VIRTUAL UNIVERSITY OF PAKISTAN

Digital Logic Design

CS302

Course Handouts

GRADUP

Practice. Pass. Graduate.

thegradup.com

Table of Contents

Lesson No. 01 . . . . . **1**
An Overview & Number Systems 1
Programmable Logic Devices (PLDs) 8
Fractions in Binary Number System 13
Binary Number System 12
Caveman number system 11
Decimal Number System 10
Number Systems and Codes 10
Analogue to Digital and Digital to Analogue conversion and Interfacing 9
Sequential logic and implementation 8
Combinational Logic Circuits and Functional Devices 7
Binary Number System 4
Digital Systems and Digital Values 4
Electronic Processing of Continuous and Digital Quantities 3
Digital representing of quantities 1
Analogue versus Digital 1
Lesson No. 02 . . . . . **14**
Number Systems. 14
Binary to Decimal conversion 14
Decimal to Binary conversion 15
Converting Decimal fractions to Binary 16
Binary Arithmetic 17
Signed and Unsigned Binary Numbers 19
1's & 2's complement 20
Lesson No. 03 . . . . . **23**
Floating-Point Numbers 24
Hexadecimal Numbers. 27
Lesson No. 04 . . . . . **31**
Octal Numbers 31
The Excess Code 34
The BCD Code 34
The Gray Code 36
Alphanumeric Codes 37
ASCII Code 38
Extended ASCII Code 38
Parity Method 38
Lesson No. 05 . . . . . **40**
Logic Gates 40

© Virtual University of Pakistan

AND Gate . . . 40
OR Gate. . . . 42
NOT Gate . . . 43
NAND Gate . . . 45
NOR Gate . . . 47
Lesson No. 06 . . . **50**
Logic Gates & Operational Characteristics. . . . 50
Exclusive-OR and Exclusive-NOR Gates . . . 53
Digital Circuits and Operational Characteristics . . . 56
TTL/CMOS NOT Gate Operation . . . 57
Integrated Circuit Technologies . . . 57
Lesson No. 07 . . . **61**
Digital Circuits & Operational Characteristics . . . 61
Lesson No. 08 . . . **71**
Boolean Algebra & Logic Simplification . . . 71
Laws of Boolean Algebra . . . 72
Rules of Boolean Algebra. . . . 73
Demorgan's Theorems . . . 74
Simplification using Boolean Algebra . . . 76
Standard Form of Boolean Expressions . . . 77
Lesson No. 09 . . . **79**
Standard SOP form . . . 84
Standard POS form . . . 85
Converting to Standard SOP and POS forms. . . . 85
Minterms and Maxterms . . . 85
Lesson No. 10 . . . **89**
Karnaugh Map & Boolean Expression Simplification . . . 89
The 3-variable Karnaugh Map . . . 89
The 4-variable Karnaugh Map . . . 89
Grouping and Adjacent Cells . . . 90
Simplification of SOP expressions using the Karnaugh Map . . . 93
Don't care Conditions . . . 96
Lesson No. 11 . . . **99**
Five-Variable Karnaugh Map . . . 102
7-Segment Display . . . 103
Lesson No. 12 . . . **109**
Comparators . . . 109
Quine-McCluskey Simplification Method . . . 110
Comparator Circuit . . . 116
Lesson No. 13 . . . **118**

Odd-Prime Number Detector . . . 118
Lesson No. 14 . . . 131
Operation of Odd-Parity Generator Circuit . . . 132
XOR and XNOR Gates . . . 133
Half Adder and Full Adder . . . 134
Parallel Binary Adders . . . 137
Carry Propagation . . . 137
Lesson No. 15 . . . 141
BCD ADDER . . . 141
Arithmetic and Logic Unit (ALU) . . . 147
Lesson No. 16 . . . 150
16-BIT ALU. . . . 150
Comparators . . . 152
Decoders . . . 157
Binary Decoder . . . 158
MSI Decoder . . . 159
Lesson No. 17 . . . 160
THE 74XX138 3-TO-8 DECODER . . . 160
BCD to 7-Segment Decoder . . . 162
MSI Seven-Segment Decoder . . . 163
BCD-to-Decimal Decoder. . . . 163
Encoder . . . 163
Binary Encoder . . . 163
Priority Encoders. . . . 164
Decimal-to-BCD Encoder. . . . 166
Multiplexer . . . 167
Lesson No. 18 . . . 169
Applications of Multiplexers . . . 172
Lesson No. 19 . . . 178
Demultiplexer . . . 178
Applications of Demultiplexer . . . 178
Programmable Logic Devices . . . 179
Programmable Arrays of AND Gates and OR Gates . . . 179
Programmable Read-Only Memory (PROM) . . . 182
Programmable Logic Array (PLA) . . . 182
Programmable Array Logic (PAL) . . . 182
Generic Array Logic (GAL) . . . 183
Lesson No. 20 . . . 190
The GAL22V10 . . . 194
OLMC Combinational Mode . . . 196

Tri-State Buffers . . . 196
Lesson No. 21 . . . 200
ABEL . . . 203
Boolean Operations and Boolean Notations . . . 203
Test Vectors . . . 206
Lesson No. 22 . . . 209
Latches and Flip-Flops . . . 211
The NAND gate based S-R (Set-Reset) Latch . . . 211
The NOR gate based S-R (Set-Reset) Latch . . . 213
Lesson No. 23 . . . 217
The Gated S-R Latch . . . 218
The Gated D Latch . . . 219
Edge-Triggered Flip-Flop . . . 221
Edge-Triggered S-R Flip-flop . . . 222
Edge-Triggered D Flip-flop . . . 224
Edge-Triggered J-K Flip-flop. . . . 225
Master-Slave Flip-Flops . . . 230
Lesson No. 24 . . . 237
Edge-Triggered J-K Flip-flop. . . . 240
J-K flip-flop used as a counter . . . 246
Lesson No. 25 . . . 248
One-Shot Mono-stable multi-vibrator. . . . 255
Lesson No. 26 . . . 258
THE 555 TIMER . . . 258
Counters . . . 261
Asynchronous Counters (Ripple Counters) . . . 262
Mod-n Counters . . . 265
Integrated Circuit Asynchronous Counters. . . . 268
Lesson No. 27 . . . 270
Down Counters . . . 270
Synchronous Counters . . . 272
Lesson No. 28 . . . 277
Mod-n Synchronous Counter . . . 277
Integrated Circuit Synchronous Counters . . . 278
Cascading Counters . . . 279
Lesson No. 29 . . . 286
Up/Down Counters . . . 286
Design Procedure . . . 298
Lesson No. 30 . . . 305
Digital Clock . . . 305

Lesson No. 31 . . . . . **314**
S-R flip-flop based Implementation 316
Lesson No. 32 . . . . . **320**
State Reduction 325
Lesson No. 33 . . . . . **328**
State Assignment 328
Moore Machine State Diagram 330
Mealy Machine 333
Lesson No. 34 . . . . . **339**
Shift Registers 339
Serial In/Shift Right/Serial Out Operation 339
Serial In/Shift Left/Serial Out Operation 339
Serial In/Parallel Out Operation 341
Parallel In/Serial Out Operation 343
Parallel In/Parallel Out Operation 345
Rotate Right Operation 347
Rotate Left Operation 347
Shift Register Counters 347
Johnson Counter 347
Ring Counter 348
Lesson No. 35 . . . . . **349**
Serial-to-Parallel Converter 349
Keyboard Encoder 350
Lesson No. 36 . . . . . **360**
3-Bit Up/Down Counter 360
Elevator State Diagram 366
Lesson No. 37 . . . . . **370**
Traffic Signal Control System 374
Lesson No. 38 . . . . . **378**
Equation Definition 378
Lesson No. 39 . . . . . **387**
Memory 387
Memory Organization 387
Random Access Memory (RAM) 391
Static RAM 392
Lesson No. 40 . . . . . **396**
Dynamic RAM. 400
Lesson No. 41 . . . . . **405**
Types of DRAMs. 407
ROM Read-Only Memory. 407

© Virtual University of Pakistan

PROM (Programmable ROMs) . . . 410
EPROM Erasable PROM . . . 411
Programming EPROM . . . 412
EEPROM Electrically Erasable PROM . . . 412
FLASH Memory . . . 412
Lesson No. 42 . . . 416
Flash Memory Array . . . 416
First In-First Out (FIFO) Memory . . . 418
Lesson No. 43 . . . 422
Last IN - First OUT (LIFO) Memory . . . 422
Memory Map . . . 424
Address Decoders . . . 428
Introduction to FPGAs . . . 430
Lesson No. 44 . . . 432
Analogue to Digital Conversion . . . 434
Operational Amplifier (Op-Amp) . . . 439
Lesson No. 45 . . . 446
Digital to Analogue Conversion . . . 449

Lesson No. 01

AN OVERVIEW & NUMBER SYSTEMS

Analogue versus Digital

Most of the quantities in nature that can be measured are continuous. Examples include

- **Intensity of light during the da y:** The intensity of light gradually increases as the sun rises in the morning; it remains constant throughout the day and then gradually decreases as the sun sets until it becomes completely dark. The change in the light throughout the day is gradual and continuous. Even with a sudden change in weather when the sun is obscured by a cloud the fall in the light intensity although very sharp however is still continuous and is not abrupt.
- **Rise and fall in temper ature during a 24-hour period:** The temperature also rises and falls with the passage of time during the day and in the night. The change in temperature is never abrupt but gradual and continuous.
- **Velocity of a car travelling from A to B:** The velocity of a car travelling from one city to another varies in a continuous manner. Even if it abruptly accelerates or stops suddenly, the change in velocity seemingly very sudden and abrupt is never abrupt in reality. This can be confirmed by measuring the velocity in short time intervals of few milliseconds.

The measurable values generally change over a continuous range having a minimum and maximum value. The temperature values in a summer month change between 23 ^{0}C to 45 ^{0}C. A car can travel at any velocity between 0 to 120 mph.

Digital representing of quantities

Digital quantities unlike Analogue quantities are not continuous but represent quantities measured at discrete intervals. Consider the continuous signal as shown in the figure 1.1.

To represent this signal digitally the signal is sampled at fixed and equal intervals. The continuous signal is sampled at 15 fixed and equal intervals. Figure 1.2. The set of values (1, 2, 4, 7, 18, 34, 25, 23, 35, 37, 29, 42, 41, 25 and 22) measured at the sampling points represent the continuous signal. The 15 samples do not exactly represent the original signal but only approximate the original continuous signal. This can be confirmed by plotting the 15 sample points. Figure 1.3. The reconstructed signal from the 15 samples has sharp corners and edges in contrast to the original signal that has smooth curves.

If the number of samples that are collected is reduced by half, the reconstructed signal will be very different from the original. The reconstructed signal using 7 samples have missing peak and dip at 34 ^{0}C and 23 ^{0}C respectively. Figure 1.4. The reason for the difference between the original and the reconstructed signal is due to under-sampling. A more accurate representation of the continuous signal is possible if the number of samples and sampling intervals are increased. If the sampling is increased to infinity the number of values would still be discrete but they would be very close and closely match the actual signal.

© Virtual University of Pakistan
thegradup.com

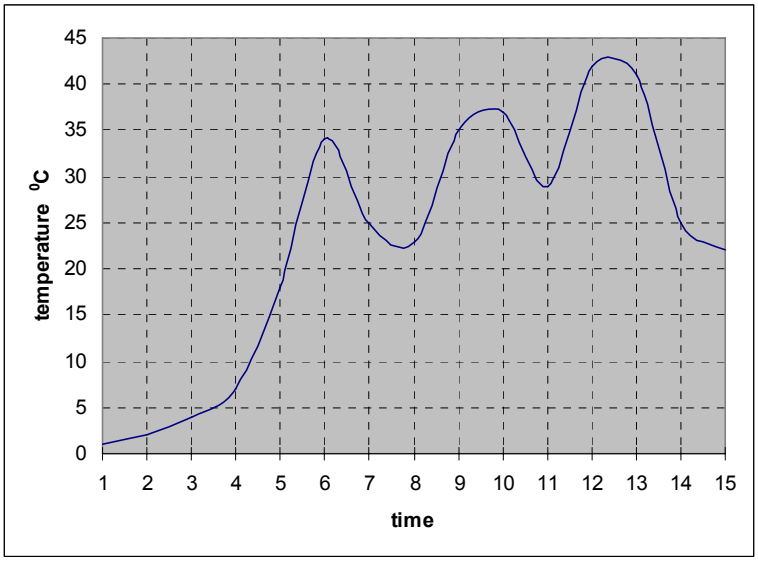


Figure 1.1 Continuous signal showing temperature varying with time

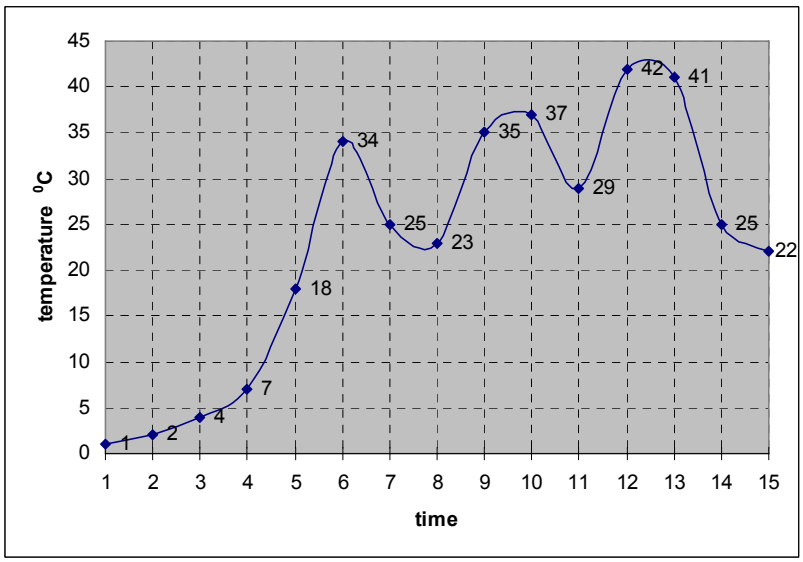


Figure 1.2 Sampling the Continuous Signal at 15 equal intervals

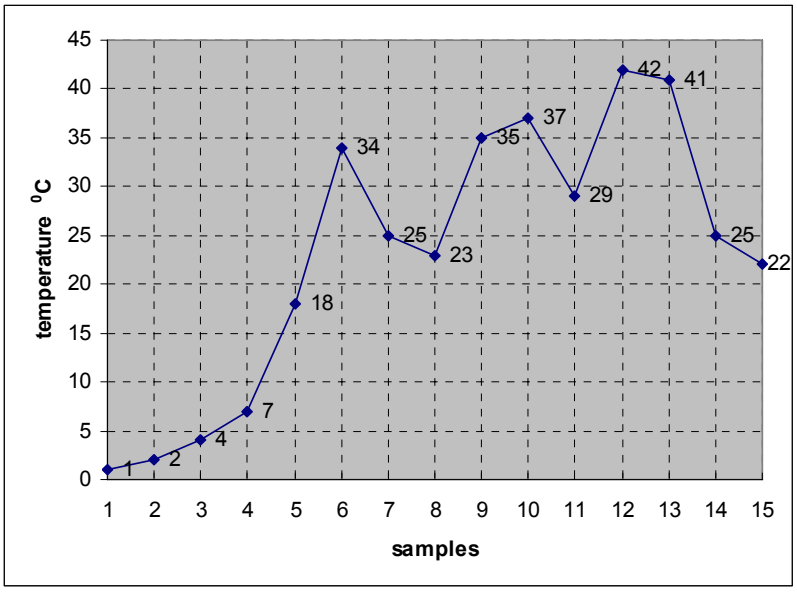


Figure 1.3 Reconstructed Signal by plotting 15 sampled values

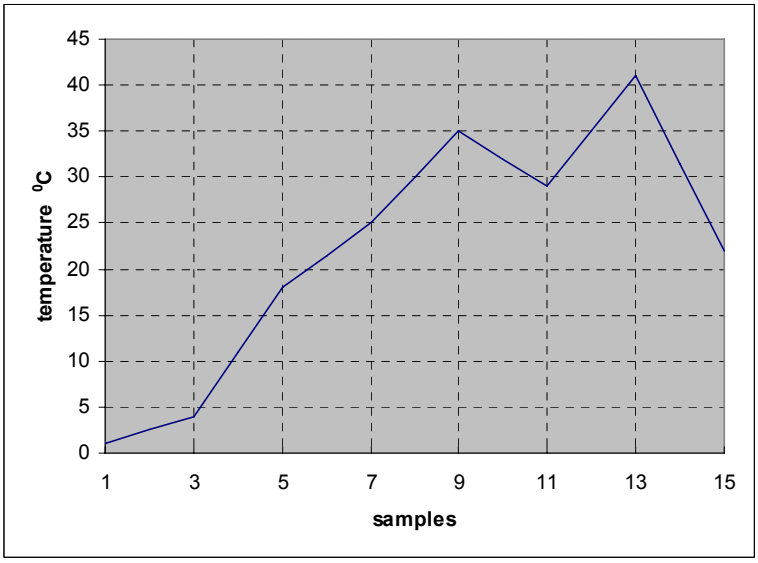


Figure 1.4 Reconstructed Signal by plotting 7 sampled values

Electronic Processing of Continuous and Digital Quantities

Electronic Processing of the continuous quantities or their Digital representation requires that the continuous signals or the discrete values be converted and represented in terms of voltages. There are basically two types of Electronic Processing Systems.

- **Analogue Electronic Systems:** These systems accept and process continuous signals represented in the form continuous voltage or current signals. The continuous quantities are converted into continuous voltage or current signals by transducers. The block diagram describes the processing by an Analogue Electronic System. Figure 1.5.

- **Digital Electronic Sy stems:** These systems accept and process discrete samples representing the actual continuous signal. Analogue to Digital Converters are used to sample the continuous voltage signals representing the original signal.

Do the Digital Electronic Systems use voltages to represent the discrete samples of the continuous signal? This question can be answered by considering a very simple example of a calculator which is a Digital Electronic System. Assume that a calculator is calibrated to represents the number 1 by 1 millivolt (mV). Thus the number 39 is represented by the calculator in terms of voltage as 39 mV. Calculators can also represent large numbers such as 6.25×10^{18} (as in 1 Coulomb = 6.25×10^{18} electrons). The value in terms of volts is 6.25×10^{15} volts! This voltage value can not be practically represented by any electronic circuit. Thus Digital Systems do not use discrete samples represented as voltage values.

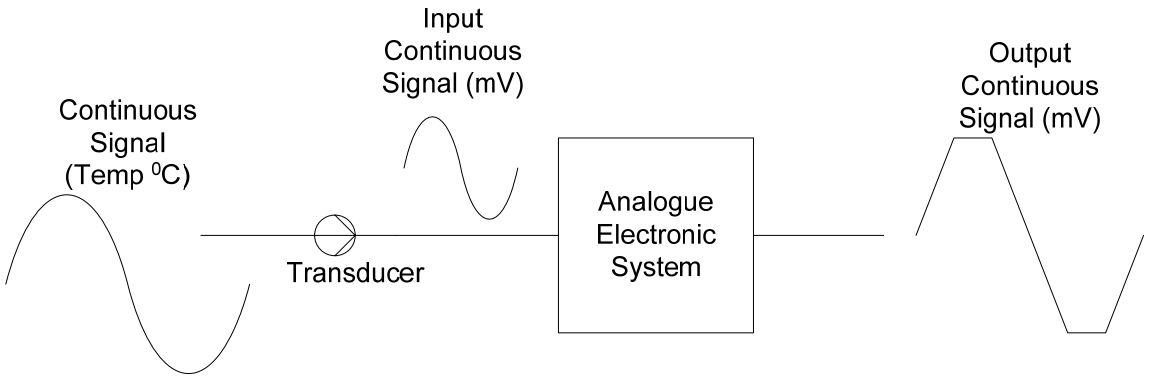


Figure 1.5 Analogue Electronic System processing continuous quantities

Digital Systems and Digital Values

Digital systems are designed to work with two voltage values. A +5 volts represents a logic high state or logic 1 state and 0 volts represents a logic low state or logic 0 state. The Digital Systems which are based on two voltage values or two states can easily represent any two values. For example,

- The numbers '0' and '1'
- The state of a switch 'on' or 'off'
- The colour 'black' and 'white'
- The temperature 'hot' and 'cold'
- An object 'moving' or 'stationary'

Representing two values or two states is not very practical, as many naturally occurring phenomenons have values or state that are more than two. For example, numbers have widely varying ranges, a colour palette might have 64 different shades of the colour red, the temperature of boiling water at room temperature varies from 30 ^{0}C to 100 ^{0}C. Digital Systems are based on the Binary Number system which allows more than two or multiple values to be represented very conveniently.

Binary Number System

The Binary Number System unlike the Decimal number system is based on two values. Each digit or bit in Binary Number system can represent only two values, a '0' and a '1'. A single digit of the Decimal Number system represents 10 values, 0, 1, 2 to 9. The Binary Number System can be used to represent more than two values by combining binary digits or bits. In a Decimal Number System a single digit can represent 10 different values (0 to 9), representing more than 10 values requires a combination of two digits which allows up to 100 values to be represented (0 to 99). A Combination of Binary Numbers is used to represent different quantities.

- **Represent Colours:** A palette of four colours red, blue, green and yellow can be represented by a combination of two digital values 00, 01, 10 and 11 respectively.
- **Representing Temperature:** An analogue value such as 39^{o}C can be represented in a digital format by a combination of 0s and 1s. Thus 39 is 100111 in digital form.

Any quantity such as the intensity of light, temperature, velocity, colour etc. can be represented through digital values. The number of digits (0s and 1s) that represents a quantity is proportional to the range of values that are to be represented. For example, to represent a palette of eight colours a combination of three digits is used. Representing a temperature range of 0^0 C to 100^0 C requires a combination of up to seven digits.

Digital Systems uses the Binary Number System to represent two or multiple values, stores and processes the binary values in terms of 5 volts and 0 volts. Thus the number 39 represented in binary as 100111 is stored electronically in as +5 v, 0v, 0v, +5 v, +5 v and +5 v.

Advantages of working in the Digital Domain

Handling information digitally offers several advantages. Some of the merits of a digital system are spelled out. Details of some these aspects will be discussed and studied in the Digital Logic Design course. Other aspects will be covered in several other courses.

- **Storing and processing data in the digital domain is more efficient:** Computers are very efficient in processing massive amounts of information and data. Computers process information that is represented digitally in the form of Binary Numbers. A Digital CD stores large number of video and audio clips. Sam number of audio and video clips if stored in analogue form will require a number of video and audio cassettes.
- **Transmission of data in the digital form is more efficient and reliable:** Modern information transmission techniques are relying more on digital transmission due to its reliability as it is less prone to errors. Even if errors occur during the transmission methods exist which allow for quick detection and correction of errors.
- **Detecting and Correcting errors in digital data is easier:** Coding Theory is an area which deals with implementing digital codes that allow for detection and correction of multi-bit errors. In the Digital Logic Design course a simple method to detect single bit errors using the Parity bit will be considered.
- **Data can be easily and precisely reproduced:** The picture quality and the sound quality of digital videos are far more superior to those of analogue videos. The reason being that the digital video stored as digital numbers can be exactly reproduced where as analogue video is stored as a continuous signal can not be reproduced with exact precision.
- **Digital systems are easy to design and implement:** Digital Systems are based on two-state Binary Number System. Consequently the Digital Circuitry is based on the two-voltage states, performing very simple operations. Complex Microprocessors are implemented using simple digital circuits. Several simple Digital Systems will be discussed in the Digital Logic course.

- **Digital circuits occupy small sp ace:** Digital circuits are based on two logical states. Electronic circuitry that implements the two states is very simple. Due to the simplicity of the circuitry it can be easily implemented in a very small area. The PC motherboard having an area of approximately 1 sq.ft has most of the circuitry of a powerful computer. A memory chip small enough to be held in the palm of a hand is able to store an entire collection of books.

Information Processing by a Digital System

A Digital system such as a computer not only handles numbers but all kinds of information.

- **Numbers:** A computer is able to store and process all types of numbers, integers, fractions etc. and is able to perform different kinds of arithmetic operations on the numbers.
- **Text:** A computer in a news reporting room is used to write and edit news reports. A Mathematician uses a computer to write mathematical equations explaining the dissipation of heat by a heat sink. The computer is able to store and process text and symbols.
- **Drawings, Diagrams and Pictures:** A computer can store very conveniently complex engineering drawings and diagrams. It allows real life still pictures or videos to be processed and edited.
- **Music and Sound:** Musicians and Composers uses\ a computer to work on a new compositions. Computers understand spoken commands.

A Digital System (computer) is capable of handling different types of information, which is represented in the form of Binary Numbers. The different types of information use different standards and binary formats. For example, computers use the Binary number system to represent numbers. Characters used in writing text are represented through yet another standard known as ASCII which allows alphabets, punctuation marks and numbers to be represented through a combination of 0s and 1s.

Digital Components and their internal working

Digital system process binary information electronically through specialized circuits designed for handling digital information. These circuits as mentioned earlier operate with two voltage values of +5 volts and 0 volts. These specialized electronic circuits are known as Logic Gates and are considered to be the Basic Building Blocks of any Digital circuit.

The commonly used Logic Gates are the AND gate, the OR gate and the Inverter or NOT Gate. Other gates that are frequently used include NOR, NAND, XOR and XNOR. Each of these gates is designed to perform a unique operation on the input information which is known as a logical or Boolean operation.

Large and complex digital system such as a computer is built using combinations of these basic Logic Gates. These basic building blocks are available in the form of Integrated Circuit or ICs. These gates are implemented using standard CMOS and TTL technologies that determine the operational characteristics of the gates such as the power dissipation, operational voltages (3.3v or 5 v), frequency response etc.

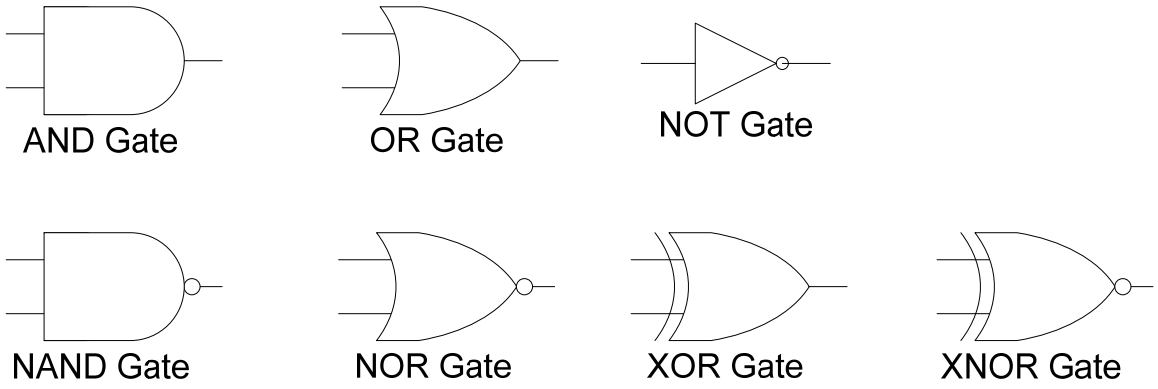


Figure 1.6 Symbolic representations of logic gates.

Combinational Logic Circuits and Functional Devices

The logic gates which form the basic building blocks of a digital system are designed to perform simple logic operations. A single logic gate is not of much use unless it is connected with other gates to collectively act upon the input data. Different gates are combined to build a circuit that is capable of performing some useful operation like adding three numbers. Such circuits are known as Combinational Logic Circuits or Combinational Circuits. An Adder Combinational Circuit that is able to add two single bit binary numbers and give a single bit Sum and Carry output is shown. Figure 1.7.

Implementing large digital system by connecting together logic gates is very tedious and time consuming; the circuit implemented occupies large space, are power hungry, slow and are difficult to troubleshoot.

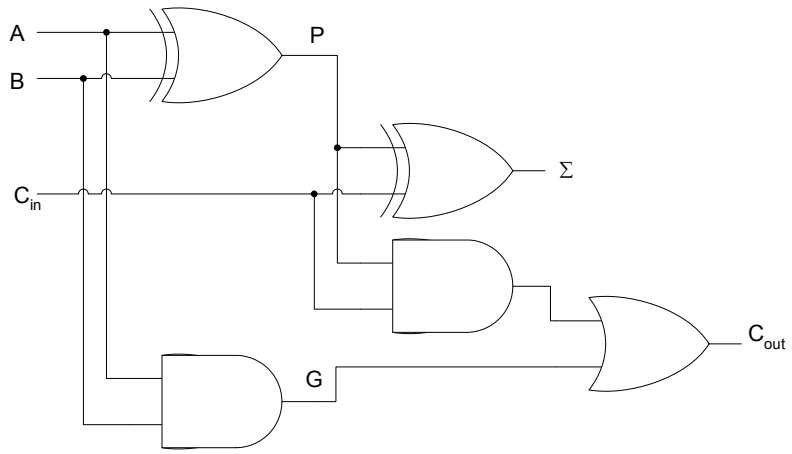


Figure 1.7 1-bit Full-Adder Combinational Circuit

Digital circuits to perform specific functions are available as Integrated Circuits for use. Implementing a Digital system in terms of these dedicated functional units makes the system more economical and reliable. Thus an adder circuit does not have to be implemented by connecting various gates, a standard Adder IC is available that can be readily used. Other

commonly used combinational functional devices are Comparators, Decoders, Encoders, Multiplexers and Demultiplexers.

Sequential logic and implementation

Digital systems are used in vast variety of industrial applications and house hold electronic gadgets. Many of these digital circuits generate an output that is not only dependent on the current input but also some previously saved information which is used by the digital circuit. Consider the example of a digital counter which is used by many digital applications where a count value or the time of the day has to be displayed. The digital counter which counts downwards from 10 to 0 is initialized to the value 10. When the counter receives an external signal in the form of a pulse the counter decrements the count value to 9. On receiving successive pulses the counter decrements the currently stored count value by one, until the counter has been decremented to 0. On reaching the count value zero, the counter could switch off a washing machine, a microwave oven or switch on an air-conditioning system.

The counter stores or remembers the previous count value. The new count value is determined by the previously stored count value and the new input which it receives in the form of a pulse (a binary 1). The diagram of the counter circuit is shown in the figure. Figure 1.8.

Digital circuits that generate a new output on the basis of some previously stored information and the new input are known as Sequential circuits. Sequential circuits are a combination of Combinational circuits and a memory element which is able to store some previous information. Sequential circuits are a very important part of digital systems. Most digital systems have sequential logic in addition to the combinational logic. An example of sequential circuits is counters such as the down-counter which generates a new decremented output value based on the previous stored value and an external input. The storage element or the memory element which is an essential part of a sequential circuit is implemented a flip-flop using a very simple digital circuit known as a flip-flop.

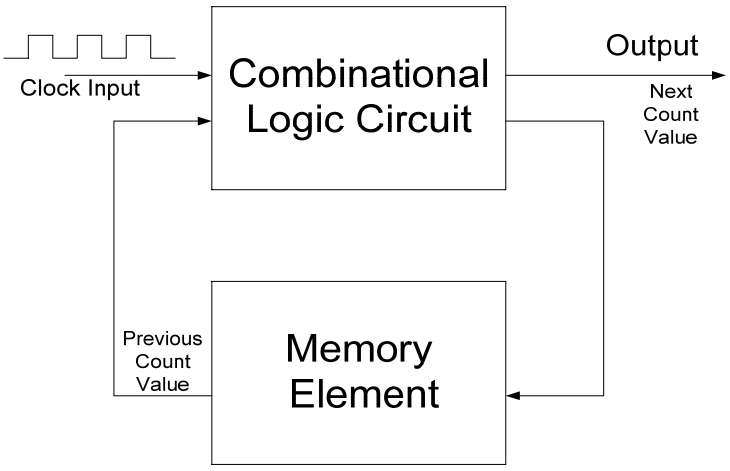


Figure 1.8 A Counter Sequential Circuit

Programmable Logic Devices (PLDs)

The modern trend in implementing specialized dedicated digital systems is through

controller for a washing machine can be implemented by connecting together pieces of combinational and sequential functional units. These implementations are reliable however they occupy considerable space. The implementation time also increases. A general purpose circuit that can be programmed to perform a certain task like controlling a washing machine reduces the implementation cost and time.

Cost is incurred on implementing a digital controller for a washing machine which requires that an inventory of all its components such as its logic circuits, functional devices and the counter circuits be maintained. The implementation time is significantly high as all the circuit components have to be placed on a circuit board and connected together. If there is a change in the controller circuit the entire circuit board has to be redesigned. A PLD based washing machine controller does not require a large inventory of components to be maintained. Most of the functionality of the controller circuit is implemented within a single PLD integrated circuit thereby considerably reducing the circuit size. Changes in the controller design can be readily implemented by programming the PLD.

