 01011101
    (c) NAND, X = 01111111

15. (a) Hex Inverter
    (b) Triple 3-input NAND
    (c) Quad 2-input NAND

16. (a) $T = X.Y.Z$
    (b) $T = X + Y + Z$
    (c) $W = \overline{\overline{X}} = X$

Tutorial 4

1 .



2. i) $Y = \bar{A}B\bar{C} + C + \bar{D} + E$

ii) $Y = \bar{P}Q + R + SP$

3. $Y = A + B$

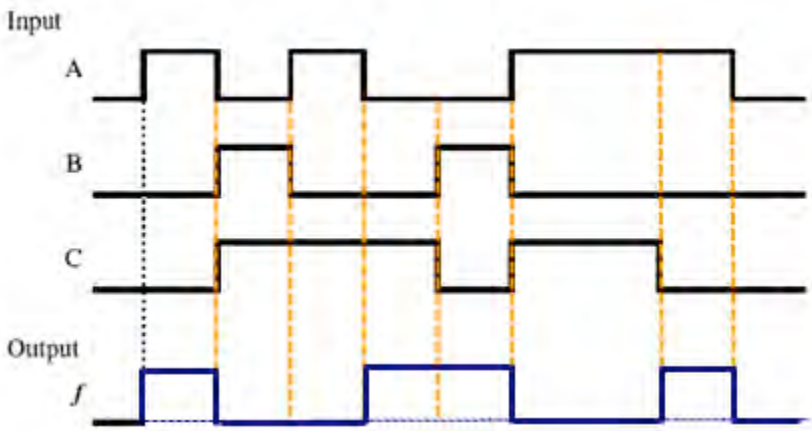
4. $f = (A + B + C)(A + B + \bar{C})(A + \bar{B} + C)(A + \bar{B} + \bar{C})$

5. $f = \bar{A}BC + ACD$

6. i) $f = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$

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

iii)



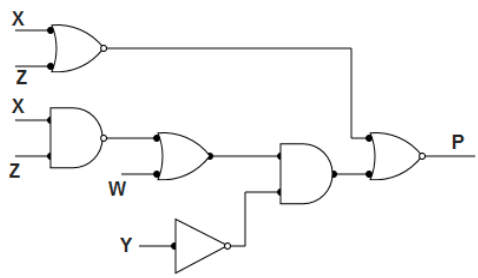
7. i) $Y = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$

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

8.

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

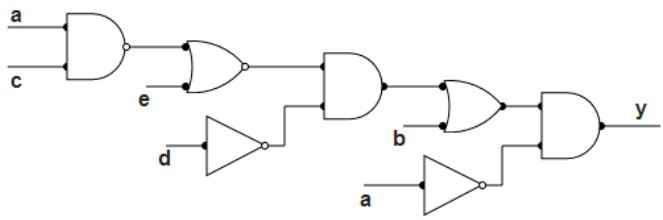
9.



10.i) $Y = \bar{R}\bar{S} + \bar{P}\bar{Q}\bar{R} + \bar{Q}\bar{S}$

ii)$G = \bar{W}\bar{X} + WXY$

11.



12 $F_{(p,q,r)} = pq + p\bar{r} + \bar{p}r$

Tutorial 8

2.

Present State			Next State			JK Transition Table					
Q2	Q1	Q0	$Q2_+$	$Q1_+$	$Q0_+$	J2	K2	J1	K1	J0	K0
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	1	0	X	1	X	X	1
0	1	0	1	1	0	1	X	X	0	1	X
0	1	1	0	1	0	0	X	X	1	X	1
1	0	0	0	0	0	X	1	0	X	0	X
1	0	1	1	0	0	X	0	0	X	X	1
1	1	0	1	1	1	X	0	X	0	1	X
1	1	1	1	0	1	X	0	X	1	X	0

Q2 \ Q1Q0	00	01	11	10
0	0	0	0	1
1	X	X	X	X

$J2=Q1\overline{Q0}$

Q2 \ Q1Q0	00	01	11	10
0	X	X	X	X
1	1	0	0	0

$K2=\overline{Q1}\ \overline{Q0}$

Q2 \ Q1Q0	00	01	11	10
0	0	1	X	X
1	0	0	X	X

$J1=\overline{Q2}Q0$

Q2 \ Q1Q0	00	01	11	10
0	X	X	1	0
1	X	X	1	0

K1= Q0

Q2 \ Q1Q0	00	01	11	10
0	1	X	X	1
1	0	X	X	1

J0=Q1

Q2 \ Q1Q0	00	01	11	10
0	X	1	1	X
1	X	X	0	X

3.

Present State				Next State			D FF Transition		
I	**Q2**	**Q1**	**Q0**	**$Q2_+$**	**$Q1_+$**	**$Q0_+$**	**D2**	**D1**	**D0**
0	0	0	0	1	0	0	1	0	0
0	0	0	1	0	0	0	0	0	0
0	0	1	0	0	1	1	0	1	1
0	0	1	1	0	0	1	0	0	1
0	1	0	0	1	0	1	1	0	1
0	1	0	1	1	1	1	1	1	1
0	1	1	0	0	1	0	0	1	0
0	1	1	1	1	1	0	1	1	0
1	0	0	0	0	0	1	0	0	1
1	0	0	1	0	1	1	0	1	1
1	0	1	0	1	1	0	1	1	0
1	0	1	1	0	1	0	0	1	0
1	1	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	1	0	0
1	1	1	0	1	1	1	1	1	1
1	1	1	1	1	0	1	1	0	1

Digital Logic : Laboratory Manual

Fifth Edition

Abd Bahrim Yusoff | Mazleena Salleh | Mohd Fo'ad Rohani | Ismail Fauzi Isnin | Raja Zahilah Raja Mohd. Radzi | Marina Md Arshad | Mohd Murtadha Mohamad | Siti Hajar Othman | Firoz Yusuf Patel Dawoodi | Muhalim Mohamed Amin

School of Computing
Faculty of Engineering

UNIVERSITI TEKNOLOGI MALAYSIA

Printed @ 2018

Student's Name : _____

Section : _____

Semester : _____

Introduction

Two hours per week are allocated for laboratory session in Digital Logic. All laboratory experiments are marked and contributed to 10% of the overall mark for the course.

- Students must attend all scheduled laboratory sessions.
- Students are required to be punctual – latecomers may be refused entry.
- Pre-lab questions must be answered prior to the laboratory session.
- Answer all questions in the sections of the laboratory manual.

Here are some guidelines to help you perform the experiments:

- Read all instructions carefully and carry them all out.
- Ask the demonstrator if you are unsure of anything.
- Don't depend on the Digital kit settings, always measure!
- Record actual results (comment on them if they are unexpected!)
- If required, write up full and suitable conclusions for each experiment.
- If student have any doubt about the safety of any procedure, contact the demonstrator beforehand.
- **THINK** about what you are doing!

Most experiments involve the use of the Digital kit and your breadboard.

The breadboard

In general the breadboard consists of two terminal strips and two bus strips (often broken in the centre). Each bus strip has two rows of contacts. Each of the two rows of contacts are a node. That is, each contact along a row on a bus strip is connected together (inside the breadboard). Bus strips are used primarily for power supply connections, but are also used for any node requiring a large number of connections. Each terminal strip has 60 rows and 5 columns of contacts on each side of the centre gap. Each row of 5 contacts is a node.

You will build your circuits on the terminal strips by inserting the leads of circuit components into the contact receptacles and making connections with 22-26 gauge wire. There are wire cutter/strippers and a spool of wire in the lab. It is a good practice to wire +5V and 0V power supply connections to separate bus strips.

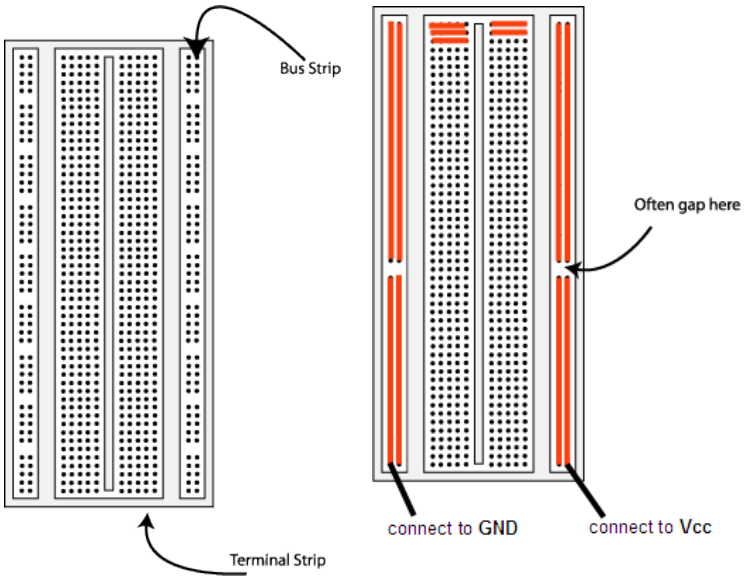


Fig 1. The breadboard. The orange lines indicate connected holes.

The 5V supply **MUST NOT BE EXCEEDED** since this will damage the ICs (Integrated circuits) used during the experiments. Incorrect connection of power to the ICs could result in them exploding or becoming very hot - with the **possible serious injury occurring to the people working on the experiment!**

Building the Circuit

Throughout these experiments we will use TTL chips to build circuits. The steps for wiring a circuit should be completed in the order described below:

- **Turn the power (Digital kit) off before you build anything!**
- **Make sure the power is off before you build anything!**
- Connect the +5V (Vcc) and ground (GND) leads of the power supply to the power and ground bus strips on your breadboard. The +5V supply may be found on the top left of the Digital kit with the black switch at the +5V fixed position. Before connecting up, use a voltmeter to check that the voltage does not exceed 5V.

- Plug the chips you will be using into the breadboard. Point all the chips in the same direction with pin 1 at the upper-left corner. (Pin 1 is often identified by a dot or a notch next to it on the chip pa e)

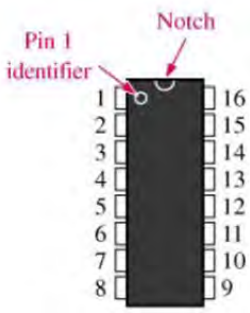


Fig 2. Pin 1 of an IC

- Connect +5V and GND pins of each chip to the power and ground bus strips on the breadboard.
- Select a connection on your schematic and place a piece of hook-up wire between corresponding pins of the chips on your breadboard. It is better to make the short connections before the longer ones. Mark each connection on your schematic as you go, so as not to try to make the same connection again at a later stage.
- Get one of your group members to check the connections, **before you turn the power on**.
- If an error is made and is not spotted before you turn the power on. Turn the power off immediately before you begin to rewire the circuit.
- At the end of the laboratory session, put your hook-up wires and chips in the component tray under your workbench.
- **Tidy the area that you were working in and leave it in the same condition as it was before you started.**

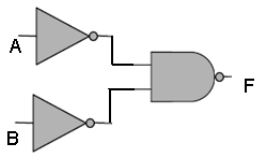
Common Causes of Problems

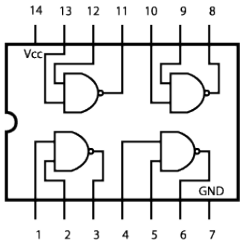
- Not connecting the ground and/or power pins (Vcc) for all chips.
- Not turning on the power supply before checking the operation of the circuit.
- Leaving out wires.
- Plugging wires into the wrong holes.
- Driving a single gate input with the outputs of two or more gates
- Modifying the circuit with the power on.

Please inform the demonstrator if you locate faulty equipment. If you damage a chip, inform the demonstrator in-charge, he/she will replace it with a new one, but you have to bear the cost.

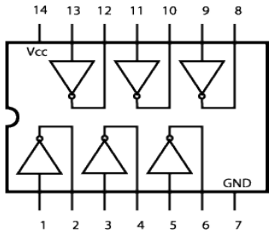
Implementation of a Logic Circuit

Build a circuit to implement the Boolean function F=$\overline{\bar{A}.\bar{B}}$





Quad 2 Input 74LS00



Hex 74LS04 Inverter

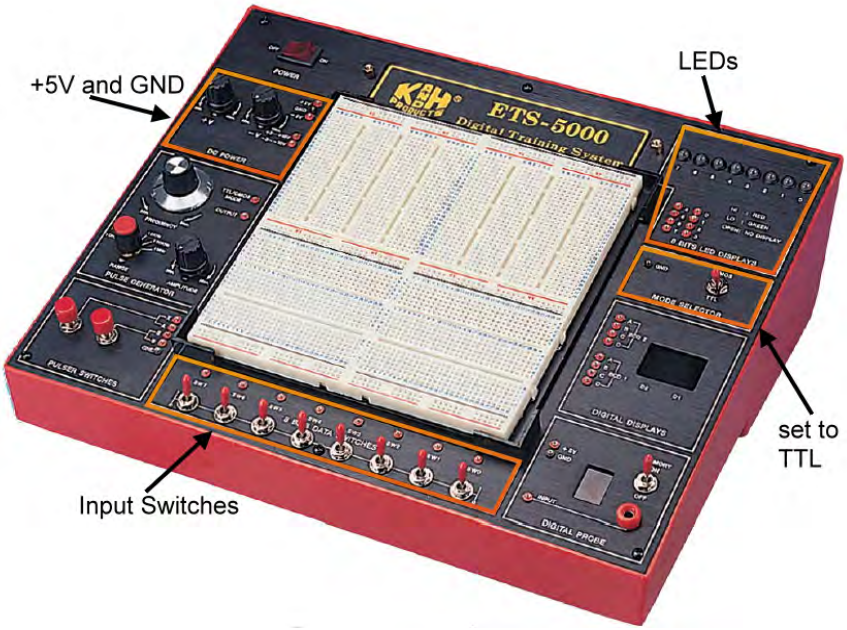


Fig 3. ETS 5000 Digital Training Systems

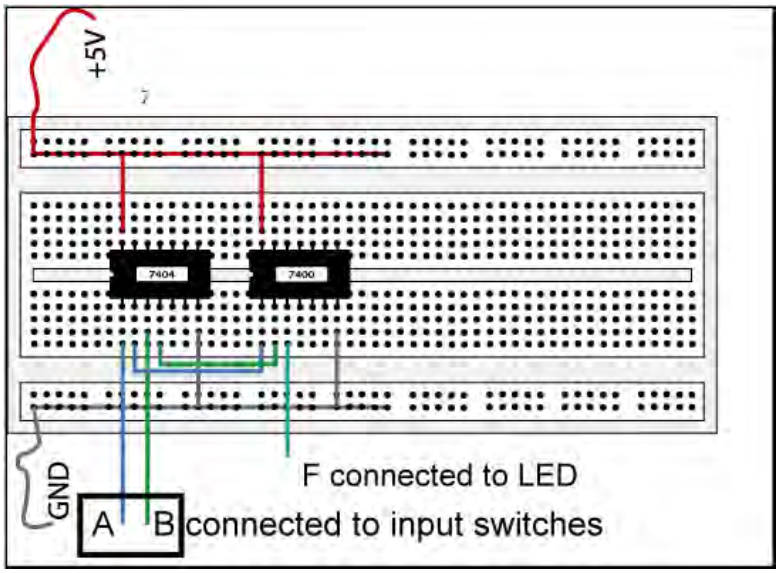
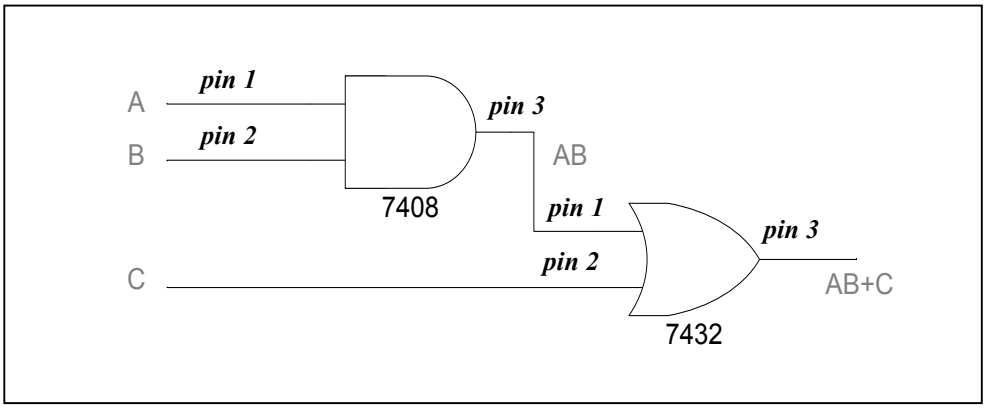


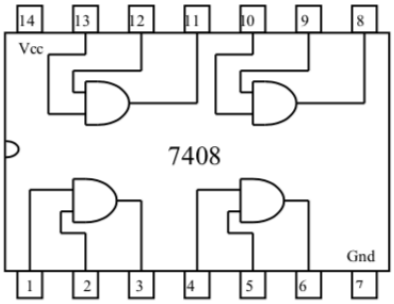
Fig 4. The complete designed and connected circuit, LED switches, +5V and GND from ETS 5000 Digital Training Systems

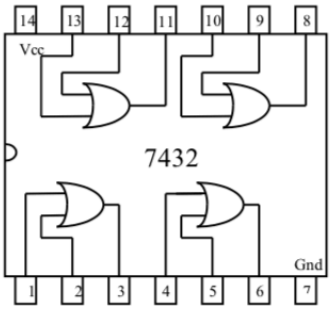
Sometimes the chip manufacturer may denote the first pin by a small indented circle above the first pin of the chip. Place your chips in the same direction, to save confusion at a later stage. Remember that you must connect power to the chips to get them to work.

Another example how to make a circuit connection on a breadboard from a schematic diagram

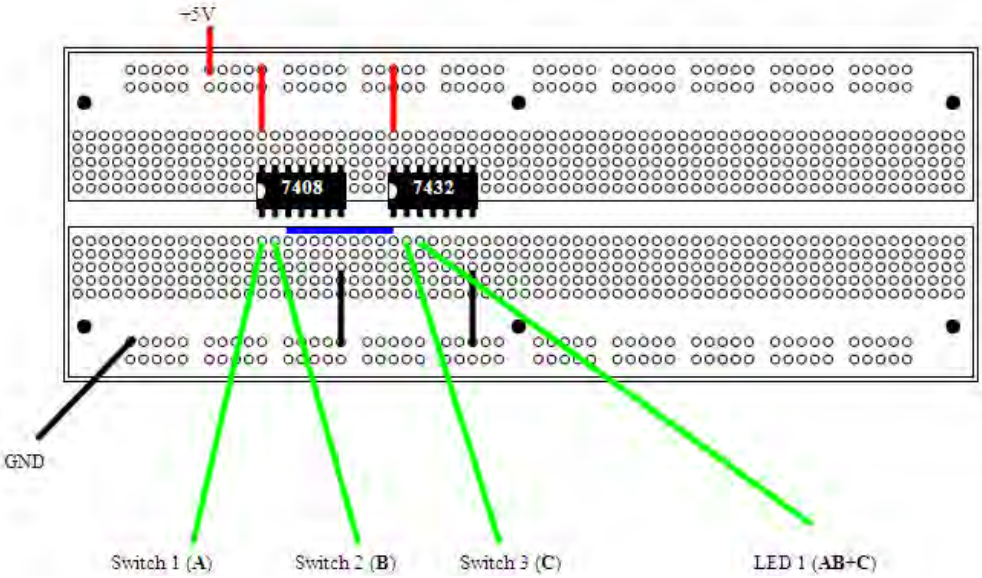
1. Label all the pins of an ICs that will be use in the lab. Refer to the pin out of an ICs







2. Put the ICs on the breadboard and connect the selected pin by using a wire. **Don't forget that each IC must be connected to the power supply**. Vcc connected to 5V and Gnd connected to Gnd of a digital kit. All input connected to a switch and all output connected to the LEDs.