Programmable Logic Devices can be used to implement Combinational and Sequential Digital Circuits.

Memory

Memory plays a very important role in Digital systems. A research article being edited by a scientist on a computer is stored electronically in the digital memory whilst changes are being made to the document. Once the document has be finalized and stored on some media for subsequent printing the memory can be reused to work on some other document. Memory also needs to store information permanently even when the electrical power is turned off. Permanent memories usually contain essential information required for operating the digital system. This important information is provided by the manufacturer of a digital system.

Memory is organized to allow large amounts of data storage and quick access. Memory (ROM) which permanently stores data allows data to be read only. The Memory does not allow writing of data. Volatile memory (RAM) does not store information permanently. If the power supplied to the RAM circuitry is turned off, the contents of the RAM are permanently lost and can not be recovered when power is restored. RAM allows reading and writing of data. Both RAM and ROM are an essential part of a digital system.

Analogue to Digital and Digital to Analogue conversion and Interfacing

Real-world quantities as mention earlier are continuous in nature and have widely varying ranges. Processing of real-world information can be efficiently and reliably done in the digital domain. This requires real-world quantities to be read and converted into equivalent digital values which can be processed by a digital system. In most cases the processed output needs to be converted back into real-world quantities. Thus two conversions are required, one from the real-world to the digital domain and then back from the digital domain to the real-world.

Modern digitally controlled industrial units extensively use Analogue to Digital (A/D) and Digital to Analogue (D/A) converters to covert quantities represented as an analogue voltage into an equivalent digital representation and vice versa. Consider the example of an industrial controller that controls a chemical reaction vessel which is being heated to expedite the chemical reaction. Figure 1.9. Temperature of the vessel is monitored to control the chemical

level. An electronic temperature sensor (transducer) converts the temperature into an equivalent voltage value. This voltage value is continuous and proportion to the temperature. The voltage representing the temperature is converted into a digital representation which is fed to a digital controller that generates a digital value corresponding to the desired amount of heat. The digitized output representing the heat is converted back to a voltage value which is continuous and is used to control a valve that regulates the heat. An A/D converter converts the analogue voltage value representing the temperature into a corresponding digital value for processing. A D/A converter converts back the digital heat value to its corresponding continuous value for regulating the heater.

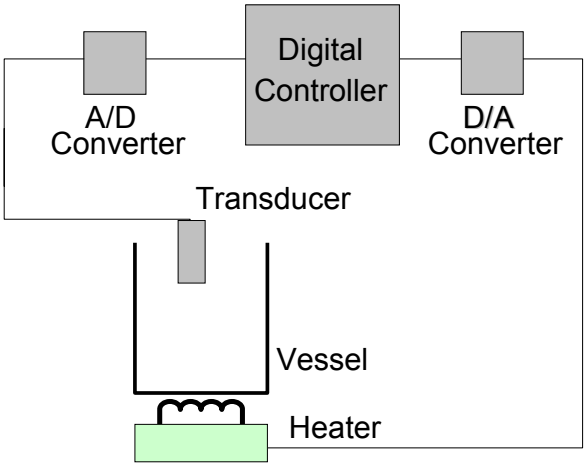


Figure 1.9 Digitally Controlled Industrial Heater Unit

A/D and D/A converters are an important aspect of digital systems. These devices serve as a bridge between the real and digital world allow the two to communicate and interact together.

Number Systems and Codes

Decimal Number System

The decimal number system has ten unique digits 0, 1, 2, 3… 9. Using these single digits, ten different values can be represented. Values greater than ten can be represented by using the same digits in different combinations. Thus ten is represented by the number 10, two hundred seventy five is represented by 275 etc. Thus same set of numbers 0,1 2… 9 are repeated in a specific order to represent larger numbers.

The decimal number system is a positional number system as the position of a digit represents its true magnitude. For example, 2 is less than 7, however 2 in 275 represents 200, whereas 7 represents 70. The left most digit has the highest weight and the right most digit has the lowest weight. 275 can be written in the form of an expression in terms of the base value of the number system and weights.

$$2 \times 10^2 + 7 \times 10^1 + 5 \times 10^0 = 200 + 70 + 5 = 275$$

where, 10 represents the base or radix

Fractions in Decimal Number System

In a Decimal Number System the fraction part is separated from the Integer part by a decimal point. The Integer part of a number is written on the left hand side of the decimal point. The Fraction part is written on the right side of the decimal point. The digits of the Integer part on the left hand side of the decimal point have weights 10^0, 10^1, 10^2 etc. respectively starting from the digit to the immediate left of the decimal point and moving away from the decimal point towards the most significant digit on the left hand side. Fractions in decimal number system are also represented in terms of the base value of the number system and weights. The weights of the fraction part are represented by 10^{-1}, 10^{-2}, 10^{-3} etc. The weights decrease by a factor of 10 moving right of the decimal point. The number 382.91 in terms of the base number and weights is represented as

$3 \times 10^2 + 8 \times 10^1 + 2 \times 10^0 + 9 \times 10^{-1} + 1 \times 10^{-2} = 300 + 80 + 2 + 0.9 + 0.01 = 382.91$

Caveman number system

A number system discovered by archaeologists in a prehistoric cave indicates that the caveman used a number system that has 5 distinct shapes ∑, Δ, >, Ω and ↑. Furthermore it has been determined that the symbols ∑ to ↑ represents the decimal equivalents 0 to 5 respectively.

Centuries ago a caveman returning after a successful hunting expedition records his successful hunt on the cave wall by carving out the numbers Δ↑. What does the number Δ↑ represent? The table 1.1 indicates that the Caveman numbers Δ↑ represents decimal number 9.

Decimal Number	Caveman Number	Decimal Number	Caveman Number
0	∑	10	>∑
1	Δ	11	>Δ
2	>	12	>>
3	Ω	13	>Ω
4	↑	14	>↑
5	Δ∑	15	Ω∑
6	ΔΔ	16	ΩΔ
7	Δ>	17	Ω>
8	ΔΩ	18	ΩΩ
9	Δ↑	19	Ω↑
		20	↑∑

Table 1.1 Decimal equivalents of the Caveman Numbers

The Caveman is using a Base-5 number system. A Base-5 number system has five unique symbols representing numbers 0 to 4. To represent numbers larger than 4, a combination of 2, 3, 4 or more combinations of Caveman numbers are used. Therefore, to represent the decimal number 5, a two number combination of the Caveman number system is used. The most significant digit is Δ which is equivalent to decimal 1. The least significant digit is ∑ which is equivalent to decimal 0. The five combinations of Caveman numbers having the most significant digit Δ, represent decimal values 5 to 9 respectively. This is similar to the Decimal Number system, where a 2-digit combination of numbers is used to represent values

greater than 9. The most significant digit is set to 1 and the least significant digit varies from 0 to 9 to represent the next 10 values after the largest single decimal number digit 9.

The Caveman number Δ↑ can be written in expression form based on the Base value 5 and weights 5^0, 5^1, 5^2 etc.

$= \Delta \times 5^1 + \uparrow \times 5^0 = \Delta \times 5 + \uparrow \times 1$

Replacing the Caveman numbers Δ and ↑ with equivalent decimal values in the expression yields

$= \Delta \times 5^1 + \uparrow \times 5^0 = 1 \times 5 + 4 \times 1 = 9$

The number ΔΩ↑∑ in decimal is represented in expression form as

$\Delta \times 5^3 + \Omega \times 5^2 + \uparrow \times 5^1 + \sum \times 5^0 = \Delta \times 125 + \Omega \times 25 + \uparrow \times 5 + \sum \times 1$

Replacing the Caveman numbers with equivalent decimal values in the expression yields

$= (1) \times 125 + (3) \times 25 + (4) \times 5 + (0) \times 1 = 125 + 75 + 20 + 0 = 220$

Binary Number System

The Caveman Number system is a hypothetical number system introduced to explain that number system other than the Decimal Number system can exist and can be used to represent and count numbers. Digital systems use a Binary number system. Binary as the name indicates is a Base-2 number system having only two numbers 0 and 1. The Binary digit 0 or 1 is known as a 'Bit'. Table 1.2

Decimal Number	Binary Number	Decimal Number	Binary Number
0	0	10	1010
1	1	11	1011
2	10	12	1100
3	11	13	1101
4	100	14	1110
5	101	15	1111
6	110	16	10000
7	111	17	10001
8	1000	18	10010
9	1001	19	10011
		20	10100

Table 1.2 Decimal equivalents of Binary Number System

Counting in Binary Number system is similar to counting in Decimal or Caveman Number systems. In a decimal Number system a value larger than 9 has to be represented by 2, 3, 4 or more digits. In the Caveman Number System a value larger than 4 has to be represented by 2, 3, 4 or more digits of the Caveman Number System. Similarly, in the Binary Number System a Binary number larger than 1 has to be represented by 2, 3, 4 or more binary

Any binary number comprising of Binary 0 and 1 can be easily represented in terms of its decimal equivalent by writing the Binary Number in the form of an expression using the Base value 2 and weights 2^0, 2^1, 2^2 etc.

The number 10011_2 (the subscript 2 indicates that the number is a binary number and not a decimal number ten thousand and eleven) can be rewritten in terms of the expression

$$\begin{aligned} 10011_2 &= (1 \times 2^4) + (0 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) \\ &= (1 \times 16) + (0 \times 8) + (0 \times 4) + (1 \times 2) + (1 \times 1) \\ &= 16 + 0 + 0 + 2 + 1 \\ &= 19 \end{aligned}$$

Fractions in Binary Number System

In a Decimal number system the Integer part and the Fraction part of a number are separated by a decimal point. In a Binary Number System the Integer part and the Fraction part of a Binary Number can be similarly represented separated by a decimal point. The Binary number 1011.101_2 has an Integer part represented by 1011 and a fraction part 101 separated by a decimal point. The subscript 2 indicates that the number is a binary number and not a decimal number. The Binary number 1011.101_2 can be written in terms of an expression using the Base value 2 and weights 2^3, 2^2, 2^1, 2^0, 2^{-1}, 2^{-2} and 2^{-3}.

$$\begin{aligned} 1011.101_2 &= (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3}) \\ &= (1 \times 8) + (0 \times 4) + (1 \times 2) + (1 \times 1) + (1 \times 1/2) + (0 \times 1/4) + (1 \times 1/8) \\ &= 8 + 0 + 2 + 1 + 0.5 + 0 + 0.125 \\ &= 11.625 \end{aligned}$$

Computers do handle numbers such as 11.625 that have an integer part and a fraction part. However, it does not use the binary representation 1011.101. Such numbers are represented and used in Floating-Point Numbers notation which will be discussed latter.

Lesson No. 02

NUMBER SYSTEMS

Binary to Decimal conversion

Most real world quantities are represented in Decimal Number System. Digital Systems on the other hand are based on the Binary Number System. Therefore, when converting from the Digital Domain to the real-world, Binary numbers have to be represented in terms of their Decimal equivalents.

The method used to convert from Binary to Decimal is the Sum-of-Weights method. The Sum-of-Weights method has been used to represent the Caveman numbers $\Delta\uparrow$, $\Delta\Omega\uparrow\sum$ and the Binary numbers 10011 and 1011.101 in the first lecture.

1. Sum-of-Weights Method

Sum-of-weights as the name indicates sums the weights of the Binary Digits (bits) of a Binary Number which is to be represented in Decimal. The Sum-of-Weights method can be used to convert a Binary number of any magnitude to its equivalent Decimal representation.

In the Sum-of-Weights method an extended expression is written in terms of the Binary Base Number 2 and the weights of the Binary number to be converted. The weights correspond to each of the binary bits which are multiplied by the corresponding binary value. Binary bits having the value 0 do not contribute any value towards the final sum expression.

The Binary number 10110_2 is therefore written in the form of an expression having weights 2^0, 2^1, 2^2, 2^3 and 2^4 corresponding to the bits 0, 1, 1, 0 and 1 respectively. Weights 2^0 and 2^3 do not contribute in the final sum as the binary bits corresponding to these weights have the value 0.

$$10110_2 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$
$$= 16 + 0 + 4 + 2 + 0$$
$$= 22$$

2. Sum-of-non-zero terms

In the Sum-of-Weights method, the Binary bits 0 do not contribute towards the final sum representing the decimal equivalent. Secondly, the weight of each binary bit increases by a factor of 2 starting with a weight of 1 for the least significant bit. For example, the Binary number 10110_2 has weights 2^0=1, 2^1=2, 2^2=4, 2^3=8 and 2^4=16 corresponding to the bits 0, 1, 1, 0 and 1 respectively.

The Sum-of-non-zero terms method is a quicker method to determine decimal equivalents of binary numbers without resorting to writing an expression. In the Sum-of-non-zero terms method the weights of non-zero binary bits are summed, as the weights of zero binary bits do not contribute towards the final sum representing the decimal equivalent.

The weights of binary bits starting from the right most least significant bit is 1, The next significant bit on the left has the weight 2, followed by 4, 8, 16, 32 etc. corresponding to higher significant bits. In binary number system the weights of successive bits increase by an order of 2 towards the left side and decrease by an order of 2 towards the right side. Thus a binary number can be quickly converted into its decimal equivalent by adding weights of non-zero terms which increase by a factor of 2. Binary numbers having an integer and a fraction part can similarly be converted into their decimal equivalents by applying the same method.

A quicker method is to add the weights of non-zero terms. Thus for the numbers
- $10011_2 = 16 + 2 + 1 = 19$
- $1011.101_2 = 8 + 2 + 1 + ½ + 1/8 = 11 + 5/8 = 11.625$

Decimal to Binary conversion

Conversion from Decimal to Binary number system is also essential to represent real-world quantities in terms of Binary values. The Sum-of-weights and repeated division by 2 methods are used to convert a Decimal number to equivalent Binary.

1. Sum-of-Weights

The Sum-of-weights method used to convert Binary numbers into their Decimal equivalent is based on adding binary weights of the binary number bits. Converting back from the decimal number to the original Binary number requires finding the highest weight included in the sum representing the decimal equivalent. A Binary 1 is marked to represent the bit which contributed its weight in the Sum representing the decimal equivalent. The weight is subtracted from the sum decimal equivalent. The next highest weight included in the sum term is found. A binary 1 is marked to represent the bit which contributed its weight in the sum term and the weight is subtracted from the sum term. This process is repeated until the sum term becomes equal to zero. The binary 1s and 0s represent the binary bits that contributed their weight and bits that did not contribute any weight respectively.

The process of determining Binary equivalent of a Decimal number 392 and 411 is illustrated in a tabular form. Table 2.1.

Sum Term	Highest Weight	Binary Number	Sum Term = Sum Term – Highest Weight
411	256	100000000	155
155	128	110000000	27
27	16	110010000	11
11	8	110011000	3
3	2	110011010	1
1	1	110011011	0

Table 2.1a Converting Decimal to Binary using Sum-of-Weights Method

Sum Term	Highest Weight	Binary Number	Sum Term = Sum Term – Highest Weight
392	256	100000000	136
136	128	110000000	8
8	8	110001000	0

Table 2.1b Converting Decimal to Binary using Sum-of-Weights Method

The Sum of weights method requires mental arithmetic and is a quick way of converting small decimal numbers into binary. With practice large Decimal numbers can be converted into Binary equivalents.

2. Repeated Division-by-2

Repeated Division-by-2 method allows decimal numbers of any magnitude to be

equivalent is repeatedly divided by 2. The divisor is selected as 2 because the decimal number is being converted into Binary a Base-2 Number system. Repeated division method can be used to convert decimal number into any Number system by repeated division by the Base-Number. For example, the decimal number can be converted into the Caveman Number system by repeatedly dividing by 5, the Base number of the Caveman Number System. The Repeated Division method will be used in latter lectures to convert decimal into Hexadecimal and Octal Number Systems.

In the Repeated-Division method the Decimal number to be converted is divided by the Base Number, in this particular case 2. A quotient value and a remainder value is generated, both values are noted done. The remainder value in all subsequent divisions would be either a 0 or a 1. The quotient value obtained as a result of division by 2 is divided again by 2. The new quotient and remainder values are again noted down. In each step of the repeated division method the remainder values are noted down and the quotient values are repeatedly divided by the base number. The process of repeated division stops when the quotient value becomes zero. The remainders that have been noted in consecutive steps are written out to indicate the Binary equivalent of the Original Decimal Number.

Number	Quotient after division	Remainder after division
392	196	0
196	98	0
98	49	0
49	24	1
24	12	0
12	6	0
6	3	0
3	1	1
1	0	1

Table 2.2 Converting Decimal to Binary using Repeated Division by 2 Method

The process of determining the Binary equivalent of a Decimal number 392 is illustrated in a tabular form. Table 2.2. Reading the numbers in the Remainder column from bottom to top 110001000 gives the binary equivalent of the decimal number 392

Converting Decimal fractions to Binary

Two methods are used to Convert Decimal fractions to Binary. The Sum-of-Weights method, which has been described and used to convert Decimal Integers into Binary Equivalents is applied to convert Decimal fractions into Binary fractions. This method requires mental arithmetic and is suitable for small numbers. The conversion of Decimal fraction 0.625 into Binary equivalent is illustrated in a tabular form. Table 2.3

Sum Term	Highest Weight	Binary Number	Sum Term = Sum Term – Highest Weight
0.625	0.500	0.100	0.125
0.125	0.125	0.101	0

Table 2.3 Converting Decimal to Binary using Sum-of-Weights Method

Repeated Multiplication-by-2 Method

An alternate to the Sum-of-Weights method used to convert Decimal fractions to

number to be converted is repeatedly multiplied by the Base Number to which the number is being converted to, in this case 2. A new number having an Integer part and a Fraction part is generated after each multiplication. The Integer part is noted down and the fraction part is again multiplied with the Base number 2. The process is repeated until the fraction term becomes equal to zero.

Repeated Multiplication-by-2 method allows decimal fractions of any magnitude to be easily converted into binary. The conversion of Decimal fraction 0.625 into Binary equivalent using the Repeated Multiplication-by-2 method is illustrated in a tabular form. Table 2.4. Reading the Integer column from top to bottom and placing a decimal point in the left most position gives 0.101 the binary equivalent of decimal fraction 0.625

Number	Integer part after multiplication	Fraction part after multiplication
0.625	1	0.25
0.25	0	0.5
0.5	1	0.0

Table 2.4 Converting Decimal to Binary using repeated Multiplication-by-2 Method

Binary Arithmetic

Digital systems use the Binary number system to represent numbers. Therefore these systems should be capable of performing standard arithmetic operations on binary numbers.

1. Binary Addition

Binary Addition is identical to Decimal Addition. By adding two binary bits a Sum bit and a Carry bit are generated. The only difference between the two additions is the range of numbers used. In Binary Addition, four possibilities exist when two single bits are added together. The four possible input combinations of two single bit binary numbers and their corresponding Sum and Carry Outputs are specified in table 2.5.

First Number	Second Number	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Table 2.5 Addition of two Single Bit Binary Numbers

The first three additions give a result 0, 1 and 1 respectively which can be represented by a single binary digit (bit). The fourth addition results in the number 2, which can be represented in binary as 10_2. Thus two digits (bits) are required. This is similar to the addition of 9 + 3 in decimal. The answer is 12 which can not be represented by a single digit, thus two digits are required. The number 2 is the sum part and 1 is the carry part.

Any number of binary numbers having any number of digits can be added together. Thus the number 1011, 110, 1000 and 11 can be added together. Most significant digits (bits) of second and fourth numbers are assumed to be zero.

Carry		1	10	1		Decimal Equivalent
1st Number		1	0	1	1	(11)
2nd Number			1	1	0	(06)
3rd Number		1	0	0	0	(08)
4th Number				1	1	(03)
Result	1	1	1	0	0	(28)

Table 2.6 Adding multiple binary numbers of different sizes

2. Binary Subtraction

Binary Subtraction is identical to Decimal Subtraction. The only difference between the two is the range of numbers. Subtracting two single bit binary numbers results in a difference bit and a borrow bit. The four possible input combinations of two single bit binary numbers and their corresponding Difference and Borrow Outputs are specified in table 2.7. It is assumed that the second number is subtracted from the first number.

First Number	Second Number	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Table 2.7 Subtraction of two Single Bit Binary Numbers

The second subtraction subtracts 1 from 0 for which a Borrow is required to make the first digit equal to 2. The Difference is 1. This is similar to decimal subtraction when 17 is subtracted from 21. The first digit 7 can not be subtracted from 1, therefore 10 is borrowed from the next significant digit. Borrowing a 10 allows subtraction of 7 from 11 resulting in a Difference of 4.

3. Binary Multiplication

Binary Multiplication is similar to the Decimal multiplication except for the range of numbers. Four possible combinations of two single bit binary numbers and their products are listed in table 2.8.

First Number	Second Number	Product
0	0	0
0	1	0
1	0	0
1	1	1

Table 2.8 Multiplication of two Single Bit Binary Numbers

Multiplying two binary numbers such as 1101 x 101 is performed by a shift and add operation. The binary multiplication shifts and adds partial product terms.

```
   1101
 x  101
   1101    1st product term
  0000     2nd product term
 1101       3rd product term
1000001
```

4. Binary Multiplication by shifting left

Binary Multiplication can be performed by shifting the binary number towards left. A left shift by a single bit is equivalent to multiplication by 2. A left shift by two bits is equivalent to multiplication by 4. Generally, the multiplication factor is determined by 2^n where n is the number of bit shifts.

00011	(3)	original binary number
00110	(6)	binary number shifted left by 1 bit
01100	(12)	binary number shifted left by 2 bits
11000	(24)	binary number shifted left by 3 bits

5. Binary Division

Division in binary follows the same procedure as in the division of decimal numbers. An example illustrates the division of binary numbers. Figure 2.1.

```
    10
101 | 1101
    101
     011
     000
      11
```

Figure 2.1 Binary Division

6. Binary Division by Shifting right

Binary Division can be performed by shifting the binary number towards right. A right shift by a single bit is equivalent to division by 2. A right shift by two bits is equivalent to division by 4. Generally, the division factor is determined by 2^n where n is the number of bit shifts.

10100	(20)	original binary number
01010	(10)	binary number shifted right by 1 bit
00101	(5)	binary number shifted right by 2 bits

Signed and Unsigned Binary Numbers

Digital systems not only handle positive numbers but both positive and negative numbers. In the decimal number system positive numbers are identified by the + sign and negative numbers are represented by the – sign.

In a digital system which uses the Binary number system, the positive and negative signs can not be represented as + and -. As mentioned in the Overview all forms of numbers, text, punctuation marks etc. are represented in the form of 1s and 0s. Thus the positive and negative signs are also presented in terms of binary 0 and 1.

To handle positive and negative binary numbers, the digital system sets aside the most significant digit (bit) to represent the sign

- MSB set to 1 indicates a negative number
- MSB set to 0 indicates a positive number

Thus +13 and -13 are represented as 01101and 11101 respectively. The bits 1101 represent the number 13 and the MSBs 0 and 1 represent positive and negative signs respectively. Thus binary numbers having the MSB signifying the Sign bit are treated as Signed Binary Numbers. This representation is known as the Signed-Magnitude representation.

Digital systems also handle binary numbers which are assumed to be positive and therefore do not have the most significant sign bit. Such numbers are known as Unsigned numbers. Digital system thus have to handle two different types of binary numbers, signed and unsigned. Thus 11101_2 represents -13 in signed binary and 29 in unsigned binary. How should a Digital System treat a binary number? Should it consider it as a signed or unsigned number? A digital system on its own can not decide how to handle a binary number. The digital system has to be notified beforehand to deal with a certain binary representation as signed or unsigned.

1's & 2's complement

Informing the digital system how to treat a binary number is not very efficient. A better way is to represent negative signed numbers in their 2's complement form. Using 2's Complement form to represent signed numbers, allows direct manipulation of positive as well as negative numbers without having to worry about setting the most significant sign bit to indicate positive and negative numbers.

A 2's complement of a number is obtained by first taking the 1's complement of a number and then adding a 1 to change the 1's complement to 2's complement. 1's complement of a number is obtained by simply inverting all its bits. Obtaining the 2's complement of 13 is described in the example below.

```
01101      The number 13
10010      1's complement of 13 is obtained by inverting all the five bits.
+    1
-----
10011      2's complement of 13 is obtained by adding a 1 to its 1's complement.
```

In a 2's complement number system all negative numbers are represented in their 2's complement form and all positive numbers are represented in their actual form. Negative numbers can be readily identified by their MSBs which are set to 1. Thus in a 2's complement representation +13 is represented as 01101 and -13 is represented as 10011.

By having numbers represented in their 2's complement form addition and subtraction operations can easily be performed without having to worry about the sign bits. Thus +13 added to -13 should result in a zero value. This can be verified by directly adding the +13 and -13 in their 2's complement forms.

```
 01101
 10011
------
100000
```

The most significant carry bit is discarded; retaining only the first 5 bits proves that adding +13 and -13 results in a zero value. Similarly it can be shown that adding the numbers +7 and -13 results in -6.

```
10011      (-13)
00111      (+7)
11010      (-6)
```

The binary 2's complement number 11010 has its most significant bit set to 1 indicating that the number is negative. The actual magnitude of the negative number is determined by taking the 2's complement of 11010.

```
11010   Original number
00101   1's complement of Original number
  + 1
00110   2's complement of Original number is equal to 6.
```

Addition and Subtraction Operations with Signed Binary

An additional benefit of using 2's complement representation for signed numbers is that both add and subtract operations can be performed by addition. In the above example 13 was subtracted from 7 by adding 2's complement of -13 to 2's complement of +7. Four cases of adding and subtracting numbers using the 2's complement representation are shown below.

- Both numbers are positive

```
0101      +5
0010      +2
0111      +7
```

- Both numbers are negative

```
1011      -5
1110      -2
11001     -7    the carry generated from the msb is discarded
```

- One number is positive and its magnitude is larger than the negative number

```
0101      +5
1110      -2
10011     +3    the carry generated from the msb is discarded
```

- One number is positive and its magnitude is smaller than the negative number

```
1011      -5
0010      +2
1101      -3
```

The four examples show that add and subtract operations can be carried out by an adder circuit if numbers are represented in their 2's complement form. A separate circuit to perform subtractions is not required.

Range of Signed and Unsigned Binary numbers

Three different types of Binary representations have been discussed. The Unsigned Binary representation can only represent positive binary numbers. The Sign-Magnitude can

represent both positive and negative numbers. The 2's complement signed representation also allows positive and negative numbers to be handled.

Each of the three binary number representations can represent certain range of binary numbers determined by the total number of bits used.

The maximum range of values that can be represented in any number system depends upon the number of digits assigned to represent the value. A 5-digit car odometer can only count up to 99,999 and then it rolls back to 00000. Similarly an 8-digit calculator can only handle integer numbers of the magnitude 99,999,999. A calculator that reserves the most significant digit to write + or – can only handle a maximum range of integer numbers from -9,999,999 to +9,999,999.

A 3-bit unsigned binary number can have values ranging between 000 and 111. Adding 100 and 111 unsigned numbers results in 1011, this result is considered to be out of range as 4 bits are required. Similarly a 4-bit sign magnitude number can handle a number range between -7 and +7. -8 can not be represented as 5-bits are required 11000. A 4-bit 2's complement based signed number range is between -8 to +7.

The table 2.9 shows the range of values that can be represented by the three Binary representations using 4-bits.

Decimal Number	Sign-Magnitude form	2's complement form	Unsigned form
-8		1000	
-7	1111	1001	
-6	1110	1010	
-5	1101	1011	
-4	1100	1100	
-3	1011	1101	
-2	1010	1110	
-1	1001	1111	
0	0000	0000	000
1	0001	0001	001
2	0010	0010	010
3	0011	0011	011
4	0100	0100	100
5	0101	0101	101
6	0110	0110	110
7	0111	0111	111

Table 2.9 Range of values represented by 4-bit Binary representations

- Signed Magnitude representation can represent positive and negative numbers in the range $(2^{n-1}-1)$ and $-(2^{n-1}-1)$ where n represents the number of bits.
- 2's complement signed representation can represent positive and negative numbers in the range $(2^{n-1}-1)$ and $-(2^{n-1})$ where n represents the number of bits.
- The unsigned representation can represent positive numbers in the range 0 to 2^n-1, where n represents the number of bits.

Lesson No. 03

NUMBER SYSTEMS

Range of Numbers and Overflow

When arithmetic operation such as Addition, Subtraction, Multiplication and Division are performed on numbers the results generated may exceed the range of values specified by the Binary representations. The values that exceed the specified range can not be correctly represented and are considered as Overflow values.

For example, a 3-bit Unsigned representation can correctly represent Unsigned Binary values in the range 0 to 2^3-1 (0 to 7). Adding 3-bit Unsigned 010 (2) to another 3-bit Unsigned 111 (7) results in 1001 (9) which exceeds the 3-bit unsigned range and is considered to be an Overflow. Similarly, 1011 (-5) and 1100 (-4) values represented in 4-bit 2's complement form when added together result in 10111 (-9) which exceeds the 4-bit 2's complement range of values (2^{4-1}-1) and -(2^{4-1}) (7 to -8) and is considered as an Overflow.

Determining Overflow Conditions for 2's Complement Numbers

The Overflow condition can be easily determined when two numbers represented in 2's Complement form are added together. Consider the four examples described below. All numbers are represented in 4-bit 2's Complemented form.

- Both numbers are positive

0101	+5
0100	+4
1001	-7

The result indicates a negative number as the most significant bit is a 1. The answer is incorrect as the result should be positive. The result indicates -7. The correct answer +9 can not be represented using 4-bit 2's complemented form, thus an Overflow has occurred.

- Both numbers are negative

1011	-5
1100	-4
10111	+7

The carry generated is discarded. The result indicates a positive number as the most significant bit is a 0. The answer is incorrect as the result should be negative. The result indicates +7. The correct answer -9 can not be represented using 4-bit 2's complement form, thus an Overflow has occurred.

- One number is positive and its magnitude is larger than the negative number

0101	+5
1100	-4
10001	+1

The carry generated is discarded. The result is correct.

- One number is positive and its magnitude is smaller than the negative number

1011	-5
0100	+4

The result is correct. As 1111 represents -1.

Analysis of the four addition operation indicates that Overflow conditions can be determined by looking at the most significant sign bits of the two numbers to be added together and the most significant sign bits of the sum result. In the first two examples where an Overflow has occurred the sign bits of both the numbers are the same indicating both numbers to be positive or negative respectively. The sign bit of the sum term in both cases is opposite to the signs of the two numbers being added together which can never be. Thus the erroneous sign bits indicate the Overflow conditions.

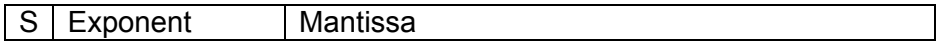
Floating-Point Numbers

Modern computers can handle large binary numbers such as 64-bit unsigned number, the maximum decimal number that can be represented using the 64-bit unsigned representation is 2^{64}-1 which is nearly equal to1.84×10^{19}.

How does a computer handle numbers larger than 2^{64}-1 or 1.84×10^{19} decimal? Secondly, numbers used routinely are not only integer numbers but numbers such as 3.14 which have an integer part and a fraction part. Thirdly, how can very small numbers such as 1.84×10^{-19} can be represented in Digital Systems?

The floating-point number system, based on scientific notation is capable of representing very large and very small numbers without having to increase the number of bits. Numbers having an integer part and a fraction part are also easily represented using the Floating-Point representation.

Floating point numbers are defined using certain standards. The ANSI/IEEE Standard 754 defines a 32-bit Single-Precision Floating Point format for binary numbers. The 32-bit Single-Precision F.P. format is shown in Figure 3.1.

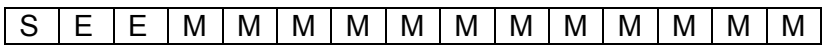


- The single Sign (S) bit represents the sign of the number (0=positive 1=negative)
- The Exponent (E) 8 bits represent the exponent
- The Mantissa 23 bits represent the magnitude of the number

Figure 3.1 Single-Precision 32-bit Floating Point Number Format

Decimal Number Floating-Point Format

To help understand how numbers are represented in the 32-bit Single Precision Floating Point format. Consider a similar 15 digit Decimal Number format to represent very large and very small decimal numbers. The 15-digit floating point format to represent decimal numbers is shown in Figure 3.2.



- The Sign (S) 1 digit represents the sign of the number (+/–)
- The Exponent (E) 2 digits represent the exponent
- The Mantissa 12 digits represent the magnitude of the number

Figure 3.2 15-digit Decimal Floating Point Number Format

The number 6918.3125 can be written as 6.9183125×10^3.

- 69183125 represents the magnitude of the number (mantissa)
- 3 represents the exponent
- The decimal point is moved to the extreme left of the number (normalized) so that the magnitude is represented by a fraction part.