Department of Computer Science
Faculty of Computing
UNIVERSITI TEKNOLOGI MALAYSIA

SUBJECT	:	SCSR1013 DIGITAL LOGIC
SESSION/SEM	:	…….
LAB 1	**:**	**COMBINATIONAL LOGIC**
NAME 1	:	…….
NAME 2	:	…….
DATE	:	…….

REMARKS :

MARKS:

Lab # 1

Introduction to Logic Circuits

A. Objectives

The objectives of this laboratory are to introduce the student to:

- basic bread boarding and wiring techniques
- the use of input switches and output LEDs in generating truth tables for a combinational logic circuit
- to verify the characteristic of the basic gates

B. Materials

- Breadboard
- 7408 Quad 2-input AND - 1
- 7404 Hex Inverter - 1
- 7400 Quad 2-input NAND - 1
- 7432 Quad 2 –input OR - 1
- ETS-5000 Digital Training kit

C. Introduction

This lab focuses on several practical issues related to bread boarding and testing combinational logic circuits. Several helpful points are made below.

Wire gauge - Use only 22 gauge wire. The breadboard may be damaged by forcing smaller gauge (larger diameter) wire into the holes.

Wire color - Use organized color schemes when wiring circuits. For example, use RED wire for all V_{cc} connections, BLACK wire for all ground connections, BLUE wire for all input switches, and YELLOW wire for all intermediate signal connections.

Wire length and placement - Use wires that are the appropriate length so that they can lie flat on the breadboard. Avoid running wires over IC's in case the IC's need to be removed.

Testing IC's - Chip tester is available in lab, always check your IC's before you begin wiring the circuit.

Inserting IC's - IC's are not difficult to insert in the socket strips once they have been properly adjusted. Brand new IC's are shipped with their pins bent apart from the vertical (typically 15° outward) in order to facilitate handling by automatic insertion equipment. Therefore, before an IC is used for the first time its pins must be bent back so that their spacing is vertical.

IC Orientation - Arrange all IC's in the same direction. This will facilitate connecting V_{cc} on each IC to a 5V strip on the breadboard and GND on each IC to a ground strip. If an IC is reversed (thus V_{cc} and ground are reversed), it may be destroyed. It is recommended that you begin wiring by making all V_{cc} and ground connections.

IC Removal - It is recommended that you use some sort of extraction tool for removing IC's. Attempts to remove IC's by hand may result in bent pins.

Monitoring the logic level of the circuit

Use the LED (at the top right of the digital kit) to monitor the logic level of the digital circuit. A GREEN indicates that the logic level is LOW (0) and a RED means the logic level is HIGH (1). If the LED does not lit, there is something wrong with the circuit. Switch off the power supply and recheck the circuit.

Note: Please be certain that the MODE switch is flip to TTL not CMOS.

You can use a logic probe (at the bottom right of the digital circuit) to monitor the logic level of any node (point) of the circuit. **H** indicates logic HIGH (1), **L** means LOW (0) and **P** means pulse (the signal keeps on changing between HIGH and LOW).

Switches

There are 8 toggle switches at the bottom row of the digital circuit. These switches will provide a logic input of HIGH (1) and LOW (0) to the circuit.

D. Preiminary Work

1. Draw a symbol, determine the IC number and produce a truth table for the following gate.

AND

Symbol:

IC Number: ……..

Truth Table 1

Input		Output
A	B	F

NAND

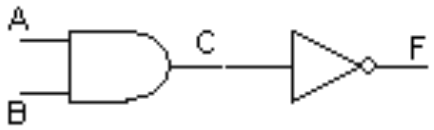
Symbol:

IC Number: ……..

Truth Table 2

Input		Output
A	B	F

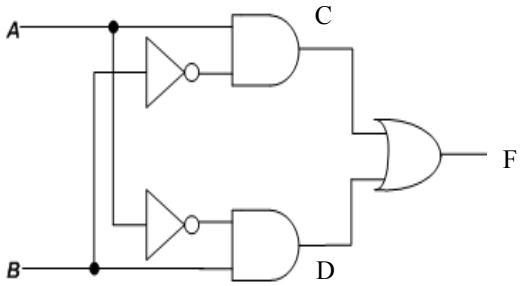
2. Complete the truth table for the following circuit.



Truth Table 3

A	B	C	F

3. Write the Boolean expression for output C, D and F the following circuit.



C =

D =

F =

4. Complete the truth table for the circuit in (3) based on the Boolean expression produced for C, D and F.

Truth Table 4

A	B	C	D	F

E. Laboratory Work

Part 1

1. Construct Circuit 1 on the breadboard. Connect all inputs (A, B) to a switches and output F to LEDs.

Circuit 1

Truth Table 5

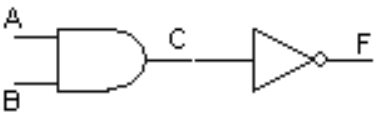
Input		Output
A	B	F

2. Test Circuit 1 and fill in Truth Table 5 for the circuit response to all possible input combinations. The Truth Table 5 should match the Truth Table 1 prepared in the Preliminary Work.

Fully Completed ☐ *Partially Completed* ☐ *Checked by:* _____

Part 2

3. Construct Circuit 2 on the breadboard. Connect all inputs (A, B) to a switches and output C and F to LEDs.



Circuit 2

Truth Table 6

A	B	C	F

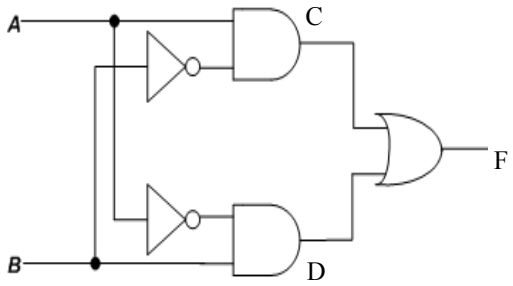
4. Test Circuit 2; fill in Truth Table 6, for the circuit response to all possible input combinations.

5. Compare Truth Table 6 to Truth Table 2. What conclusion can you make?

Fully Completed ☐ *Partially Completed* ☐ *Checked by:* _____

Part 3

6. Construct circuit 3 on the breadboard. Connect all inputs (A, B) to a switches and output C, D and F to LEDs.



Circuit 3

Truth Table 7

A	B	C	D	F

7. Test Circuit 3; fill in Truth Table 7 for the circuit outputs (C, D, and F) for all possible input combinations.

8. What single gate does Circuit 3 represent?

Fully Completed ☐ *Partially Completed* ☐ *Checked by:* _____

Department of Computer Science
Faculty of Computing
UNIVERSITI TEKNOLOGI MALAYSIA

SUBJECT	:	SCSR1013 DIGITAL LOGIC
SESSION/SEM	:	…….
LAB 2	**:**	**COMBINATIONAL LOGIC CIRCUIT DESIGN SIMULATION**
NAME 1	:	…….
NAME 2	:	…….
DATE	:	…….

REMARKS :

MARKS:

Lab # 2

Combinational Digital Circuit Design Simulation Using Deeds Simulator

A. Objective

i) To expose student with producing digital logic circuit, generating truth table and Timing Diagram with Deeds Simulator.

ii) To expose student with a complete cycle process of a combinatorial circuit design and simulate with Deeds Simulator.

B. Material

Install Deeds Software for Windows.

C. Introduction

Deeds Simulator

The Digital Circuit Simulator ***d-DcS*** appears to the user as a graphical schematic editor, with a library of simplified logic components, specialized toward pedagogical needs and not describing specific commercial products. As described before, the schematic editor allows building a simple digital networks composed of gates, flip-flops, pre-defined combinational and sequential circuits and custom-defined components (defined as Finite state machine).

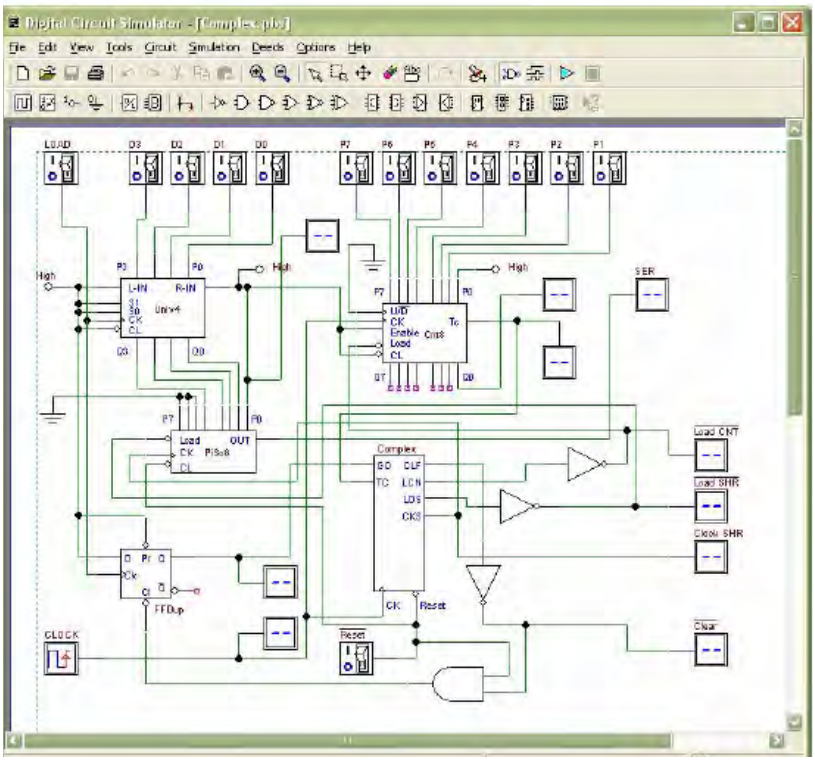


Figure 1 Circuit Editor of Digital Circuit Simulator (d-DcS)

Simulation can be interactive or in timing-mode. In **interactive mode**, the student can *"animate"* the digital system in the editor, controlling its inputs and observing the results. This is the simplest mode to examine a digital network, and this way of operation can be useful for the beginners. In **timing mode**, the behavior of the circuit can be analyzed by a timing diagram window, in which the user can define graphically an input signal sequence and observe the simulation results.

Digital Circuit Simulator (d-DcS): A Simple Example

In the following screen shots (Figure 2a, 2b, and 2c), student can see the circuit during the drawing and then simulate it by animation with following simple steps:

a) student picks-up components from the *Component Tool Bar*.
b) connects them using *Wires*.
c) student activates the animation.

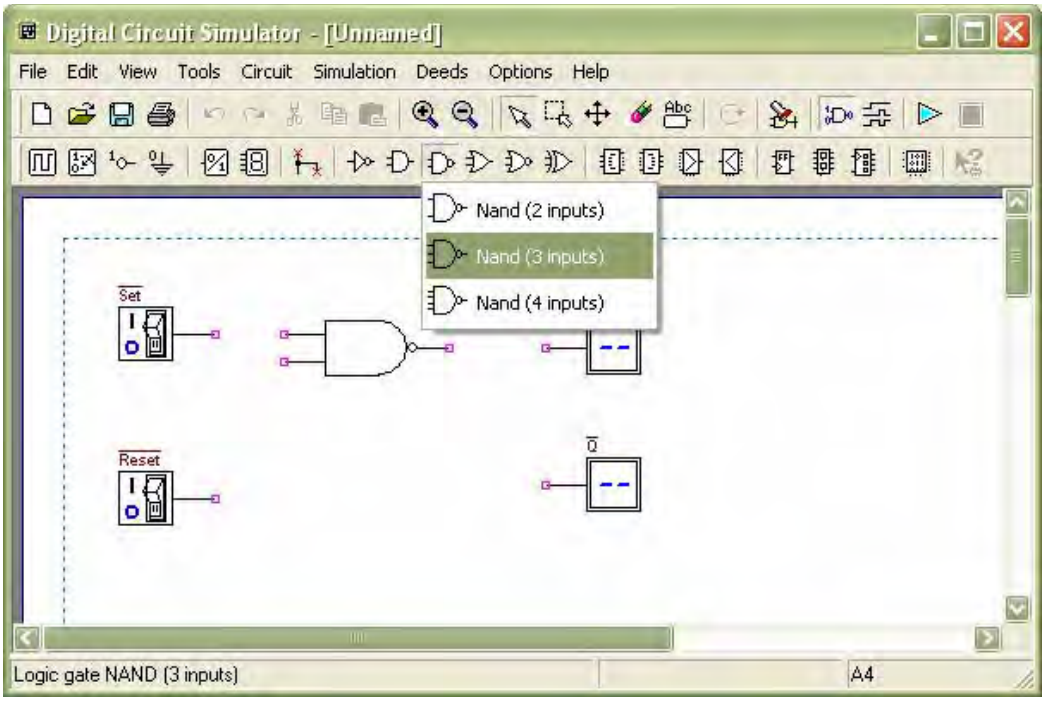


Figure 2a Drawing Phase of the Digital Circuit Editor: Insertion of Components

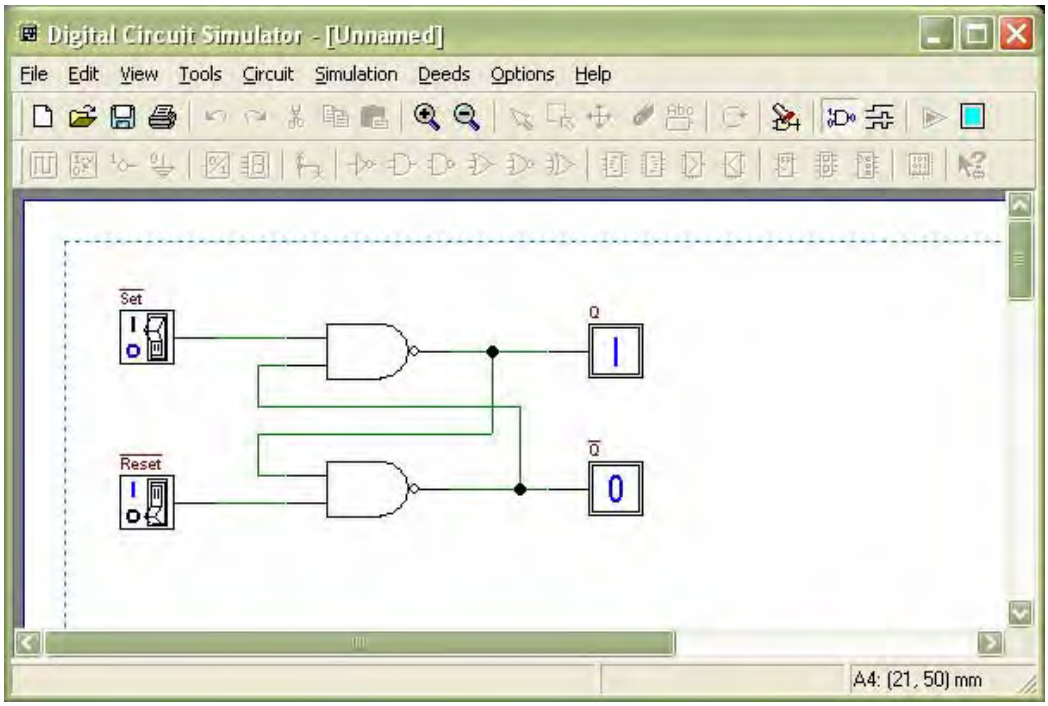


Figure 2b Next Phase of the Work: Connection of Components using *Wires*

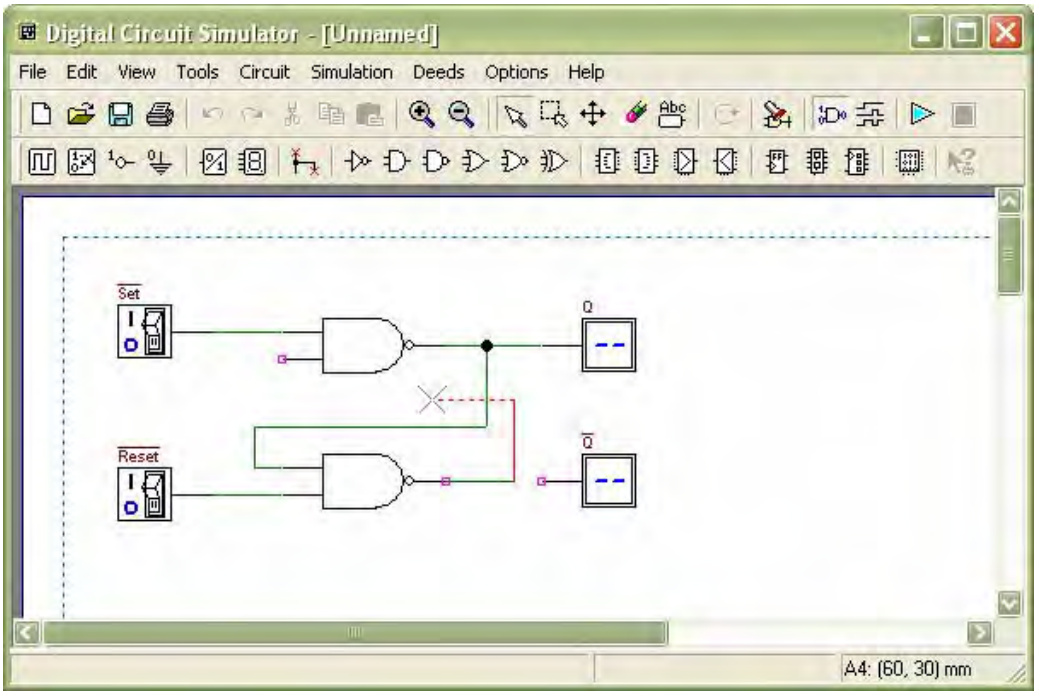


Figure 2c Animation: User Switches Inputs and the Circuit Shows Changes on Outputs

To exit the animation, it is necessary to click on the square *Stop* button. Instead, if the timing simulation is to be performed, student should click on the *Timing Simulation* button. This will show the Timing Diagram simulation window (Figure 3).