The number 0.69183125×10^4 is represented in decimal f.p. notation as

+	0	4	6	9	1	8	3	1	2	5	0	0	0	0

- Using this 15 digit (including the sign digit) notation the largest number that can be represented is $0.999,999,999,999 \times 10^{99}$

Representing Negative Exponent Values

The 15-digit decimal floating-point format does not allow negative exponents to be represented. There are two options available

- Increase the Exponent field by one digit to allow for the sign to represent positive and negative exponents. The total number of digits increases to 16.
- Used a Biased Exponent scheme. Instead of writing the exponent value directly add the value 50 to the exponent and write the result in the exponent field. Using this biased scheme the maximum positive exponent value that can be represented is 49 (49 + 50 = 99). The smallest exponent that can be represented is -50 (-50 + 50 = 0).

After allowing positive and negative exponent values to be represented, the range of positive and negative decimal numbers that can be represented using the decimal f.p. notation is $0.999,999,999,999 \times 10^{49}$ to $0.999,999,999,999 \times 10^{-50}$

Representing Zero and Infinity Values

How should the number Zero and the value Infinity be represented using the 15-digit decimal floating point format?

- The number zero can be represented by setting al the Mantissa digits to 0. The Biased exponent field can be set to any number and the sign field can be set to + or –
- The number infinity can not be represented.

The solution to represent infinity is to set aside a biased exponent value to represent infinity. There are two options available

- Allow numbers having the maximum and minimum exponent values to be 48 and -49 instead of 49 and -50. Thus the Biased exponent values would range between 98 (50 + 48 = 98) and 01 (-49 + 50 = 1). The biased exponent value 00 can be used to represent the number zero whatever the value of the mantissa. The biased exponent value 99 can be used to represent the number infinity what ever the value of mantissa.
- Allow numbers having the maximum and minimum exponent values to be 49 and -48 instead of 49 and -50 and selecting 49 as the biased number. Thus the Biased exponent values would range between 98 (49 + 49 = 98) and 01 (-48 + 49 = 1). The biased exponent value 00 can be used to represent the number zero whatever the value of the mantissa.

The biased exponent value 99 can be used to represent the number infinity what ever the value of mantissa. This approach is perhaps better as the range of maximum positive exponent remains 49 and the range of values having a negative exponent have been reduced to -48.

Representing a Decimal fraction number in 32-bit Single-Precision Floating Point format

The 32-bit Single Precision Floating Point format represents the Exponent value as a Biased Number, reserving the exponent values 0 and 255 to represent the value zero and infinity respectively. The range of exponent value is from +127 to -126.

The step wise representation of a decimal number 6918.3125 in 32-bit Floating Point format

- Convert Decimal number into equivalent Binary representation: Binary equivalent of Decimal number 6918.3125 is 1101100000110.0101
- Normalizing the binary number: $1.1011000001100101 \times 2^{12}$
- Representing the exponent in Biased 127: exponent is 12 + 127 =139 = 10001011

0	10001011	10110000011001010000000

- The Mantissa is 10110000011001010000000 instead of 11011000001100101000000 as all binary numbers that are normalized always have a leading 1. In the f.p. format the leading 1 is not written, however it is taken into account in all calculations. The leading 1 which is not written is known as a hidden 1.

Arithmetic Operations on Floating Point Numbers

Arithmetic operations can be directly performed on floating point numbers by manipulating the mantissa and exponent parts of the floating point numbers.

Two floating point numbers can be added by adding together their mantissas ensuring that the exponent parts of both the numbers are the same. If the exponents of the two floating point numbers that are to be added together are not the same than decimal point has to be adjusted for one of the floating point number to make both the exponents equal. Similarly, two floating point numbers having the same exponents can be subtracted by subtracting their corresponding mantissas. If the exponents of the two numbers to be subtracted are not equal, then decimal point is adjusted to make the two exponents equal.

Multiplication is performed by multiplying the mantissas together and adding their corresponding exponents. Division is performed by dividing the mantissa parts and subtracting the corresponding exponents. The examples illustrate arithmetic operations on floating point numbers.

```
   723   represented in f.p. as  exponent 2   mantissa 7.23
 + 134   represented in f.p. as  exponent 2   mantissa 1.34
   857   Adding together the mantissa part results in
                                 exponent 2   mantissa 8.57

   723   represented in f.p. as  exponent 2   mantissa 7.23
+ 2015   represented in f.p. as  exponent 3   mantissa 2.015
  2738   Adjusting the decimal point of the first number
                                 exponent 3   mantissa 0.723
         Adding together the mantissa pert results in
                                 exponent 3   mantissa 2.738
```

723 represented in f.p. as exponent 2 mantissa 7.23
- 134 represented in f.p. as exponent 2 mantissa 1.34
589 Subtracting together the mantissa part results in
exponent 2 mantissa 5.89

2015 represented in f.p. as exponent 3 mantissa 2.015
- 723 represented in f.p. as exponent 2 mantissa 7.23
1292 Adjusting the decimal point of the second number
exponent 3 mantissa 0.723
Subtracting the mantissa pert results in
exponent 3 mantissa 1.292

723 represented in f.p. as exponent 2 mantissa 7.23
x 34 represented in f.p. as exponent 1 mantissa 3.4
24582 Multiplying the mantissa parts and adding the exponents results in
exponent 4 mantissa 24.582

697 represented in f.p. as exponent 2 mantissa 6.97
÷ 41 represented in f.p. as exponent 1 mantissa 4.1
17 Dividing the mantissa part and subtracting the exponents results in

exponent 1 mantissa 1.7

64-bit Double-Precision Floating Point format

The 32-bit Single precision floating point representation can represent largest positive or negative number of the order of 2^{127} and the smallest positive or negative number of the order of 2^{-126}. To represent numbers larger than 2^{127} and numbers smaller than 2^{-126}, 64- bit Double Precision floating point format is used.

The 64-bit Double-Precision format sets aside 11 bits to represent the exponent as Biased-1023 and a mantissa of 52 bits. A single bit, the most significant bit, is set aside for the sign.

Hexadecimal Numbers

Representing even small number such as 6918 requires a long binary string (1101100000110) of 0s and 1s. Larger decimal numbers would require lengthier binary strings. Writing such long string is tedious and prone to errors.

The Hexadecimal number system is a base 16 number system and therefore has 16 digits and is used primarily to represent binary strings in a compact manner. Hexadecimal number system is not used by a Digital System. The Hexadecimal number system is for our convenience to long binary strings in a short and concise form. Each Hexadecimal Number digit can represent a 4-bit Binary Number. The Binary Numbers and the Hexadecimal equivalents are listed in Table 3.1

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

Table 3.1 Hexadecimal Equivalents of Decimal and Binary Numbers

Counting in Hexadecimal

Counting in Hexadecimal is similar to the other number systems already discussed. The maximum value represented by a single Hexadecimal digit is F which is equivalent to decimal 15. The next higher value decimal 16 is represented by a combination of two Hexadecimal digits 10_{16} or 10 H. The subscript 16 indicates that the number is Hexadecimal 10 and not decimal 10. Hexadecimal Numbers are also identified by appending the character H after the number. The Hexadecimal Numbers for Decimal numbers 16 to 39 are listed in Table 3.2.

Decimal	Hexadecimal	Decimal	Hexadecimal	Decimal	Hexadecimal
16	10	24	18	32	20
17	11	25	19	33	21
18	12	26	1A	34	22
19	13	27	1B	35	23
20	14	28	1C	36	24
21	15	29	1D	37	25
22	16	30	1E	38	26
23	17	31	1F	39	27

Table 3.2 Counting using Hexadecimal Numbers

Binary to Hexadecimal Conversion

Converting Binary to Hexadecimal is a very simple operation. The Binary string is divided into small groups of 4-bits starting from the least significant bit. Each 4-bit binary group is replaced by its Hexadecimal equivalent.

```
11010110101110010110      Binary Number
1101 0110 1011 1001 0110  Dividing into groups of 4-bits
  D    6    B    9    6   Replacing each group by its Hexadecimal equivalent
```

Thus 11010110101110010110 is represented in Hexadecimal by D6B96

Binary strings which can not be exactly divided into a whole number of 4-bit groups are assumed to have 0's appended in the most significant bits to complete a group.

```
1101100000110          Binary Number
1 1011 0000 0110       Dividing into groups of 4-bits
0001 1011 0000 0110    Appending three 0s to complete the group
  1    B    0    6     Replacing each group by its Hexadecimal equivalent
```

Hexadecimal to Binary Conversion

Converting from Hexadecimal back to binary is also very simple. Each digit of the Hexadecimal number is replaced by an equivalent binary string of 4-bits.

```
FD13                   Hexadecimal Number
1111 1101 0001 0011    Replacing each Hexadecimal digit by its 4-bit binary equivalent
```

There are two methods to convert from Decimal to Hexadecimal. The first method is the Indirect Method and the second method is the Repeated Division Method.

1. Indirect Method

A decimal number can be converted into its Hexadecimal equivalent indirectly by first converting the decimal number into its binary equivalent and then converting the binary to Hexadecimal.

2. Repeated Division-by-16 Method

The Repeated Division Method has been discussed earlier and used to convert Decimal Numbers to Binary by repeatedly dividing the Decimal Number by 2. A decimal number can be directly converted into Hexadecimal by using repeated division. The decimal number is continuously divided by 16 (base value of the Hexadecimal number system).

The conversion of Decimal 2096 to Hexadecimal using the Repeated Division-by-16 Method is illustrated in Table 3.3. The hexadecimal equivalent of 2096_{10} is 830_{16}.

Number	Quotient after division	Remainder after division
2096	131	0
131	8	3
8	0	8

Table 3.3 Hexadecimal Equivalent of Decimal Numbers using Repeated Division

Hexadecimal to Decimal Conversion

Converting Hexadecimal Numbers to Decimal is done using two Methods. The first Method is the Indirect Method and the second method is the Sum-of-Weights method.

1. Indirect Method

The indirect method of converting Hexadecimal number to decimal number is to first convert Hexadecimal number to Binary and then Binary to Decimal.

2. Sum-of-Weights Method

A Hexadecimal number can be directly converted into Decimal by using the sum of weights method. The conversion steps using the Sum-of-Weights method are shown.

CA02	Hexadecimal number
$C \times 16^3 + A \times 16^2 + 0 \times 16^1 + 2 \times 16^0$	Writing the number in an expression
(C x 4096) + (A x 256) + (0 x 16) + (2 x 1)	
(12 x 4096) + (10 x 256) + (0 x 16) + (2 x 1)	Replacing Hexadecimal values with Decimal equivalents
49152 + 2560 + 0 + 2	Summing the Weights
51714	Decimal equivalent

Hexadecimal Addition and Subtraction

Numbers represented in Hexadecimal can be added and subtracted directly without having to convert them into decimal or binary equivalents. The rules of Addition and Subtraction that are used to add and subtract numbers in Decimal or Binary number systems apply to Hexadecimal Addition and Subtraction. Hexadecimal Addition and Subtractions allows

1. **Hexadecimal Addition**

Carry		1		
Number 1	2	A	C	6
Number 2	9	2	B	5
Sum	B	D	7	B

2. **Hexadecimal Subtraction**

Borrow		1	1	1
Number 1	9	2	B	5
Number 2	2	A	C	6
Difference	6	7	E	F

Lesson No. 04

NUMBER SYSTEMS & CODES

Octal Numbers

Octal Number system also provides a convenient way to represent long string of binary numbers. The Octal number is a base 8 number system with digits ranging from 0 to 7. Octal number system was prevalent in earlier digital systems and is not used in modern digital systems especially when the Hexadecimal number is available. Each Octal Number digit can represent a 3-bit Binary Number. The Binary Numbers and the Octal equivalents are listed in Table 4.1

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

Table 4.1 Octal Equivalents of Decimal and Binary Numbers

Counting in Octal Number System

Counting in Octal is similar to counting in any other Number system. The maximum value represented by a single Octal digit is 7. For representing larger values a combination of two or more Octal digits has to be used. Thus decimal 8 is represented by a combination of 10_8. The subscript 8 indicates the number is Octal 10 and not decimal ten. The Octal Numbers for Decimal numbers 8 to 30 are listed in Table 4.2

Decimal	Octal	Decimal	Octal	Decimal	Octal
8	10	16	20	24	30
9	11	17	21	25	31
10	12	18	22	26	32
11	13	19	23	27	33
12	14	20	24	28	34
13	15	21	25	29	35
14	16	22	26	30	36
15	17	23	27	31	37

Table 4.2 Counting using Octal Numbers

Binary to Octal Conversion

Converting Binary to Octal is a very simple. The Binary string is divided into small groups of 3-bits starting from the least significant bit. Each 3-bit binary group is replaced by its Octal equivalent.

```
11101011010111001011 0      Binary Number
111 010 110 101 110 010 110  Dividing into groups of 3-bits
 7   2   6   5   6   2   6   Replacing each group by its Octal equivalent
```

Thus 11101011010111001011 0 is represented in Octal by 7265626

Binary strings which can not be exactly divided into a whole number of 3-bit groups are assumed to have 0's appended in the most significant bits to complete a group.

1101100000110	Binary Number
1 101 100 000 110	Dividing into groups of 3-bits
001 101 100 000 110	Appending three 0s to complete the group
1 5 4 0 6	Replacing each group by its Octal equivalent

Octal to Binary Conversion

Converting from Octal back to binary is also very simple. Each digit of the Octal number is replaced by an equivalent binary string of 3-bits

1726	Octal Number
001 111 010 110	Replacing each Octal digit by its 3-bit binary equivalent

Decimal to Octal Conversion

There are two methods to convert from Decimal to Octal. The first method is the Indirect Method and the second method is the Repeated Division Method.

1. Indirect Method

A decimal number can be converted into its Octal equivalent indirectly by first converting the decimal number into its binary equivalent and then converting the binary to Octal.

2. Repeated Division-by-8 Method

The Repeated Division Method has been discussed earlier and used to convert Decimal Numbers to Binary and Hexadecimal by repeatedly dividing the Decimal Number by 2 and 16 respectively. A decimal number can be directly converted into Octal by using repeated division. The decimal number is continuously divided by 8 (base value of the Octal number system).

The conversion of Decimal 2075 to Octal using the Repeated Division-by-8 Method is illustrated in Table 4.3. The Octal equivalent of 2075_{10} is 4033_8.

Number	Quotient after division	Remainder after division
2075	259	3
259	32	3
8	4	0
4	0	4

Table 4.3 Octal Equivalent of Decimal Numbers using Repeated Division

Octal to Decimal Conversion

Converting Octal Numbers to Decimal is done using two Methods. The first Method is the Indirect Method and the second method is the Sum-of-Weights method.

1. Indirect Method

The indirect method of converting Octal number to decimal number is to first convert Octal number to Binary and then Binary to Decimal.

2. Sum-of-Weights Method

An Octal number can be directly converted into Decimal by using the sum of weights method. The conversion steps using the Sum-of-Weights method are shown.

4033	Octal number
$4 \times 8^3 + 0 \times 8^2 + 3 \times 8^1 + 3 \times 8^0$	Writing the number in an expression
$(4 \times 512) + (0 \times 64) + (3 \times 8) + (3 \times 1)$	
$2048 + 0 + 24 + 3$	Summing the Weights
2075	Decimal equivalent

Octal Addition and Subtraction

Numbers represented in Octal can be added and subtracted directly without having to convert them into decimal or binary equivalents. The rules of Addition and Subtraction that are used to add and subtract numbers in Decimal or Binary number systems apply to Octal Addition and Subtraction. Octal Addition and Subtractions allows large Binary numbers to be quickly added and subtracted.

1. Octal Addition

Carry		1			
Number 1		7	6	0	2
Number 2		5	7	7	1
Sum	1	5	5	7	3

3. Octal Subtraction

Borrow		1	1	
Number 1	7	6	0	2
Number 2	5	7	7	1
Difference	1	6	1	1

Working with different Binary representations

There are different ways of representing numbers in binary. Four ways of representing binary numbers have been already discussed.

- Unsigned binary
- Signed-Magnitude form
- 2's Complement form
- Floating point notation

The different representations help in processing of numbers. For example 2's complement based signed numbers help in handling positive and negative numbers. Floating point notations help in handling numbers having an integer and a fraction part. Digital systems generally allow processing of multiple data values that are of the same type. For example, one number represented using unsigned binary can not be used to perform arithmetic operations with another number represented using signed notation. Therefore before a digital system like a computer is able to process data it has to be explicitly informed the types of data and the manner in which they have been represented within the machine.

When computer Programs are written, usually as a first step of the program different variables and their data types are declared and defined. During program execution when ever a particular variable is accessed by the Computer it knows exactly the data type and the type

Alternate forms of Binary representations

There are many different ways to represent binary numbers, other than the 4 representation that we have discussed. Many of these alternate representations are used to support specific applications and requirements. Biased Code or Excess Code is used by floating point numbers to represent positive and negative exponent values.

In many applications in which Digital Systems are used, the Digital systems interact with the real world. For example, a digital controller controls a motor which positions a solar panel to point towards the sun to extract maximum solar energy. The controller needs to accurately know the angle at which the panel is pointing; this can be determined by the position of the shaft of the motor with respect to some reference point. The shaft position has to be encoded in some suitable format to be of use to the controller. A shaft encoder based on the Gray Code is used to read the angular position of the motor shaft.

The angular position of the motor shaft can be displayed on a 7-segment display panel in terms of Decimal Numbers. BCD Code is used to display decimal digits on 7-Segment Display Panels.

The Excess Code

Consider the decimal number range +7 to -8. These positive and negative decimal numbers can be represented by the 2's complement representation. The magnitude of positive and negative numbers can not be easily compared as the positive and negative numbers represented in 2's complement form are not represented on a uniformly increasing scale.

The decimal number range +7 to -8 is represented using an Excess-8 code that assigns 0000 to -8 the lowest number in the range and 1111 to +7 the highest number in the range. Excess-8 code is obtained by adding a number to the lowest number -8 in the range such that the result is zero. The number is 8. The number 8 is added to all the remaining decimal numbers from -7 up to the highest number +7. The Excess-8 represented is presented in Table 4.4.

Decimal	2's Complement	Excess-8	Decimal	2's Complement	Excess-8
0	0000	1000	-8	1000	0000
1	0001	1001	-7	1001	0001
2	0010	1010	-6	1010	0010
3	0011	1011	-5	1011	0011
4	0100	1100	-4	1100	0100
5	0101	1101	-3	1101	0101
6	0110	1110	-2	1110	0110
7	0111	1111	-1	1111	0111

Figure 4.4 Excess-8 Code Representation of decimal numbers in the range 7 to -8

The BCD Code

Binary Coded Decimal (BCD) code is used to represent decimal digits in binary. BCD code is a 4-bit binary code; the first 10 combinations represent the decimal digits 0 to 9. The

remaining six 4-bit combinations 1010, 1011, 1100, 1101, 1110 and 1111 are considered to be invalid and do not exist.

The BCD code representing the decimal digits 0 to 9 is shown in Table 4.4

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

Table 4.4 BCD representation of Decimal digits 0 to 9

To write 17, two BCD code for 1 and 7 are used 0001 and 0111. The two digits are considered to be separate. The conventional method of representing decimal 17 using unsigned binary is 10001. A telephone keypad having the digits 0 to 9 generates BCD codes for the keys pressed.

Most digital systems display a count value or the time in decimal on 7-segment LED display panels. Since the numbers displayed are in decimal, therefore the BCD Code is used to display the decimal numbers. Consider a 2-digit 7-segment display that can display a count value from 0 to 99. To display the two decimal digits two separate BCD codes are applied at the two 7-segment display circuit inputs.

BCD Addition

Multi-digit BCD numbers can be added together.

```
23      0010 0011
45      0100 0101
68      0110 1000
```

The two 2-digit BCD numbers are added and generate a result in BCD. In the example the least significant digits 3 and 5 add up to 8 which is a valid BCD representation. Similarly the most significant digits 2 and 4 add up to 6 which also is a valid BCD representation.

Consider the next example where the least significant numbers add up to a number greater than 9 for which there is no valid BCD code

```
23      0010 0011
48      0100 1000
71      0110 1011
```

For BCD numbers that add up to an invalid BCD number or generate a carry the number 6 (0110) is added to the invalid number. If a carry results, it is added to the next most significant digit. Thus

```
  0011
  1000
  1011        11 is generated which is an invalid BCD number
  0110        6 is added
1 0001
```

A carry is generated which is added to the result of the next most significant digits

```
   1
0110
0111
```

The answer is 0111 0001

The Gray Code

The Gray code does not have any weights assigned to its bit positions. The Gray Code is not a positional code. The Gray code is different from the unsigned binary code as successive values of Gray code differ by only one bit. Table 4.5 shows the Gray Code representation of Decimal numbers 0 to 9.

Decimal	Gray	Binary
0	0000	0000
1	000**1**	0001
2	00**1**1	0010
3	001**0**	0011
4	0**1**10	0100
5	011**1**	0101
6	01**0**1	0110
7	010**0**	0111
8	**1**100	1000
9	110**1**	1001

Table 4.5 Gray Code representation of Decimal values

The bits in **bold** change in successive values of Gray code representation

Gray Code Application

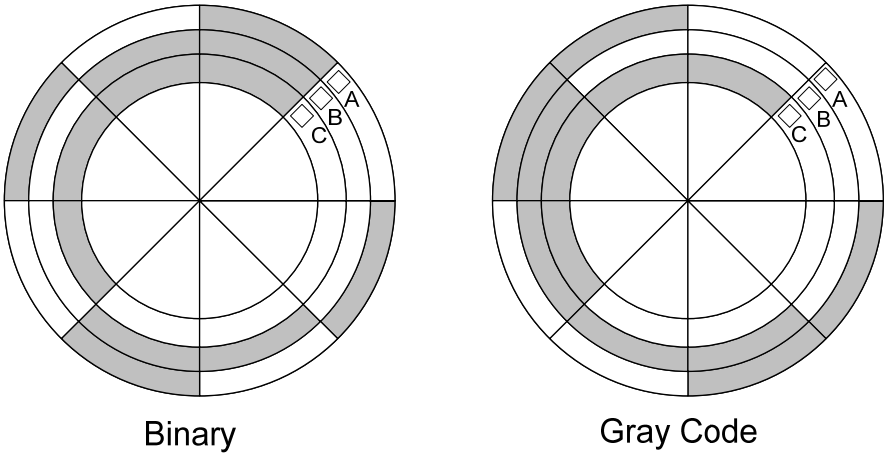


Figure 4.1 Binary and Gray Code based Shaft Encoders

The diagram shows a disk connected to the shaft of a rotating machine. The shaded areas on the disk indicate conducting area at a voltage of +5 volts. The non-shaded areas indicate a non-conducting area. Three stationary brushes A, B and C touch the surface of the rotating disk. The three brushes are connected to three LED lamps through wires. As the disk rotates the brushes come in contact with the conducting area and the insulated area. The three LEDs display the position of the rotating shaft in terms of 3-bit numbers. Thus if the disk on the right rotates in the anti-clockwise direction by 45^0 the Brush A comes in contact with the conducting strip at 5 volts, which turns on the LED indicating Binary 001.

If the disk continuous its rotation, after a rotation of another 45^0, brush B comes in contact with the conducting strip and brush A comes in contact with the non-conducting strip. Thus LED connected to brush B lights up indicating binary 010. Thus at any instant of time, the LEDs indicate the angular position of the rotating shaft.

Assume that the three brushes A, B and C are not aligned properly and Brush B is slightly ahead of brushes A and C. Now if the disk rotates 90^0 from its start position. Brush A would be in contact with the conducting strip, Brush B due to its misalignment would also be in contact with the conducting strip and brush C would be in contact with the insulated strip. Thus when the disk rotates the LEDs will show a 001, followed by a 011 for a short duration when the disk rotates from 90^0 to 91^0 and then to 010. Thus due to misalignment the count value jumped from 1 to 3 and then back to 2.

Consider the disk shown on the right. The conducting and non-conducting strips follow a Gray Code pattern 000, 001, 011, 010, 110, 111, 101 and 100 representing decimal 0, 1, 2, 3, 4, 5, 6 and 7. Now even if the brushes are misaligned, the LEDs would always display the correct count value. Thus a Gray Code based shaft encoder allows angular position of the shaft to be determined even when the brushes are misaligned.

Alphanumeric Codes

All the representation studied so far allow decimal numbers to be represented in binary. Digital systems also process text information as in editing of documents. Thus each

have a representation. Numbers are also written in textual form such as 2nd June 2003. The ASCII Code is a universally accepted code that allows 128 characters and symbols to be represented.

ASCII Code

The ASCII Code (American Standard Code for Information Interchange) is a 7-bit code representing 128 unique codes which represent the alphabet characters A to Z in lower case and upper case, the decimal numbers 0 to 9, punctuation marks and control characters.

- ASCII codes 011 0000 (30h) to 011 1001 (39h) represents numbers 0 to 9
- ASCII codes 110 0001 (61h) to 111 1010 (7Ah) represent lower case alphabets a to z
- ASCII codes 100 0001 (41h) to 101 1010 (5Ah) represent upper case alphabets A to Z
- ASCII codes 000 0000 (0h) to 001 1111 (1Fh) represent the 32 Control characters.

Extended ASCII Code

The 7-bit ASCII code only has 128 unique codes which are not enough to represent some graphical characters displayed on Computer screens. An 8-bit code Extended ASCII code gives 256 unique codes. The extended 128 unique codes represent graphic symbols which have become an unofficial standard as vendors use their own interpretation of these graphic codes.

Parity Method

Binary information which can be text or numbers is processed, stored and transmitted. Although digital systems are extremely reliable but still there is a possibility that one bit gets corrupted. That is, a 1 changes to 0 or 0 changes to 1. Many systems use a parity bit to detect errors. A single parity based error detection scheme is not very practically efficient and more elaborate and robust schemes have been designed and implemented to detect and correct multiple bit errors. However, the use of a parity bit does help in understanding the basic concept of error detection.

Consider that the 8-bit Extended ASCII Code is used to transmit text messages from one location to another remote location. An extra bit is appended with the 8 data bits making a total of nine bits. The 8-bits comprise the information that is to be stored or transmitted and the extra parity bit is appended to check for any errors that might occur during the storage or transmission of the information. Two schemes are used, Even Parity or Odd Parity essentially the two schemes are identical except for a very minor difference.

Even Parity Method

The information 10001101 is to be transmitted to a remote location. A parity bit error detection method is adopted to indicate if the information has been corrupted when it reaches the other end. In the Even Parity method the number of 1s is counted in the information and depending upon the number of 1s in the message the appended parity bit is either set to 0 or 1 to make the total number of 1s to be even (Even Parity)

The 8-bit data 10001101 has even number of 1s, therefore the parity bit which is appended is set to 0. The 9-bit message is 10001101**0**. The parity bit is indicated in Bold. Suppose the message received at the other end of the wire shows the bits to be 10<u>1</u>011101**0**, the underlined bit has changed from 0 to1. Before transmitting the message, the users at both

parity. Thus the receiver on receiving the 9-bit message does a quick parity check. The total number of bits including the parity bit should add up to an even number. However, in this case the numbers of 1 in the message add up to 5 which indicates that a bit has been corrupted. There is no way that the receiver can know the location of the corrupted bit in the message. The only solution is to request the sender to retransmit the message. If two bits get corrupted during the transmission, 10100101**0** then the total number of 1s remains the same and the receiver would not be able to detect an error. If 3-bits get corrupted, 10100001**0** the user would still be able to detect that an error has occurred, however there is no way to determine if a single bit or 3-bit, or 5-bit or 7-bit error has occurred.

Odd parity is identical except that both the sender and receiver agree to send information using the Odd parity and the parity bit is set or cleared so that the total number of 1s in the message including the Parity bit sums up to an Odd Number.

Lesson No. 05

LOGIC GATES

The Digital Systems should be able to process or perform operations on the numbers that are represented in the Binary Number System. The simplest operations that come to mind are the arithmetic operations like add and subtract. There are many more operations and functions that Digital Systems are able to perform.

Digital Logic Gates provide the basic building blocks; these Logic Gates perform different operations on the Binary information. These Logic Gates are used in different combinations to implement large complex systems. Digital Logic Gates are represented and identified by unique symbols. These symbols are used in circuit diagrams to describe the function of a digital circuit.

Digital Logic Gates function is represented by a function table or a truth table that describes all the Logic gate outputs for every possible combination of inputs. As the logic Gates operate on binary values therefore these function tables describes the relationship between the input and output in terms of binary values. The function of a Logic Gate is also described in terms of an expression.

Logic Gates are practically used in circuits where the inputs to the Logic Gates vary in time. Timing diagrams are used to describe the response of the Logic Gates in a certain period of time with respect to the changing input. Timing diagrams graphically show the actual performance (behavior) of the logic gate to the changing inputs for a predetermined period of time or sequence of input signals.

The three fundamental Gates are the AND, OR and NOT Gates.

AND Gate

The AND Gate performs a logical multiplication function. An AND Gate has multiple inputs and a single output. Most commonly used AND Gates are two input AND gates. An AND Gate is represented by the symbols shown in Figure 5.1



Figure 5.1 Symbolic representation of AND Gate

The multiplication function performed by the AND Gate is shown in the function table for a two input AND Gate. Figure 5.2. The function table for a 3, 4 or multiple input AND Gate is similar. The output is 1 when all the inputs are at logic level 1. For all other input combinations the output is zero.

Logical AND Operation		
Inputs		Output
A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

The expression describing the operation of a two inputs AND Gate is $F = A.B$ The '.' is an AND Operator and the expression represents an AND operation between inputs A and B. Expression for multiple input AND Gates is $F = A.B.C.\cdots N$, where N is the total number of inputs.

The timing diagram of the two input AND gate with the input varying over a period of 7 time intervals is shown in the diagram. Figure 5.3.

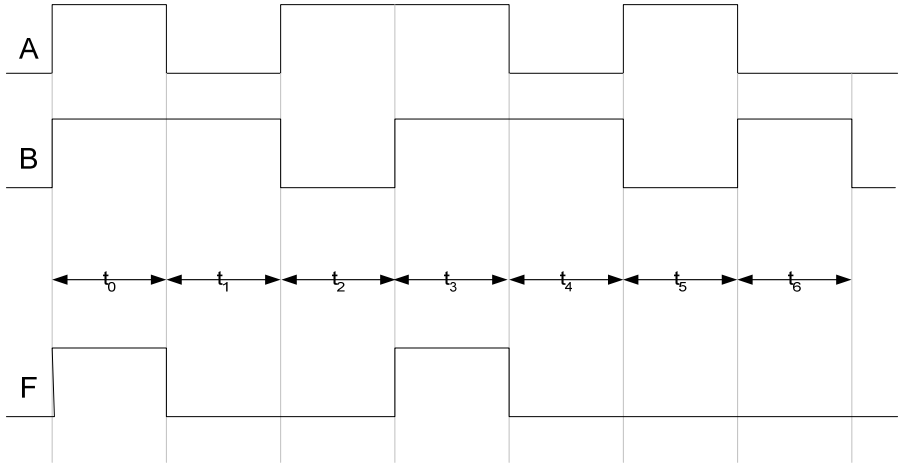


Figure 5.3 Timing diagram of operation of an AND gate

An important use of an AND gate in addition to the multiplication operation is its use to disable or enable a device. Figure 5.4. A Counter device counts from 0 to 100. The counter device increments its current count value to the next when it receives a pulse at its clock input. To allow the Counter device to count continuously from 0 to 100, continuous pulses are applied at the clock input of the Counter Device. The continuous pulses are shown as Clock pulses.

The counter can be stopped from counting by stopping the clock pulses from reaching the clock input of the Counter Device. A 2-input AND gate is connected to the Counter Clock input. The clock pulses are applied at the Input A of the AND Gate. Input B of the AND Gate is connected to an Enable/Disable signal. When the Counter Device is stopped from counting the enable/disable signal ay Input B is set to 0.

The Function Table, figure 5.2, indicates that when ever an input of the AND gate is set to 0 the output also becomes 0. Thus by applying the disable signal 0 at Input B, the output of the gate becomes zero and therefore clock signals are prevented from reaching the Counter device. To allow the Counter Device to count, the enable/disable signal at input B of the AND gate is set to 1. The Function Table of the AND gate indicates that when an Input of the AND gate is 1, the output follows the input signal applied at the input A of the AND Gate. Thus the clock signal at Output of the AND gate follows the clock signal at Input A of the AND Gate.

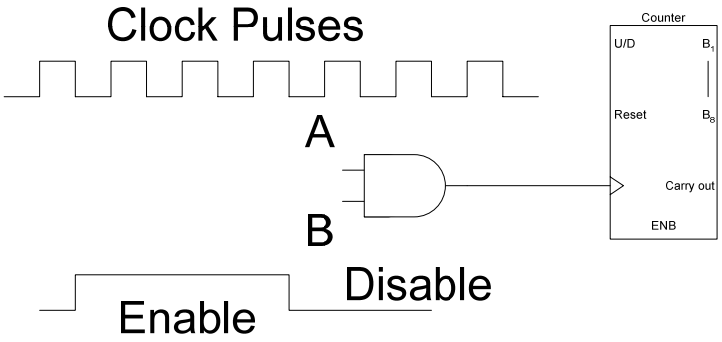


Figure 5.4 Enabling a Counter using an AND Gate

OR Gate

The OR Gate performs a Boolean add function. An OR Gate has multiple inputs and a single output. Most commonly used OR Gates are two input OR gates. An OR Gate is represented by symbols as shown in figure 5.5.