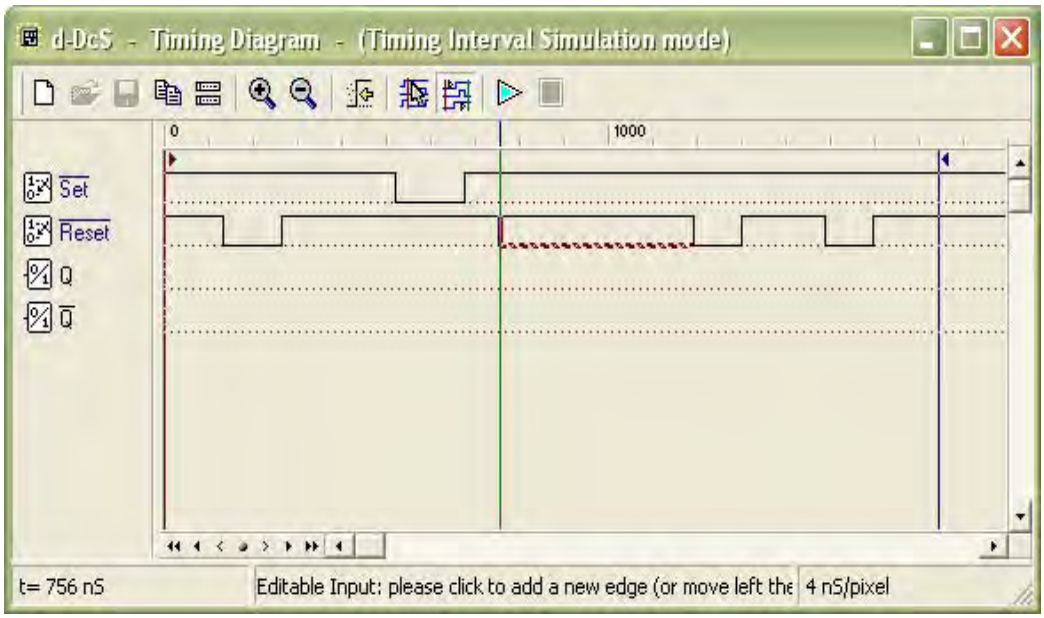


Figure 3 Timing Diagram Simulation Window

In this window, first student should define the timing of the input signals, drawing them on the diagram with the mouse. A vertical line cursor permits to define the 'end time' of the simulation. When student clicks on the triangular *play* button on the toolbar, the simulation is executed, and its results are displayed in the same window (Figure 4).

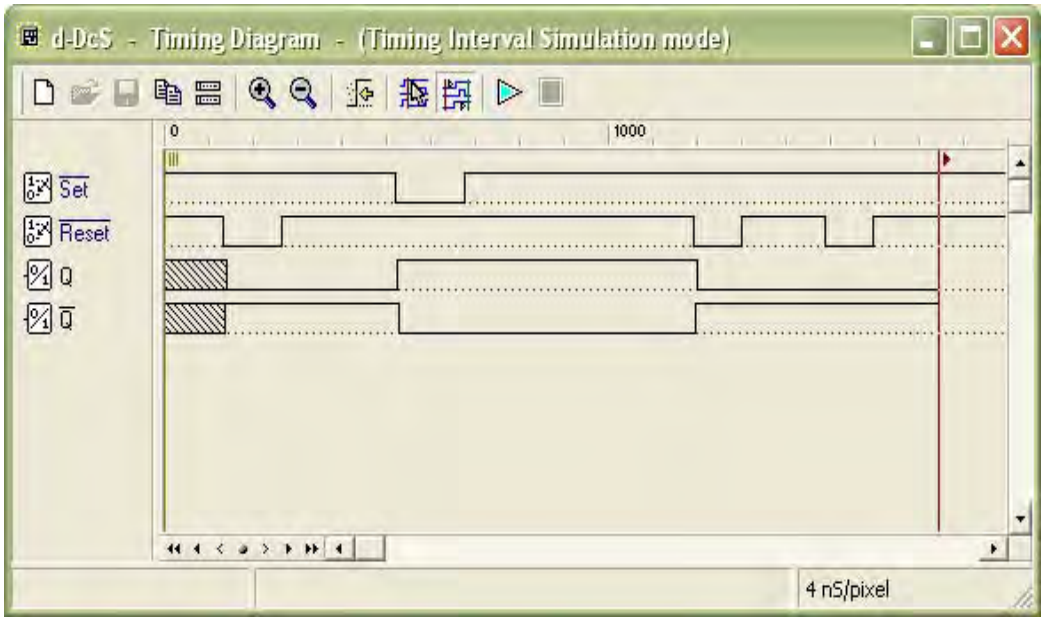


Figure 4 Timing Simulation Results, Displayed in Timing Diagram Window

Student can verify the correct behavior of the digital circuit, comparing simulation results with reasoning and theory concepts.

D. Lab Activities

Part 1

Simulating logic circuit, construct truth table and timing diagram with Deeds.

Given Boolean expression as follow:

$$Y = AB + BC + AC$$

1. Convert the non-standard Boolean expression into standard form.

2. Based on standard form expression, complete the following truth table.

INPUT			OUTPUT
A	**B**	**C**	**Y**

3. Using Deeds Simulator, draw the following circuits:

a) Circuit (i) for non-standard form (based on the given expression).

Circuit (i)

b) Circuit (ii) for standard form (from your answer in question (1)).

Circuit (ii)

4. Simulate these two circuits in step (3) and complete their truth table.

Compare the simulation result for these two truth tables. What is your conclusion?

Circuit (i)

INPUT			OUTPUT
A	B	C	Y

Circuit (ii)

INPUT			OUTPUT
A	B	C	Y

Conclusion:

_____

_____

_____

5. Simulate output of circuit (ii) with Timing Diagram. Illustrate some examples of different inputs and output.

Part 2

Combinational circuit design process and simulate with Deeds Simulator.

Design Process

i) Determine Parameter Input / Output and their relations.
ii) Construct Truth Table.
iii) Using K-Map, get the SOP optimized form of all Boolean equation outputs.
iv) Draw the circuit and use duality symbol; convert AND-OR circuit to NAND gates ONLY.
v) Simulate the design using Deeds Simulator. Check the results according to Truth Table and Timing Diagram Operation.

Problem Situation

A new digital fault diagnoses circuit is requested to be designed for analyzing four bit 2's complement input binary number from sensors A, B, C, and D. Sensor A represents input MSB and sensor D represents input LSB. As shown in the following Figure 5, bit pattern analysis from input sensors A, B, C, and D will trigger four different output errors (active HIGH) of type E1, E2, E3, and E4.

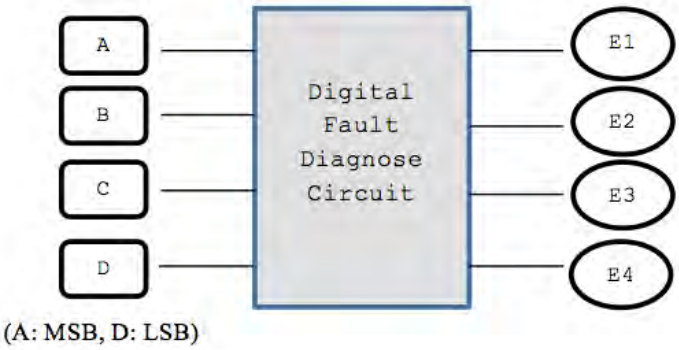


Figure 5

The following rules are used to activate the error's signal type:

RULE 1: E1 is activated if the input number is positive ODD and the majority of the bits is '0'.
RULE 2: E2 is activated if the input number is positive EVEN and the majority of the bits is '0'.
RULE 3: E3 is activated if the input number is negative ODD and the majority of the bits is '1'.
RULE 4: E4 is activated if the input number is negative EVEN and the majority of the bits is '1'.
RULE 5: The output of error signal is invalid if the input has equal bit '0' and bit '1'.
(**NOTE:** Positive ODD is positive numbers that are odd and negative EVEN is negative numbers that are even).

Experimental Steps

1. Complete Truth Table 1 for Digital Fault Diagnose Circuit. Use variables A, B, C and D as inputs; E1, E2, E3 and E4 as outputs.

Truth Table 1

INPUTS				OUTPUTS			
A	**B**	**C**	**D**	**E1**	**E2**	**E3**	**E4**

2. Using K-MAP, get minimized SOP Boolean expressions for E1, E2, E3 and E4 circuits.

3. From the Boolean expression in the step (2), draw your final E1, E2, E3 and E4 circuits using 2 input basic gates (AND, OR, NOT). Use Deeds Simulator.

4. Simulate the Deeds circuit in step (3):

a) Update Truth Table 2 based on the simulation result.

Truth Table 2

INPUTS				OUTPUTS			
A	**B**	**C**	**D**	**E1**	**E2**	**E3**	**E4**

Compare the output results in Truth Table 2 with Truth Table 1. What is your conclusion?

b) Timing Diagram

Explain some analysis values based on your timing diagram:

5. Using dual symbol concept, convert your circuit in step (3) to NAND gates only. Use Deeds Simulator.

6. Simulate the Deeds circuit in step (5):

a) Update Truth Table 3 based on the simulation result.

Truth Table 3

INPUTS				OUTPUTS			
A	**B**	**C**	**D**	**E1**	**E2**	**E3**	**E4**

Compare the output results in Truth Table 3 with Truth Table 2. What is your conclusion?

b) Timing Diagram

Explain some analysis values based on your timing diagram:

Fully Completed ☐ Partially Completed ☐ Checked by: _____

Department of Computer Science
Faculty of Computing
UNIVERSITI TEKNOLOGI MALAYSIA

SUBJECT	:	SCSR1013 DIGITAL LOGIC
SESSION/SEM	:	……
LAB 3	**:**	**SYNCHRONOUS DIGITAL COUNTER**
NAME 1	:	……
NAME 2	:	……
DATE	:	……

REMARKS :

MARKS:

Lab #3

Identifying The Properties of a Synchronous Counter

A. Aims

1) Expose the student with experience on constructing synchronous counter circuit using Flip-Flop IC, Basic Gate ICs, Breadboard and ETS-5000 Digital Kit.
2) Promote critical thinking among students by analyzing the given circuit and identifying the behavior of the digital circuit.

B. Objectives

The objectives of this lab activity are to:

1) Implement a synchronous counter circuit into physical circuit using Breadboard, Flip-Flops, Basic Gates and Switches.
2) Completing the next-state table of the counter circuit.
3) Sketch the state diagram of the counter circuit.
4) Identify the properties of the counter.

C. Materials And Equipments

Materials and equipment required for this lab are as follows:

Item Name	Number of Item
1. Breadboard	1
2. 7408 Quad 2-Input AND	1
3. 7404 Hex Inverter	1
4. 7432 Quad 2-input OR	1
5. 7476 Dual J-K Flip Flop	1
6. ETS-5000 Digital Kit	1

D. Introduction

Flip-Flops

A flip-flop is the simplest type of memory cell. Its output, Q, does not depend solely upon its inputs, but also depends on the order in which they are applied. Thus, the flip-flop is not a combinational circuit, but is a sequential circuit. The flip-flop is the key building block of most synchronous sequential circuits. There are four common types of flip-flops. The symbol and truth table for each is shown in Figure 1.

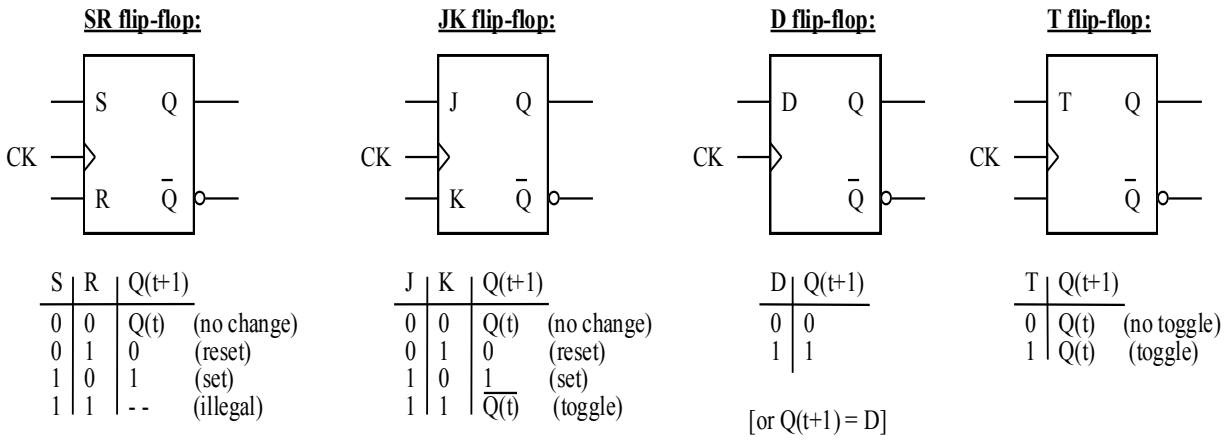


Figure 1: Four common types of flip-flops

Flip-flops are synchronous devices meaning that the output responds to the synchronous inputs (S, R, J, K, D or T) only on certain clock edges. There are three main types of triggering:

1) **positive-edge triggering** - the output Q can only change on the positive (rising) edge of the clock (due to the values of the synchronous inputs).
2) **negative-edge triggering** - the output Q can only change on the negative (falling) edge of the clock (due to the values of the synchronous inputs).
3) **master-slave triggering** - the synchronous inputs are "*read*" on the positive edge of the clock, but the output Q does not respond until the negative edge of the clock.

The type of triggering is sometimes indicated by the symbol. Shown below in Figure 2 are JK flip-flops with all three types of triggering.

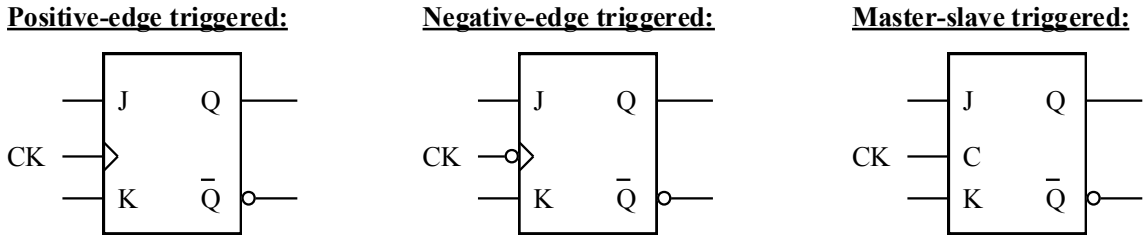


Figure 2: JK Flip-flops with different types of triggering

Flip-flops often have asynchronous inputs available also. These inputs are not synchronized with the clock, therefore, the output may respond immediately to changes in these inputs. There are two types of asynchronous inputs commonly used:

1) **PRESET** (also called **SET**) - used to preset the output Q to 1
2) **CLEAR** (also called **RESET**) - used to clear the output (set Q to 0)

Asynchronous inputs are often active-LOW. Therefore, they are typically tied HIGH for normal flip-flop operation. The PRESET or CLEAR may be momentarily set LOW to initialize the flip-flop to some desired initial value.

The symbol for a flip-flop often shows the asynchronous inputs as indicated below in Figure 3.

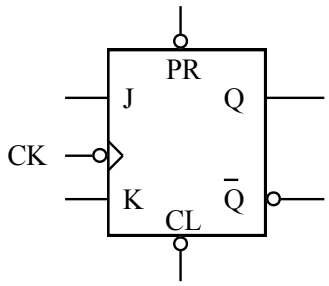


Figure 3: JK Flip-flops with asynchronous PRESET and CLEAR inpu

E. Preliminary Works

1) Using 7476 IC, connect the synchronous input (J, K) of a JK flip-flop to switches and its output (Q) to an LED. Connect the CLK input to a pulser switch, $\overline{A}$. Determine the logic level for each input combinations in Table 1 so that the desired result can be realized.

Table 1

Desired Result	$\overline{PRE}$	$\overline{CLR}$	J	K	CLK	Q
Set initial value Q = 1			X	X	--	
Output Q stays the same	1	1			⇓	
Output Q become 0, no change in asynchronous input					⇓	
Output Q is not the previous Q	1	1			⇓	
RESET Q	1	1			⇓	
SET Q	1	1			⇓	

2) Answer all questions.

a) Which state that JK flip-flop has, but not on SR flip-flop.

_____

b) Identify whether the JK flip flop in 7476, is a positive-edge triggered or negative-edge triggered flip flop.

_____

F. Lab Activities

1) You are given a counter circuit as shown in Figure 4.

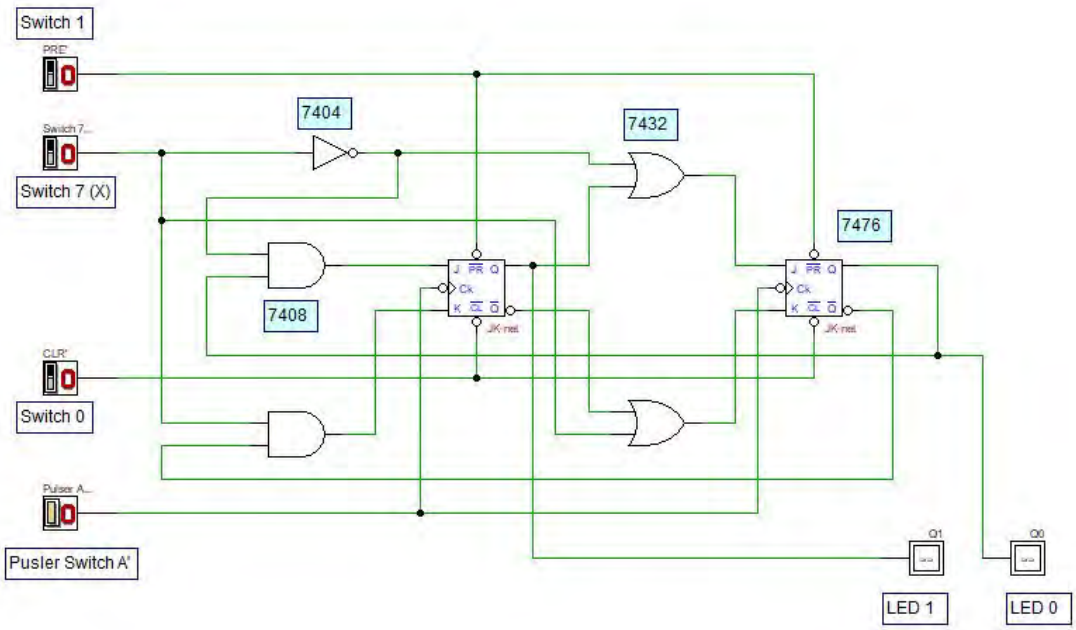


Figure 4: A Synchronous Counter Circuit

2) By using all materials and equipment's listed in section C, construct the physical circuit of Figure 4. (Make sure all ICs are connected to Vcc and GND).

3) Investigate the behavior of the counter by observing the next state of the counter for all combination of *Present State* and *X* values. Complete the *Next-State* table of the counter in Table 2. Ensure the Switch 0 is in HIGH state. (0=LOW, 1=HIGH)

Table 2

Switch 7	Present State		Next State	
X	Q1 LED 1	Q0 LED 0	Q1 LED 1	Q0 LED 0
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Fully Completed ☐ *Partially Completed* ☐ *Checked by:* _____

4) By referring to the *Next-State* in Table 2, sketch the state diagram of the counter.

5) By referring to the *Next-State* in Table 2 and the state diagram in (4), answer all questions.

   a) What is the main indicator to decide that the counter is a synchronous counter?

   b) How many states are available for the counter and what are they?

   c) What is the function of Switch 7 (X) in the circuit?

   d) What is the function of Switch 0 and Switch 1 in the circuit?

   e) Is the counter a saturated counter or recycle counter?

Fully Completed ☐ *Partially Completed* ☐ *Checked by:* _____

Department of Computer Science
Faculty of Computing
UNIVERSITI TEKNOLOGI MALAYSIA

SUBJECT	:	SCSR1013 DIGITAL LOGIC
SESSION/SEM	:	…….
LAB 4	**:**	**MINI PROJECT** **PHOTOCOPYING (XEROX) MACHINE**
NAME 1	:	…….
NAME 2	:	…….
DATE	:	…….