Figure 5.5 Symbolic representation of OR Gate

The addition function performed by the OR Gate is shown in the function table for a two input OR Gate. Figure 5.6. The function table for a 3, 4 or multiple input OR Gate is similar. The output is 1 when any one input is at logic level 1. The output is 0 when all inputs are zero.

The expression describing the operation of the two inputs OR Gate is $F = A + B$. The '+' is an OR Operator and the expression represents an OR operation between inputs A and B. Expression for multiple input OR Gates is $F = A + B + C + ...N$, where N is the total number of inputs.

Logical OR Operation		
Inputs		Output
A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

Figure 5.6 Function Table of an OR Gate

The timing diagram of the two input OR gate with the input varying over a period of 7 time intervals is shown in the diagram 5.7.

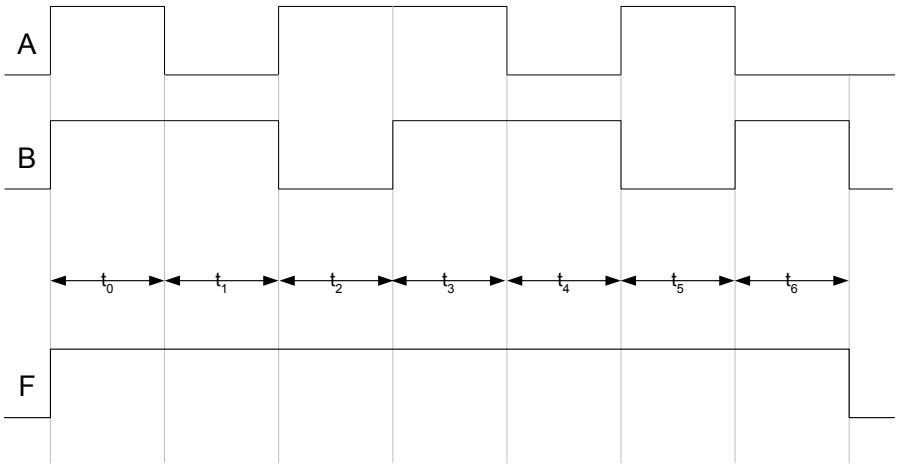


Figure 5.7 Timing diagram of operation of an OR gate

The OR Gate is used in applications where the output signal is a 1 when any one input is a 1. An example of such an application is an alarm circuit for car door locks shown in diagram, figure 5.8. Four circuits are connected to each of the four doors of a car. The door circuit generates a 1 when the door is open and a 0 when it is closed. The four outputs of each of the four door circuits are connected to the four inputs of an OR Gate. The output of the OR gate is connected to an Alarm.

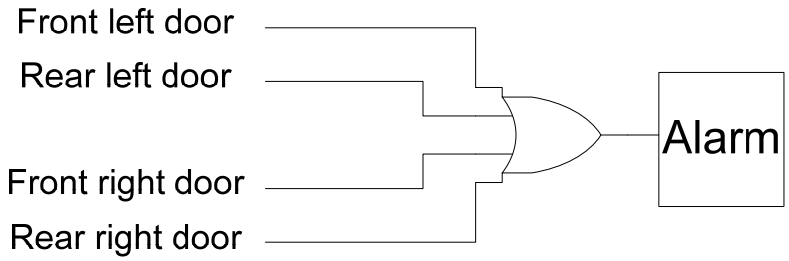


Figure 5.8 Car door Alarm System based on a 4-input OR Gate

When any one or more doors are open the inputs of the OR Gate have a 1. The output of the OR gate is a 1, according to the Function Table of an OR Gate, figure 5.6, which enables the Alarm.

NOT Gate

NOT Gate is also known as an Inverter. The name indicates that the NOT Gate should be performing an inversion function. The Not Gate has a single input and a single output. The NOT Gate is represented by the symbol shown in Figure 5.9.

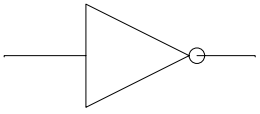


Figure 5.9 NOT Gate

The input signal applied across the single input of the OR gate is inverted and is available at the output. The function of the NOT Gates is described by the Function Table or the Truth Table represented in Figure 5.10.

Logical NOT Operation	
Input	Output
A	F
0	1
1	0

Figure 5.10 Function Table of a NOT Gate

The expression describing the behavior of a NOT gate in terms of the Input and Output shown in the Function Table, Figure 5.10 is $F = \overline{A}$ where $\overline{A}$ indicates invert of A

The timing diagram of a NOT gate with the input varying over a period of 7 time intervals and its corresponding output is shown in the Figure 5.11.

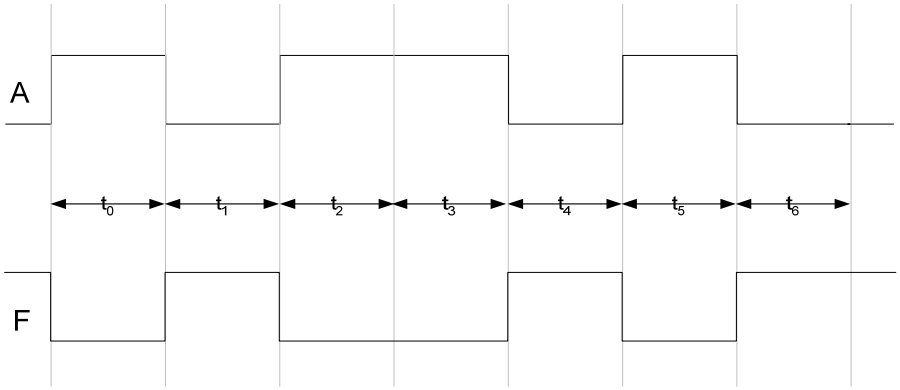


Figure 5.11 Timing diagram of operation of a NOT gate

The NOT Gate is used in circuits to generate the 1's Complement of a number by inverting all its bits. Figure 5.12. It is also used to invert an incoming signal '1' as per requirements of another circuit which requires the signal to be '0'.

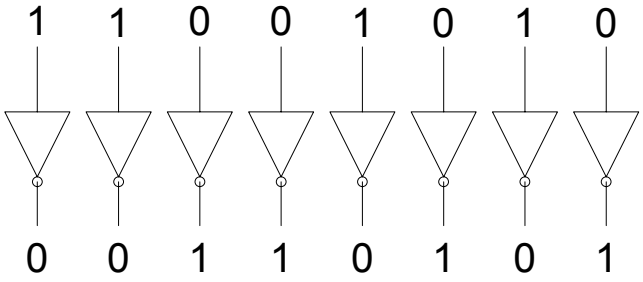


Figure 5.12 A 1's Complement Circuit using 8 NOT Gates

In addition to the three Fundamental Gates which perform AND, OR and NOT operations, two other important gates that are commonly used in Digital Logic are the NAND and NOR Gates. These two gates do not perform any new functions. The NAND Gate performs an AND-NOT function and the NOR gate performs the OR-NOT function.

AND & OR Gate alternate symbols

The AND gate and the OR gate can also be represented by alternate symbols. The two fundamental symbols, the AND Gate symbol and the OR gate symbol complement each other. Thus a gate can be represented by its complementary symbol. The inputs and outputs of the complementary symbol are inverted by placing or removing bubbles. Figure 5.13.



Figure 5.13 Alternate Symbolic representation of AND & OR gates

The AND gate is represented by its complementary OR gate symbol, the two inputs and the output are inverted by placing bubbles. The OR gate is represented by its complementary AND gate symbol, the two inputs and the output are inverted by placing bubbles.

NAND Gate

The NAND Gate performs a function that is equivalent to the function performed by the combination of an AND gate and a NOT gate. Figure 5.14

A NAND Gate has multiple inputs and a single output. Most commonly used NAND Gates are two input NAND gates. A NAND gate is represented by the symbols shown in figure 5.15, the NOT gate connected at the output of the AND gate is represented by a circle, in Digital Logic terminology a ‘bubble’.

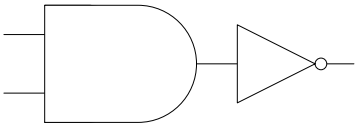


Figure 5.14 NAND Gate function

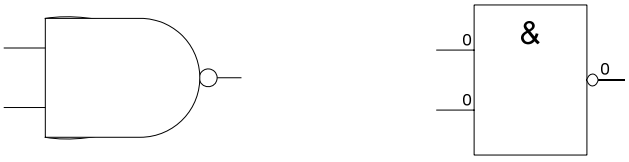


Figure 5.15 Symbolic representation of NAND Gate

The function performed by the NAND Gate is described by the Function Table for a two input NAND Gate. Figure 5.16. The function table for a 3, 4 or multiple input NAND Gate is

similar. The output is 0 when all inputs are 1s. For all other combinations of inputs the output logic level is 1.

Logical NAND Operation		
Inputs		Output
A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

Figure 5.16 Function Table of a NAND Gate

The expression describing the operation of the two inputs NAND Gate is $F = \overline{A.B}$. Expression for multiple input NAND Gates is $F = \overline{A.B.C.....N}$, where N is the total number of inputs.

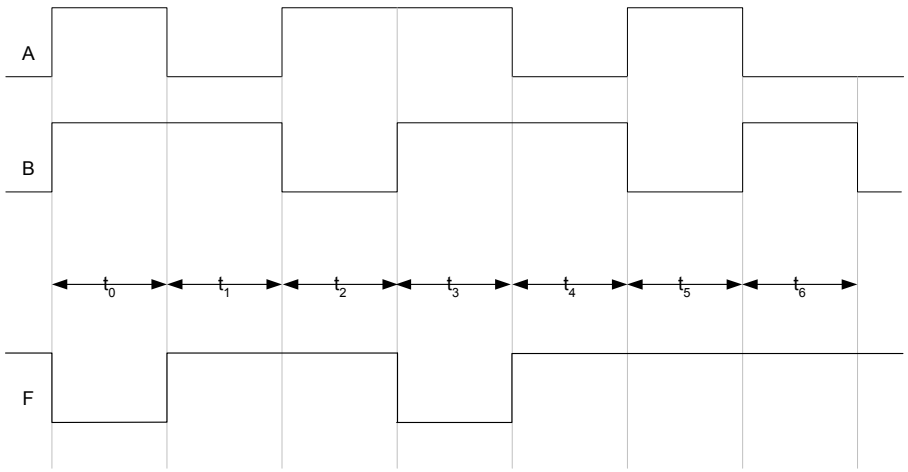
The timing diagram of the two input NAND gate with the input varying over a period of 7 time intervals is shown in the diagram. Figure 5.17.

NAND Gate as a Universal Gate

The NAND gate is also used as a Universal Gate as the NAND Gate can be used in a combination to perform the function of a AND, OR and NOT gates.

1. NOT Gate Implementation

A NOT gate can be implemented using a NAND gate by connecting both the inputs of the NAND gate together. By connecting the two inputs together, the input combinations where the inputs are dissimilar become redundant. The Function Table of the 2-input NAND Gate reduces to that of the NOT gate. Figure 5.18



Logical NAND Operation		
Inputs		Output
A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

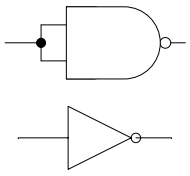


Figure 5.18 Implementing a NOT Gate using a NAND gate

2. AND Gate Implementation

A NAND Gate performs the AND-NOT function. Removing the NOT gate at the output of the NAND gate results in an AND gate. The effect of the NOT gate at the output of the NAND gate can be cancelled by connecting a NOT gate at the output of the NAND Gate. The two NOT gates cancel each other out. A NOT Gate implemented using a NAND gate (2) is connected to the output of a NAND gate (1). Figure 5.19.

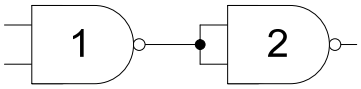


Figure 5.19 Implementing an AND Gate using two NAND gates

3. OR Gate Implementation

An OR Gate can be implemented using a combination of three NAND gates. The implementation is based on the alternate symbolic representation of the OR gate. The OR gate is represented as an AND gate with bubbles at the inputs and outputs. Figure 5.13. The two bubbles at the input can be replaced by two NOT gates (1) & (2) implemented using two NAND gates. If the two bubbles are removed from the two inputs, the AND gate with the bubble at the output represents a NAND gate (3). Figure 5.20

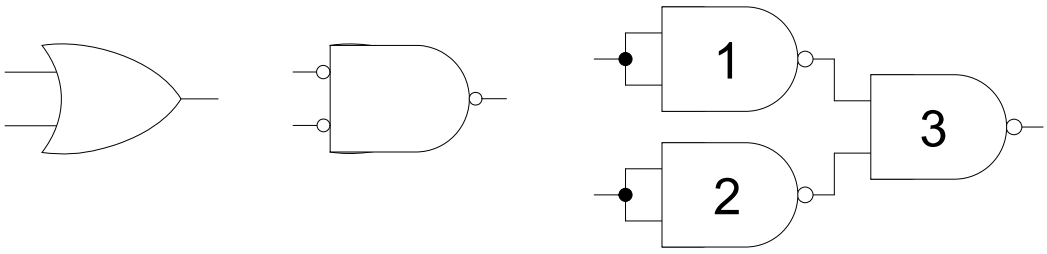


Figure 5.20 Implementing an OR Gate using three NAND gates

NOR Gate

The NOR Gate performs a function that is equivalent to the function performed by a combination of an OR gate and a NOT gate. Figure 5.21

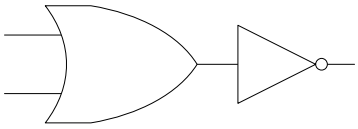


Figure 5.21 NOR Gate function

A NOR Gate has multiple inputs and a single output. Most commonly used NOR Gates are two input NOR gates. A NOR gate is represented by the symbols shown in figure 5.22, the NOT gate connected at the output of the OR gate is represented by a circle.

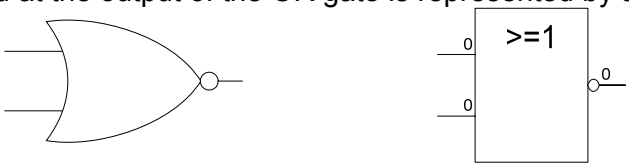


Figure 5.22 Symbolic representation of NOR Gate

The function performed by the NOR Gate is described by the Function Table for a two input NOR Gate. Figure 5.23. The function table for a 3, 4 or multiple input NOR Gate is similar. The output is 1 when all inputs are 0s. For all other combinations of inputs the output logic level is 0.

Logical NOR Operation		
Inputs		Output
A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

Figure 5.23 Function Table of a NOR Gate

The expression describing the operation of the two inputs NOR Gate is $F = \overline{A + B}$. Expression for multiple input NOR Gates is $F = \overline{A + B + C + ...N}$, where N is the total number of inputs.

The timing diagram of the two input NOR gate with the input varying over a period of 7 time intervals is shown in the diagram. Figure 5.24.

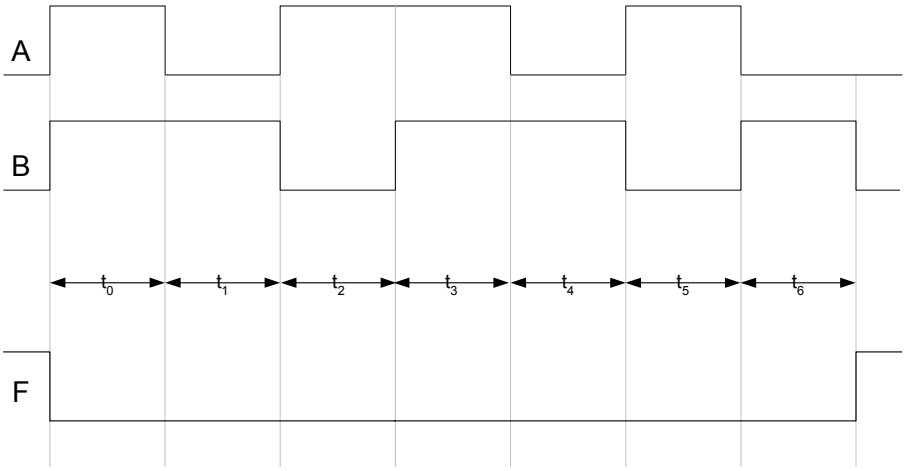


Figure 5.24 Timing diagram of operation of a NOR gate

Lesson No. 06

LOGIC GATES & OPERATIONAL CHARACTERISTICS

NOR Gate as a Universal Gate

The NOR gate is also used as a Universal Gate as the NOR Gate can be used in a combination to perform the function of a AND, OR and NOT gates.

4. NOT Gate Implementation

A NOT gate can be implemented using a NOR gate by connecting both the inputs of the NOR gate together. By connecting the two inputs together, the combinations with dissimilar inputs become redundant. The Function Table of the 2-input NOR Gate reduces to that of the NOT gate. Figure 6.1

Logical NOR Operation		
Inputs		Output
A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

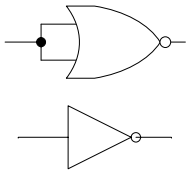


Figure 6.1 Implementing a NOT Gate using a NOR gate

5. OR Gate Implementation

A NOR Gate performs the OR-NOT function. Removing the NOT gate at the output of the NOR gate results in an OR gate. The effect of the NOT gate at the output of the NOR gate can be cancelled by connecting a NOT gate at the output of the NOR Gate. The two NOT gates cancel each other out. A NOT Gate implemented using a NOR gate (2) is connected to the output of a NOR gate (1). Figure 6.2.

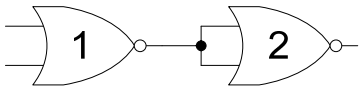


Figure 6.2 Implementing an OR Gate using two NOR gates

6. AND Gate Implementation

An AND Gate can be implemented using a combination of three NOR gates. The implementation is based on the alternate symbolic representation of the AND gate. The AND gate is represented as an OR gate with bubbles at the inputs and outputs. Figure 5.13. The two bubbles at the input can be replaced by two NOT gates (1) & (2) implemented using two NOR gates. If the two bubbles are removed from the two inputs, the OR gate with the bubble at the output represents a NOR gate (3). Figure 6.3

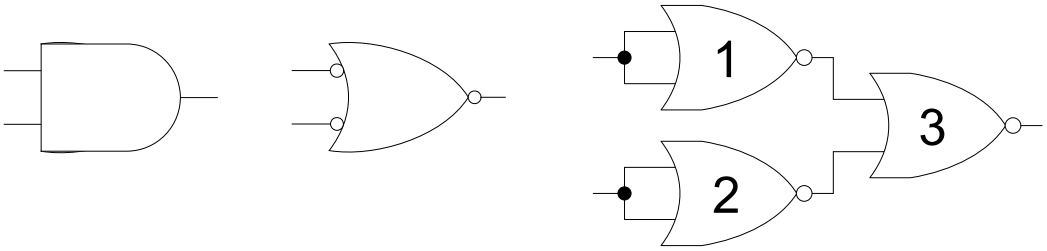


Figure 6.3 Implementing an OR Gate using three NOR gates

NAND-NOR Universal Gates

NAND and NOR gates are known as Universal Gates as they can be used to implement any of the three fundamental gates, AND, OR and NOT. The NAND Universal Gate can also be used to implement a NOR gate. Similarly, a NOR gate can be used to implement a NAND gate.

1. NAND gate Implementation using NOR gates

The AND gate implementation using three NOR gates is shown in figure 6.3. A NAND gate implementation requires addition of an inverter (NOT) gate at the output. The NOT gate is implemented using a NOR gate. Figure 6.4. NOR gates 1, 2 and 3 implement the AND gate. NOR gate 4 implements the NOT gate connected at the output of the NAND gate.

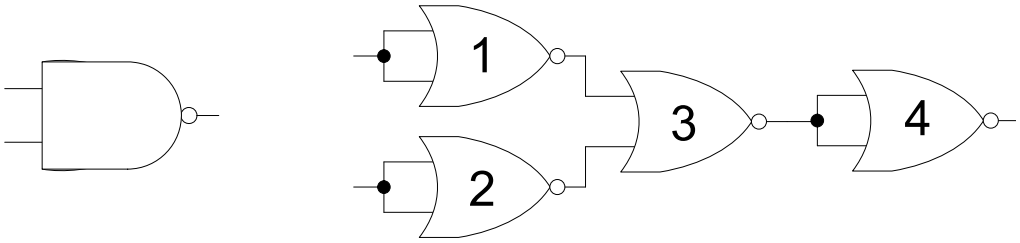
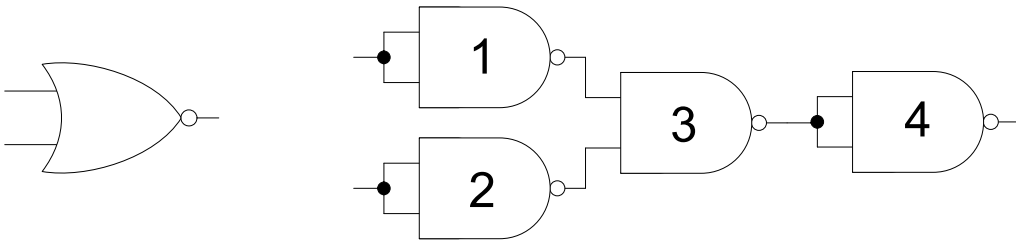


Figure 6.4 Implementing a NAND Gate using four NOR gates

2. NOR gate Implementation using NAND gates

The OR gate implementation using three AND gates is shown in figure 5.20. A NOR gate implementation requires addition of an inverter (NOT) gate at the output. The NOT gate is implemented using a NAND gate. Figure 6.5. NAND gates 1, 2 and 3 implement the OR gate. NAND gate 4 implements the NOT gate connected at the output of the NOR gate.



NAND and NOR Gate Applications

The output of a NAND is 0 when all inputs to the NAND gate are 1s. This property of the NAND gate can be used to activate an operation when any of the inputs to the NAND gate are deactivated. A NOR gate on the other hand generates an output of 1 when all inputs to NOR gate are deactivated. The output is deactivated when any input is activated.

A warehouse is used to store industrial chemicals. Toxic fumes produced by the chemicals are removed from the ware house and dispersed in the atmosphere through three exhaust fans. The three exhaust fans should be continuously working to remove the dangerous toxic fumes. If any one or more fans fail an alarm should be activated to signal the failure of one or more exhaust fans.

An electronic circuit connected to each fan generates a 1 to indicate a working fan. If the fan fails the circuit generates a 0 output. The outputs of the three fans are connected to the three inputs of a NAND gate. When all fans are working the input to the 3-input NAND gate is 111 and the corresponding output is a 0. When any one fan fails the output of NAND gate becomes 1 activating an alarm connected to the output of the NAND gate. Figure 6.6

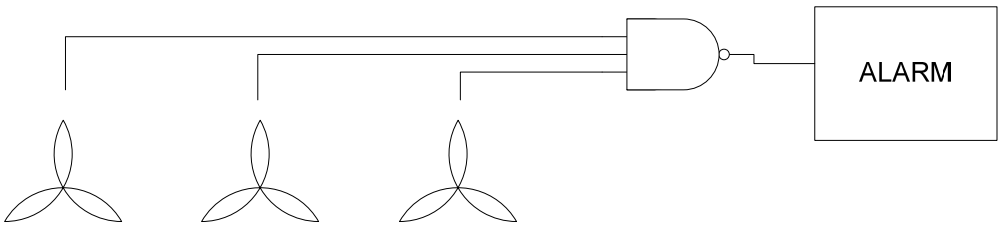


Figure 6.6 A NAND gate based exhaust fan failure detection system

A Washing Machine has three sensors to check for washing machine lid open, washing tub filled to minimum level and weight of cloths and water in the tub. If the lid of the Washing machine is open or the water is below the minimum level or the washing machine has been overloaded the appropriate sensor generates an output of 1. The outputs of the three sensors are connected to the inputs of a 3-input NOR gate. During the normal operation of the Washing Machine all the sensors output a 0. The corresponding output of the NOR gate is a 1. If an erroneous condition is detected by any one or more sensors, the corresponding sensor output(s) is set to 1, setting the NOR gate output to a 0. The NOR gate output is connected to the main switch which switches off the washing machine. Figure 6.7.

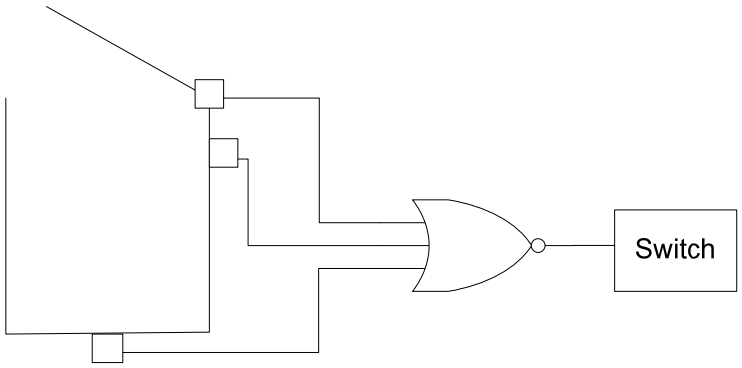


Figure 6.7 A NOR gate based Washing Machine Controller

Exclusive-OR and Exclusive-NOR Gates

The XOR and XNOR gates are frequently used in Digital Logic. These two additional gates are used to detect dissimilar and similar inputs respectively.

1. Exclusive-OR Gate

The Exclusive-OR Gate or XOR Gate performs a function that is equivalent to the combination of NOT, AND and OR gates. XOR gates are extensively used in digital applications; therefore XOR gates are available as basic components. Most commonly used XOR Gates have two inputs. The XOR gate is represented by symbol shown in figure 6.8.

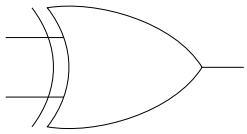


Figure 6.8 Symbolic representation of XOR Gate

The function performed by the XOR gate is represented by the Function Table for a two input XOR Gate. Figure 6.9. The function table for a 3, 4 or multiple input XOR Gate is similar. The output of an XOR gate is 1 when the inputs are dissimilar and a 0 when all the inputs are the same.

Logical XOR Operation		
Inputs		Output
A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

Figure 6.9 Function Table of an XOR Gate

The expression describing the operation of the two inputs XOR Gate is $F = A \oplus B$. The $\oplus$ is an XOR operator and the expression for multiple input XOR Gates is $F = A \oplus B \oplus C \oplus .....N$, where N is the total number of inputs.

The timing diagram of the two input XOR gate with the input varying over a period of 7 time intervals is shown in the diagram. Figure 6.10.

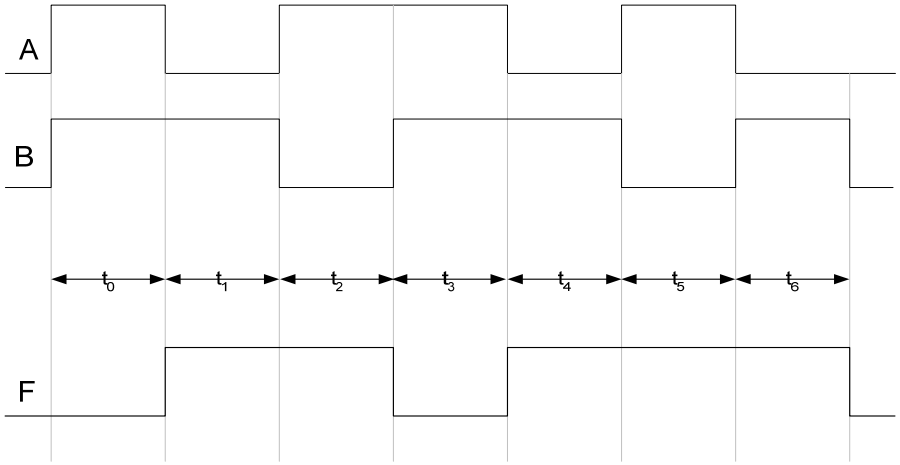


Figure 6.10 Timing diagram of operation of a XOR gate

2. Exclusive-NOR Gate

The Exclusive-NOR Gate or XNOR Gate performs a function that is equivalent to the combination of NOT, AND and OR gates. XNOR gate is extensively used in digital applications; therefore XNOR gates are available as basic components. Most commonly used XNOR Gates have two inputs. The XNOR gate is represented by symbol shown in figure 6.11.

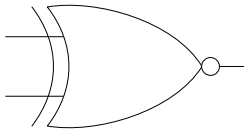


Figure 6.11 Symbolic representation of XNOR Gate

The function performed by the XNOR Gate is represented by the Function Table for a two input XNOR Gate. Figure 6.12. The function table for a 3, 4 or multiple input XNOR Gate is similar. The output of an XNOR gate is 1 when the all the inputs are same and a 0 when the inputs are dissimilar.

Logical XNOR Operation		
Inputs		Output
A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

Figure 6.12 Function Table of an XNOR Gate

The expression describing the operation of the two inputs XNOR Gate is $F = \overline{A \oplus B}$. The expression for multiple input XNOR Gates is $F = \overline{A \oplus B \oplus C \oplus .....N}$, where N is the total number of inputs.

The timing diagram of the two input XNOR gate with the input varying over a period of

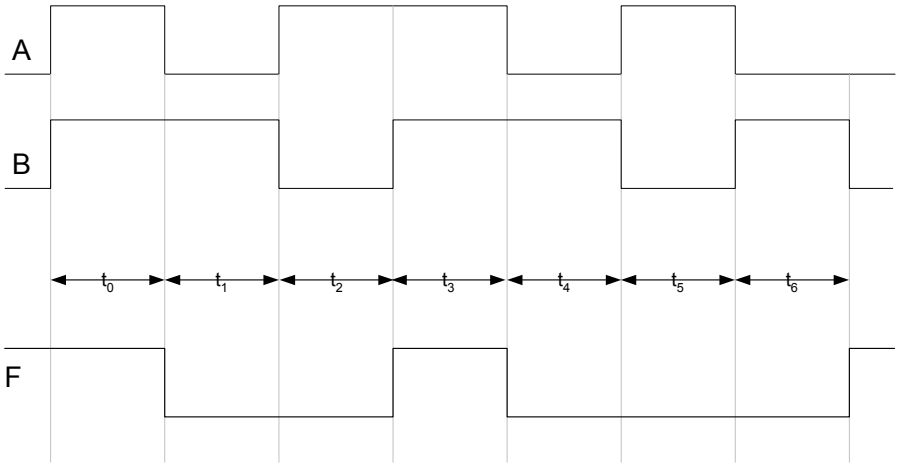


Figure 6.13 Timing diagram of operation of a XNOR gate

XOR and XNOR Gate Applications

XOR and XNOR gates are used to detect dissimilar and similar inputs. This property of XOR and XNOR gates is used to detect odd and even number of 1s in a Parity Detection Circuit.

Consider the three XOR gate logic circuit which is used to detect odd number of 1's in a 4-bit binary input combination. Figure 6.14

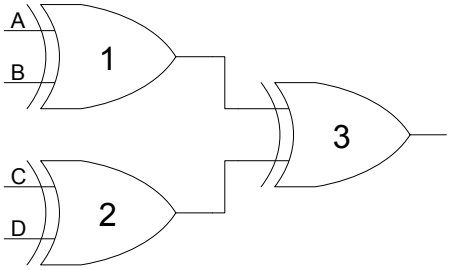


Figure 6.14 XOR gate based Odd number of 1s detector

A 4-bit binary number 0000 applied at the inputs A, B, C and D respectively of XOR gates 1 and 2. The output of XOR Gates 1 and 2 is 0 and 0. The output of XOR gate 3 is also zero. Similarly, a binary number 0011 applied at the inputs A, B, C and D respectively. The output of XOR gate 1 with inputs 00 is 0. The output of XOR gate 2 with inputs 11 is 0. The output of gate 3 is 0. Thus the output indicates that the binary number 0011 does not have odd number of 1's. Consider the binary number 1011 applied at the inputs A, B, C and D respectively. The output of XOR gate 1 with inputs 10 is 1. The output of XOR gate 2 with inputs 11 is 0. The output of gate 3 is 1. Thus the output indicates that the binary number 1011 has odd number of 1's

The logic circuit based on two XOR and a single XNOR gate which is used to detect even number of 1's in a 4-bit binary input combination. Figure 6.15

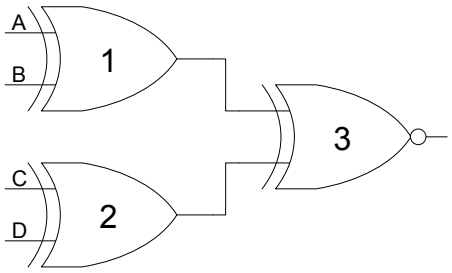


Figure 6.15 XOR-XNOR gate based Even number of 1s detector

A 4-bit binary number 0000 applied at the inputs A, B, C and D respectively of XOR gates 1 and 2. The output of XOR Gates 1 and 2 is 0 and 0. The output of XNOR gate 3 is a 1. Similarly, a binary number 0011 applied at the inputs A, B, C and D respectively. The output of XOR gate 1 with inputs 00 is 0. The output of XOR gate 2 with inputs 11 is 0. The output of XNOR gate 3 is also a 1. Thus the output indicates that the binary number 0011 has even number of 1's. Consider the binary number 1011 applied at the inputs A, B, C and D respectively. The output of XOR gate 1 with inputs 10 is 1. The output of XOR gate 2 with inputs 11 is 0. The output of XNOR gate 3 is 0 because of dissimilar inputs. Thus the output indicates that the binary number 1011 does not have even number of 1's.