REMARKS :

MARKS:

Mini Project: Programmable Logic Device (PLD)
Photocopying (Xerox) Machine

A. Objectives

The objectives of this laboratory are to introduce the students to:

- The development of a PLD device.
- A simple Hardware Description Language.

B. Materials

Breadboard
GAL 22V10
ETS-5000 Digital Training kit
Wellon Universal Programmer & Tester
WinCUPL 5.0 Software
Handouts:

- ***"WinCUPL user manual"***
- ***"GAL22V10 Data Sheet"***
- ***"How to use Wellon Programmer"***
- ***"How to use Win CUPL 5"***

As an appendix in your Lab Book

C. Mini Project : Photocopying (Xerox) Machine

Background

Read the appendix in your lab book to familiarize yourself with the WinCUPL compiler and the universal programmer. If you need any help please ask your lecturer or the teaching assistants.

This mini project will implement 3 different components on a single GAL device, those components are

- 3-bit Count Up Counter
- 3-bit Comparator
- Clock Disabler

Problem:

User will initially enter amount of copies, the counter will count the number of copies that has been photocopied. The machine will stop once the required number of copies produced.

Proposed Solution:

A block diagram of a component required is shown in Figure 1. Three core components are counter, comparator and clock disabler. Counter will determine the number of copies that has been made while the comparator will determine whether the required number of copies has been met. Once the number of copies has been met, the clock disabler will disable the clock and stop the counter from counting. The machine will display the required number of copies and the amount that has been produced.

To implement this, the user has to key in the required number of copies by using 3 switches, which allow the required number from 0 to 7. He/she will reset the counter (assuming pressing a START button), the counter will count and it's output will be compared with the value of the switches, if the value is not the same it will continue counting up. A signal will be generated to stop the counting if the output of the counter has the same value as the digital value of the switches.

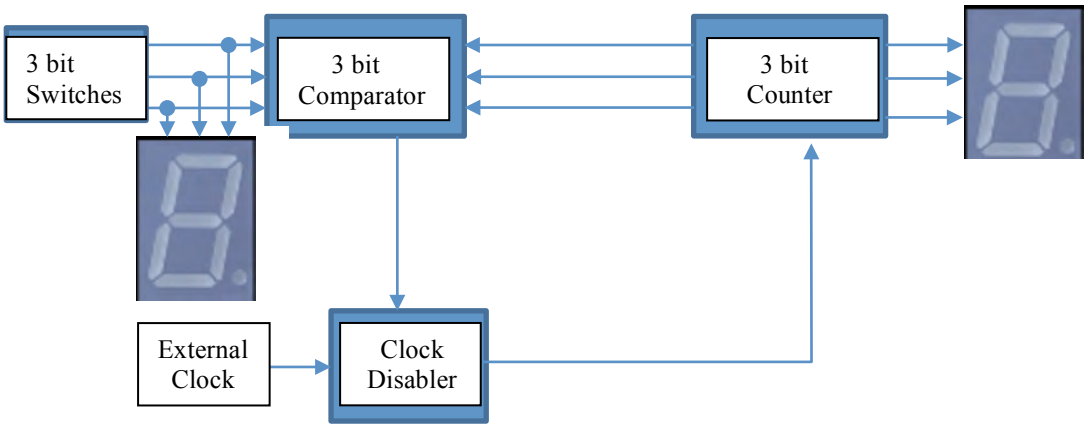


Figure 1 A block diagram of a complete system

Components & Requirement:

- ✓ **Switches:** to set the required no of copies
- ✓ **Counter:** to count the number of copies that has been made
- ✓ **Comparator:** to compare the number of copies required with the number of copies produced
- ✓ **Clock Disabler:** to stop the operation of the counter

If Copies produced < Required Copies, Counter will count up
Else
Counter will stop counting

Implementation

Switches: Use switches provided by the ETS 5000 Digital Training Kit.

Counter: Counter is a count up 3 bits counter that was controlled by the clock. Counter is implemented as a WinCUPL code that will be programmed into GAL.

```
/** 2 bits count up saturated counter **/
/** Define state counter     **/
field countP = [Q1..0];
$define s0 'b'00
$define s1 'b'01
$define s2 'b'10
$define s3 'b'11

/** Define Asynchronous input: asynchronous input Active High  **
/
countP.sp = preset;
countP.ar = reset;

/** State Counter Sequence       **/
sequence countP {
present s0 if clkEn next s1;     /** 0 --> 1 **/
        default next s0;
present s1 if clkEn next s2;     /** 1 --> 2 **/
        default next s1;
present s2 if clkEn next s3;     /** 2 --> 3 **/
        default next s2;
present s3 if clkEn next s4;     /** 3 --> 3 **/
        default next s3;
}
```

Comparator: A comparator is a combinational circuit that can be designed using *XNOR* gate. Figure 3 shows a 2 bit equality Comparator, you need to modify this circuit to suit the requirement of your design.

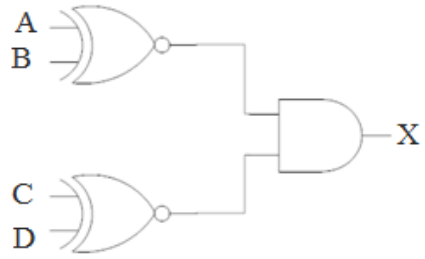


Figure 3 2bits Equality Comparator

The circuit in Figure 3 can be translated to a WinCUPL code as follows:

```
/** 2 bit comparator **/
X = !(A $ B)&!(C $ D);
```

Clock Disabler: Clock Disabler is a circuit to disable the clock so that the counter will stop counting. Design your own Clock Disabler.

Hint: *The counter will count if there are an edge of the clock signal, how to eliminate that edge when you want to stop the counting and allow it when you want the counter to keep on counting.*

D. <u>Mini Project: The Practical Work</u>

1. a) Type your code using WinCUPL.
   b) Compile the program. The compiler will generate .JED (jedec) file.
   c) Burn the .JED file to GAL22V10D using a Wellon programmer in the lab
   d) Test the circuit by using a Digital Training Kit.
      i. Connect Clock pin to a pulser switch
      ii. Connect switches input to a toggle switch
      iii. Monitor the logic value of the switches and the output of the counter by connecting it to the 7 segment display (digital display on the kit)
      iv. Don't forget to connect the Vcc and GND
      v. ***GAL22V10 is a CMOS device, handle with care***. Flip Mode selector switch to CMOS. If your counter miss a counting sequence you can try to flip the switch to TTL

2. You must bring your source code (.PLD) and a jedec file (.JED) in electronic form to the lab before lab begins.

Practical Works

Part I

PLD Code: 2 Bit Comparator (Detect same value of 2 input)

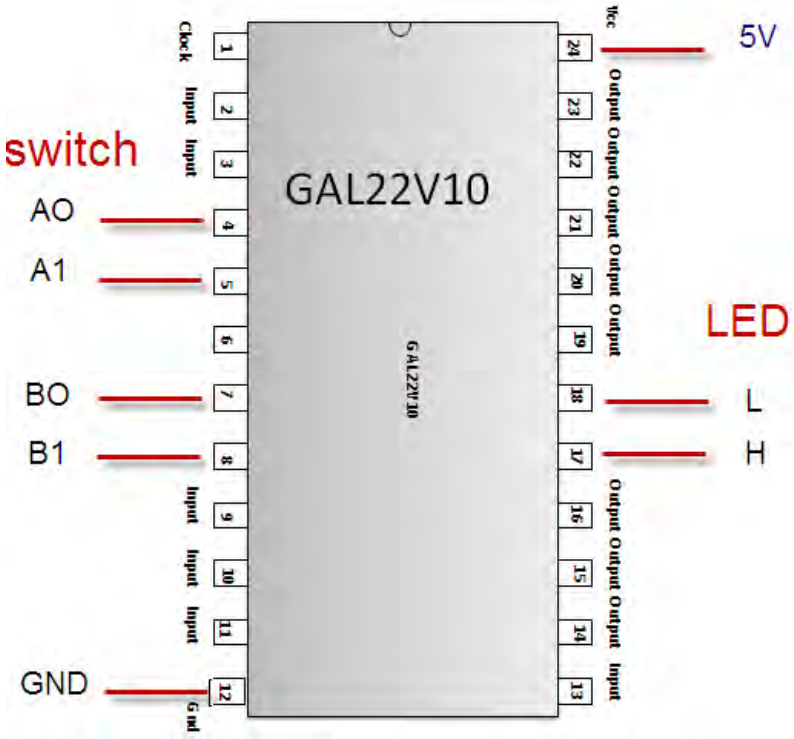
- ✓ A=B, Output=1 (Output active HIGH)
- ✓ A=B, Output=0 (Output active LOW)

The steps to code PLD programming and circuit implementation on ETS5000 training kit are as follows:

1. Start a new project in WinCUPL, overwrite the generated code
2. Compile the code and bring the *XXX.JED* file to the lab and program the GAL22V10 with the code
3. Test your circuit by connecting the IC as follows, connect the comparator output to LEDs

```
Name     lab3 2011 ;
PartNo   00 ;
Date     05/12/2011 ;
Revision 01 ;
Designer Engineer ;
Company  UTM ;
Assembly None ;
Location  ;
Device   G22V10 ;

/* *************** COMPARATOR  ******************/
/* *************** INPUT PINS *********************/
PIN 4 = A0;
PIN 5 = A1;
PIN 7 = B0;
PIN 8 = B1;

/** OUTPUTS  PIN  ********************************/

PIN 17 = H;       /** comparator output: Output Active High  **/
PIN 18 = L;       /** comparator output: Output Active High  **/

/** 2 bit comparator: If A=B, Output Comparator=1 **/
H = !(A1 $ B1)& !(A0 $ B0);
/** 2 bit comparator: If A=B, Output Comparator=0 **/
L = !( !(A1 $ B1)& !(A0 $ B0));
```

Circuit implementation on ETS5000 Training kit



Testing steps:

a. Set the number of required copies by flipping the switch (eg**:** A_1A_0 = 10 ; 2 copies)
b. Set different value of B_1B_0 and compare the value in A_1A_0 . What is the value of L and H.

A_1	A_0	B_1	B_0	L	H
1	0	0	0		
		0	1		
		1	0		
		1	1		

Part II

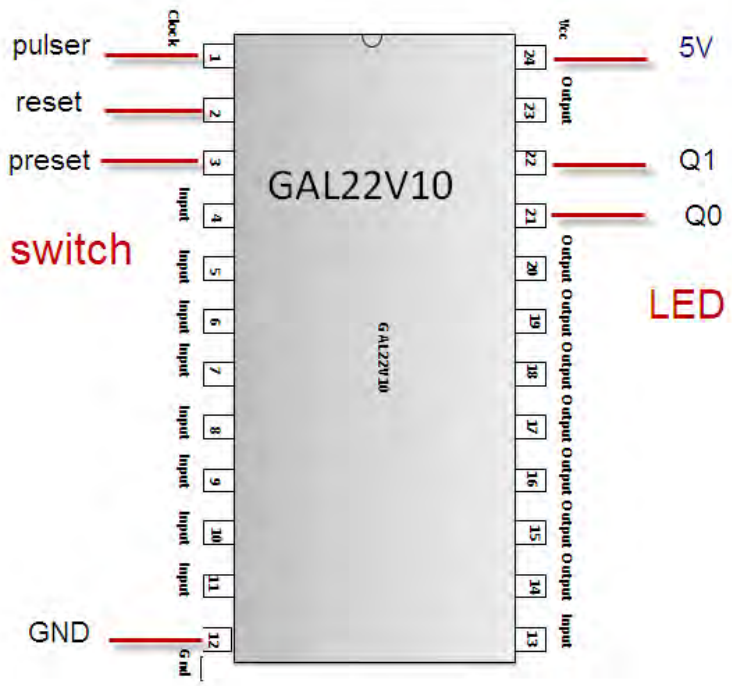
The steps to code PLD programming and circuit implementation on ETS5000 training kit are as follows:

1. Start a new project in WinCUPL, overwrite the generated code
2. Compile the code and bring the *XXX.JED* file to the lab and program the GAL22V10 with the code
3. Test your circuit by connecting the IC as follows, connect the 2 bit Synchronous Count-up Counter output to LEDs

PLD Code : 2 bit Synchronous Count-up Counter

```
Name     lab3 2011 ;
PartNo   00 ;
Date     05/12/2011 ;
Revision 01 ;
Designer Engineer ;
Company  UTM ;
Assembly None ;
Location  ;
Device   G22V10 ;


/* *************** INPUT PINS *********************/
/*counter input */
PIN 1 = clk;
PIN 2 = reset;
PIN 3 = preset;
/** counter output **/
PIN 21 = Q0;
PIN 22 = Q1;

/** 2 bits count up counter **/
/** Define state counter     **/
field countP = [Q1..0];
$define s0 'b'00
$define s1 'b'01
$define s2 'b'10
$define s3 'b'11

/** Define Asynchronous input: asynchronous input Active High  **/
countP.sp = preset;
countP.ar = reset;

/** State Counter Sequence       **/
sequence countP {
present s0 if clkEn next s1;     /** 0 --> 1 **/
        default next s0;
present s1 if clkEn next s2;     /** 1 --> 2 **/
        default next s1;
present s2 if clkEn next s3;     /** 2 --> 3 **/
        default next s2;
present s3 if clkEn next s4;     /** 3 --> 3 **/
         default next s3;
 }
```

Circuit implementation on ETS5000 Training kit



Testing steps:

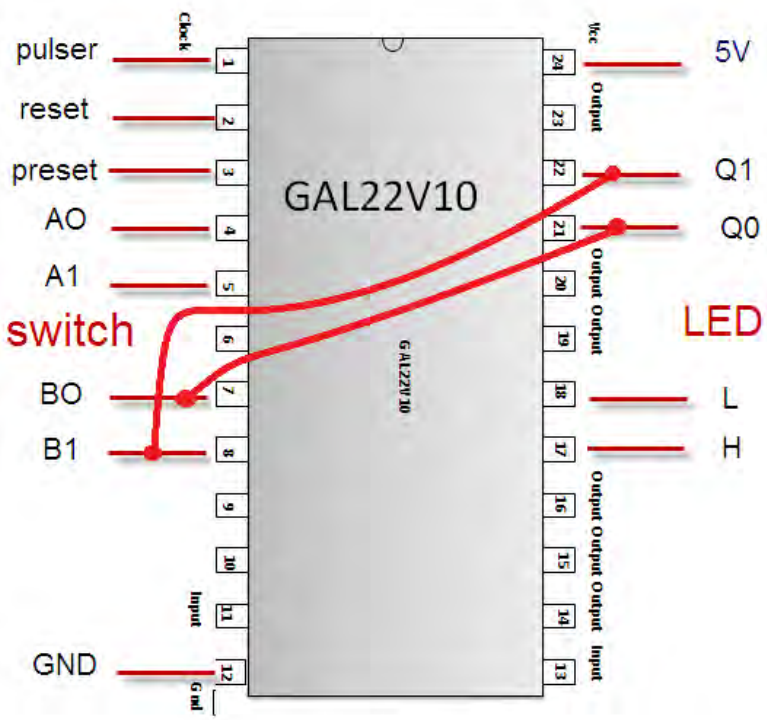
a. Set the Asynchronous pin switch input to the correct setting.
   Make sure the initial output of the counter in ZERO.
b. Press the pulser switch, it will increment the counter (00 → 01 → 10)
c. Counter output equals comparator will change as count up counter.

Part III

Combination of Comparator and Counter

```
Name     lab3 2017 ;
PartNo   00 ;
Date     05/11/2017 ;
Revision 01 ;
Designer Engineer ;
Company  UTM ;
Assembly None ;
Location  ;
Device   G22V10 ;

/* *************** COMPARATOR  ******************/
/* *************** INPUT PINS *********************/
PIN 4 = A0;
PIN 5 = A1;
PIN 6 = B0;
PIN 7 = B1;
/* *************** OUTPUT PINS ********************/
PIN 17 = H;       /** comparator output: Output Active High  **/
PIN 18 = L;       /** comparator output: Output Active High  **/

/** 2 bit comparator: If A=B, Output Comparator=1 **/
H = !(A1 $ B1)& !(A0 $ B0);
/** 2 bit comparator: If A=B, Output Comparator=0 **/
L = !( !(A1 $ B1)& !(A0 $ B0));

/* *************** INPUT PINS *********************/
/*counter input */
PIN 1 = clk;
PIN 2 = reset;
PIN 3 = preset;
/** counter output **/
PIN 21 = Q0;
PIN 22 = Q1;

/** 2 bits count up counter **/
/** Define state counter     **/
field countP = [Q1..0];
$define s0 'b'00
$define s1 'b'01
$define s2 'b'10
$define s3 'b'11

/** Define Asynchronous input: asynchronous input Active High  **/
countP.sp = preset;
countP.ar = reset;

/** State Counter Sequence       **/
   sequence countP {
   present s0 if clkEn next s1;     /** 0 --> 1 **/
           default next s0;
   present s1 if clkEn next s2;     /** 1 --> 2 **/
           default next s1;
   present s2 if clkEn next s3;     /** 2 --> 3 **/
           default next s2;
   present s3 if clkEn next s4;     /** 3 --> 3 **/
            default next s3;
}
```

Circuit implementation on ETS5000 Training kit



Testing steps:

a. Set the Asynchronous pin switch input to the correct setting. Make sure the initial output of the counter in ZERO.
b. Set the number of required copies by flipping the switch (eg: A_1A_0 = 10 ; 2 copies)
c. Press the pulser switch, it will increment the counter (00 → 01 → 10)
d. When the counter output equals to 2 (B_1B_0= 10) the output of the comparator will change (for L: red to green, for H: green to red)

Comment: For the second part of your assignment either one (only one!!) of the output of the comparator must be connected to the clock disabler.

Part IV

Extend your works in Part III.

Do the following:

a) Change the 2-bits up counter from Figure 2 to 3-bits up counter
b) Using Figure 3 as a guide design for a 3 bits comparator and write the equation into your WinCUPL code.
   Hint: "if find too many product terms, DeMorgan might be your best friend"
c) Design a clock disabler (a very simple circuit) and translate the circuit into a WinCUPL code.

(* You can choose the pin to be assigned to each variable by referring to GAL22V10 manual.*)**

Test your circuit on ETS5000 Training kit.

Part V

Enhanced Design Features Using Deeds Software

Based on the original design in Figure 1, add more components with different functions (features) to the existing circuit by using components that you have learnt in this course.

Some of the ideas are:

a. Security/Authentication (Pattern Recognition-Decoder/Encoder)
b. More indicator –
   i. Count-up (increasing seq)
   ii. Count-down (decreasing seq)
c. Output more than 1 level (Mux/DeMux)
d. Online monitoring
e. Alert Messagging/Buzzer
f. Etc

Your design must be uniqued and you can discussed with your lecturer.

Simulate your enhanced circuit design.