Digital Circuits and Operational Characteristics

The Logic Gates discussed provide the basic building blocks for implementing the large digital systems. The logic gates discussed so far has been described in terms of the functions they perform. Practical implementation of digital systems by using the logic gates in combination requires some additional information. For example, theoretically the output of an Inverter can be connected to the inputs of an unlimited number of AND Gates. However, the practical limitation to the circuit shown is that the total current sourced by the Inverter is distributed amongst the 10 AND Gates. The Inverter is not able to provide the total current required by the ten AND gates. The current sunk by each AND gate is not enough to drive the AND gate circuitry thus its behavior is unpredictable resulting in unpredictable behavior of the system.

The binary 1 and 0 are represented by +5V and 0 V. What if the output of an AND Gate is +3 V? Does this output voltage level represent a binary 1 or 0? If the output of the AND Gate is connected to the input of an Inverter, what would be the response of the Inverter? Another important aspect is the frequency of the input signal. Electronic circuits operate at certain frequencies. If the frequency of the input signal increases beyond the operational specification of the circuit, the circuit will not be able to respond fast enough resulting in unpredictable behavior.

Digital circuits that depend upon battery for their power should consume low power to allow the device to function for longer periods of time before replacing or recharging the battery. Thus the digital system should be implemented keeping in view the power requirements of the application.

TTL/CMOS NOT Gate Operation

Logic Gates are implemented using transistors. These transistors are connected in various combinations to form a switching circuit. The transistor itself is configured to work like a switch. On the application of a bias voltage the transistor is switched on and by removing the bias voltage the transistor is turned off. Different technologies are used to manufacture the Logic Gates based on the transistors. The performance or the Operational characteristics of a Logic Gate are determined by the transistors and the technologies used to implement the switching transistors. Certain technologies allow transistor and thereby the Logic Gates to operate at high frequencies. Other technologies allow transistors to operate with low voltages, consuming minimal power, similarly certain other implementation technologies allow very dense logic circuits to be manufactured.

The Inversion function of the NOT gate is performed by the switching circuit shown in figure 6.16. The Bipolar Junction Transistor (BJT) based NOT shown on the left is switched on when a Voltage is applied at the base of the BJT. The transistor when switched on short circuits the V_{CC}, the output voltage is therefore 0 volts. When the BJT base pin is connected to 0 volts, the transistor is switched off. The $V_{o/p}$ is at potential V_{CC} = 5 Volts. The actual implementation is different.

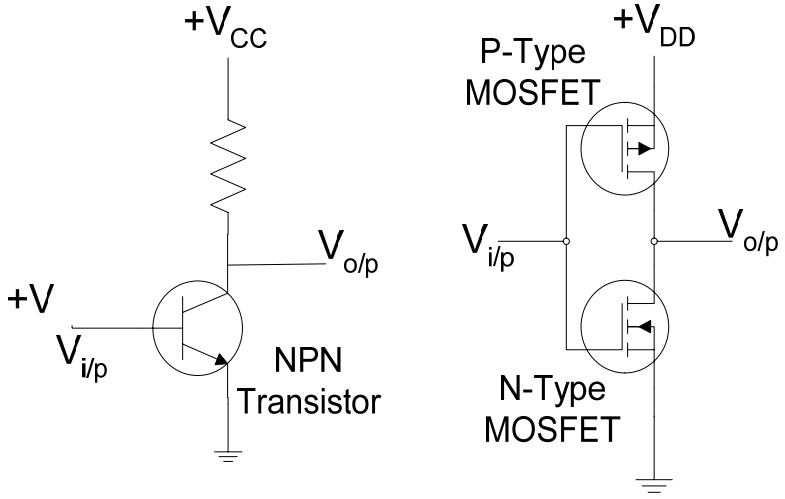


Figure 6.16 BJT & CMOS based NOT Gate Implementation

The CMOS based implementation, shown on the right, uses a P-type and a N-type MOSFETs. When the input is connected to +V, the P-type MOSFET is switched off and the N-type MOSFET is switched on. The $V_{o/p}$ is at ground potential. When the input is connected to ground, the P-type and N-type MOSFETs are switched on and off respectively. The $V_{o/p}$ is at potential V_{DD} = 5 Volts.

Integrated Circuit Technologies

The practical implementation of the Logic gates is through the Integrated Circuits (IC) technologies. The logic gates implemented through these technologies are available to be connected and practical implementation of a digital circuit. Different types of Integrated Circuit technologies are used to implement the digital circuits. These technologies differ in terms of the circuit density, power consumptions, frequency response etc.

- CMOS: Complementary Metal-Oxide Semiconductor

  - The most extensively used technology, characterized by low power consumption, switching speed which is slower but comparable to TTL. Has higher chip density than TTL. Due to high input impedance is easily damaged due to accumulated static charge
- TTL: Transistor-Transistor Logic
  - Extensively used technology, characterized by fast switching speed and high power consumption
  - Offers a wide variety of gates, devices, arithmetic units etc.
- ECL: Emitter-Coupled Logic
  - Used in specialized applications where switching speed is of prime importance such as high speed transmission, high speed memories and high speed arithmetic units.
- PMOS: p-channel and NMOS: n-channel MOS transistor
  - PMOS and NMOS technologies are used in LSI requiring high chip density. Large memories and microprocessors are implemented using these technologies
  - These ICs have very low power consumption.
- E^2CMOS: a combination of CMOS and NMOS technologies
  - Used to implement Programmable Logic Devices

Types of IC Logic Gates

The most common form of logic Gate ICs are listed. To identify and use the Integrated Circuits or ICs in implementing logic circuits, some sort of identification code has to be used which is printed on the IC package.

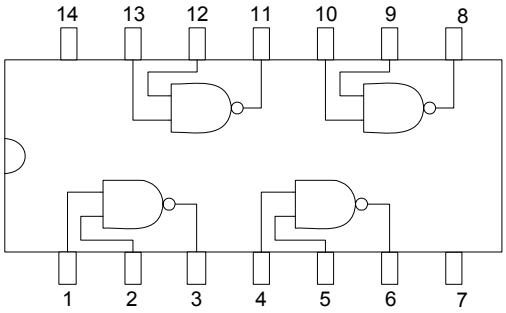
Logic Gates are identified by the codes. The prefix 74 is used to identify a commercial version of the device from the military version device identified by the prefix 54. Military versions are designed to withstand harsh and severe environmental conditions. The XX part of the code identifies the switching speed of the gate.

- 74XX00 Quad 2-input NAND Gate
- 74XX02 Quad 2-input NOR Gate
- 74XX04 Hex Inverter
- 74XX08 Quad 2-input AND Gate
- 74XX10 Triple 3-input NAND Gate
- 74XX11 Triple 3-input AND Gate
- 74XX20 Dual 4-input NAND Gate
- 74XX21 Dual 2-input AND Gate
- 74XX27 Triple 3-input NOR Gate
- 74XX30 Single 8-input NAND Gate
- 74XX32 Quad 2-input OR Gate
- 74XX86 Quad 2-input XOR Gate
- 74XX133 Single 13-input NAND Gate

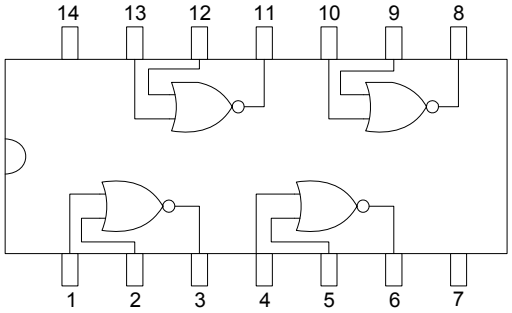
The Integrated Circuit packages of the seven gates that have been discussed so far are shown. Figure 6.17. The 7408 14-pin chip has 4 or Quad, 2-input AND gates. The input pins and the output pins of each of the four gates are shown. To use any one or all four gates the appropriate pins are connected. Pins 7 and 14 are connected to ground and Supply voltage respectively.

The 7432 14-pin IC package has 4 or Quad, 2-input OR Gates. Connections to the OR gates are identical to those of the 7408 AND gate IC. The 7404 14-pin chip has 6 or hex, inverters. The input and output connections of each of the 6 NOT gates are shown. Pins 7 and

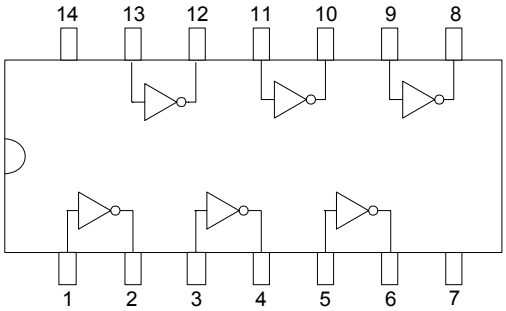
The 7400, Quad, 2-input NAND Gate IC, the 7402, Quad, 2-input NOR Gate IC, the 7486, Quad, 2-input XOR Gate IC and the 74266, Quad, 2-input XNOR Gate IC are similar.



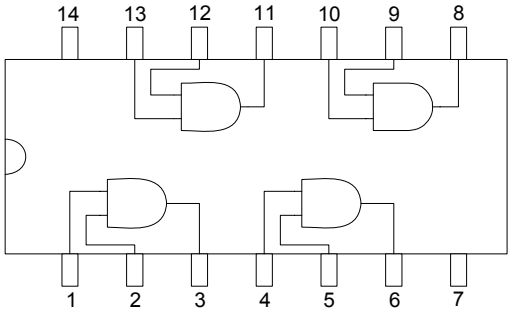
7400
Four 2-Input NAND Gate



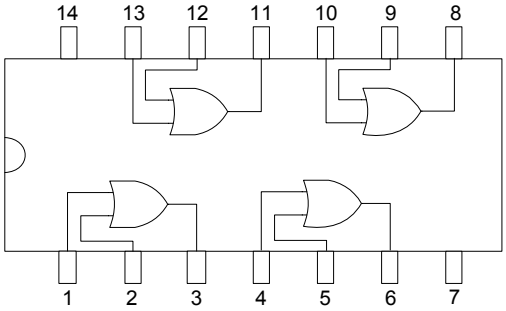
7402
Four 2-Input NOR Gate



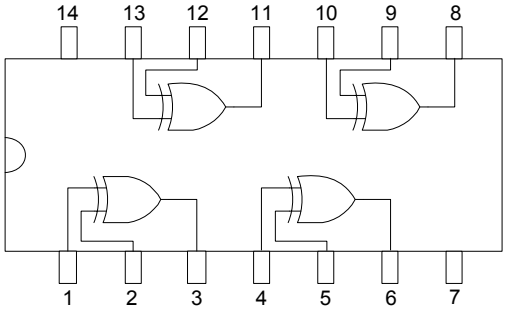
7404
Hex Inverters



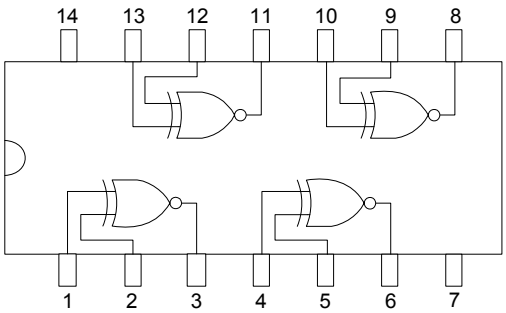
7408
Four 2-Input AND Gate



7432
Four 2-Input OR Gate



7486
Four 2-Input XOR Gate



74266
Four 2-Input XNOR Gate

Performance Characteristics and Parameters

A number of performance characteristics and parameters determine the suitability of a particular IC technology for a particular application. The important parameters that are considered whilst designing Digital Logic Circuits are mentioned briefly.

- DC Supply Voltage:
  - The supply voltage at which the Gate operates
- Noise Margin:
  - The maximum and minimum voltages that represent binary 0 and 1 respectively. These voltage ranges determine the suitability of a gate to work in noisy environments.
- Power Dissipation:
  - Gates consume power during their operation. The power dissipation varies with the frequency at which these gates operate.
- Frequency Response and Propagation Delay:
  - Gates do not instantaneously switch to a new output state after the inputs are changed. The delay between the input and output limits the frequency at which the inputs to a logic gate can be changed and the logic circuit can operate.
- Fan-Out:
  - The number of gates that can be connected to the output of a single gate.

Lesson No. 07

DIGITAL CIRCUITS AND OPERATIONAL CHARACTERISTICS

1. DC Supply Voltage

TTL based devices work with a dc supply of +5 Volts. TTL offers fast switching speed, immunity from damage due to electrostatic discharges. Power consumption is higher than CMOS. The TTL family has six different types of devices characterized by different power dissipation and switching speeds. The series of TTL chips are:

- 74 Standard TTL
- 74S Schottky TTL
- 74AS Advanced Schottky TTL
- 74LS Low-Power Schottky TTL
- 74ALS Advanced Low-Power Schottky TTL
- 74F Fast TTL

The Standard, the Schottky, the Advanced Schottky, the Low-Power Schottky, the Advanced Low-Power Schottky and the FAST TTL series are characterized by their switching speed and power dissipation. The Standard TTL is the slowest and consumes more power and the Advanced low power Schottky has the fastest switching speed and low power requirements.

CMOS technology is the dominant technology today and used in large scale ICs and microprocessors. CMOS technology is characterized by low power dissipation with slow switching speeds. There are two categories of CMOS in terms of the dc supply voltage. The 3.3 v CMOS series is characterized by fast switching speeds and very low power dissipation as compared to the 5 v CMOS series.

- +5 V CMOS
  - 74HC and 74HCT High-Speed
  - 74AC and 74ACT Advanced CMOS
  - 74AHC and 74AHCT Advanced High Speed
- 3.3 V CMOS
  - 74LV Low voltage CMOS
  - 74LVC Low-voltage CMOS
  - 74ALVC Advanced Low voltage CMOS

2. Logic Levels and Noise Margin

The TTL and CMOS circuit operating at +5 or 3.3 Volts respectively are designed to accept voltages in a certain range as logic 1 and 0. The input and output logic levels for CMOS and TTL are shown in the figure 7.1. The V_{IH} and V_{IL} indicate the acceptable voltage ranges for the input logic high and low respectively. Similarly V_{OH} and V_{OL} indicate the acceptable output voltage range for logic high and low respectively.

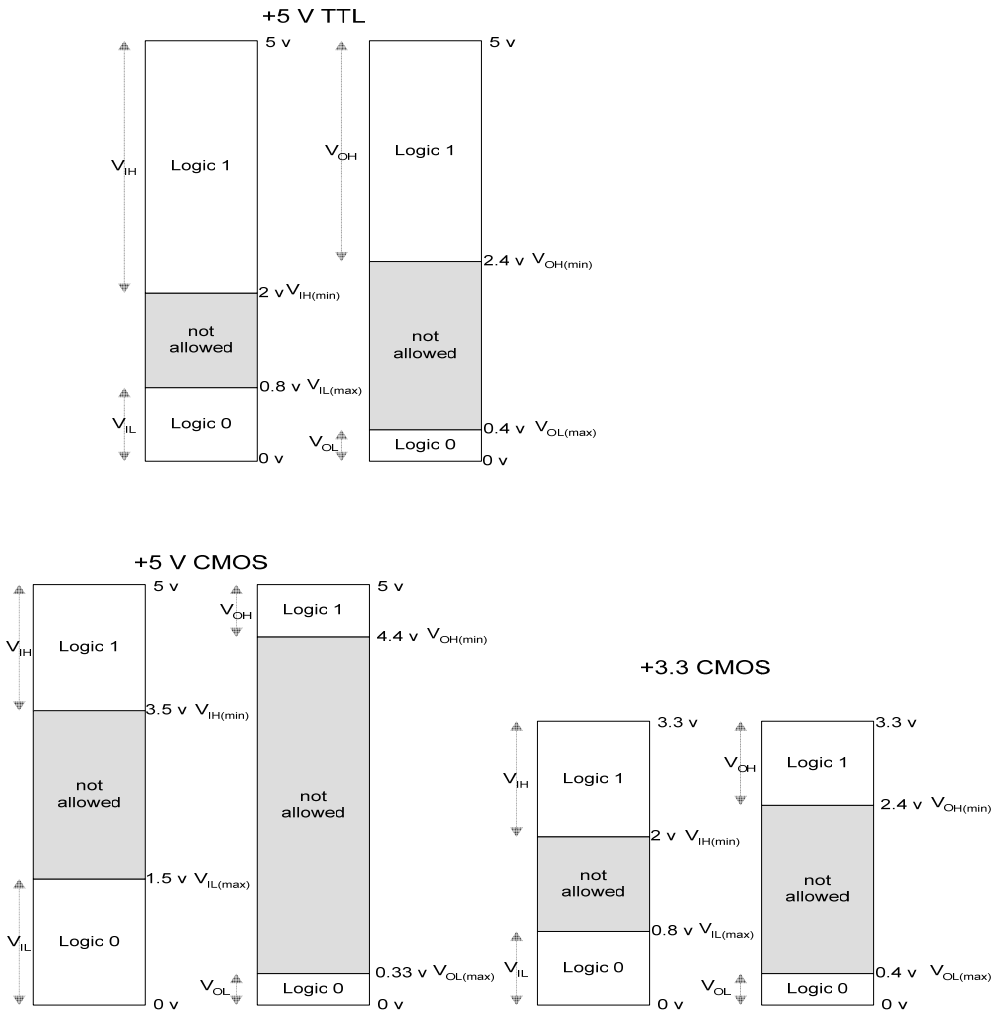


Figure 7.1 Logic Levels for TTL and CMOS Series

a) TTL Logic Levels

At the input of any TTL logic gate logic high '1' or a logic low '0' signal is applied.

- V_{IH} is the input voltage range of Logic high signal with a range of 2 to 5 volts.
- $V_{IH(min)}$ is the minimum acceptable input range for a logic high signal. (2 volts)
- V_{IL} is the input voltage range of Logic low signal with a range of 0 to 0.8 volts.
- $V_{IL(max)}$ is the maximum acceptable input range for a logic low signal. (0.8 volts)

The output of any TTL logic gate can be at logic high '1' or logic low '0'.

- V_{OH} is the output voltage range of Logic high signal with a range of 2.4 to 5 volts.
- $V_{OH(min)}$ is the minimum acceptable output range for a logic high signal. (2.4 volts)
- V_{OL} is the output voltage range of Logic low signal with a range of 0 to 0.4 volts.
- $V_{OL(max)}$ is the maximum acceptable output range for a logic low signal. (0.4 volts)

b) CMOS 5 Volt series Logic Levels

At the input of any CMOS 5 volt series logic gate logic high '1' or a logic low '0' signal is

- V_{IH} is the input voltage range of Logic high signal with a range of 3.5 to 5 volts.
- $V_{IH(min)}$ is the minimum acceptable input range for a logic high signal. (3.5 volts)
- V_{IL} is the input voltage range of Logic low signal with a range of 0 to 1.5 volts.
- $V_{IL(max)}$ is the maximum acceptable input range for a logic low signal. (1.5 volts)

The output of any CMOS 5 volt series logic gate can be at logic high '1' or logic low '0'
- V_{OH} is the output voltage range of Logic high signal with a range of 4.4 to 5 volts.
- $V_{OH(min)}$ is the minimum acceptable output range for a logic high signal.(4.4 volts)
- V_{OL} is the output voltage range of Logic low signal with a range of 0 to 0.33 volts.
- $V_{OL(max)}$ is the maximum acceptable output range for a logic low signal. (0.33 volts)

c) CMOS 3.3 Volt series Logic Levels

At the input of any CMOS 3.3 volt series logic gate a logic high '1' or a logic low '0' signal is applied.
- V_{IH} is the input voltage range of Logic high signal with a range of 2 to 3.3 volts.
- $V_{IH(min)}$ is the minimum acceptable input range for a logic high signal. (2 volts)
- V_{IL} is the input voltage range of Logic low signal with a range of 0 to 0.8 volts.
- $V_{IL(max)}$ is the maximum acceptable input range for a logic low signal. (0.8 volts)

The output of any CMOS 3.3 volt series logic gate can be at logic high '1' or logic low '0'
- V_{OH} is the output voltage range of Logic high signal with a range of 2.4 to 3.3 volts
- $V_{OH(min)}$ is the minimum acceptable output range for a logic high signal. (2.4 volts).
- V_{OL} is the output voltage range of Logic low signal with a range of 0 to 0.4 volts.
- $V_{OL(max)}$ is the maximum acceptable output range for a logic low signal. (0.4 volts).

The valid output voltages representing logic high and low are confined to certain voltage ranges. For example, low-power 3.3 volt CMOS chips output logic high voltage ranges between 2.4-3.3 volts and logic low ranges between 0-0.4 volts. Output voltages that are not within the specified ranges can cause logic circuits to malfunction.

A low-power 3.3v CMOS AND gate will accept a voltage of 2.1 volts as a valid logic high input. However, a voltage of 1.9 volts is unacceptable as an input between 0.8-2.0 volts will give unpredictable results, therefore input voltages within this range is not allowed.

Wires in electronic circuits pick up noise from adjacent conductors. Noise is unwanted voltage that is induced in the circuit due to high-frequency electromagnetic radiation. The unwanted noise can affect the performance of a logic gate and the digital circuit.

Effect of Noise on the Operation of a CMOS AND Gate

A CMOS 5 volt series AND gate is shown. Figure 7.2. Input A of the AND gate is permanently connected to logic high of +5 volts. Input B of the AND gate is connected to the output of some other gate. The signal at input B of the AND gate can vary between logic 0 and logic 1.

Consider that the input B is at logic High state with V_{IH} = 4.2 volts which is within the valid CMOS V_{IH} voltage range of 5 to 3.5 volts. A voltage generated due to some external noise (shown by the zigzag line) rides on the 4.2 volt signal. A sharp dip in the input voltage due to the noise brings the input voltage down to 3 volts for a very short duration. The 3 volt input is below the minimum input voltage limit of 3.5 volts for logic high input voltage and within

the not allowed voltage range. This dip in the voltage even for a short duration will result in an output of logic low for a short interval of time.

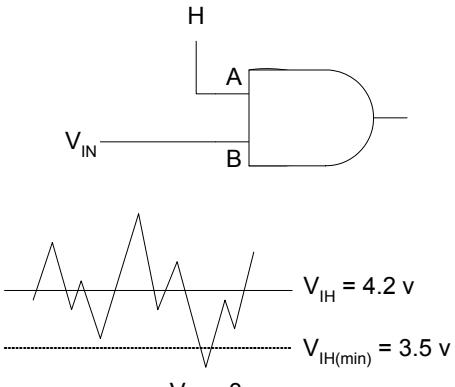


Figure 7.2 Effect of Noise on CMOS AND gate

Effect of Noise on the Operation of a CMOS AND Gate circuit

Two CMOS 5 volt series AND gates are connected together. Figure 7.3 The first AND gate has both its inputs connected to logic high, therefore the output of the gate is guaranteed to be logic high. The logic high voltage output of the first AND gate is assumed to be 4.6 volts well within the valid V_{OH} range of 5-4.4 volts. Assume the same noise signal (as described earlier) is added to the output signal of the first AND gate.

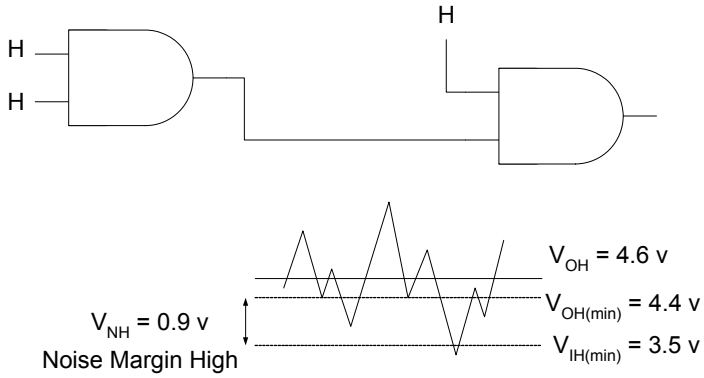


Figure 7.3 Effect of Noise on CMOS AND gate circuit

The sharp dip due to noise brings the V_{OH} voltage down to 3.4 volts with reference to the V_{OH} of 4.6 volts. 3.4 volts is lower than the $V_{IH(min)}$ of 3.5 volts required by the input of the second AND gate, the circuit will thus malfunction.

Since $V_{OH(min)}$ is guaranteed to be at 4.4 volts therefore a noise signal being added to 4.4 volts can bring V_{OH} voltage down to a minimum of 3.5 volts which is the acceptable minimum range for V_{IH}. Anything below 3.5 will cause the second gate to malfunction. Thus the second AND gate can tolerate a maximum variation of 0.9 volts for its logic high input or has a 'Noise Margin' of 0.9 volts.

Noise Margin

Noise margin is a measure of the circuit's immunity to noise. The high-level and low-level noise margins are represented by V_{NH} and V_{NL} respectively.

- $V_{NH} = V_{OH(min)} – V_{IH(min)}$
- $V_{NL} = V_{IL(max)} – V_{OL(max)}$

CMOS 5 volt series Noise Margins
- $V_{NH} = V_{OH(min)} – V_{IH(min)} =$ 4.4 - 3.5 = 0.9 v
- $V_{NL} = V_{IL(max)} – V_{OL(max)}$ = 1.5 – 0.33 = 1.17 v

CMOS 3.3 volt series Noise Margins
- $V_{NH} = V_{OH(min)} – V_{IH(min)}$ = 2.4 – 2.0 = 0.4 v
- $V_{NL} = V_{IL(max)} – V_{OL(max)}$ = 0.8 – 0.4 = 0.4 v

TTL 5 volt Noise Margins
- $V_{NH} = V_{OH(min)} – V_{IH(min)} =$ 2.4 - 2.0 = 0.4 v
- $V_{NL} = V_{IL(max)} – V_{OL(max)}$ = 0.8 – 0.4 = 0.4 v

The CMOS 5 volts and the 3.3 volts series can not be mixed.

For CMOS 5 volt series the high-level noise margin is 0.9 volts. That is, the logic high output of the gate would never be below 4.4 volts. Even if it is below 4.4 volts due to some external noise, the input will consider any voltage above 3.5 volts to be logic high. So CMOS 5 volt series gates can withstand noisy signals riding on logic high inputs up to a noise margin of 0.9 volts. Similarly, low-level noise margin is 1.17 volts (1.5-0.33).

The V_{NH} high-level and V_{NL} low-level noise margins for TTL 5 volt and CMOS 3.3 series are 0.4 volts and 0.4 volts respectively. Therefore in noisy environments, CMOS 5 volt series based digital system perform better.

3. Power Dissipation

Logic Gates and Logic circuits consume varying amount of power during their operation. Ideally, logic gates and logic circuit should consume minimal power. Advantages of low power consumption are circuits that can be run from batteries instead of mains power supplies. Thus portable devices that run on batteries use Integrated circuits that have low power dissipation. Secondly, low power consumption means less heat is dissipated by the logic devices; this means that logic gates can be tightly packed to reduce the circuit size without having to worry about dissipating the access heat generated by the logic devices. Microprocessors for example generate considerable heat which has to be dissipated by mounting small fans.

Generally, the Power dissipation of TTL devices remains constant throughout their operation. CMOS device on the other hand dissipate varying amount power depending upon the frequency of operation.

a) Power Dissipation of TTL Devices

When a TTL logic gate output is in a logic high state it draws out a current from the dc power supply. It is said to be sourcing current. The high current is designated by I_{CCH}, typical value for I_{CCH} is 1.5 mA when V_{CC} = 5 V. When a TTL logic gate output is in a logic low state it sinks a current designated by I_{CCL} = 3.0 mA when V_{CC} = 5 V. The figure 7.4 shows an AND gate connected to output a logic high ‘1’. It thus draws a current I_{CCH} from the voltage supply V_{CC}.

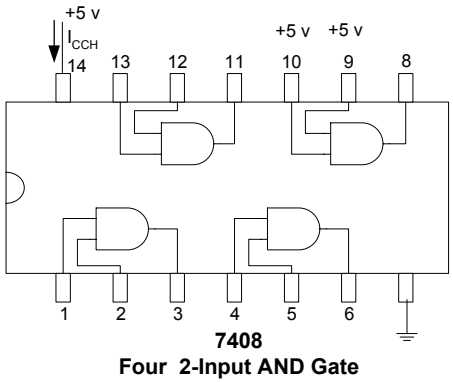


Figure 7.4 Power dissipation of a TTL AND gate

When any one of the AND gate input is connected to low, the output becomes low and it sinks current I_{CCL}. An AND Gate which has one of its input connected to a clock which continuously changes from logic high to low sets the AND gate output to high and low respectively for every one half of the clock cycle. Thus the AND gate sources and sinks currents I_{CCH} and I_{CCL} respectively.

The power dissipated by a gate is V_{CC} x I_{CC}. The power dissipated would be different for a gate having a logic high output and logic low output. The average power dissipated is determined, based on a 50% duty cycle, that is, the gate is pulsed and its output switches between high and low for every one half of the cycle.

$P_D = V_{CC}(I_{CCH} + I_{CCL})/2$

Power Dissipation in TTL circuits is constant over its range of operating frequencies. For example, the power dissipation of a LS TTL gate is a constant 2.2 mW.

b) Power Dissipation of CMOS Devices

The transistors used in CMOS logic present a capacitive load instead of the resistive load in TTL based logic. Each time a CMOS logic gate switches between low and high, current has to be supplied to the capacitive load. The typical supply current is 5 mA for a duration of 20-30 nsec. As the frequency of operation increases, there would be more of these current spikes occurring per second, thus the average current drawn from the voltage source increases.

Power Dissipation in CMOS circuits is frequency dependent. It is extremely low under static (dc) conditions and increases as the frequency increases. Total Dynamic Power dissipation of a CMOS circuit is

$P_D = P_T + P_L$

where P_T is the internal power dissipation of the gate
P_L is the external power dissipation due to the external capacitive load

$P_D = C_{PD}.V_{DD}^2.f + C_L.V_{DD}^2.f$
$P_D = (C_{PD+} C_L).V_{DD}^2.f$
where C_{PD} is the internal power dissipation capacitance
C_L is the external load dissipation capacitance
V_{DD} is the supply voltage

The power dissipation of a HCMOS gate is 2.75 μW under static conditions and 170 μW at 100 KHz.

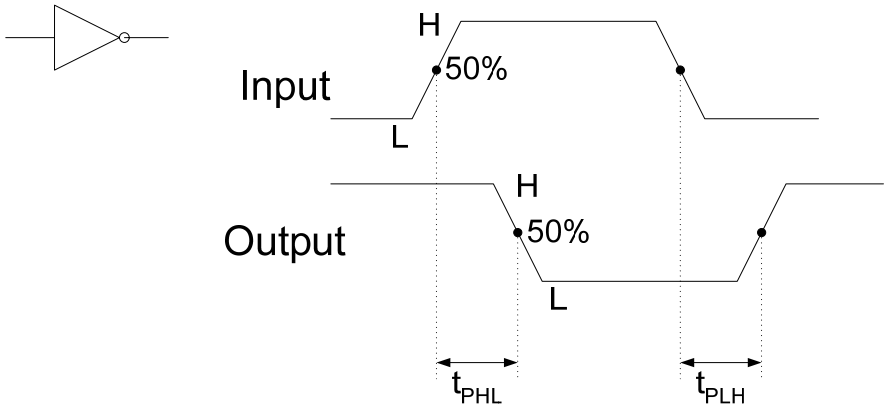
4. Propagation Delay

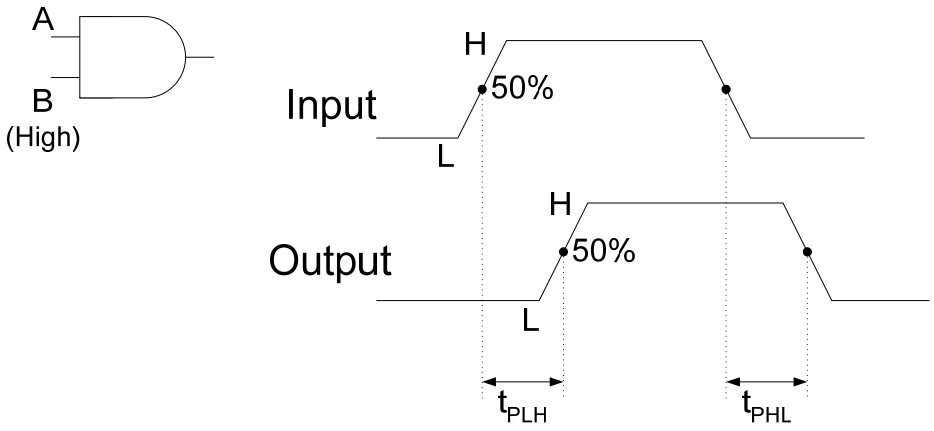
When ever a signal passes through a gate it experiences a delay. That is, a signal applied to the input of a gate does not result in an instantaneous response. The output of a gate is delayed with respect to the input. The delay in the output is known as the Propagation Delay.