LAB REFERENCES

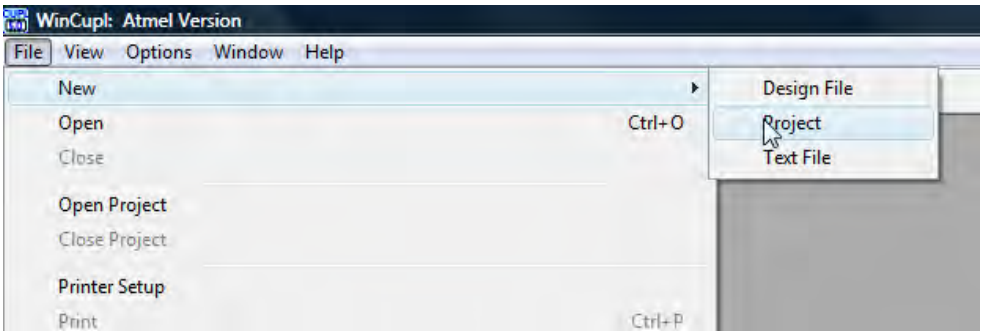
#1 How to use WinCUPL 5.30.4

How to use WinCUPL 5.30.4

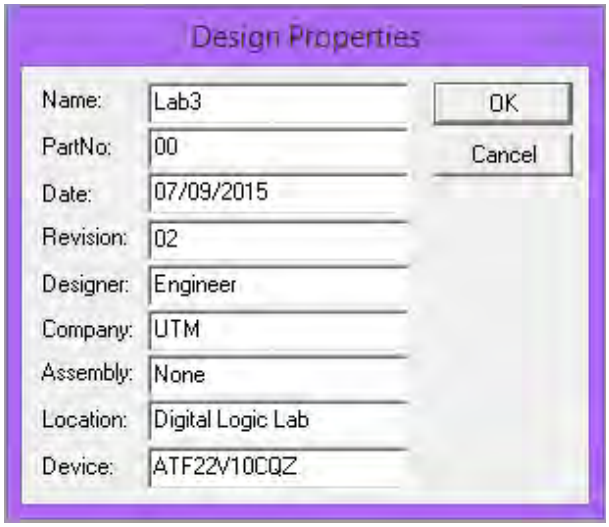
1. Install the software using the serial number given in the folder.

2. Start WinCUPL.

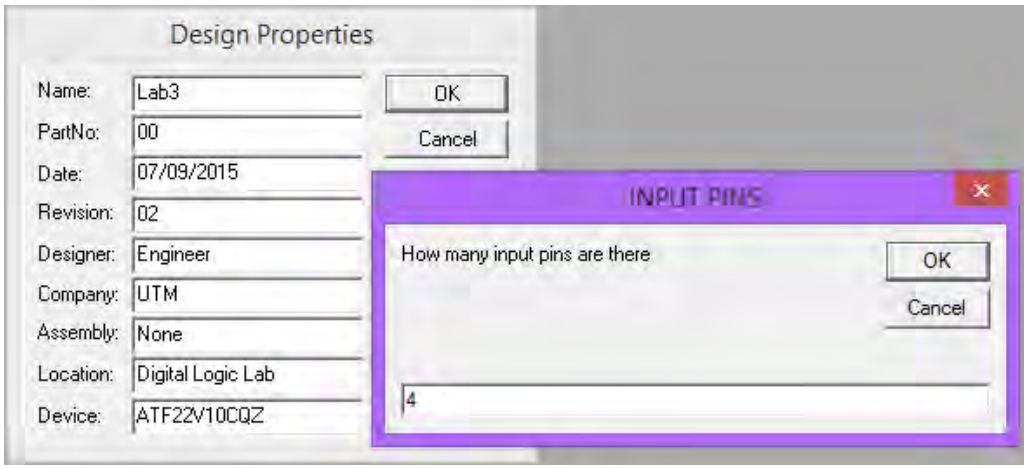
3. Start a new project : File → New → Project



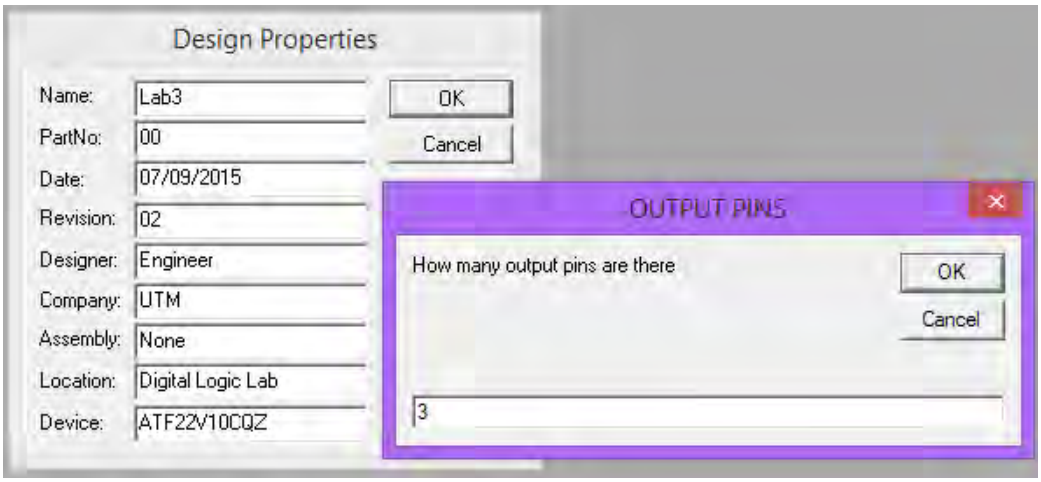
4. Key in the information, the most important is the **Name** and **Device.** The device must be **ATF22V10CQZ.**



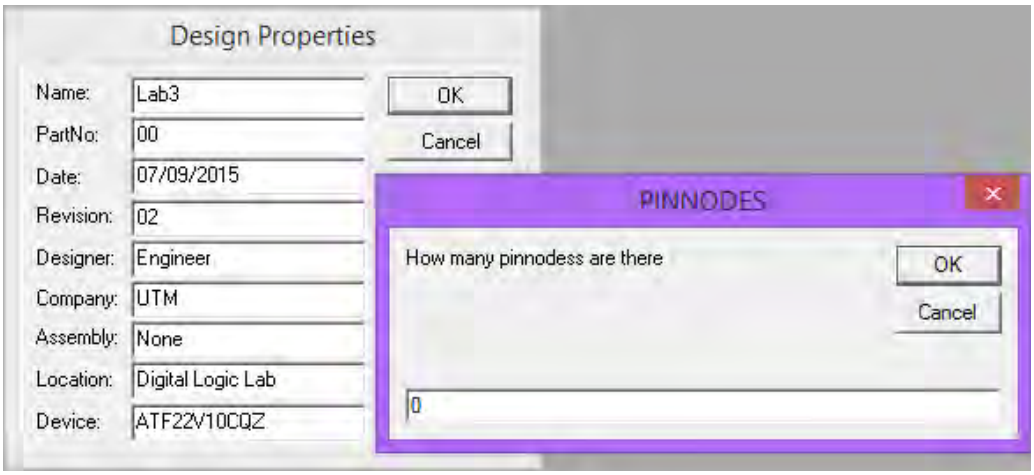
5. Key in the number of input pins. Depending on your design (program in the Appendix), the number of input pin is 4.



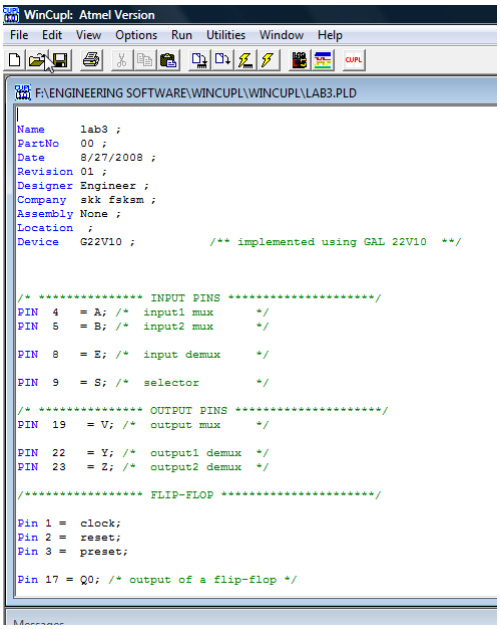
6. Key in the number of output pins. Depending on your design (program in the Appendix), the number of output pin is 3.



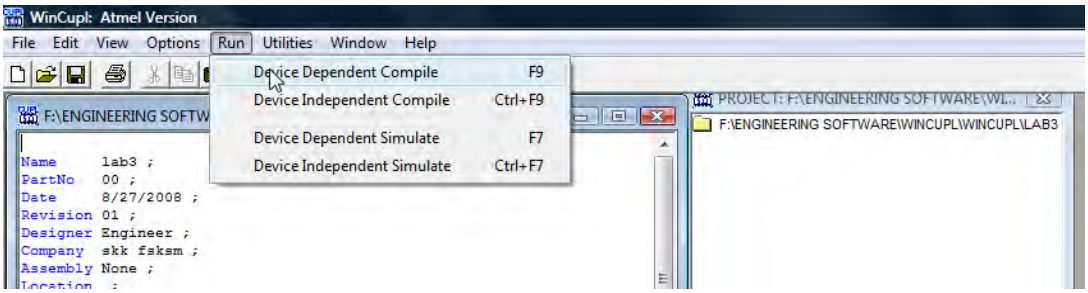
7. Since there is no pin nodes in the program, key in 0.



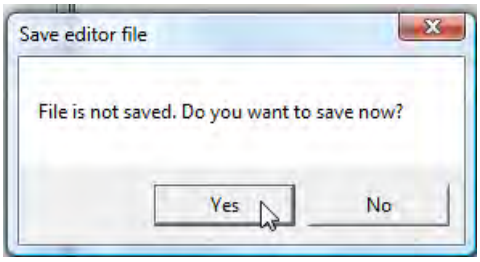
8. A template will appear in the editor. Type in your program.



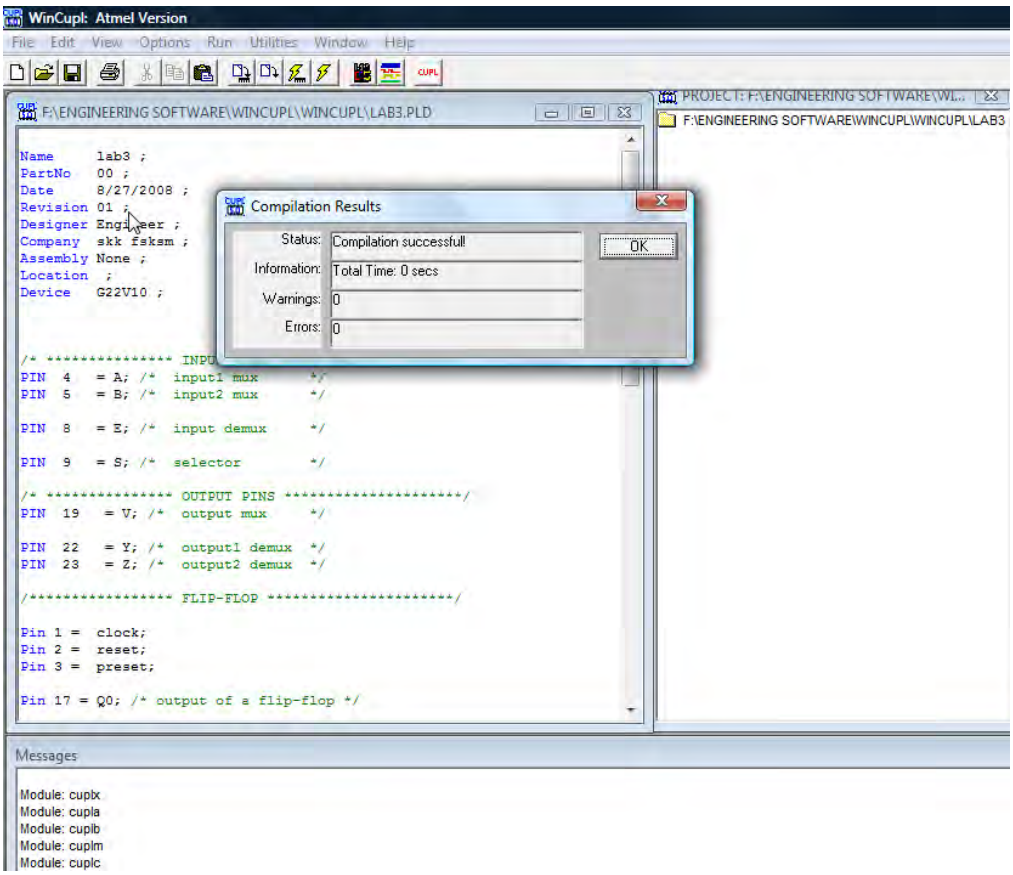
9. After you finish editing, you need to compile the program. Press F9 or Click Run → Device Dependent Compile.



10. Press **Yes** when asked to save your file.

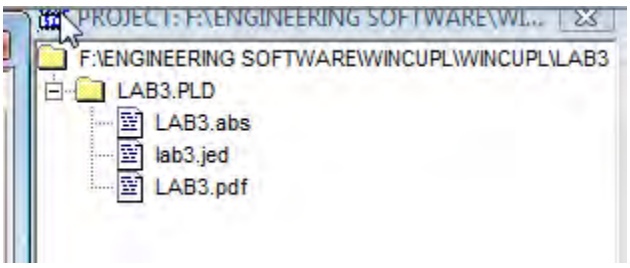


11. WinCUPL will compile the program and display a compilation result.



12. After a successful compilation, a jedec (.JED) file will be created. This file will be needed to program the PLD device by using a universal programmer such as Wellon Programmer.

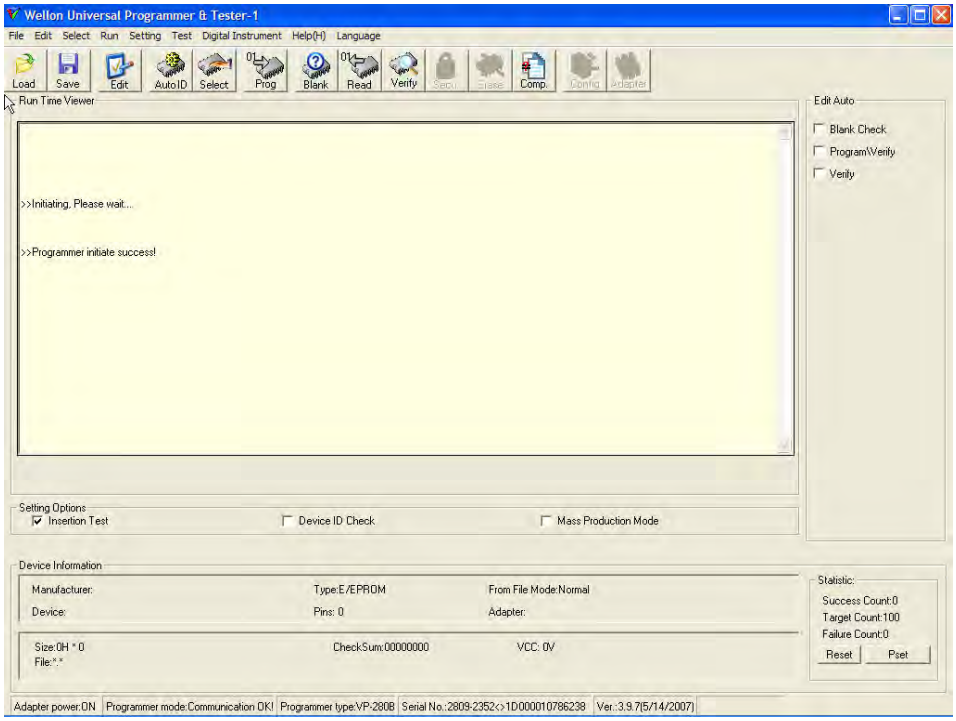
Please read “How to use Wellon Programmer”.



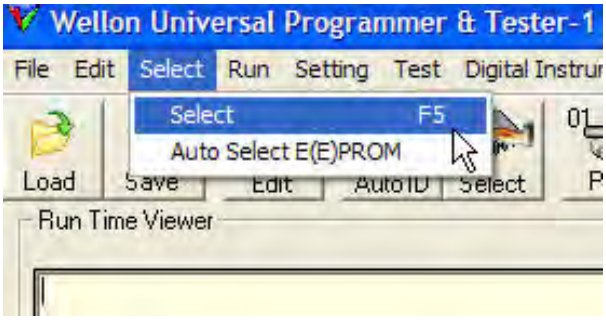
#2 How To Use Wellon Programmer

1. Leave the socket on the programmer empty and the lever straight up.

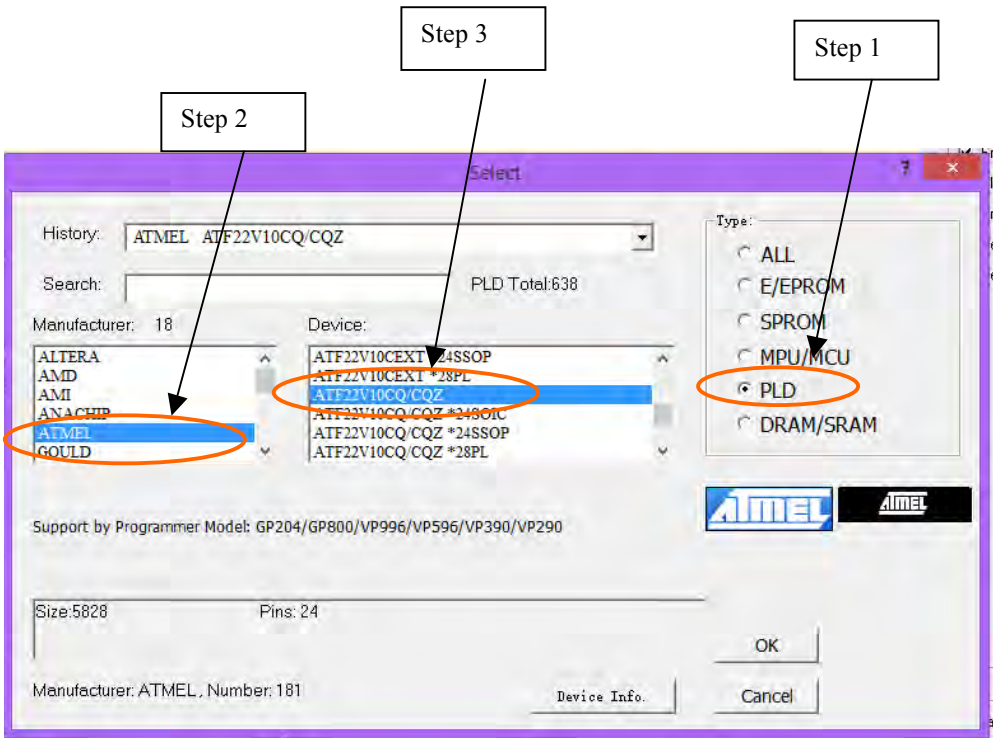
   Double click on the Wellon icon ( ), that will execute the software and initialize it.



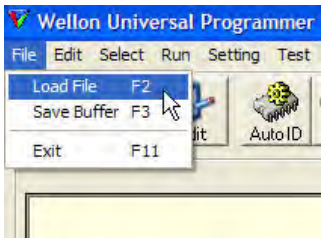
2 Select type of IC that you want to program by clicking Select → Select.