The Propagation Delay of a gate limits the frequencies at which the gate can work. Higher the Propagation Delay lower is the frequency at which the gate can operate. Smaller the Propagation Delay higher the frequency at which the gate can operate. A Gate with a Propagation Delay of 3 nsec is faster than a gate with a 10 nsec delay.

There are two Propagation Delay times specified for Logic Gates. Figure 7.5

- t_{PHL} The time between a specified reference point on the input pulse and a corresponding reference point on the resulting output pulse, with the output changing from high level to low level.
- t_{PLH} The time between a specified reference point on the input pulse and a corresponding reference point on the resulting output pulse, with the output changing from low level to high level.





The output of the NOT gate changes from high to low after a delay of time specified by t_{PHL} after the input changes from low to high. The output of the NOT gate changes from low to high after a delay of time specified by t_{PLH} after the input changes from high to low. The delay time is measured at the 50% transition mark.

The input B of the AND gate is permanently connected to logic high, where as input A varies between High and Low. The output of the AND gate changes from low to high after a delay of time specified by t_{PLH} after the input changes from low to high. The output of the AND gate changes from high to low after a delay of time specified by t_{PHL} after the input changes from high to low. The delay time is measured at the 50% transition mark. Generally, the t_{PLH} and t_{PHL} propagation delay times are same.

The effect of Propagation Delay on the operation of a digital circuit can be explained with the help of an example. Consider a Cricket Stadium, entry to the Cricket Stadium is through three gates, each manned by a security guard who allows the spectator into the stadium after checking the ticket. Assume that the security guards at Gates A, B and C take 1, 1.5 and 2 minutes respectively to check the ticket and allow the spectator into the stadium. Assuming equal number of spectators queuing up at the three gates, the queue at gate C after 30 minutes is the longest as the guard at Gate C has the longest Propagation Delay.

5. Speed-Power Product (SPP)

An important parameter is the Speed-Power Product which is used as a measure of performance of a logic circuit taking into account the propagation delay and the power dissipation.

The SPP = $t_P P_D$ expressed in Joules (J), the unit of energy. Lower the SP product better is the performance.

6. Fan-Out and Loading

The fan-out of a logic gate is the maximum numbers of inputs of the same series in an IC family that can be connected to a gate's output and still maintain the output voltage levels within the specified limits. Fan-out parameter is associated with TTL technology. CMOS circuits have very high impedance therefore fan-out of CMOS circuits is very high but depends upon the frequency because of capacitance effects.

Fan-out is specified in terms of unit loads. A unit load for a logic gate equals one input to a like circuit. Consider a 7400 NAND gate. The output current at logic high is I_{OH} = 400 µA. The input current at logic high is I_{IH} = 40 µA. Thus a gate at logic high can source current to another gate connected to its output.

Similarly, the output current at logic low is I_{OL} = 16 mA. The input current at logic low is I_{IL} = 1.6 mA. Thus a gate output at logic low can sink current from another gate connected to its output.

Unit Loads = $I_{OH}/I_{IH} = I_{OL}/I_{IL}$ = 400 µA/40 µA = 16 mA/1.6 mA = 10

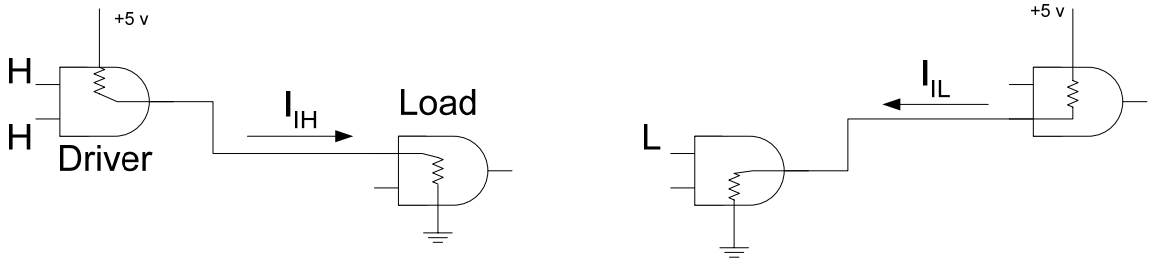


Figure 7.6 AND Gate Sourcing and Sinking Current

As more gates (Loads) are connected to the driving gate the loading on the driving gate increases. The total current sourced by the driving gate increases. As the current increases the internal voltage drop increases causing the output voltage V_{OH} to decrease. If excessive number of gates are connected the output voltage V_{OH} drops below the $V_{OH(min)}$ reducing the High-level noise margin, thus compromising the circuit operation. Also as the source current increases the power dissipation increases. Figure 7.7.

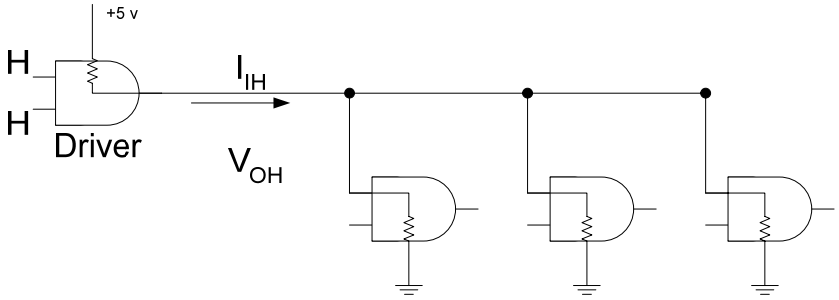


Figure 7.7 AND Gate Sourcing Current

The total sink current also increases with each load gate that is added. As the sink current increases the internal voltage drop of the driving gate increases causing V_{OL} to increase. If excessive number of loads are connected, V_{OL} exceeds $V_{OL(max)}$ and the Low noise margin is reduced.

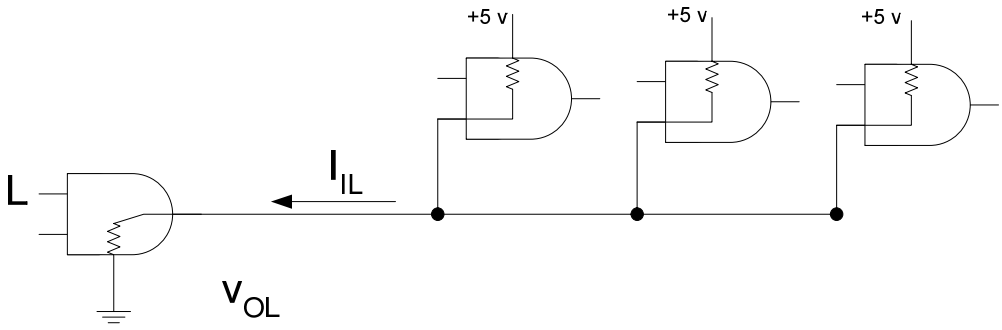


Figure 7.8 AND Gate Sinking Current

CMOS loading is different from TTL loading as the type of transistors used in CMOS circuits presents a capacitive load to the driving gate. When the output of the driving gate is

is low the load gate is discharging. When more load gates are added the input capacitance increases as input capacitances are being connected in parallel. With the increase in the capacitance, charging and discharging time increases, reducing the maximum frequency at which the gate can operate. Figure 7.9

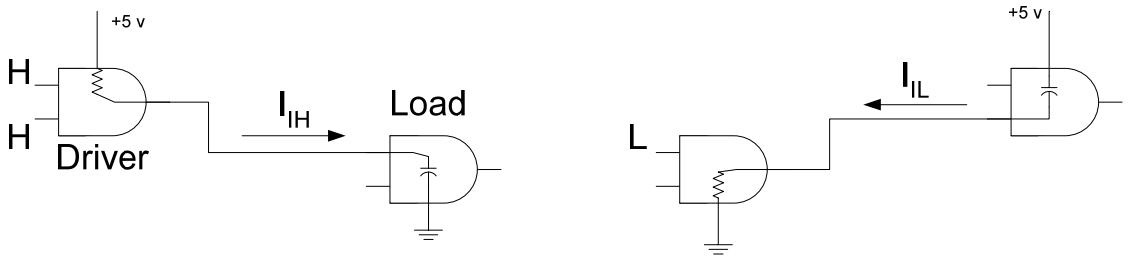


Figure 7.9 CMOS AND Gate Sourcing and Sinking Current

The fan-out of a CMOS gate depends upon the maximum frequency of operation. Fewer the load gates, greater the maximum frequency of operation.

Different TTL series are characterized by switching speed and power consumption as shown in the table. Table 7.1

	74	74S	74LS	74AS	74ALS	74F
Performance Rating						
Propagation Delay (ns)	9	3	9.5	1.7	4	3
Power Dissipation (mW)	10	20	2	8	1.2	6
Speed-Power product (pJ)	90	60	19	13.6	4.8	18
Max. Clock Rate (MHz)	35	125	45	200	70	100
Fan-out (same series)	10	20	20	40	20	33

	74HC	74AC	74AHC
Performance Rating			
Propagation Delay (ns)	18	5	3.7
Power Dissipation (mW) Static	0.00275	0.0055	0.00275
Power Dissipation (mW) Dynamic 100KHz	0.0625	0.08	0.0625
Speed-Power product (pJ) at 100KHz	1.125	0.4	0.23
Max. Clock Rate (MHz)	50	160	170

	74LV	74LVC	74ALVC
Performance Rating			
Propagation Delay (ns)	9	4.3	3
Power Dissipation (mW) Static	0.0016	0.0008	0.0008
Max. Clock Rate (MHz)	90	100	150

Table 7.1 Operational Characteristics of TTL and CMOS families

Lesson No. 08

BOOLEAN ALGEBRA AND LOGIC SIMPLIFICATION

Any digital circuit no matter how complex can be described by Boolean Expressions. Boolean algebra is the mathematics of Digital Systems. Knowledge of Boolean algebra is indispensable to the study and analysis of logic gates. AND, OR, NOT, NAND and NOR gates perform simple Boolean operations and Boolean expressions represent the Boolean operations performed by the logic gates.

- AND gate $F = A.B$
- OR gate $F = A + B$
- NOT gate $F = \overline{A}$
- NAND gate $F = \overline{A.B}$
- NOR gate $F = \overline{A + B}$

Boolean expressions which represent Boolean functions help in two ways. The function and operation of a Logic Circuit can be determined by Boolean expressions without implementing the Logic Circuit. Secondly, Logic circuits can be very large and complex. Such large circuits having many gates can be simplified and implemented using fewer gates. Determining a simpler Logic circuit having fewer gates which is identical to the original logic circuit in terms of the function it performs can be easily done by evaluating and simplifying Boolean expressions.

Boolean Algebra expressions are written in terms of variables and literals using laws, rules and theorems of Boolean Algebra. Simplification of Boolean expressions is also based on the Boolean laws, rules and theiorems.

Boolean Algebra Definitions

1. Variable

A variable is a symbol usually an uppercase letter used to represent a logical quantity. A variable can have a 0 or 1 value.

2. Complement

A complement is the inverse of a variable and is indicated by a bar over the variable. Complement of variable X is $\overline{X}$. If X = 0 then $\overline{X}$ = 1 and if X = 1 then $\overline{X}$ = 0.

3. Literal

A Literal is a variable or the complement of a variable.

Boolean Addition

Boolean Addition operation is performed by an OR gate. In Boolean algebra the expression defining Boolean Addition is a sum term which is the sum of literals.

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

- A sum term is 1 when any one literal is a 1
- A sum term is 0 when all literals are a 0.

Boolean Multiplication

Boolean Multiplication operation is performed by an AND gate. In Boolean algebra the expression defining Boolean Multiplication is a product term which is the product of literals.

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

- A product term is 1 when all literal terms are a 1
- A product term is 0 when any one literal is a 0.

Laws of Boolean Algebra

The basic laws of Boolean Algebra are the same as ordinary algebra and hold true for any number of variables.

1. Commutative Law for addition and multiplication
2. Associative Law for addition and multiplication
3. Distributive Law

1. Commutative Law for Addition and Multiplication

- Commutative Law for Addition A + B = B + A
- Commutative Law for Multiplication A.B = B.A

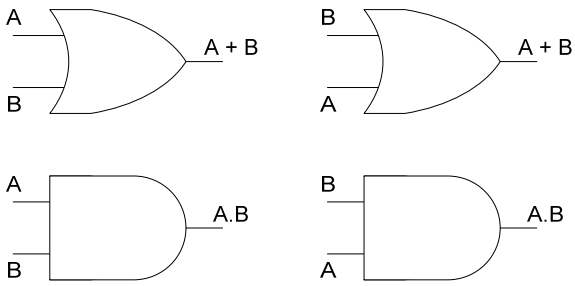
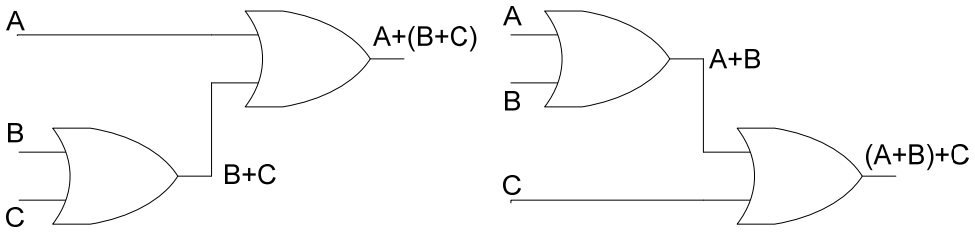


Figure 8.1 Implementation of Commutative Laws

In terms of implementation, the Boolean Addition and Multiplication of two or more literals is the same no matter how they are ordered at the input of an OR and AND Gates respectively. Commutative law for Addition and Multiplication holds true for n number of literals.

2. Associative Law for Addition and Multiplication

- Associative Law for Addition A + (B + C) = (A + B) + C
- Associative Law for Multiplication A.(B.C) = (A.B).C



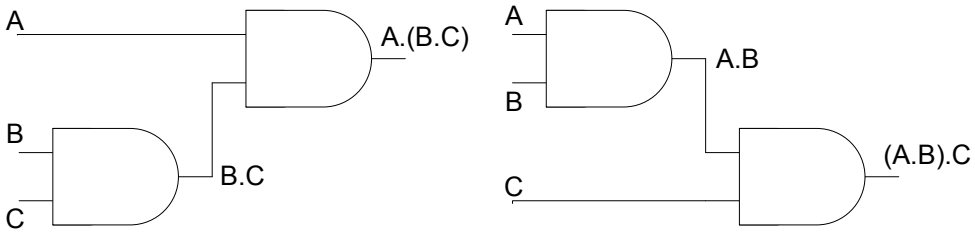


Figure 8.2 Implementation of Associative Laws

In terms of implementation, the Associative ordering of literals for Boolean Addition and Multiplication is the same at the input of an OR and AND gates. Commutative law for Addition and Multiplication holds true for n number of literals. The addition of literals B and C followed by the addition of literal A with the result of B+C is the same as adding literals A and B followed by the addition of literal C.

The multiplication of literals B and C followed by the multiplication of the result of B.C with literal A is the same as multiplying literals A and B followed by the multiplication of literal C.

3. Distributive Law

- Distributive Law A.(B + C) = A.B + A.C

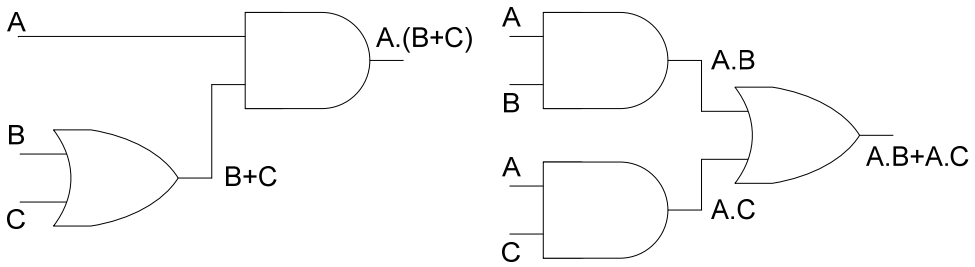


Figure 8.3 Implementation of Distributive Law

Distributive law holds true for any number of literals. Adding literals B and C followed by multiplying the result with literal A is the same as multiplying literal A with literal B and adding the result to the product of literals A and C.

Rules of Boolean Algebra

Rules of Boolean Algebra can be proved by replacing the literals with Boolean values 0 and 1.

1. A + 0 = A
2. A + 1 = 1
3. A.0 = 0
4. A.1 = A
5. A + A = A
6. $A + \overline{A} = 1$
7. $A.\overline{A} = A$

9. $\overline{\overline{A}} = A$
10. A + A.B = A
    = A.(1 + B) where (1+B) according to Rule 2 is equal to 1
    = A
11. A + $\overline{A}$.B = A + B
    = A(B+1) + $\overline{A}$.B according to Rule 2 (B+1) = 1
    = AB +A + $\overline{A}$.B
    = B(A+ $\overline{A}$) +A according to Rule 6 A + $\overline{A}$ = 1
    = B + A
12. (A+B).(A+C) = A+B.C
    = AA+AC+AB+BC applying the Distributive Law
    = A(1+C+B) +BC according to Rule 2 (1+B+C) = 1
    = A+BC

Demorgan's Theorems

Demorgan's First Theorem states: The complement of a product of variables is equal to the sum of the complements of the variables.

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

Demorgan;s Second Theorem states: The complement of sum of variables is equal to the product of the complements of the variables.

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

Demorgan's two theorems prove the equivalency of the NAND and negative-OR gates and the NOR and negative-AND gates respectively. Figure 8.4

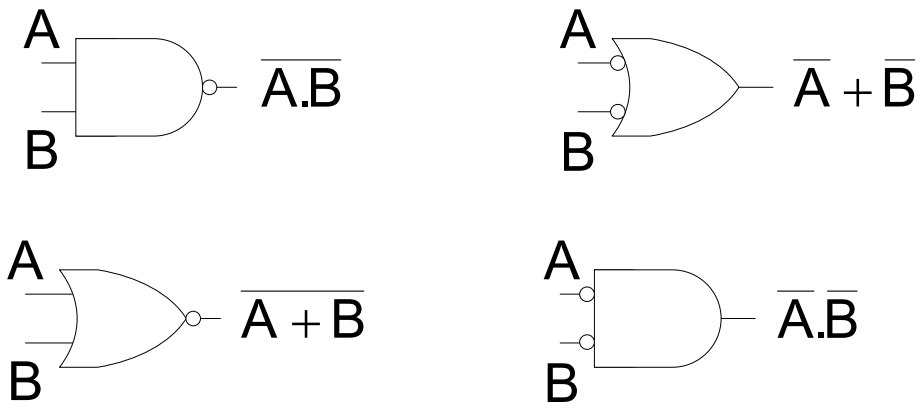


Figure 8.4 Implementation of Demorgan's Theorems

Demorgan's Theorems can be applied to expressions having any number of variables

- $\overline{X.Y.Z} = \overline{X} + \overline{Y} + \overline{Z}$

Demorgan's Theorem can be applied to a combination of other variables

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

Boolean Analysis of Logic Circuits

Boolean algebra provides a concise way to represent the operation of a logic circuit. The complete function of the logic circuit can be determined by evaluating the Boolean expression using different input combinations.

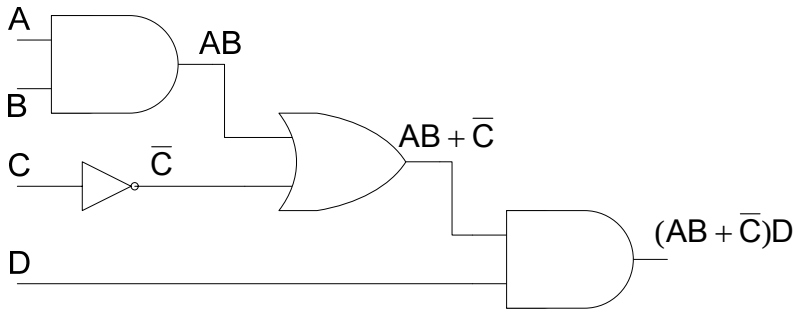


Figure 8.5 Boolean expression representing a Logic Circuit

The expression $(AB + \overline{C})D$ can be derived from the circuit by starting from the left hand, input side of the Logic Circuit. The AND gate provides the output AB. The OR gate adds the product term AB and the complement C to result in $(AB + \overline{C})$ term. The AND gate on the right hand side of the circuit performs a multiplication operation between the term $(AB + \overline{C})$ and the literal D resulting in $(AB + \overline{C})D$.

There are four variables, therefore the function table or truth table for the logic circuit has 16 possible input combinations. The expression can be evaluated by trying out the 16 combinations. Alternately, the input combinations A, B, C and D that set the output of the expression $(AB + \overline{C})D$ to 1 can be easily determined.

From the expression, the output is a 1 if both variable D = 1 and term $(AB + \overline{C})$ =1.

The term $(AB + \overline{C})$ =1 only if AB=1 or C=0.

Thus expression $(AB + \overline{C})D$ =1 if D=1 AND (C=0 OR AB=1)

Inputs				Output
A	**B**	**C**	**D**	**F**
0	0	0	0	0
0	0	**0**	**1**	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	**0**	**1**	1
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**	1
1	1	1	0	0
1	**1**	1	**1**	1

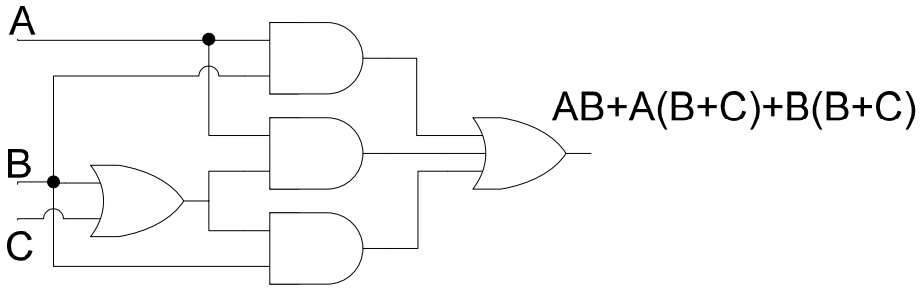
Table 8.1 Function table for expression $(AB + \overline{C})D$

In the function table the input conditions for variables A, B, C and D that satisfy the condition D=1 AND C=0 are 0001, 0101, 1001. The condition D=1 AND AB=1 are satisfied by input combination 1111. The condition D=1 AND (C=0 OR AB=1) is satisfied by the input combination 1101.

Simplification using Boolean Algebra

Many times a Boolean expression has to be simplified using laws, rules and theorems of Boolean Algebra. The simplified expression results in fewer variables and a simpler circuit. Consider the Boolean expression AB + A(B+C) + B(B+C) and the Logic Circuit represented by the expression. Figure 8.6. The simplification of the expression results in an expression B + AC represented by a simpler circuit having fewer gates. Figure 8.7

= AB + A(B+C) + B(B+C)
= AB + AB + AC + BB +BC using Distributive Law
= AB + AC + B + BC BB = B using rule 7
= AB + AC + B (B+BC) = B using rule 10
= B + AC (B+AB) = B using rule 10



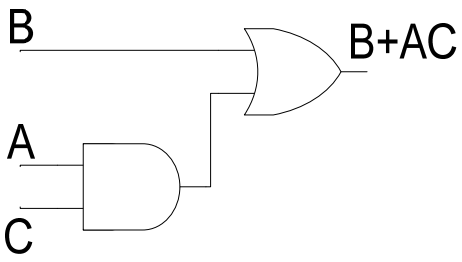


Figure 8.7 Simplified Logic Circuit represented by Boolean expression B+AC

Standard Form of Boolean Expressions

All Boolean expressions can be converted into and represented in one of the two standard forms

- Sum-of-Products form
- Product-of-Sums form

1. Sum of Product form

When two or more product terms are summed by Boolean addition, the result is a Sum-of-Product or SOP expression.

- AB + ABC
- ABC + CDE + $\overline{B}C\overline{D}$
- $\overline{A}B + \overline{A}B\overline{C} + AC$

The Domain of an SOP expression is the set of variables contained in the expression, both complemented and un-complemented. A SOP expression can have a single variable term such as A. A SOP expression can not have a term of more than one variable having an over bar extending over the entire term, such as $\overline{AB} + C$.

2. Product of Sums form

When two or more sum terms are multiplied by Boolean multiplication, the result is a Product-of-Sum or POS expression.

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

The Domain of a POS expression is the set of variables contained in the expression, both complemented and un-complemented. A POS expression can have a single variable term such as A. A POS expression can not have a term of more than one variable having an over bar extending over the entire term such as $(\overline{A + B})(A + \overline{B} + C)$.

Implementation of an SOP and POS expression

A SOP expression can be implemented by an AND-OR combination of gates. The product terms are implemented by an AND gate and the SOP expression is implemented by OR gate connected to the outputs of the AND gates. Figure 8.8

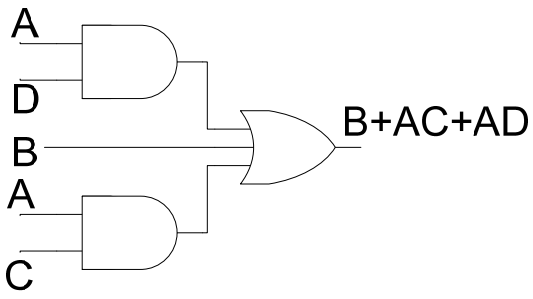


Figure 8.8 SOP Implementation of Boolean expression B+AC+AD

A POS expression can be implemented by an OR-AND combination of gates. The sum terms are implemented by OR gates and the POS expression is implemented by AND gate connected to the outputs of the OR gates.

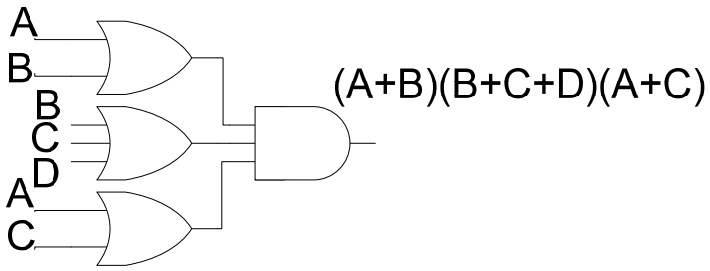


Figure 8.9 POS Implementation of Boolean expression (A+B)(B+C+D)(A+C)

Conversion of a general expression to SOP form

Any logical expression can be converted into SOP form by applying techniques of Boolean Algebra

- $AB + B(CD + EF) = AB + BCD + BEF$
- $(A + B)(B + C + D) = AB + AC + AD + B + BC + BD = AC + AD + B$
- $\overline{\overline{(A + B)} + C} = \overline{\overline{A + B}}\,\overline{C} = (A + B)\overline{C} = A\overline{C} + B\overline{C}$

Lesson No. 09

BOOLEAN ALGEBRA AND LOGIC SIMPLIFICATION

Boolean Analysis of Logic Circuits, evaluating of Boolean expressions, representing the operation of Logic circuits and Boolean expressions in terms of Function tables and representing Boolean expressions in SOP and POS forms are inter-related. Boolean laws, rules and theorems are used to readily change from one form of representation to the other.

Two examples are considered which illustrate the analysis, simplification and representation of Logic Circuits and Boolean expressions. In both the examples, a Boolean expression representing the Logic Circuit is developed, the Boolean expression is evaluated and a function table is implemented that represents the Boolean expression and the function of the Logic Circuit. Each Boolean expression is also simplified into SOP or POS form, the simplified expression is presented in a function table format. The original and the simplified expressions are verified to show identical functions.

Example 1

1. Finding the Boolean Expression

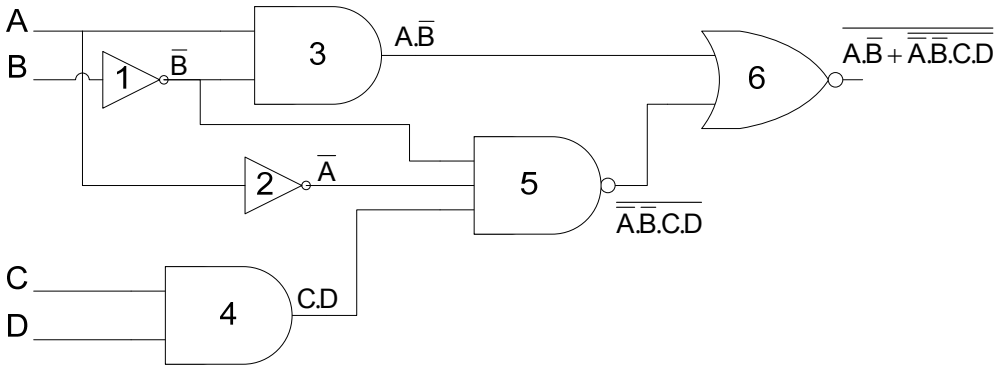


Figure 9.1 Logic Circuit represented by Boolean expression $\overline{A.\overline{B} + \overline{A.\overline{B}.C.D}}$

The circuit can be represented by a Boolean Expression. Starting from the left hand side

- Output of NOT gate 1 is $\overline{B}$
- Output of NOT gate 2 is $\overline{A}$
- Output of two input AND gate 3 is $\overline{A}.B$ (product of $\overline{A}$ and B
- Output of two input AND gate 4 is C.D (product of C and D)
- Output of three input NAND gate 5 is $\overline{\overline{A}.\overline{B}.C.D}$ NOT(product of $\overline{A}$, $\overline{B}$ and CD)
- Output of two input NOR gate 6 is $\overline{A.\overline{B} + \overline{A.\overline{B}.C.D}}$ NOT(sum of $\overline{A}.B$ and $\overline{A.\overline{B}.C.D}$)

2. Evaluating the Expression

Considering that $X = A.\overline{B}$ and $Y = \overline{\overline{A}.\overline{B}.C.D}$. The expression $\overline{A.\overline{B} + \overline{A.\overline{B}.C.D}}$ can be represented by $\overline{X + Y}$.

The output of the logic circuit is 1 when X = 0 and Y = 0

- X=0 NOR Y=0 Output = 1
- X=0 NOR Y=1 Output = 0
- X=1 NOR Y=0 Output = 0
- X=1 NOR Y=1 Output = 0

X= $A.\overline{B}$ = 0 when any literal is zero. That is, A =0 or $\overline{B}$ =0 (B=1)

Y= $\overline{\overline{A}.\overline{B}.C.D}$ = 0 when $\overline{A}.\overline{B}.C.D = 1$

$\overline{A}.\overline{B}.C.D = 1$ when all literals are one. That is $\overline{A}$ =1 (A=0), $\overline{B}$ =1 (B=0), C=1 and D=1

The expression output is 1 for the input conditions
(A=0 OR B=1) AND (A=0 AND B=0 AND C=1 AND D=1)
That is, A=0, B=0, C=1and D=1.

3. Putting the Results in Truth Table Format

Input				Output
A	B	C	D	F
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	0
1	0	0	0	0
1	0	0	1	0
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	0

Table 9.1 Truth Table representing function of Logic Circuit (fig. 9.1)

4. Simplification of Boolean Expression

The $\overline{A.\overline{B} + \overline{\overline{A}.\overline{B}.C.D}}$ expression can be simplified by applying Demorgan's second theorem $\overline{A + B} = \overline{A}.\overline{B}$.

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

Apply Demorgan's first theorem to the first term and Rule 9 to the second term

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

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

Using the Distributive Law

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

Applying Rule 8 to the second term

$= \overline{A}.\overline{B}.C.D$

expression =1 when all literals are one

that is $\overline{A}$ =1 AND $\overline{B}$ =1 AND C=1 AND D=1

or A=0 AND B=0 AND C=1 AND D=1

5. Putting the result in Truth Table format

Input				Output
A	B	C	D	F
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	0
1	0	0	0	0
1	0	0	1	0
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	0

Table 9.2 Truth Table representing function of simplified expression

6. Implementing Logic Circuit from Simplified Boolean expression

$F = \overline{A}.\overline{B}.C.D$

The expression F represents a product term having four literals. Product term is implemented using AND gates. Since, the product has four literals therefore a 4-input AND gate is used. The literals $\overline{A}$ and $\overline{B}$ are implemented using NOT gates.

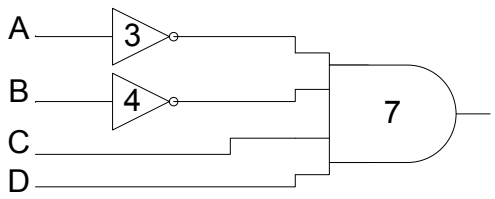


Figure 9.2 Simplified Logic Circuit

Example 2

1. Finding the Boolean Expression

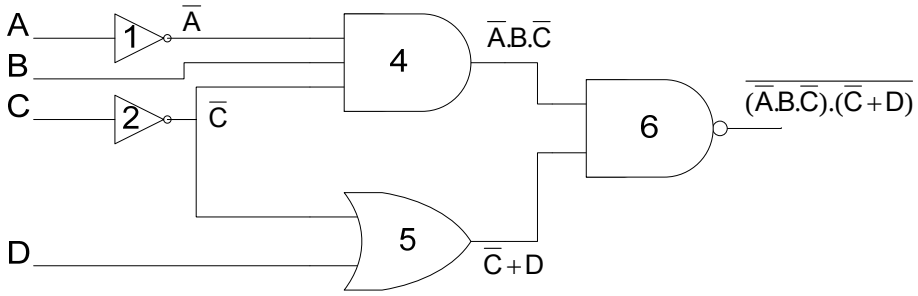


Figure 9.3 Logic Circuit represented by Boolean expression $\overline{(\overline{A}.B.\overline{C}).(\overline{C}+D)}$

The circuit can be represented by a Boolean Expression. Starting from the left hand side

- Output of NOT gate 1 is $\overline{A}$
- Output of NOT gate 2 is $\overline{C}$
- Output of three input AND gate 4 is $\overline{A}.B.\overline{C}$ (product of $\overline{A}$, B and $\overline{C}$)
- Output of two input OR gate 5 is $\overline{C}+D$ (sum of $\overline{C}$ and D)
- Output of NAND gate 6 is $\overline{(\overline{A}.B.\overline{C}).(\overline{C}+D)}$ NOT(product of $\overline{A}.B.\overline{C}$ and $\overline{C}+D$)

2. Evaluating the Expression

Considering that $X=\overline{A}.B.\overline{C}$ and $Y=\overline{C}+D$. The expression $\overline{(\overline{A}.B.\overline{C}).(\overline{C}+D)}$ can be represented by $\overline{X.Y}$.

The output of the logic circuit is 1 when X=0 or Y=0

- X=0 NAND Y=0 Output = 1
- X=0 NAND Y=1 Output = 1
- X=1 NAND Y=0 Output = 1
- X=1 NAND Y=1 Output = 0

X= $\overline{A}.B.\overline{C}$ = 0 when any literal is zero. That is, $\overline{A}$ =0 (A=1) or B =0 or $\overline{C}$ =0 (C=1)

Y= $\overline{C}+D$ = 0 when both literals are zero. That is $\overline{C}$ =0 (C=1) and D=0

The expression output is 1 for the input conditions
(A=1 OR B=0 OR C=1) OR (C=1 AND D=0)

3. Putting the Results in Truth Table Format

Input				Output
A	B	C	D	F
0	**0**	0	0	1
0	**0**	0	1	1
0	**0**	**1 0**		1
0	**0**	**1**	1	1
0	1	0	0	0
0	1	0	1	0
0	1	**1**	**0**	1
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	1

Table 9.2 Truth Table representing function of Logic Circuit (fig. 9.3)

The truth table shows that the variable D has no effect on the output of the circuit. The truth table reduces to a three variable truth table. Table 9.3

4. Simplification of Boolean Expression

The $\overline{(\overline{A}.B.\overline{C}).(\overline{C}+D)}$ expression can be simplified by applying Demorgan's first theorem $\overline{A.B} = \overline{A} + \overline{B}$.

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

Input			Output
A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Table 9.3 Alternate Truth Table representing function of Logic Circuit (fig. 9.3)

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

Applying Rule 9

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

$= A + \overline{B} + C(1 + \overline{D})$

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

expression =1 when any one literal is one

that is A=1 OR $\overline{B}$=1 OR C=1

or A=1 OR B=0 OR C=1

5. Putting the result in Truth Table format

Input			Output
A	B	C	F
0	**0**	0	1
0	**0**	**1**	1
0	1	0	0
0	1	**1**	1
1	**0**	0	1
1	**0 1**		1
1	1	0	1
1	1	**1**	1

Table 9.4 Truth Table representing function of simplified expression

6. Implementing Logic Circuit from Simplified Boolean expression

$F = A + \overline{B} + C$

The expression F represents a sum term having three literals. Sum term is implemented using OR gates. Since, the sum has three literals therefore a 3-input OR gate is used. The literal $\overline{B}$ is implemented using NOT gate.

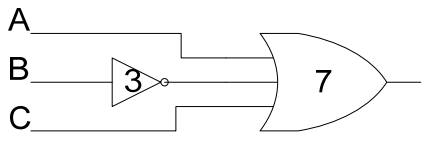


Figure 9.4 Simplified Logic Circuit

Standard SOP form

A standard SOP form has product terms that have all the variables in the domain of the expression. The SOP expression $A\overline{C} + B\overline{C}$ is not a standard SOP as the domain of the expression has variables A, B and C.

A non-standard SOP is converted into a standard SOP by using the rule $A + \overline{A} = 1$

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

Standard POS form

A standard POS form has sum terms that have all the variables in the domain of the expression. The POS expression $(A + \overline{B} + C)(A + B + \overline{D})(A + \overline{B} + \overline{C} + D)$ is not a standard POS as the domain of the expression has variables A, B, C and D.

A non-standard POS is converted into a standard POS by using the rule $A\overline{A} = 0$

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

Converting to Standard SOP and POS forms

There are several reasons for converting SOP and POS forms into standard SOP and POS forms respectively.

Any logic circuit can be implemented by using either the SOP, AND-OR combination of gates or POS, OR-AND combination of gates. It is very simple to convert from standard SOP to standard POS or vice versa. This helps in selecting an implementation that requires the minimum number of gates. Secondly, the simplification of general Boolean expression by applying the laws, rules and theorems does not always result in the simplest form as the ability to apply all the rules depends on ones experience and knowledge of all the rules.

A simpler mapping method uses Karnaugh maps to simplify general expressions. Mapping of all the terms in a SOP form expression and the sum terms in a POS form can be easily done if standard forms of SOP and POS expressions are used. Karnaugh maps will be discussed latter in the chapter.

Lastly, the PLDs are implemented having a general purpose structure based on AND-OR arrays. A function represented by an expression in Standard SOP form can be readily programmed.

Minterms and Maxterms

The Product terms in the Standard SOP form are known as Minterms and the Sum terms in the Standard POS form are known as Maxterms.

A	B	C	Minterms	Maxterms
0	0	0	$\overline{A}.\overline{B}.\overline{C}$	$A + B + C$
0	0	1	$\overline{A}.\overline{B}.C$	$A + B + \overline{C}$
0	1	0		

0	1	1	$\overline{A}.B.C$	$A+\overline{B}+\overline{C}$
1	0	0	$A.\overline{B}.\overline{C}$	$\overline{A}+B+C$
1	0	1	$A.\overline{B}.C$	$\overline{A}+B+\overline{C}$
1	1	0	$A.B.\overline{C}$	$\overline{A}+\overline{B}+C$
1	1	1	$A.B.C$	$\overline{A}+\overline{B}+\overline{C}$

Table 9.5 Table of Minterms and Maxterms

Binary representation of a standard Product term or Minterm

A standard product term is equal to one for only one combination of variable values. For all other variable values the standard product term is equal to zero.

For the expression $AB\overline{C}+A\overline{B}\overline{C}+\overline{A}B\overline{C}$

$AB\overline{C}$ =1 if A=1, B=1 and C=0

$A\overline{B}\overline{C}$ =1 if A=1, B=0 and C=0

$\overline{A}B\overline{C}$ =1 if A=0, B=1 and C=0

An SOP expression is equal to 1 when one or more product terms in the expression are equal to 1.

Binary representation of a standard Sum term or Maxterm

A standard sum term is equal to zero for only one combination of variable values. For all other variable values the standard sum term is equal to one.

For the expression

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

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

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

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

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

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

A POS expression is equal to 0 when one or more product terms in the expression are equal to 0.

Converting Standard SOP into Standard POS

The binary values of the product terms in a given standard SOP expression are not present in the equivalent standard POS expression. Also, the binary values that are not represented in the SOP expression are present in the equivalent POS expression.

$\overline{A}\overline{B}\overline{C}+\overline{A}B\overline{C}+\overline{A}BC+A\overline{B}C+ABC$ has the binary values 000, 010,011,101 and 111

Canonical Sum = $\sum_{A,B,C}(0,2,3,5,7) = \overline{A}\overline{B}\overline{C}+\overline{A}B\overline{C}+\overline{A}BC+A\overline{B}C+ABC$

The missing binary values are 001, 100 and 110.

The POS expression is $(A+B+\overline{C})(\overline{A}+B+C)(\overline{A}+\overline{B}+C)$

Canonical Product = $\prod_{A,B,C}(1,4,6)$

Verifying POS expression is equivalent to SOP expression

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

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

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

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

$=A.\overline{A}.C+A.\overline{B}.C+A.C.C+\overline{A}.B+B.\overline{B}+B.C+\overline{A}.\overline{A}.\overline{C}+\overline{A}.\overline{B}.\overline{C}+\overline{A}.C.C$

$=A.\overline{B}.C+A.C+\overline{A}.B+B.C+\overline{A}.\overline{C}+\overline{A}.\overline{B}.\overline{C}$

Converting into standard SOP

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

$=A.\overline{B}.C+A.B.C+A.\overline{B}.C+\overline{A}.B.C+\overline{A}.B.\overline{C}+A.B.C+\overline{A}.B.C+\overline{A}.B.\overline{C}+\overline{A}.\overline{B}.\overline{C}+\overline{A}.B.\overline{C}$

Simplifies to

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

Therefore $\sum(0,2,3,5,7) = \prod(1,4,6)$

Boolean Expressions and Truth Tables

All standard Boolean expressions can be easily converted into truth table format using binary values for each term in the expression. Standard SOP or POS expressions can also be determined from a truth table.

Converting SOP expression to Truth Table format

A truth table is a list of possible input variable combinations and their corresponding output values. An SOP expression having a domain of 2 variables will have a truth table having 4 combinations of inputs and corresponding output values.

To convert an SOP expression in a Truth table format, a truth table having input combinations proportional to the domain of variables in the SOP expression is written. Next the SOP expression is written in a standard SOP form. In the last step all the sum terms present in the standard SOP expression are marked as 1 in the output.

$A\overline{B}+BC$ has a domain of three variables thus a truth table having 8 input and output combinations is required. The SOP expression is converted into standard SOP expression $A\overline{B}(C+\overline{C})+BC(A+\overline{A}) = A\overline{B}C+A\overline{B}\overline{C}+ABC+\overline{A}BC = \overline{A}BC+A\overline{B}\overline{C}+A\overline{B}C+ABC$

Marking the outputs in the truth table as 1 for sum terms that are present in the standard SOP.

Input			Output
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1

1	1	0	0
1	1	1	1

Table 9.6 Mapping SOP expression to Truth Table

Canonical Sum F= $\sum_{A,B,C}(3,4,5,7) = \overline{A}.B.C + A.\overline{B}.\overline{C} + A.\overline{B}.C + A.B.C$

Converting POS expression to Truth Table format

An POS expression having a domain of 2 variables will have a truth table having 4 combinations of inputs and corresponding output values. To convert a POS expression in a Truth table format, a truth table having input combinations proportional to the domain of variables in the POS expression is written. Next the POS expression is written in a standard POS form. In the last step all the product terms present in the standard POS expression are marked as 0 in the output.

$(A + \overline{B})(B + \overline{C})$ has a domain of three variables thus a truth table having 8 input and output combinations is required. The POS expression is converted into standard POS expression

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

Marking the outputs in the truth table as 0 for product terms that are present in the standard POS

Input			Output
A	B	C	F
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

Table 9.7 Mapping POS expression to Truth Table

Canonical Product F= $\prod_{A,B,C}(1,2,3,5) = (A + B + \overline{C})(A + \overline{B} + C)(A + \overline{B} + \overline{C})(\overline{A} + B + \overline{C})$

Lesson No. 10

KARNAUGH MAP & BOOLEAN EXPRESSION SIMPLIFICATION

Simplifying Boolean Expressions using the laws, rules and theorems do not guarantee the simplest form of expression as sometimes simplification of certain terms is not so obvious or the person doesn't have the necessary experience in applying the laws and rules. The Karnaugh Map provides a systematic method for simplifying Boolean expressions.

A Karnaugh Map is organized in the form of an array. Adjacent cells of the array can be grouped together to result in simplification of a given expression. Karnaugh Maps can be used to simplify expressions of 2, 3, 4 and 5 variables.

The 3-variable Karnaugh Map

AB\C	0	1
00	0	1
01	2	3
11	6	7
10	4	5

A\BC	00	01	11	10
0	0	1	3	2
1	4	5	7	6

Figure 10.1 Column and Row based 3-variable Karnaugh Maps

- A 3-variable K-Map has an array of 8 cells. The 8 cells can be arranged in 2 columns and 4 rows representing the column form of the Karnaugh Map.
- Alternately, the 8 cells can be organized in 2 rows and 4 columns representing the row form of the Karnaugh map.
- Any of the two forms of the Karnaugh Map can be used to simplify Boolean expressions. The simplified expressions using either of the two K-maps are identical.
- Considering first the column based 3-variable Karnuagh map. The binary values 00, 01, 11 and 10 in the left most column of the K-map represent the binary values of variables A and B. The binary values 0 and 1 in the top row of the K-map represent the binary values of variable C.
- The 3-variable K-Map based on the row representation is considered next. The binary values 0 and 1 in the left most column of the K-map represent the binary values of variable A. The binary values 00, 01, 11 and 10 in the top row of the K-map represent the binary values of variables B and C
- The numbers in the cells represent the Minterms or Maxterms of an expression that is to be represented using the K-map. The cell marked 0 for example, represents the minterm 0 or the maxterm 0 having binary value of variables A, B and C equal to 000. Similarly cell marked 5 represents the minterm 5 or the maxterm 5 having binary values of variables A, B and C equal to 101.

The 4-variable Karnaugh Map

AB\CD	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

- A 4-variable K-Map has an array of 16 cells
- The numbers in the cells represent the Minterms and Maxterms of an expression that is to be represented using the K-map.
- The 4-variable K-Map has a square format with four rows and four columns of cells.
- The binary values 00, 01, 11 and 10 in the left most column of the K-map represent the binary values of variables A and B. The binary values 00, 01, 11 and 10 in the top row of the K-map represents the binary values of variables C and D
- The 16 cells marked with numbers 0 to 15 represent the cells 0 to 15 corresponding to the minterms 0 to 15 or the maxterms 0 to 15 in a 4 variable Boolean expression.
- The cell marked 6 for example, represents the minterm 6 or the maxterm 6 having binary value of variables A, B, C and D equal to 0110. Similarly cell marked 13 represents the minterm 13 or the maxterm 13 having binary values of variables A, B, C and D equal to 1101.

Grouping and Adjacent Cells

Karnaugh Map Array is considered to be wrapped around were all sides are adjacent to each other. Groups of 2, 4, 8, 16, 32 etc. adjacent cells are formed. Adjacent cells can be

- row wise
- column wise
- four corner cells
- row-column groups of 4, 8, 16, 32 etc

Groups are formed on the basis of 1s (Minterms) or 0s (maxterms). A group is selected to have maximum number of cells of Minterms or Maxterms, keeping in view that the size of the group should be a power of 2. The idea is to form minimal number of largest groups that uniquely cover all the cells, thereby ensuring that all minterms or maxterms are included.

Mapping a standard SOP Expression

The first step in simplification of Boolean expressions is to map the expressions to the Karnaugh maps. For a Standard SOP expression, a 1 is placed in the cell corresponding to the product term (Minterm) present in the expression. The cells that are not filled with 1s have 0s.

The Standard SOP expression having a Domain of three variables $AB\overline{C} + A\overline{B}\overline{C} + \overline{A}B\overline{C}$ is mapped to a 3-Variable Karnaugh Map. The product terms or the Minterms are 2, 4 and 6. The expression is mapped on a K-Map by placing a 1 at Minterm cells 2, 4 and 6 and placing 0 at remaining cells.

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

A\BC	00	01	11	10
0	0	0	0	1
1	1	0	0	1

Figure 10.3 Mapping the expression $AB\overline{C} + A\overline{B}\overline{C} + \overline{A}B\overline{C}$ to a 3-variable K-Map

The Standard SOP expression having a domain of four variables $\overline{A}.\overline{B}.\overline{C}.D + \overline{A}.B.\overline{C}.\overline{D} + \overline{A}.B.\overline{C}.D + \overline{A}.B.C.\overline{D} + A.\overline{B}.\overline{C}.\overline{D} + A.B.\overline{C}.D + A.B.C.\overline{D}$ is mapped to a 4-

expression is mapped on a K-Map by placing a 1 at Minterm cells 1, 4, 5, 6, 8, 13 and 14 and placing 0 at remaining cells.

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

Figure 10.4 Mapping the 7 term SOP expression to a 4-variable K-Map

Mapping a non-standard SOP Expression

In many practical cases, SOP expressions are not in a standard format. To map them to K-maps they have to be either converted into Standard SOP expressions or they can be directly mapped.

Example 1

The expression $A + B\overline{C}$ is a non-standard SOP expression having a domain of 3 variables. If the expression is converted into a standard SOP expression then there will be four product terms having the variable A. Similarly, there would be two product terms having the variable combination $B\overline{C}$. Two of the product terms $AB\overline{C}$ are identical. The expression $A + B\overline{C}$ can be directly mapped to a K-map without first converting the expression to the standard form.

The term A is mapped first. A '1' is marked in cells where the variable A is present.

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

A\BC	00	01	11	10
0				
1	1	1	1	1

Figure 10.5 Mapping the expression A to a 3-variable K-Map

Consider the mapping of the term $B\overline{C}$. A '1' is marked in cells where the variable $B\overline{C}$ is present. The cells are marked with **1**. One of the cells $AB\overline{C}$ has already been marked when mapping the terms containing variable A.

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

A\BC	00	01	11	10
0	0	0	0	**1**
1	1	1	1	**1**

The K-map shows that if the non-standard SOP expression $A + B\overline{C}$ is converted into a standard SOP expression it would have five product terms as represented by the K-map cells.

Example 2

The expression $\overline{A} + \overline{C}$ is a non-Standard SOP expression having a domain of 3 variables. It is mapped directly to a 3-variable K-map. The term $\overline{A}$ is mapped first by marking cells having $\overline{A}$ with '1'.

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

A\BC	00	01	11	10
0	1	1	1	1
1	0	0	0	0

Figure 10.7 Mapping the expression $\overline{A}$ to a 3-variable K-Map

The term $\overline{C}$ is mapped next. A '**1**' is marked in cells where the term $\overline{C}$ is present.

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

A\BC	00	01	11	10
0	**1**	1	1	**1**
1	**1**	0	0	**1**

Figure 10.8 Mapping the expression $\overline{C}$ to a 3-variable K-Map

Mapping of non-standard SOP expressions having a domain of 4 variables is similar. Consider the expression $D + A\overline{C} + BC$. The terms D, $A\overline{C}$ and BC are mapped one after the other by marking cells with '1's where these terms are present.

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

Figure 10.9a Mapping the expression D to a 4-variable K-Map

AB\CD	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	**1**	**1**	1	0
10	**1**	**1**	1	0

AB\CD	00	01	11	10
00	0	1	1	0
01	0	1	**1**	**1**
11	1	1	**1**	**1**
10	1	1	1	0

Figure 10.9c Mapping the expression BC to a 4-variable K-Map

Simplification of SOP expressions using the Karnaugh Map

SOP expressions can be very easily simplified using the K-Map method. In the first step of the simplification process, the SOP expression is mapped on the K-map. In the next step, groups of 1s are formed starting with the largest group of 1s. The group should be of size 2, 4, 8, 16 etc. having adjacent 1s. Multiple (unique) groups of 1s are formed. All the groups formed can either be separate groups or they could share common 1s each having at least a single 1 that is not common to any other group. A single 1 that is not adjacent to any other 1 is considered as a group having only a single cell.

In the next step minimal product terms are determined. Each group, including a group having a single cell, represents a product term having variables that occur in only one form either complemented or un-complemented.

A 3-variable K-map yields

- A product term of three variables for a group of 1 cell
- A product term of two variables for a group of 2 cell
- A product term of one variable for a group of 4 cell
- A group of 8 cells yields a value of 1 for the expression.

A 4-variable K-map yields

- A product term of four variables for a group of 1 cell
- A product term of three variables for a group of 2 cell
- A product term of two variables for a group of 4 cell
- A product term of one variable for a group of 8 cell
- A group of 16 cells yields a value of 1 for the expression.

Example 1 & 2

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

A\BC	00	01	11	10
0	0	1	1	1
1	1	0	0	0

Figure 10.10 Simplification of SOP expression using a 3-variable K-Map

An SOP expression having 5 minterms is mapped to a 3-variable column based K-map. Three groups of two cells each are formed.

- The first group of 1s comprising of cells 2 and 6 forms the product term $B\overline{C}$
- The second group of 1s comprising of cells 5 and 7 forms the product term AC
- The third group of 1s comprising of cells 1 and 5 forms the product term $\overline{B}C$

The five term SOP expression simplifies to a 3 term SOP expression $B.\overline{C} + A.C + \overline{B}.C$

An SOP expression having 4 minterms is mapped to a 3-variable row based K-map. Two groups of 2 cells each and a third group of single cell are formed.

- The single cell group comprising of cell 4 forms the product term $A\overline{B}\overline{C}$
- The second group of 1s comprising of cells 1 and 3 forms the product term $\overline{A}C$
- The third group of 1s comprising of cells 2 and 3 forms the product term $\overline{A}B$

The four term SOP simplifies to a 3 term SOP expression $A.\overline{B}.\overline{C} + \overline{A}.C + \overline{A}.B$

Example 3 & 4

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

A\BC	00	01	11	10
0	0	0	1	1
1	1	1	1	0

Figure 10.11 Simplification of SOP expression using a 3-variable K-Map

An SOP expression having 5 minterms is mapped to a 3-variable column based K-map. One group of 4 cells and another group of 2 cell are formed.

- The first group of 1s comprising of cells 2, 3, 6 and 7 forms the product term B
- The second group of 1s comprising of cells 5 and 7 forms the product term AC

The five term SOP simplifies to a 2 term SOP expression $B + AC$

An SOP expression having 5 minterms is mapped to a 3-variable row based K-map. Three groups of 2 cells each are formed.

- The first group of 1s comprising of cell 4 and 5 forms the product term $A.\overline{B}$
- The second group of 1s comprising of cells 3 and 7 forms the product term $B.C$
- The third group of 1s comprising of cells 2 and 3 forms the product term $\overline{A}.B$

The five term SOP simplifies to a 3 term SOP expression $A.\overline{B} + B.C + \overline{A}.B$

Example 5

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

An SOP expression having 11 minterms is mapped to a 4-variable based K-map. Three groups of 4 cells each are formed.

- The first group of 1s comprising of cells 8, 9, 12 and 13 forms the product term $A.\overline{C}$
- The second group of 1s comprising of cells 1, 3, 9 and 11 forms the product term $\overline{B}.D$
- The third group of 1s comprising of cells 6, 7, 14 and 15 forms the product term $B.C$

The eleven term SOP expression has simplified to a 3 term expression $A.\overline{C} + \overline{B}.D + B.C$

Example 6

An SOP expression having 8 minterms is mapped to a 4-variable based K-map. One group of two cells and two groups of four cells are formed.

- The first group of 1s comprising of cells 8 and 12 forms the product term $A.\overline{C}.\overline{D}$
- The second group of 1s comprising of cells 3, 7, 11 and 15 forms the product term $C.D$
- The third group of 1s comprising of cells 6, 7, 14 and 15 forms the product term $B.C$

The eight term SOP expression has simplified to a 3 term expression $A.\overline{C}.\overline{D} + C.D + B.C$

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

Figure 10.13 Simplification of SOP expression using a 4-variable K-Map

Example 7

An SOP expression having 9 minterms is mapped to a 4-variable based K-map. Two group of two cells and two groups of four cells are formed.

- The first group of 1s comprising of corner cells 0, 2, 8 and 10 forms the product term $\overline{B}.\overline{D}$
- The second group of 1s comprising of cells 2, 3, 10 and 11 forms the product term $\overline{B}.C$
- The third group of 1s comprising of cells 13 and 15 forms the product term $A.B.D$
- The fourth group of 1s comprising of cells 2 and 6 forms the product term $\overline{A}.C.\overline{D}$

The nine term SOP expression has simplified to a 4 term SOP expression $\overline{B}.\overline{D} + \overline{B}.C + A.B.D + \overline{A}.C.\overline{D}$

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

Figure 10.14 Simplification of SOP expression using a 4-variable K-Map

Mapping Directly from Function Table

Practically, when a digital circuit is to be implemented to perform some operation, its

mapped to a K-map of appropriate variables which is then simplified. The simplified expression obtained from the K-map is directly implemented using logic Gates.

Consider a logical circuit that accepts 4-bit binary numbers representing decimal numbers 0 to 15. The circuit checks the four bit binary equivalent of the decimal number. If the number is odd and it is a prime number the function outputs a one. Before designing the logic circuit a function table is implemented with all the input output combinations. The function table for the odd prime number checker is shown. Table 10.1 The output is a 1 for inputs 1, 3, 5, 7, 11 and 13.

Input				Output	Input				Output
A	B	C	D	F	A	B	C	D	F
0	0	0	0	0	1	0	0	0	0
0	0	0	1	1	1	0	0	1	0
0	0	1	0	0	1	0	1	0	0
0	0	1	1	1	1	0	1	1	1
0	1	0	0	0	1	1	0	0	0
0	1	0	1	1	1	1	0	1	1
0	1	1	0	0	1	1	1	0	0
0	1	1	1	1	1	1	1	1	0

Table 10.1 Function Table for Odd-Prime Checker Circuit

The 4 variable Function Table, Table 10.1 can be directly mapped to a 4 variable K-map by marking the K-map cells with 1s corresponding to the minterms marked as 1s in the function table. Figure 10.14. Simplifying the expression using the K-map results in $\overline{A}.D + B.\overline{C}.D + \overline{B}.C.D$. The expression can be directly implemented using logic gates.

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

Figure 10.14 Simplification of expression using a 4-variable K-Map

Don’t care Conditions

Function Tables represent the function by listing all the possible inputs and marking the corresponding outputs with 1s and 0s. Thus a circuit having four inputs can be described by a 4-variable function table having 16 possible input combinations. For each of the 16 possible input conditions the corresponding output bits are marked as 1s and 0s depending upon the minterms or maxterms. It is however, possible that out of the 16 possible input combinations, three input combinations never occur. Since these three input combinations never occur so should their corresponding outputs be marked as 0s or 1s? Since these inputs never care therefore we don’t need to worry about the output of these input states. They are considered to be don’t care conditions.

Don’t care conditions are marked as x in the output column of the function table

corresponding K-map, the don't care conditions are marked as x. However during the grouping process for simplification of the SOP expression the x outputs can be considered as 0 or 1. By assigning a 0 or 1 to the cells marked with x, the final expression can be significantly simplified.

Reconsider the last example of the Odd-Prime Number checker circuit. Assuming that only the first ten input (0 to 9) states can occur and the last 6 inputs never occur. The function table for the conditions that never occur is shown. Table 10.2

Input				Output	Input				Output
A	B	C	D	F	A	B	C	D	F
0	0	0	0	0	1	0	0	0	0
0	0	0	1	1	1	0	0	1	0
0	0	1	0	0	1	0	1	0	x
0	0	1	1	1	1	0	1	1	x
0	1	0	0	0	1	1	0	0	x
0	1	0	1	1	1	1	0	1	x
0	1	1	0	0	1	1	1	0	x
0	1	1	1	1	1	1	1	1	x

The function table can be directly mapped to a 4 variable K-map. Figure 10.15. The cells marked with x are considered to be 0s. Thus the function expression is simplified to $\overline{A}.D$

AB\CD	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	x	x	x	x
10	0	0	x	x

Figure 10.15 Simplification of expression with Don't care states

If the Odd-Parity Checker Circuit checks for numbers between 0 and 8, and states 9 to 15 never occur then the Boolean expression representing the function reduces to a single literal D Figure 10.16. The cells 9, 10, 11, 13, 14 and 15 marked as 'x' as they represent the don't care states are considered as 1's to form a group of 8 cells. Remaining cells marked with 'x' are considered as 0's and are not involved in grouping.

AB\CD	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	x	x	x	x
10	0	x	x	x

Figure 10.16 Simplification of expression with Don't care states

Consider the K-map considered earlier in Example 6. Figure 10.13. Assume that the input conditions $\overline{A}.\overline{B}.\overline{C}.D$, $\overline{A}.\overline{B}.C.\overline{D}$ and $A.\overline{B}.C.\overline{D}$ never occur so they are marked as x in the K-

map cells corresponding to the minterms that never occur. Redefining the groups using x as 0 or 1 results in a simpler expression $C + A.\overline{D}$ instead of $A.\overline{B}.\overline{C} + C.D + B.C$. Figure 10.17.

AB\CD	00	01	11	10
00	0	x	1	x
01	0	0	1	1
11	1	0	1	1
10	1	0	1	x

AB\CD	00	01	11	10
00	0	x	1	x
01	0	0	1	1
11	1	0	1	1
10	1	0	1	x

Figure 10.17 Simplified expression by incorporating the don't care states

Lesson No. 11

KARNAUGH MAP & BOOLEAN EXPRESSION SIMPLIFICATION

Mapping a Standard POS Expression

For a Standard POS expression, a 0 is placed in the cell corresponding to the product term (maxterm) present in the expression. The cells are not filled with 0s have 1s. The Standard POS expression having a Domain of three variables $(A+B+\overline{C}).(A+\overline{B}+C).(\overline{A}+B+\overline{C}).(\overline{A}+\overline{B}+\overline{C})$ uses a 3-Variable Karnaugh Map. The sum terms or the Maxterms are 1, 2, 5 and 7. The expression can be represented by a K-Map by placing a 0 at Maxterm locations 1, 2, 5 and 7 and placing 1 at remaining places. Any of the two K-maps can be used. Figure 11.1.

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

A\BC	00	01	11	10
0	1	0	1	0
1	1	0	0	1

Figure 11.1 Mapping a Standard POS expression

Karnaugh Map simplification of POS expressions

POS expressions can be easily simplified by use of the K-Map in a manner similar to the method adopted for simplifying SOP expressions. After the POS expression is mapped on the K-map, groups of 0s are marked instead of 1s based on the rules for forming groups used for simplifying SOP.

In the next step minimal sum terms are determined. Each group, including a group having a single cell, represents a sum term having variables that occur in only one form either complemented or un-complemented.

A 3-variable K-map yields

- A sum term of three variables for a group of 1 cell
- A sum term of two variables for a group of 2 cell
- A sum term of one variable for a group of 4 cell
- A group of 8 cells yields a value of 0 for the expression.

A 4-variable K-map yields

- A sum term of four variables for a group of 1 cell
- A sum term of three variables for a group of 2 cell
- A sum term of two variables for a group of 4 cell
- A sum term of one variable for a group of 8 cell
- A group of 16 cells yields a value of 0 for the expression.

Example 1 & 2

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

A\BC	00	01	11	10
0	0	1	1	1
1	1	0	0	0

A POS expression having 3 Maxterms is mapped to a 3-variable column based K-map. A single group of two cells and a group of one cell are formed.

- The first group of 0s comprising of cells 0 and 4 forms the sum term $(B + C)$
- The second group comprising of cell 3 forms the sum term $(A + \overline{B} + \overline{C})$

The three term POS expression simplifies to a 2 term POS expression $(B + C).(A + \overline{B} + \overline{C})$.

A POS expression having 4 Maxterms is mapped to a 3-variable column based K-map. Two groups of 2 cells each and a third group of single cell are formed.

- The single cell group comprising of cell 0 forms the sum term $(A + B + C)$
- The second group of 0s comprising of cells 5 and 7 forms the sum term $(\overline{A} + \overline{C})$
- The third group of 0s comprising of cells 6 and 7 forms the sum term $(\overline{A} + \overline{B})$

The four term POS expression simplifies to a 3 term POS expression $(A + B + C).(\overline{A} + \overline{C}).(\overline{A} + \overline{B})$.