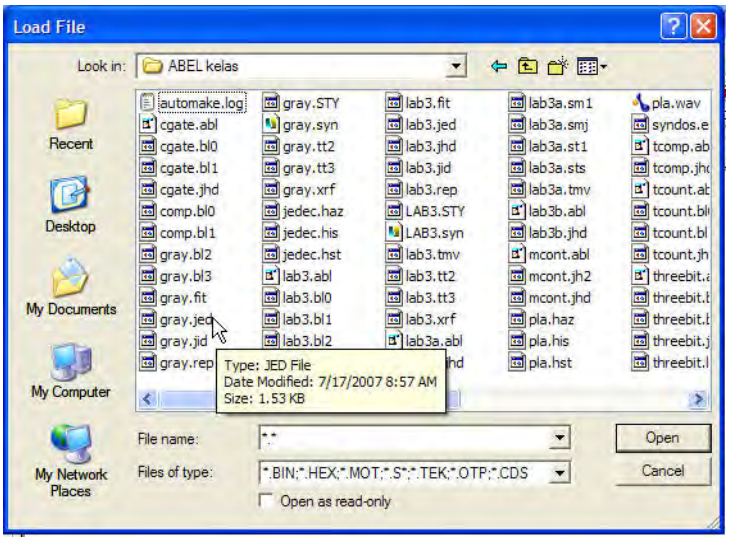
3. Then Click PLD under menu Type. The software will load the necessary driver for that particular device.



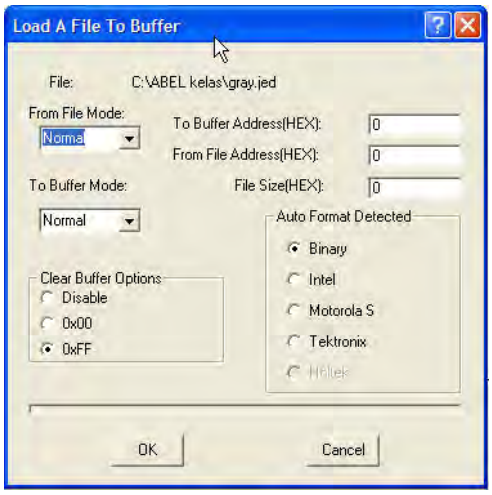
4. Load JEDEC (.JED) file. Click File → Load File.



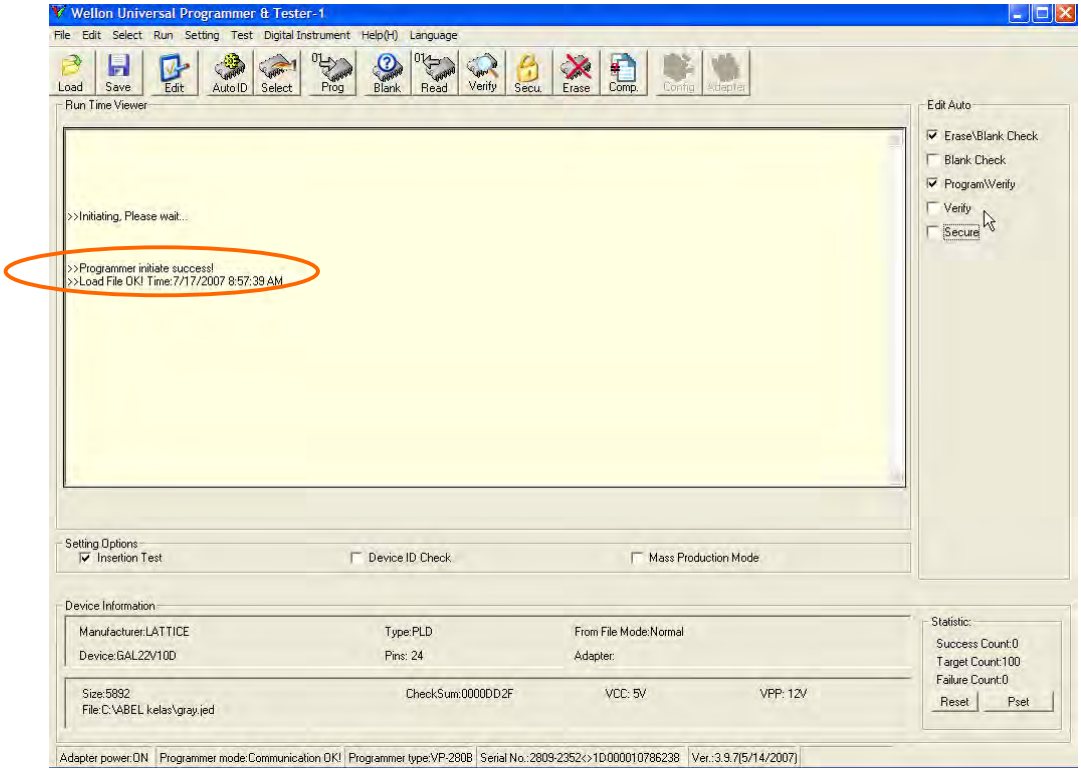
5. Select which JEDEC file you want to load.



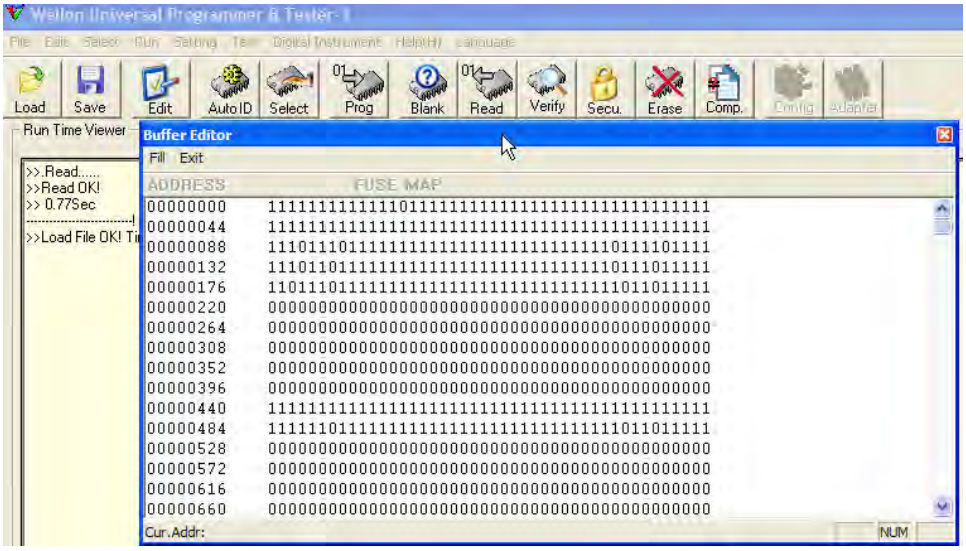
6. A new window will pop-up, simply click OK.



7. File will be loaded to the programmer's buffer. If loading of file successful then a message will be displayed.



8. You can check the content of the programmer's buffer by clicking *Edit* → *Edit Buffer*. If the content of the buffer all 0's or all 1's it mean there is a problem during loading. Your buffer should have the same pattern of 0's and 1's as in your .JED file.



9. **Programming the IC**.

Insert the IC that you want to program into the programmer socket. Push the lever from a vertical position to a horizontal position. Please be certain that the IC pin inserted properly and it makes a good contact with the programmer's socket.

Under ***Edit Auto*** field, check ***Erase/Blank*** and ***Program/Verify***. This option will blank check the IC to determine whether it is blank, if not it will erase it, then program the IC and verify that it's content is the same as in the programmer's buffer after the programming process. The entire step will be executed in sequence automatically.

To execute the sequence selected in ***Edit Auto*** field, click button . All the sequence will be executed and the result will be displayed.

Note:
If you prefer to do it manually then you can do all the steps one by one by

i. Blank check , click button

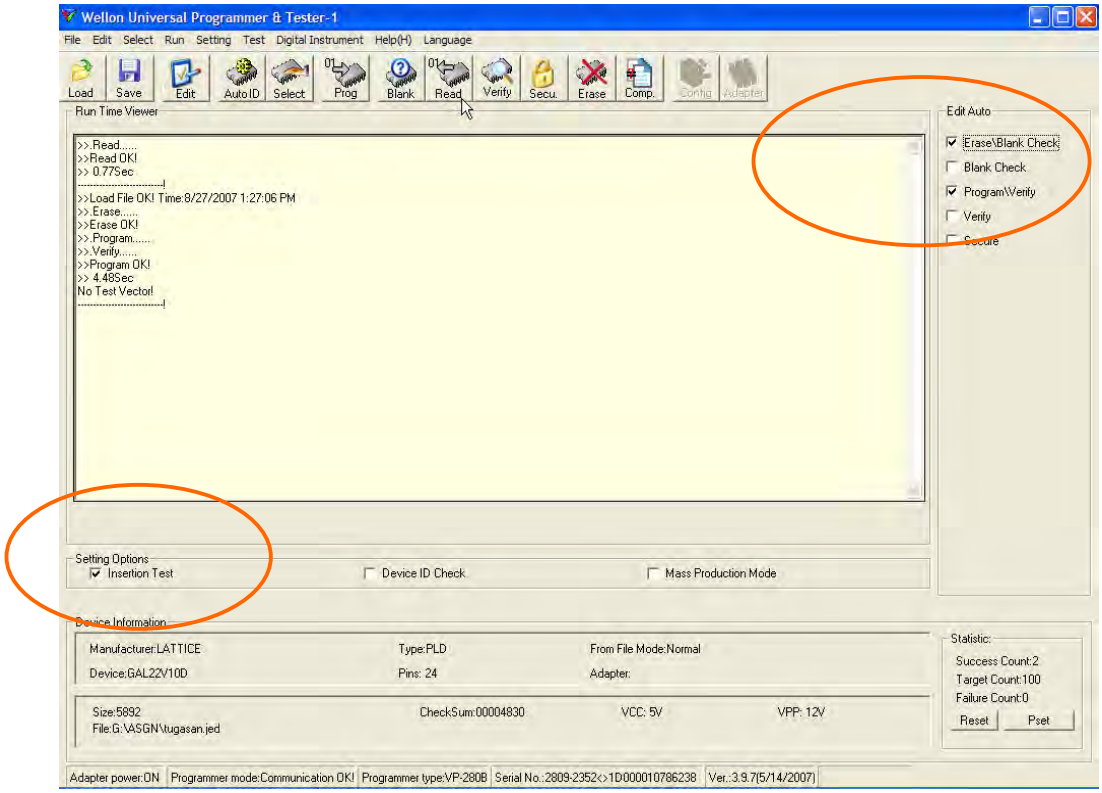
ii. Erase the IC, click button

, (careful – it will erase the content of the IC)

iii. Manually program the IC, by clicking button

, but you have to clear all the check mark in Auto Edit field first.

iv. Finally verify that the programming is successful by clicking button



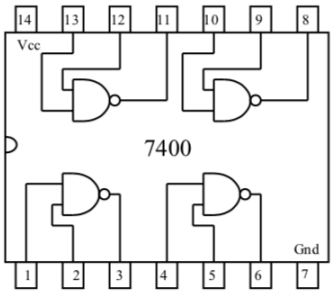
10. The programming of the IC completed. Pullback the socket lever from a horizontal position to a vertical position. Take out the IC from the socket.

If there is a communication error during the programming, please disconnect the USB cable and reconnect it again.

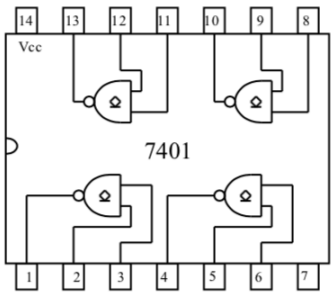
Lab Reference Sheets

This document contains useful information for constructing and testing digital logic circuits, including:

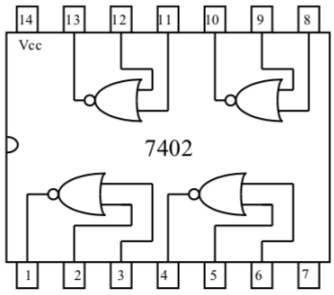
- **Pinouts**
- **Capacitor markings and values**
- **Standard 5% and 10% Resistor and the resistor color code**
- **Breadboard and an example circuits wired on the breadboard**
- **Building a 5V power supply using a 9V battery and a 5V regulator**



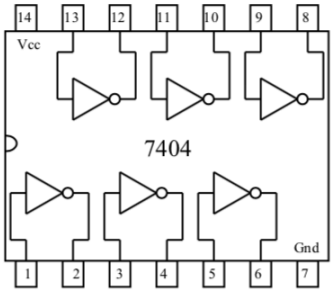
7400 Quad 2-input NAND



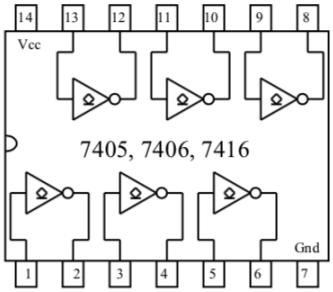
7401 Quad 2-input NAND
(open-collector outputs)



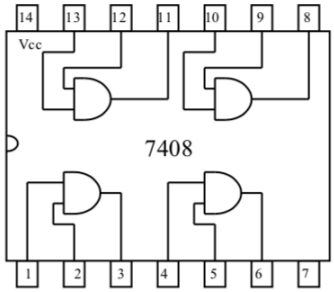
7402 Quad 2-input NOR



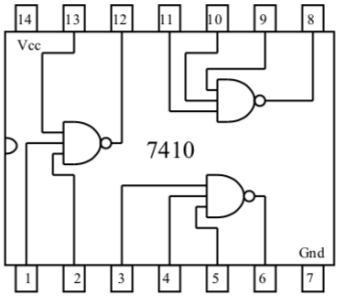
7404 Hex Inverter



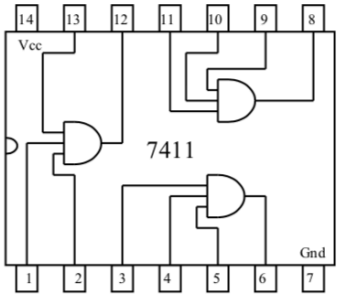
7405, 7406, 7416 Hex Inverter
(open-collector outputs)



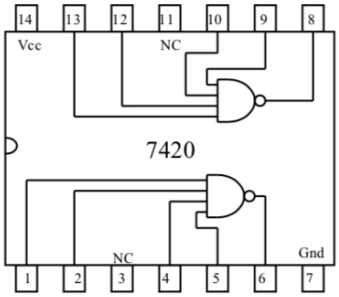
7408 Quad 2-input AND



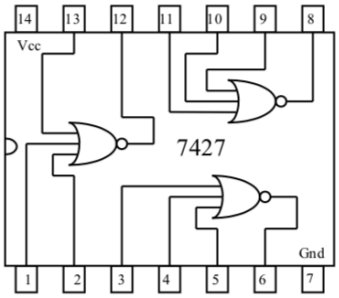
7410 Triple 3-input NAND



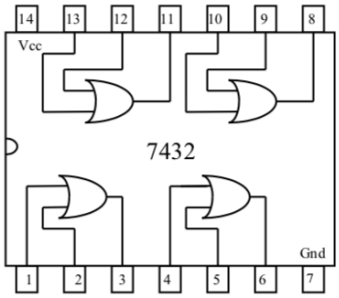
7411 Triple 3-input AND



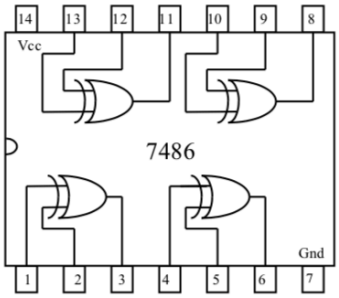
7420 Dual 4-input NAND



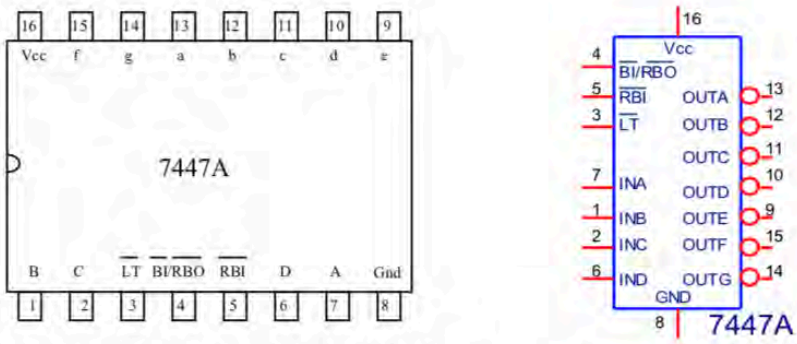
7427 Triple 3-input NOR



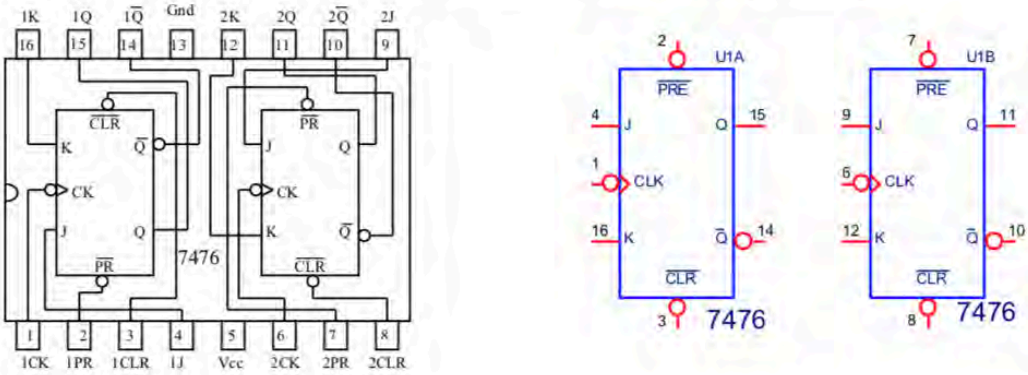
7432 Quad 2-input OR



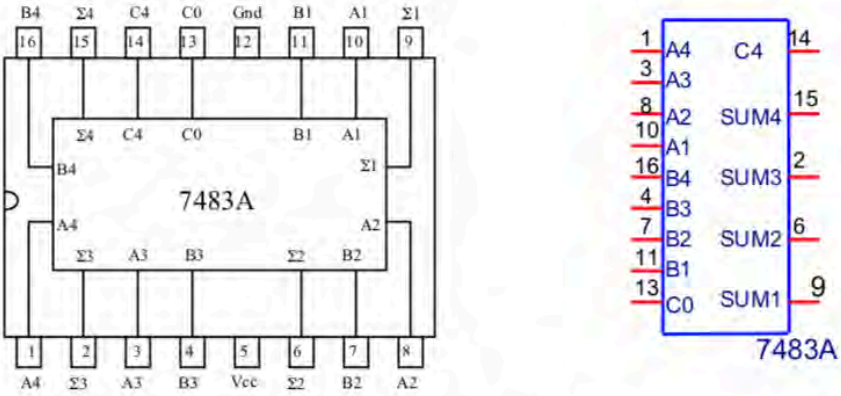
7486 Quad 2-input Exclusive-OR



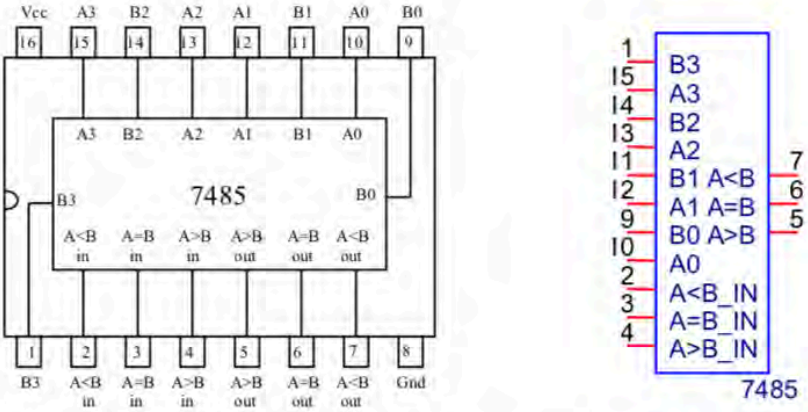
7447A BCD to 7-segment decoder/driver: Pinout and Logic Symbol



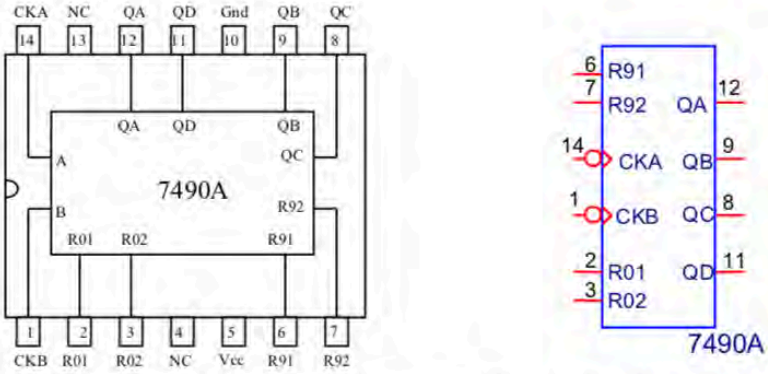
7476 Dual JK Flip-Flop: Pinout and Logic Symbols



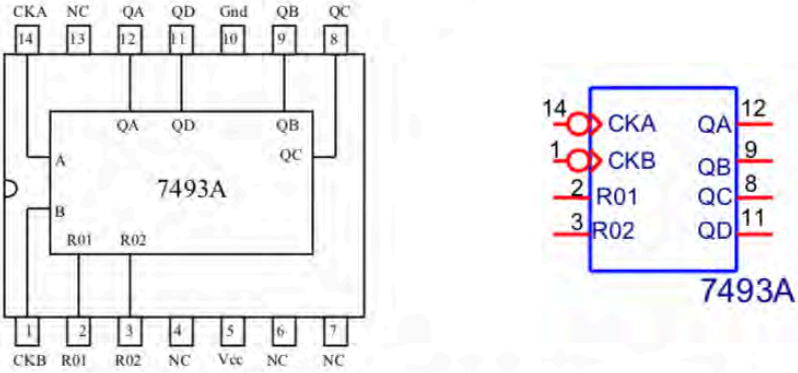
7483A 4-Bit Adder: Pinout and Logic Symbol



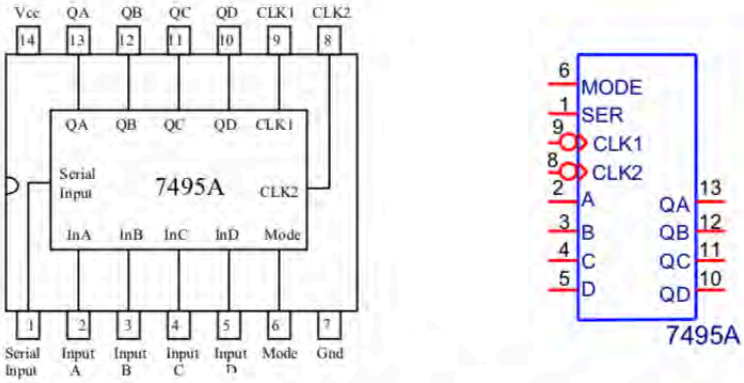
7485 4-Bit Magnitude Comparator: Pinout and Logic Symbol



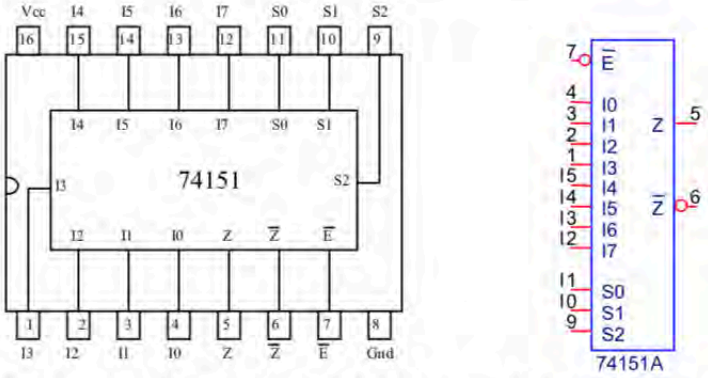
7490A Decade Counter: Pinout and Logic Symbol



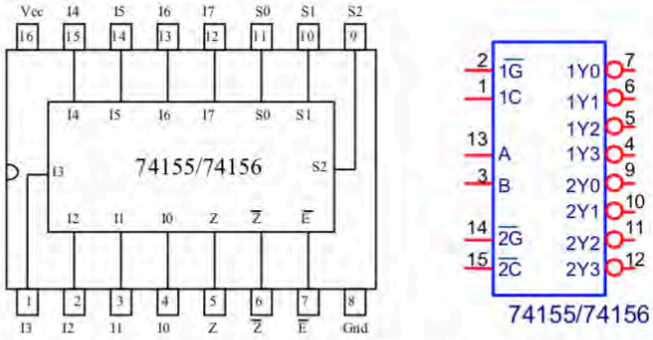
7493A 4-Bit Binary Counter: Pinout and Logic Symbol



7495A 4-Bit Parallel-Load Shift Register: Pinout and Logic Symbol

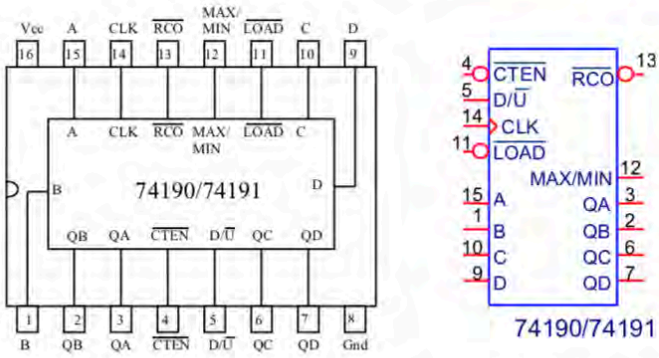


74151 8x1 Data Selector/Multiplexer: Pinout and Logic Symbol



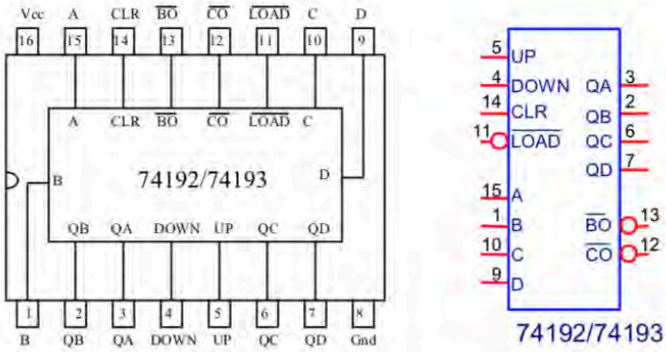
74155 – Totem Pole Outputs
74156 – Open-Collector Outputs

74155/74156 Dual 2-Line To 4-Line Decoder/Demultiplexer: Pinout and Logic Symbol



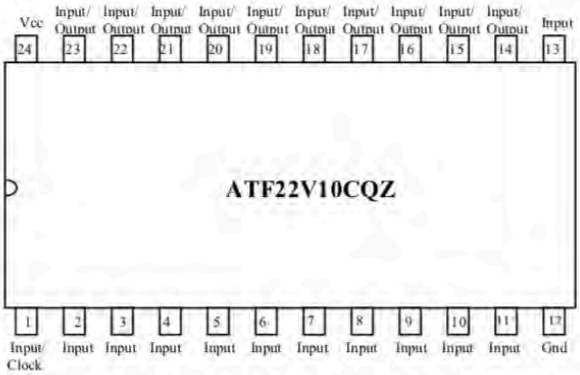
74190 – BCD Counter
74191 – 4-Bit Binary Counter

74190/74191 Synchronous Up/Down Counter: Pinout and Logic Symbol

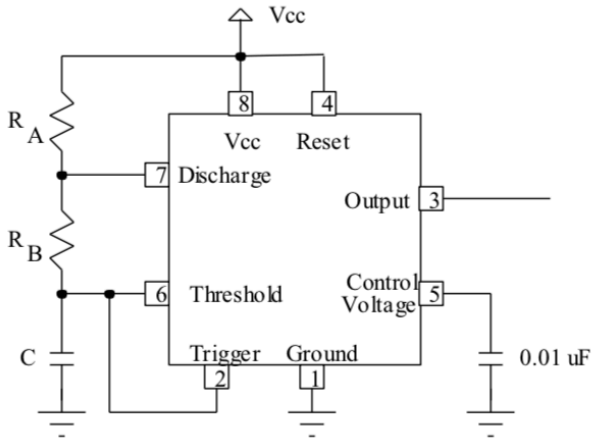


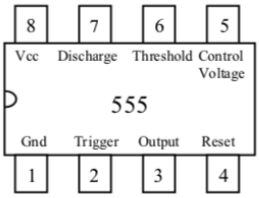
74192 – BCD Counter
74193 – 4-Bit Binary Counter

74192/74193 Synchronous Up/Down Counter: Pinout and Logic Symbol



ATMEL **ATF22V10CQZ** PLD (Programmable Logic Device)

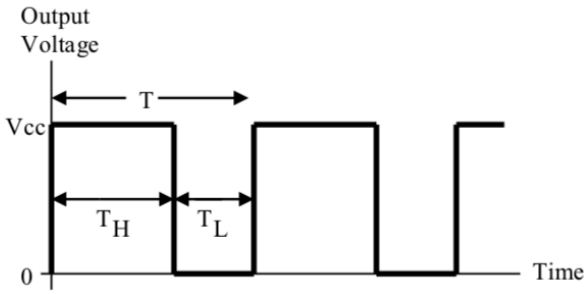


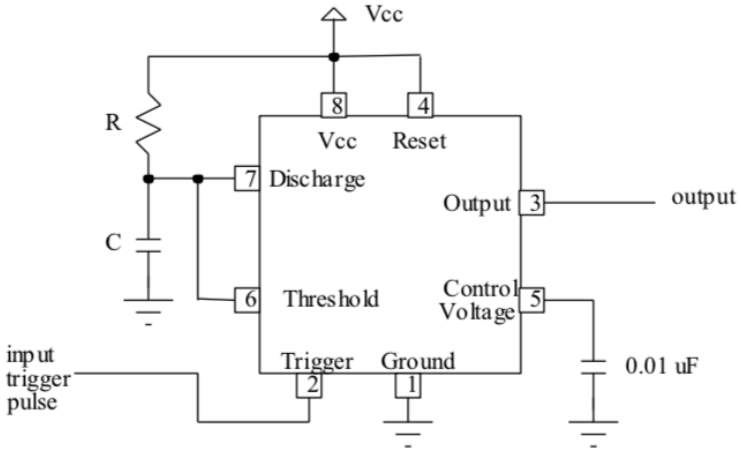


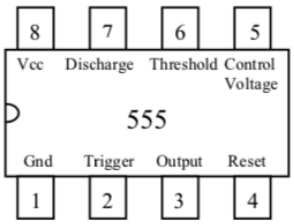
Example: If C = 0.1 μF, R_A = 5 kΩ, and R_B = 4.4 kΩ,
then T_H = 651.4 μs, T_L = 304.9 μs,
T = 956.3 μs, f = 1046 Hz, and D = 68.1 %

555 Timer connected as an astable multivibrator (clock generator)

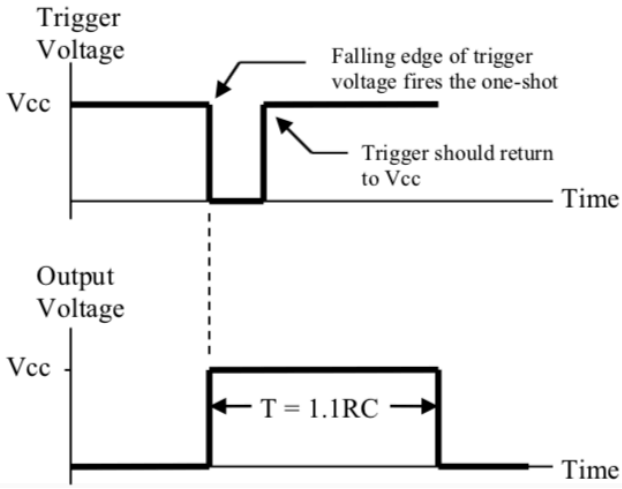
$$T_H = 0.693(R_A + R_B)C$$
$$T_L = 0.693(R_B)C$$
$$T = T_H + T_L = 0.693(R_A + 2R_B)C$$
$$f = \frac{1}{T} = \frac{1.44}{(R_A + 2R_B)C}$$
$$D = \text{Duty Cycle} = \frac{T_H}{T} = \frac{R_A + R_B}{R_A + 2R_B} \times 100\%$$



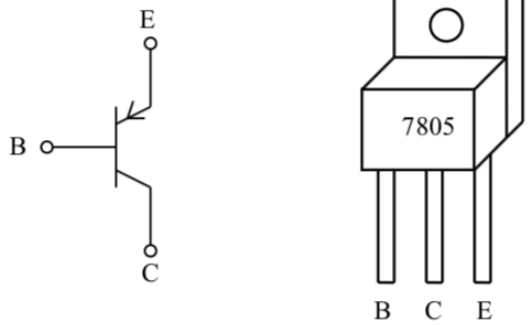




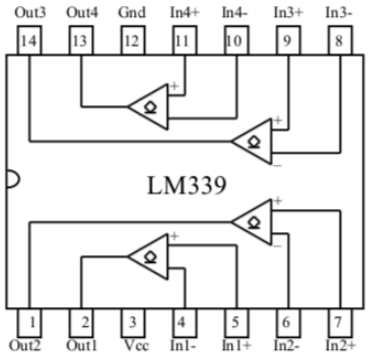
555 Timer connected as a monostable multivibrator (one-shot)



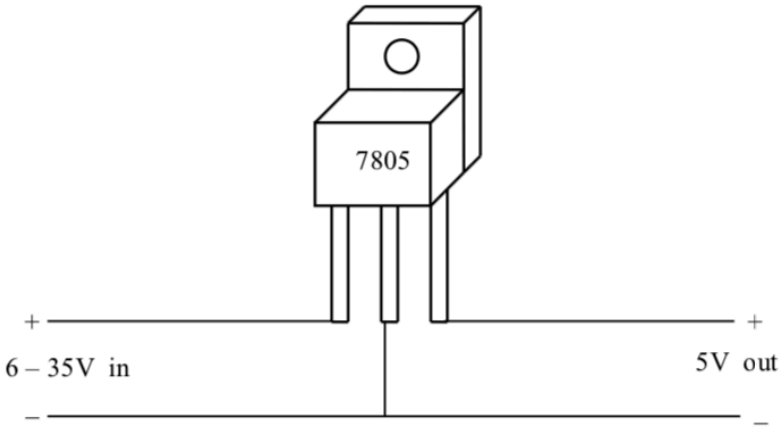
Example: If C = 0.1 μF and R = 47 kΩ, then T = 1.1(47 kΩ)(0.1 μF) = 5.17 ms.
So, each time the one-shot is triggered, an output pulse with a duration of 5.17 ms is produced.



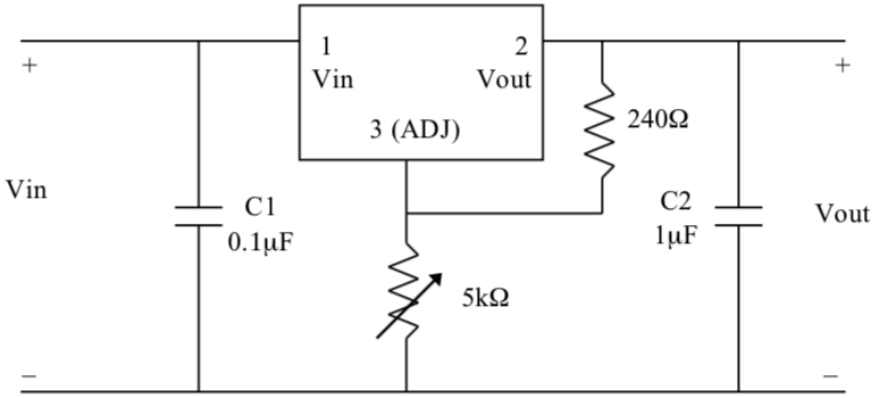
TIP30 PNP Transistor Symbol and Pin Assignment

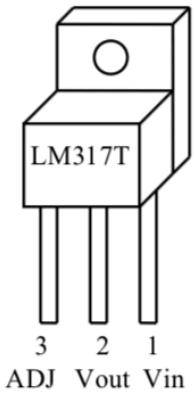


LM339 Quad Comparator (open-collector outputs)



7805 5V Regulator

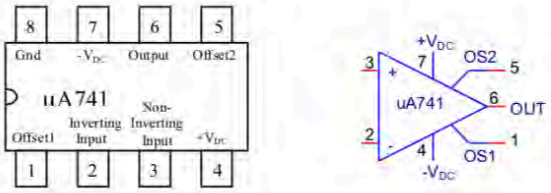




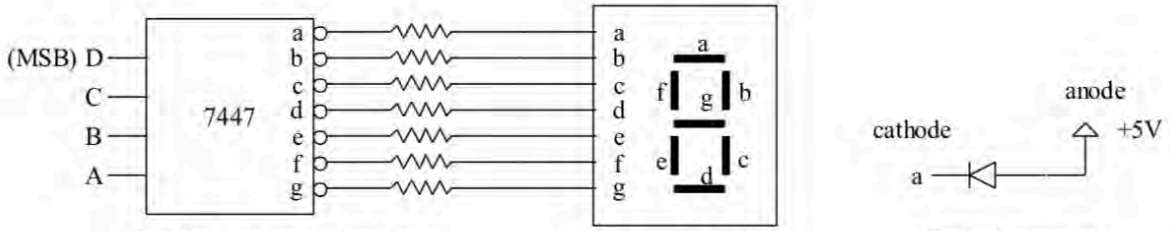
LM317T Adjustable Regulator

Notes:

1) C1 is optional – improves transient response
2) C2 is needed only if device is far from filter capacitors
3) Vout adjustable +1.2V to +37V
4) Vout is tied to the heat sink



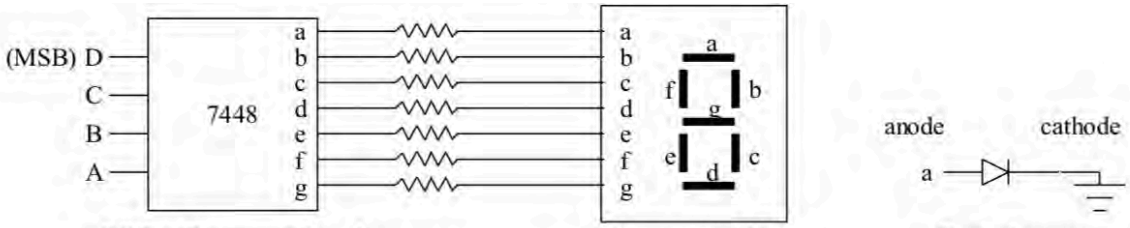
µA741 Operational Amplifier: Pinout and Symbol



BCD to 7-segment decoder
(with active-LOW outputs)

Common-anode
7-segment display

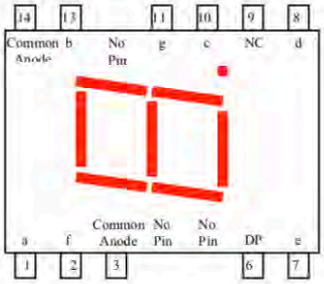
Typical segment



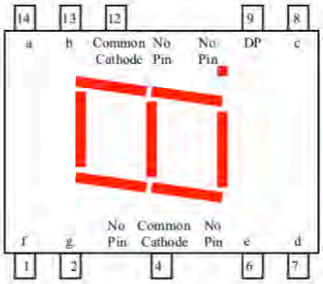
BCD to 7-segment decoder
(with active-HIGH and outputs)

Common-cathode
7-segment display

Typical segment



MAN72A Common Anode
7-segment Display
(common anode connection to Vcc)



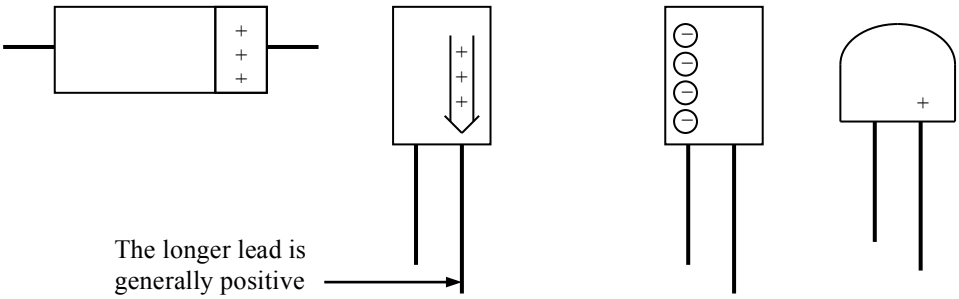
MAN74A Common Cathode
7-segment Display
(common cathode connection to ground)

Capacitor values (all values are in μF)

.001	.01	.1	1	10	100	1000
	.015	.15		12, 15	120	1500
.0022	.022	.22	2.2	22	200, 220	
	.027	.27	3	25		
.0033	.033	.33	3.3	33		
.0047	.047	.47	4.7	47	470	
		.56	5	50	500	
.0068	.068	.68	6.8			
	.082	.82				

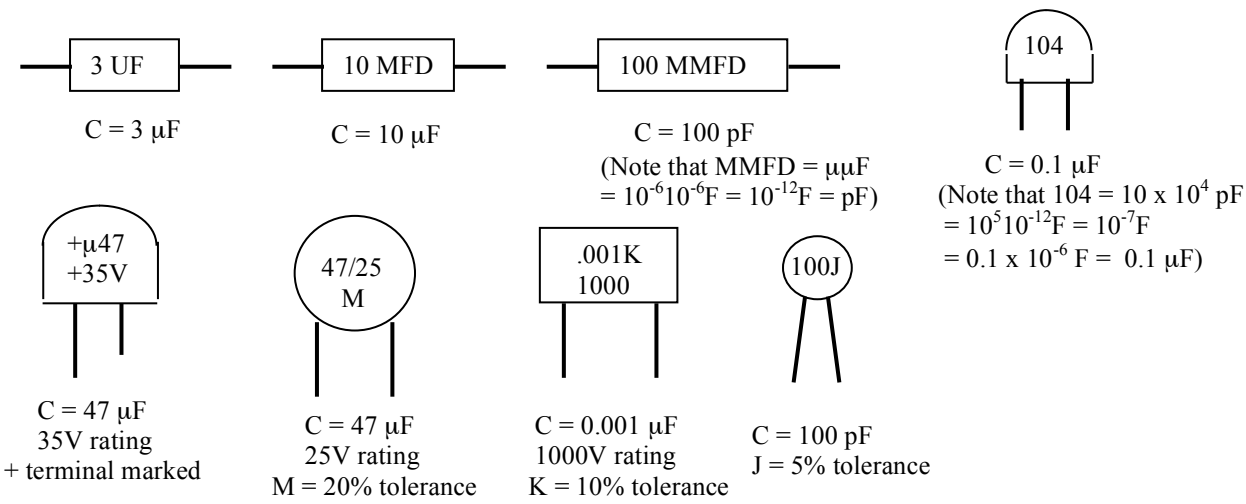
Polarized capacitors:

Polarity is important on some capacitors. Most capacitors that look like metal cans are ***electrolytic capacitors*** and are polarized. If a capacitor is polarized it should have a terminal marked as being positive (+) or negative (-). Be sure to connect it with the proper polarity. Connecting a polarized capacitor with reverse polarity can sometimes yield errors and in extreme cases the capacitors can explode! Some common markings for polarized capacitors are shown below.



Capacitors values:

Capacitors vary tremendously in size, color, shape, and material. The markings on capacitors also vary considerably and they can often be difficult to interpret since various styles are used to specify the value of capacitance. It is a good idea to check capacitor values in lab using an ***impedance bridge*** if one is available. Several examples of labels found on capacitors are shown below.



Standard values of commercially available resistors

Ohms (Ω)					Kilohms (kΩ)		Megohms (MΩ)	
0.10	**1.0**	**10**	**100**	**1000**	**10**	**100**	**1.0**	**10.0**
0.11	1.1	11	110	1100	11	110	1.1	11.0
0.12	**1.2**	**12**	**120**	**1200**	**12**	**120**	**1.2**	**12.0**
0.13	1.3	13	130	1300	13	130	1.3	13.0
0.15	**1.5**	**15**	**150**	**1500**	**15**	**150**	**1.5**	**15.0**
0.16	1.6	16	160	1600	16	160	1.6	16.0
0.18	**1.8**	**18**	**180**	**1800**	**18**	**180**	**1.8**	**18.0**
0.20	2.0	20	200	2000	20	200	2.0	20.0
0.22	**2.2**	**22**	**220**	**2200**	**22**	**220**	**2.2**	**22.0**
0.24	2.4	24	240	2400	24	240	2.4	
0.27	**2.7**	**27**	**270**	**2700**	**27**	**270**	**2.7**	
0.30	3.0	30	300	3000	30	300	3.0	
0.33	**3.3**	**33**	**330**	**3300**	**33**	**330**	**3.3**	
0.36	3.6	36	360	3600	36	360	3.6	
0.39	**3.9**	**39**	**390**	**3900**	**39**	**390**	**3.9**	
0.43	4.3	43	430	4300	43	430	4.3	
0.47	**4.7**	**47**	**470**	**4700**	**47**	**470**	**4.7**	
0.51	5.1	51	510	5100	51	510	5.1	
0.56	**5.6**	**56**	**560**	**5600**	**56**	**560**	**5.6**	
0.62	6.2	62	620	6200	62	620	6.2	
0.68	**6.8**	**68**	**680**	**6800**	**68**	**680**	**6.8**	
0.75	7.5	75	750	7500	75	750	7.5	
0.82	**8.2**	**82**	**820**	**8200**	**82**	**820**	**8.2**	
0.91	9.1	91	910	9100	91	910	9.1	

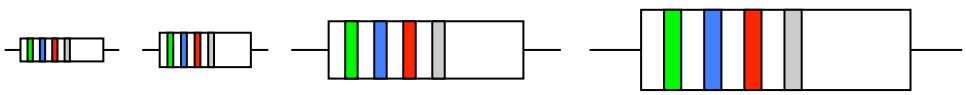
Notes: All values are available with 5% tolerance.
Only the **bold** values are available with 10% tolerance.

Resistor Color Code

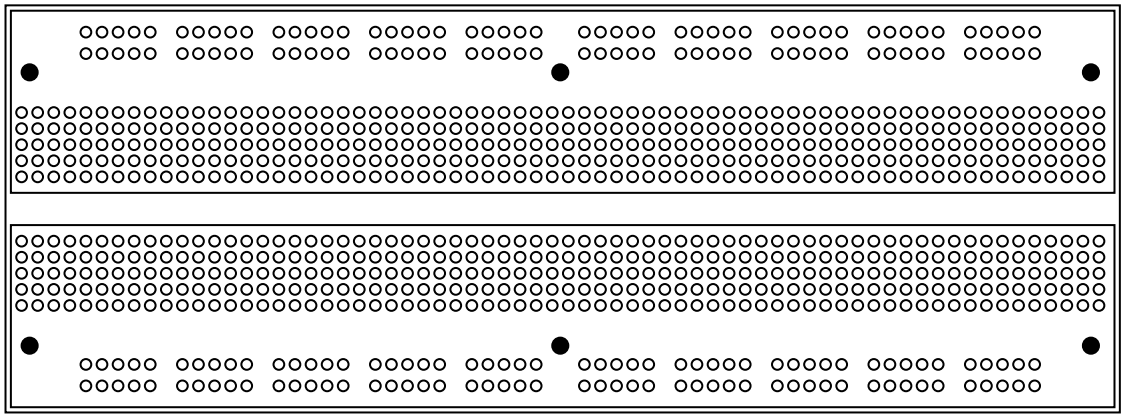
Carbon resistors are typically color-coded using four colored bands labeled A, B, C and D as indicated below.

Bands A, B, C		Band D	
Black	0	Gold	5%
Brown	1	Silver	10%
Red	2	No band	20%
Orange	3		
Yellow	4		
Green	5		
Blue	6		
Violet	7		
Gray	8		
White	9		
Gold	-1		
Silver	-2		

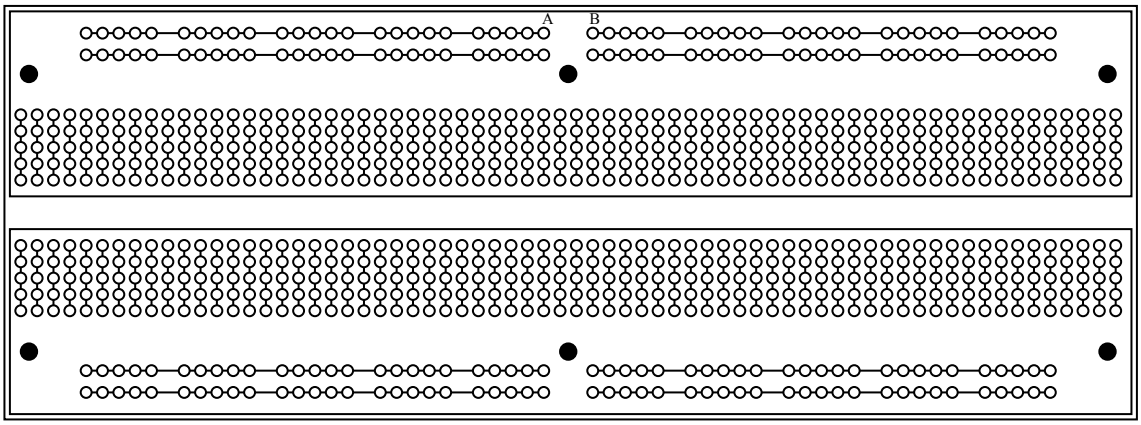
The size of a carbon resistor indicates its power rating.



¼ W ½ W 1 W 2 W



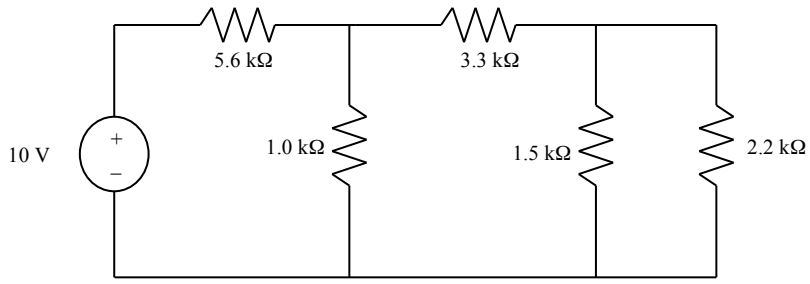
Solderless Breadboard (or equivalent)

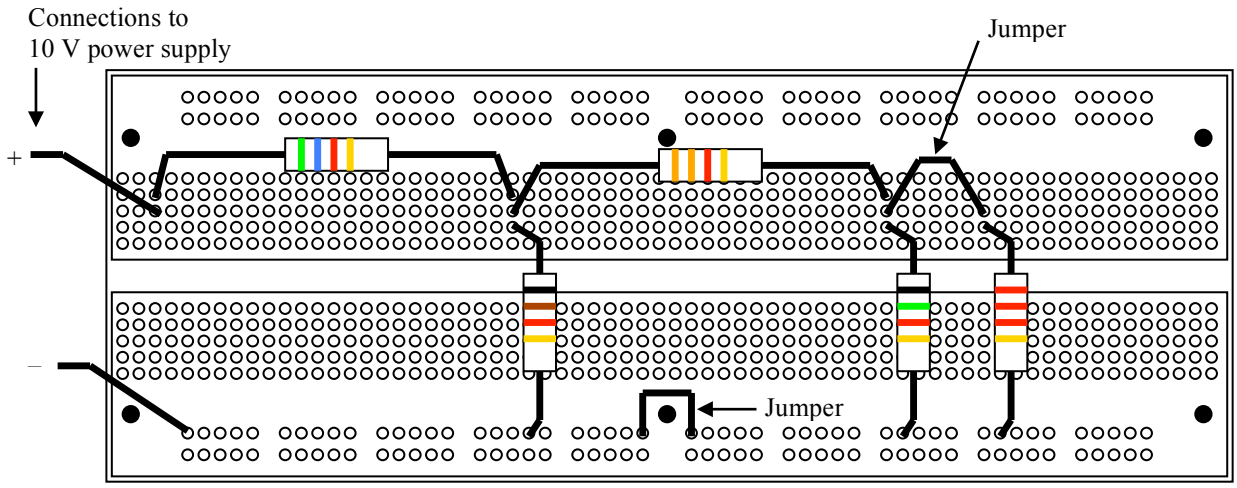


Internal Connections on the Solderless Breadboard

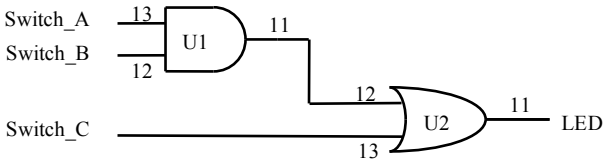
Notes:
1) Lines indicate which holes are connected under the breadboard.
2) To connect two or more wires together, plug them in the same row of holes.
3) Holes A and B are connected on some breadboards (as well as the similar holes on the other horizontal rows).

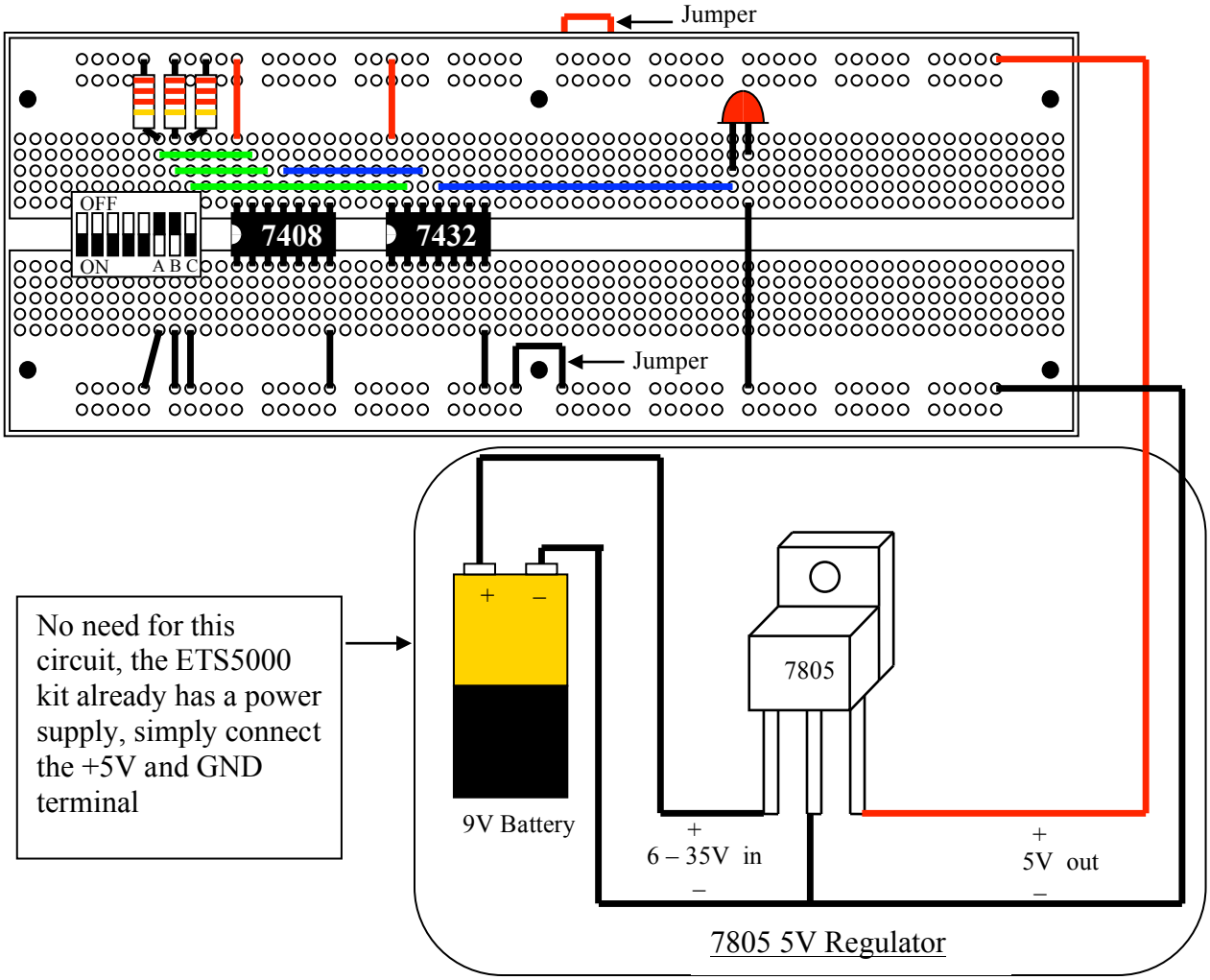
<u>Example 1</u>: Connect the following circuit using the solderless breadboard.





Example 2: Connect the following circuit using the solderless breadboard.





Notes:

1) The DIP switch operates as follows:
   ON – Provides a logical 0 (LOW). Closed switch makes connection to ground.
   OFF – Provides a logical 1 (HIGH). Open switch provides 5V through the 2.2k resistor.
   (It is often useful to install the switch ***upside down*** so that ON (LOW) is on the bottom.)

2) The switch positions shown above (darkened) provide an input of 110 causing the red LED to light indicating that the output is logical 1 (HIGH).

3) Cutting wires to length and using a wire color scheme makes for a neat circuit that is easy to troubleshoot. In the circuit above the following color scheme was used:
   RED – 5V
   BLACK – Ground (0V)
   GREEN – Switch connections
   BLUE – Other connections