Example 3 & 4

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

A\BC	00	01	11	10
0	0	0	1	1
1	1	1	1	0

Figure 11.3 Simplification of POS expression using a 3-variable K-Map

A POS expression having 3 Maxterms is mapped to a 3-variable column based K-map. Two groups of two cells are formed.

- The first group of 0s comprising of cells 0 and 1 forms the sum term $(A + B)$
- The second group of 0s comprising of cells 0 and 4 forms the sum term $(B + C)$

The three term POS expression simplifies to a 2 terms POS expression $(A + B).(B + C)$

A POS expression having 3 Maxterms is mapped to a 3-variable column based K-map. One group of 2 cells and another group of single cell are formed.

- The first group of 0s comprising of cell 0 and 1 forms the sum term $(A + B)$
- The second group comprising of cell 6 forms the sum term $(\overline{A} + \overline{B} + C)$

The three term POS expression simplifies to a 2 term POS expression $(A + B).(\overline{A} + \overline{B} + C)$

Example 5

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

Figure 11.4 Simplification of POS expression using a 4-variable K-Map

A POS expression having 5 Maxterms is mapped to a 4-variable column based K-map. Three groups of two cells are formed.

- The first group of 0s comprising of cells 4 and 5 forms the sum term $(A+\overline{B}+C)$
- The second group of 0s comprising of cells 0 and 4 forms the sum term $(A+C+D)$
- The third group of 0s comprising of cells 2 and 10 forms the sum term $(B+\overline{C}+D)$

The five term POS expression has reduced to a 3 term POS expression $(A+\overline{B}+C).(A+C+D).(B+\overline{C}+D)$

Example 6

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

Figure 11.5 Simplification of POS expression using a 4-variable K-Map

A POS expression having 8 Maxterms is mapped to a 4-variable column based K-map. Two groups of 4 cells and one group of two cells are formed.

- The first group of 0s comprising of cells 0, 1, 4 and 5 forms the sum term $(A+C)$
- The second group of 0s comprising of cells 1, 5, 9 and 13 forms the sum term $(C+\overline{D})$
- The third group of 0s comprising of cells 2 and 10 forms the sum term $(B+\overline{C}+D)$

The eight term POS expression has reduced to a 3 term POS expression $(A+C).(C+\overline{D}).(B+\overline{C}+D)$.

Example 7

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

Figure 11.6 Simplification of POS expression using a 4-variable K-Map

A POS expression having 6 Maxterms is mapped to a 4-variable column based K-map. Three groups of 2 cells and one group of a single cell are formed.

- The first group of 0s comprising of cells 4 and 5 forms the sum term $(A+\overline{B}+C)$
- The second group of 0s comprising of cells 5 and 7 forms the sum term $(A+\overline{B}+\overline{D})$
- The third group of 0s comprising of cells 1 and 9 forms the sum term $(B+C+\overline{D})$

- The fourth group comprising of cell 14 forms the sum term $(\overline{A}+\overline{B}+\overline{C}+D)$

The six term POS expression has reduced to a 4 term POS expression $(A+\overline{B}+C).(A+\overline{B}+\overline{D}).(B+C+\overline{D}).(\overline{A}+\overline{B}+\overline{C}+D)$

Converting between POS and SOP using the K-map

Converting between the two forms of standard expressions is very simple. If the 1s mapped on the K-map are grouped together they form the product terms of the SOP expression. Similarly, if the 0s mapped on the K-map are grouped together they form the sum terms of the POS expression

Consider the POS expression $(A+\overline{B}+C).(A+\overline{B}+\overline{D}).(B+C+\overline{D}).(\overline{A}+\overline{B}+\overline{C}+D)$

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

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

Figure 11.7 Converting between SOP and POS using K-map

An equivalent SOP expression can be obtained by grouping the 1s together.

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

Five-Variable Karnaugh Map

A K-map for 5 variables can be constructed by using two 4-variable K-maps. Figure 11.8. The cells 0 to 15 lie in the 4-variable map A=0 and cells 16 to 31 lie in the 4-variable map A=1.

The two, 4-variable maps are considered to be lying on top of each other. Thus a two dimensional map is formed. Rules for grouping of 0s and 1s remain unchanged. In a 2-dimensional map, the groups of adjacent 0s or 1s can also span both the maps. In a 5-variable Karnaugh map groups of 2, 4, 8, 16 and 32 can be formed.

BC\DE	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

BC\DE	00	01	11	10
00	16	17	19	18
01	20	21	23	22
11	28	29	31	30
10	24	25	27	26

Figure 11.8 5-variable Karnaugh Map using A=0 and A=1 maps

Mapping, Grouping and Simplification using 5-variable Karnaugh maps is identical to those of 3 and 4 variable Karnaugh maps.

Simplification of 5-Variable Karnaugh Map

BC\DE	00	01	11	10
00	0	1	0	1
01	0	1	0	0
11	0	0	0	1
10	0	0	1	1

BC\DE	00	01	11	10
00	1	1	0	0
01	1	1	0	0
11	0	0	0	1
10	0	1	1	1

Figure 11.9 5-variable Karnaugh Map Simplification

The 5-variable Karnaugh map is mapped with Minterms in plane A=0 and A=1 respectively. Consider the groups that are formed.

- Starting with A=0 map. The cells 1 and 5 form a group of two cells. These two cells along with cells 17 and 21 in map A=1 from a group of 4 cells. This group of 4 cells represents the term $\bar{B}\bar{D}E$
- The cell 2 in map A=0. Cell 2 does not form a group with any adjacent cells. Therefore it is a group of single cell having the product term $A\bar{B}\bar{C}D\bar{E}$
- The cells 10 and 11 in map A=0. These two cells form a group of four with adjacent cells 26 and 27 in map A=1. Therefore the group of 4 cells represents the product term $B\bar{C}D$
- Tthe cells 11 and 14 in map A=0 and cells 26 and 30 in map A=1represent a group of 4 cells representing the product term $BD\bar{E}$

Now considering the map A=1.

- The 4 cells 16, 17, 20 and 21 represent the product term $A\bar{B}\bar{D}$
- The cell 25 along with cell 27 in map A=1 represent the product term $AB\bar{C}E$

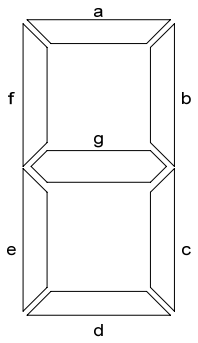
Functions having multiple outputs

In the discussions on Boolean expressions and Function Tables that represent Boolean functions it has been assumed that Logic Circuits have multiple inputs and single output. Practical Logic circuits however, have multiple inputs and multiple outputs. Circuits having a single output or multiple outputs are treated in the same manner.

Circuits having multiple outputs are represented by multiple function tables one for each output or a single function table having multiple output columns. The example of a BCD to 7-Segment Decoder circuit which has 4 inputs and 7 outputs is considered to explain functions having multiple outputs.

7-Segment Display

The 7-segment display digit is shown. Figure 11.10. 7-Segment Display is used to display the decimal numbers 0 to 9. A 7-segment display digit has 7 segments a, b, c, d, e, f and g that are turned on/off by a digital circuit depending upon the number that is to be displayed.



Digit	Segments
0	a, b, c, d, e, f
1	b, c
2	a, b, d, e, g
3	a, b, c, d, g
4	b, c, f, g
5	a, c , d, f, g
6	a, c, d, e, f, g
7	a, b, c
8	a, b, c, d, e, f, g
9	a, b, c, d, f, g

Figure 11.10 7-Segment Display

Different set of segments have to be turned on to display different digits. For example, to display the digit 3, segments a, b, c, d and g have to be turned on. To display the digit 7, segments a, b and c have to be turned on. The table indicates the segments that are turned on for each digit.

The circuit that turns on the appropriate segments to display a digit is known as a BCD to 7-Sement Decoder. The input to the BCD to 7-Segment decoder circuit is a 4-bit BCD number between 0 and 9. The seven output lines of the decoder connect to the 7 segments. Figure 11.11.

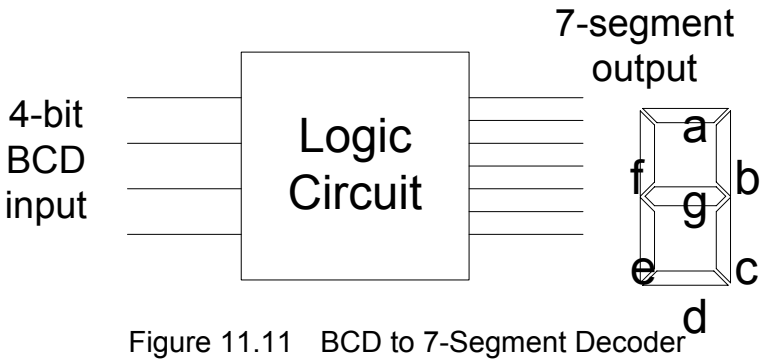


Figure 11.11 BCD to 7-Segment Decoder

To implement the decoder circuit having 4 inputs and 7 outputs, function tables have to be drawn which represent the output status of each output line for all combinations of inputs. For example, the segment a is turned on when the 4-bit input is 0, 2, 3, 5, 6, 7, 8 and 9. Similarly, the segment b is turned on for 0, 2, 3, 4, 7, 8 and 9 combinations of inputs. Thus seven expressions, one for each segment has to be be determined before the decoder circuit can be implemented.

Seven function tables are required to represent the input/output combinations for each segment. The seven function tables for segments a, b, c, d, e, f and g are shown. Figure 11.12a-g. To determine the seven expressions for each of the seven outputs, seven 4-variable Karnaugh maps are used. The Karnaugh maps and the simplified expressions are shown. Figure 11.13a-g. An alternate way of representing the seven Function tables is to have a single function table with the four columns representing the 4-bit input BCD number and seven

output columns each representing one of the seven segments a, b, c, d, e, f and g respectively.

Since the 4-bit input to the decoder circuit can have 16 possible input combinations, therefore each of the seven Function tables have sixteen input combinations. However, the last 6 input combinations are don't care as these combinations never occur because the input to the circuit is a 4-bit BCD number. The don't care states help in simplifying the Boolean expressions for the seven segments.

Input				Output	Input				Output
A	B	C	D	Seg. a	A	B	C	D	Seg. a
0	0	0	0	1	1	0	0	0	1
0	0	0	1	0	1	0	0	1	1
0	0	1	0	1	1	0	1	0	x
0	0	1	1	1	1	0	1	1	x
0	1	0	0	0	1	1	0	0	x
0	1	0	1	1	1	1	0	1	x
0	1	1	0	1	1	1	1	0	x
0	1	1	1	1	1	1	1	1	x

Figure 11.12a Function Table for Segment a

Input				Output	Input				Output
A	B	C	D	Seg. b	A	B	C	D	Seg. b
0	0	0	0	1	1	0	0	0	1
0	0	0	1	1	1	0	0	1	1
0	0	1	0	1	1	0	1	0	x
0	0	1	1	1	1	0	1	1	x
0	1	0	0	1	1	1	0	0	x
0	1	0	1	0	1	1	0	1	x
0	1	1	0	0	1	1	1	0	x
0	1	1	1	1	1	1	1	1	x

Figure 11.12b Function Table for Segment b

Input				Output	Input				Output
A	B	C	D	Seg. c	A	B	C	D	Seg. c
0	0	0	0	1	1	0	0	0	1
0	0	0	1	1	1	0	0	1	1
0	0	1	0	0	1	0	1	0	X
0	0	1	1	1	1	0	1	1	X
0	1	0	0	1	1	1	0	0	X
0	1	0	1	1	1	1	0	1	x
0	1	1	0	1	1	1	1	0	x

Figure 11.12c Function Table for Segment c

Input				Output	Input				Output
A	B	C	D	d	A	B	C	D	d
0	0	0	0	1	1	0	0	0	1
0	0	0	1	0	1	0	0	1	1
0	0	1	0	1	1	0	1	0	x
0	0	1	1	1	1	0	1	1	x
0	1	0	0	0	1	1	0	0	x
0	1	0	1	1	1	1	0	1	x
0	1	1	0	1	1	1	1	0	x
0	1	1	1	0	1	1	1	1	x

Figure 11.12d Function Table for Segment d

Input				Output	Input				Output
A	B	C	D	Seg. e	A	B	C	D	Seg. e
0	0	0	0	1	1	0	0	0	1
0	0	0	1	0	1	0	0	1	0
0	0	1	0	1	1	0	1	0	x
0	0	1	1	0	1	0	1	1	x
0	1	0	0	0	1	1	0	0	x
0	1	0	1	0	1	1	0	1	x
0	1	1	0	1	1	1	1	0	x
0	1	1	1	0	1	1	1	1	x

Figure 11.12e Function Table for Segment e

Input				Output	Input				Output
A	B	C	D	Seg. f	A	B	C	D	Seg. f
0	0	0	0	1	1	0	0	0	1
0	0	0	1	0	1	0	0	1	1
0	0	1	0	0	1	0	1	0	x
0	0	1	1	0	1	0	1	1	x
0	1	0	0	1	1	1	0	0	x
0	1	0	1	1	1	1	0	1	x
0	1	1	0	1	1	1	1	0	x
0	1	1	1	0	1	1	1	1	x

Figure 11.12f Function Table for Segment f

Input				Output	Input				Output
A	B	C	D	Seg. g	A	B	C	D	Seg. g
0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	1	0	0	1	1
0	0	1	0	1	1	0	1	0	x
0	0	1	1	1	1	0	1	1	x
0	1	0	0	1	1	1	0	0	x
0	1	0	1	1	1	1	0	1	x
0	1	1	0	1	1	1	1	0	x
0	1	1	1	0	1	1	1	1	x

Figure 11.12g Function Table for Segment g

AB\CD	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	x	x	x	x
10	1	1	x	x

$a = A + C + BD + \overline{B}\overline{D}$

AB\CD	00	01	11	10
00	1	1	1	1
01	1	0	1	0
11	x	x	x	x
10	1	1	x	x

$b = \overline{B} + \overline{C}\overline{D} + CD$

AB\CD	00	01	11	10
00	1	1	1	0
01	1	1	1	1
11	x	x	x	x
10	1	1	x	x

$c = \overline{C} + D + B$

AB\CD	00	01	11	10
00	1	0	1	1
01	0	1	0	1
11	x	x	x	x
10	1	1	x	x

$d = A + \overline{B}\overline{D} + \overline{B}C + C\overline{D} + B\overline{C}D$

AB\CD	00	01	11	10
00	1	0	0	1
01	0	0	0	1
11	x	x	x	x
10	1	0	x	x

$e = \overline{B}\overline{D} + C\overline{D}$

AB\CD	00	01	11	10
00	1	0	0	0
01	1	1	0	1
11	x	x	x	x
10	1	1	x	x

$f = B + \overline{C}\overline{D} + B\overline{C} + B\overline{D}$

AB\CD	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	x	x	x	x
10	1	1	x	x

$g = A + B\overline{C} + C\overline{D} + \overline{B}C$

Figure 11.13a-g Karnaugh Maps and Simplified Boolean Expressions for Display Segments a to g

Lesson No. 12

COMPARATOR

A comparator circuit compares two numbers and sets one of its three outputs to 1 indicating the result of the comparison operation. A Comparator circuit has multiple inputs and three outputs.

A 2-bit Comparator circuit compares two 2-bit numbers A and B. The comparator circuit has three outputs. It sets the A>B output to 1 if A>B. It sets the A=B output to 1 if A=B and sets A<B output to 1 if A < B.

- The output A>B is set to 1 when the input combinations are 01 00, 10 00, 10 01, 11 00, 11 01 and 11 10
- The output A=B is set to 1 when the input combinations are 00 00, 01 01, 10 10 and 11 11
- The output A<B is set to 1 when the input combinations are 00 01, 00 10, 00 11, 01 10, 01 11 and 10 11

The circuit has 4-bit input, 2-bits represent A and 2-bits represent B and a 3-bit output representing A>B, A=B and A<B. To represent the function of a Comparator circuit, three function tables are required for each of the three outputs. A single function table is drawn with three outputs. Table 12.1.

Input				Output		
A_1	A_0	B_1	B_0	A>B	A=B	A<B
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

Table 12.1 Function Table of a Comparator Circuit

Each of the three outputs, A>B, A=B and A<B are mapped separately using three 4-variable Karnaugh maps. The Karnuagh Maps and the simplified expressions for the three outputs are shown. Figure 12.1

$A_1A_0\backslash B_1B_0$	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

$$(A > B) = A_1\overline{B_1} + A_0\overline{B_1}\,\overline{B_0} + A_1A_0\overline{B_0}$$

$A_1A_0\backslash B_1B_0$	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

$$(A = B) = \overline{A_1}\,\overline{A_0}\,\overline{B_1}\,\overline{B_0} + \overline{A_1}A_0\overline{B_1}B_0 + A_1A_0B_1B_0 + A_1\overline{A_0}B_1\overline{B_0}$$

$A_1A_0\backslash B_1B_0$	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

$$(A < B) = \overline{A_1}B_1 + \overline{A_1}\,\overline{A_0}B_0 + \overline{A_0}B_1B_0$$

Figure 12.1a-c Simplified Boolean expressions for the A>B, A=B and A<B outputs

Quine-McCluskey Simplification Method

Karnuagh map method becomes difficult to manage when numbers of variables exceed 4. Even with a 4-varaiable K-map, grouping of 1s or 0s depends on the ability of the user to detect optimum groups. Some times some redundant groups are included which adds a product term or a sum term which is not required and thus the expression is not the simplified version.

Consider the two 4-variable K-map with the groups of 1s shown. Figure 12.2.

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

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

Figure 12.2 4-Variable Karnaugh Maps with redundant terms

In the 4-variable K-map on the left, 6 groups of 4 cells each are formed. The 6 groups form the six terms $AB, A\overline{C}, AD, BC, CD$ and $\overline{B}D$. Out of these six terms three terms are redundant and therefore they are introducing three extra product terms which are not required. The essential terms that are required are $A\overline{C}$, BC and $\overline{B}D$.

In the first K-map the group of 1s formed by cells 9, 11, 13 and 15, the group formed by cells 12, 13, 14 and 15 and the group formed by cells 3, 7, 11 and 15 are redundant.

In the 4-variable K-map on the right, 5 groups are formed. The 5 groups form the five terms $AB\overline{C}$, ACD, $\overline{A}BC$, $\overline{A}\overline{C}D$ and BD. Out of these five groups the largest group of 4 cells is redundant and therefore it is introducing an extra product term which is not required. The essential terms that are required are $AB\overline{C}, ACD$, $\overline{A}BC$ and $\overline{A}\overline{C}D$.

In both the Karnaugh maps, finding the redundant terms is not very obvious. The Quine-McCluskey approach of simplifying Boolean expression is based on an exhaustive search where each minterm is compared with every other minterm in order to remove single variables. The exhaustive search is continued until only a few terms remain which do not share any common variable that can be eliminated. From these remaining terms the minimal product terms are selected that represent the simplified form of Boolean expression.

Quine-McCluskey is a program based method that is able to carry out the exhaustive search for removing shared variables. The Quine-McCluskey method is a two step method which comprises of finding Prime Implicants and selecting a minimal set of Prime Implicants.

- Find Prime Implicants: Find by an exhaustive search all the terms that are candidates for inclusion in the simplified function. These terms are known as Prime Implicants.
- Selecting Minimal Set of Prime Implicants: Choose from amongst the Prime Implicants those that give expression with the least number of literals.

The Quine-McCluskey is explained with the help of two examples, each of which uses a slightly different variation of the exhaustive search method. The methods describe the algorithms of the Quine McCluskey method. The two expressions that are simplified using Quine-McCluskey are based on the two set of Minterms mapped to the 4-variable Karnaugh maps shown in figure 12.2

Example 1

A function is defined in Canonical Sum form $\sum_{A,B,C,D}(1,3,6,7,8,9,11,12,13,14,15)$. As the first step of the Quine McCluskey method to find the Prime Implicants through an exhaustive search, all the Minterms are listed in a tabular form. Table 12.2.

Minterm	A	B	C	D
1	0	0	0	1
3	0	0	1	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Table 12.2 Table of Minterms

Minterm	A	B	C	D	used
1	0	0	0	1	✓
8	1	0	0	0	✓
3	0	0	1	1	✓
6	0	1	1	0	✓
9	1	0	0	1	✓
12	1	1	0	0	✓
7	0	1	1	1	✓
11	1	0	1	1	✓
13	1	1	0	1	✓
14	1	1	1	0	✓
15	1	1	1	1	✓

Table 12.3 Rearranged Minterms

The Table of Minterms is reorganized and the Minterms are arranged in groups of minterms having 0, 1, 2, 3 and 4 1s. This is done to allow different minterms to be easily compared and allow for elimination of single variables. The rearranged Minterm table is shown in table 12.3. Four group of Minterms are formed.

- Minterms 1 and 8 have only single 1s
- Minterms 3, 6, 9 and 12 have two 1s each
- Minterms 7, 11, 13 and 14 have three 1s each
- Minterm 15 has 4 1s.

An extra column is added to the table of minterms which is used to mark the terms that are compared together to eliminate a variable. All pairs of minterms which can be compared together to eliminate a variable are marked as used.

When comparing minterms the rule is to compare each minterm in one group with each minterm in the other group. Thus in this example, minterms 1 and 8 in group having single 1s are compared with each of the 4 minterms 3, 6, 9 and 12 in the group having minterms of 2 1s each. Similarly, each of the 4 minterms 3, 6, 9 and 12 are compared with each of the minterms in the next group having 3 1s, that is, minterms 7, 11, 13 and 14. Finally, each of the minterms 7, 11, 13 and 14 are compared with the minterm 15 in the last group having all 1s or 4 1s.

	A	B	C	D	used
1,3	0	0	-	1	✓
1,9	-	0	0	1	✓
8,9	1	0	0	-	✓
8,12	1	-	0	0	✓
3,7	0	-	1	1	✓
3,11	-	0	1	1	✓
6,7	0	1	1	-	✓
6,14	-	1	1	0	✓
9,11	1	0	-	1	✓
9,13	1	-	0	1	✓
12,13	1	1	0	-	✓
12,14	1	1	-	0	✓
7,15	-	1	1	1	✓
11,15	1	-	1	1	✓
13,15	1	1	-	1	✓
14,15	1	1	1	-	✓

The results of the comparisons between two minterms are represented in a separate table. Table 12.4. The first column lists the minterms that have been compared together to eliminate common variables. So terms 1 and 3 forms a single term eliminating variable C, forming the product term $A\bar{B}\bar{D}$. The comparison terms 1 and 3 are marked as used in table 12.3. Similarly, terms 1 and 9 form a single term eliminating variable A, forming the product term $\bar{B}\bar{C}D$. Both these terms are marked as used in table 12.3. Similarly, terms 8, 9 eliminate variable D, terms 8, 12 eliminate variable B, terms 3, 7 eliminate variable B and so on. All these terms are marked as used in table 12.3.

As a result of comparison a total of 16, three variable product terms are formed, eliminating a single variable from each term. All the 16 terms are represented in table 12.4. All the minterms in table 12.3 are shown to be used.

The exhaustive search for finding prime implicants has not completed. The three variable terms in table 12.4 are compared to eliminate another single variable. All terms that combine to eliminate a variable are represented in table 12.5.

	A	B	C	D	used
1,3,9,11	-	0	-	1	
8,9,12,13	1	-	0	-	
3,7,11,15	-	-	1	1	
6,7,14,15	-	1	1	-	
9,11,13,15	1	-	-	1	
12,13,14,15	1	1	-	-	

Table 12.5 Compared Minterms, Two variable removed

Thus terms 1,3 and terms 9,11 in table 12.4 form the product term $\bar{B}D$ eliminating variable A. Whilst comparing terms in table 12.4, a pair of terms which are different in a single variable are used. The terms 1,3 and 9,11 are different in a single variable A only. All terms in table 12.4 which form a simpler product term eliminating a single variable are marked as used in table 12.4.

In table 12.5 there are 6 product terms of two variables each. If the terms in table 12.5 are compared, none of them form pairs to eliminate a variable, thus all the 6 terms are marked as not used. An unmarked term represents a Prime Implicant. The exhaustive search for Prime Implicants has been completed. No more terms can be eliminated therefore the terms $\bar{B}D$, $A\bar{C}$, CD, BC, AD and AB are considered to be Prime Implicants.

In the second step of Quine-McCluskey method the essential and minimal Prime Implicants are found. The Prime Implicants found in the first step are listed in left most column of the table. Table 12.6. All the original minterms are listed in the top row.

	1	3	6	7	8	9	11	12	13	14	15
$\bar{B}D$	(X)	x				x	x				
$A\bar{C}$					(x)	x		x	x		
CD		x		x			x				x
BC			(x)	x						x	x
AD						x	x		x		x
AB								x	x	x	x

Table 12.6 Table of Prime Implicants

In each cell an x is marked indicating that the Prime Implicant listed in the left column covers the minterm mentioned in the top row. Thus the Prime Implicant $\bar{B}D$ covers the minterms 1, 3, 9 and 11. In other words minterms 1, 3, 9 and 11 all have the product terms $\bar{B}D$. The table 12.6 can be directly implemented from table 12.5.

Circles are marked in cells having x, which represent minterms covered by only a single Prime Impicant. Thus the minterms 1, 6 and 8 are covered by only the Prime Implicants $\bar{B}D$, $A\bar{C}$ and BC respectively. These three Prime Implicants in fact are the three essential Prime Implicants that cover all the minterms. The simplified expression therefore has the terms $\bar{B}D$, $A\bar{C}$ and BC. The Prime Implicants CD, AD and AB are redundant product terms which are not required. The simplified expression is $\bar{B}D + A\bar{C} + BC$

Example 2

A function is defined in Canonical Sum form as $\sum_{A,B,C,D}(1,5,6,7,11,12,13,15)$. The Minterms along with variables ABCD are written in a tabular form. Each minterm is represented in terms of its binary value. Table 12.7.

Minterm	A	B	C	D
1	0	0	0	1
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
15	1	1	1	1

Table 12.7 Table of Minterms

Minterm	A	B	C	D	Used
1	0	0	0	1	✓
5	0	1	0	1	✓
6	0	1	1	0	✓
12	1	1	0	0	✓
7	0	1	1	1	✓
11	1	0	1	1	✓
13	1	1	0	1	✓
15	1	1	1	1	✓

Table 12.8 Rearranged Minterms

The table of minterms is reorganized in terms of groups of minterms having 0, 1, 2, 3 and 4 1s.

- Minterms 1 has a single 1
- Minterms 5, 6 and 12 have two 1s each
- Minterms 7, 11 and 13 have three 1s each
- Minterm 15 has 4 1s.

An extra column is added to the table of minterms which indicates which minterms have been compared together to eliminate a variable. Table 12.8. All pairs of minterms which can be compared together to eliminate a variable are marked as used.

When comparing minterms the rule is to compare each minterm in one group with each minterm in the other group. Thus, in this example, minterm 1 in group having single 1s is compared with each of the 3 minterms 5, 6 and 12 in the group having minterms of 2 1s each. Similarly, each of the 3 minterms 5, 6 and 12 are compared with each of the 3 minterms in the next group having 3 1s, that is, minterms 7, 11 and 13. Finally, each of the 3 minterms 7, 11 and 13 are compared with the minterm 15 in the last group having all 1s or 4 1s.

The results of the comparisons between two minterms are represented in a separate table. Table 12.9. The first column lists the minterms that have been compared together to eliminate common variables. The second column shows the variable in terms of its binary value. So terms 1 and 5 form a single term eliminating variable B, forming the product term $\bar{A}\bar{C}D$. Variables A, B, C and D have binary values 8, 4, 2 and 1 respectively.

Minterms	Variable removed	used
1,5	4	
5,7	2	✓
5,13	8	✓
6,7	1	
12,13	1	
7,15	8	✓
11,15	4	
13,15	2	✓

Table 12.9 Compared Minterms, Single variable eliminated

The comparison terms 1 and 5 are marked as used in table 12.8. Similarly terms 5 and 7 form a single term eliminating variable C, forming the product term $\bar{A}BD$. Both these terms are marked as used in table 12.8. Similarly, terms 5, 13 eliminate variable A, terms 6, 7 eliminate variable D, terms 12, 13 eliminate variable D and so on.

As a result of comparison a total of 8, three variable product terms are formed, eliminating a single variable from each term. All the 8 terms are represented in table 12.9. The exhaustive search for finding Prime Implicants has not completed.

Terms 5,7 and 13, 15 compare to form a product term BD eliminating variable A. The terms 5,7 and 13,15 are marked as used in table 12.9. Similarly, terms 5,13 and 7,15 compare to form an identical product term BD eliminating variable A. Both the terms 5,13 and 7, 15 are marked as used in table 12.9. To speed up the comparison process terms having the same missing or removed variables are compared. However, the comparison should eliminate only a single variable. Thus in table 12.9 terms 1,5 and terms 11,15 have their B variable eliminated. Considering that 1,5 represents the product term $\bar{A}\bar{C}D$ and terms 11, 15 represent the product term ACD can not be compared as two variables are different. Terms 5,7 and 13,15 can be compared as in both the product terms the variable C is missing and by comparing the two product terms removes variable A.

Minterms	Term removed	used
5,7,13,15	2,8	

Table 12.10 Minterms compared, two variables removed

No more comparisons of terms and elimination of variables take place. Thus the Prime Implicants have been found. There are 4 prime implicants in table 3 and another prime implicant in table 12.10. The five prime implicants are represented by product terms $\overline{A}\overline{C}D$, $\overline{A}BC$, $AB\overline{C}$, ACD and BD.

In the second step of Quine-McCluskey method the essential and minimal Prime implicants are found. The Prime Implicants found in the first step are listed in left most column of the table. Table 12.11. All the original minterms are listed in the top row. In each cell an x is marked indicating that the Prime Implicant listed in the left column covers the minterm mentioned in the top row.

The Prime Implicant $\overline{A}\overline{C}D$ covers the minterms 1 and 5. In other words minterms 1 and 5 all have the product terms $\overline{A}\overline{C}D$. The table 12.11 can be directly implemented from table 12.9 and 12.10.

Circles are marked in cells having x, which represent minterms covered by only a single Prime Impicant. Thus the minterms 1, 6, 11 and 12 are covered by only the Prime Implicants $\overline{A}\overline{C}D$, $\overline{A}BC$, $AB\overline{C}$ and ACD respectively. These 4 implicants in fact are the three essential Prime Implicants that cover all the minterms. The simplified expression is $\overline{A}\overline{C}D + \overline{A}BC + AB\overline{C} + ACD$

	1	5	6	7	11	12	13	15
$\overline{A}\overline{C}D$	(x)	x						
$\overline{A}BC$			(x)	x				
$AB\overline{C}$						(x)	x	
ACD					(x)			x
BD		x		x			x	x

Table 12.11 Table of Prime Implicants

Comparator Circuit

A 2-bit Comparator circuit that compares two 2-bit numbers A and B and activates one of its three outputs A>B, A=B and A<B depending upon the magnitudes of the numbers A and B has been discussed earlier. The function outputs of the three outputs A>B, A=B and A<B can easily be represented using truth tables which can then be written in a simplified Boolean expression form after simplifying the three function expressions using 4-variable Karnaugh maps.

A comparator circuit that compares two 3-bit numbers A and B instead of the 2-bit numbers has an input of 6-bits, which represents an input combination of 64. Writing a truth

unmanageable. A program based Quine-McCluskey method can easily handle expression of 6 variables represented in the Canonical form $\sum_{A,B,C,D,E,F}(8,16,17,24,.....)$

Odd-Prime Number Detector

A circuit that detects Odd Prime numbers between 0 and 9 has been considered earlier. The circuit is to be improved to detect Odd Prime numbers for a decimal number range represented by 5-bit binary numbers or in terms of decimal numbers between the decimal number range 0 to 31. Writing out a function table to represent the 32 input combinations and their corresponding outputs, and then simplifying the function expression using a 5-varaibale K-map can take up considerable amount of time.

Quine-McCluskey method can be used to easily simplify the 5-variable Boolean expression represented in Canonical Sum form as $\sum_{A,B,C,D,E}(1,3,5,7,11,13,17,19,23,29,31)$. The minterms 1, 3, 5, 7, 11, 13, 17, 19, 23, 29 and 31 represent the 5-bit input combinations (decimal numbers) which are Odd and Prime numbers.

Lesson No. 13

ODD-PRIME NUMBER DETECTOR

The table of minterms is represented. Table 13.1

Minterm	A	B	C	D	E
1	0	0	0	0	1
3	0	0	0	1	1
5	0	0	1	0	1
7	0	0	1	1	1
11	0	1	0	1	1
13	0	1	1	0	1
17	1	0	0	0	1
19	1	0	0	1	1
23	1	0	1	1	1
29	1	1	1	0	1
31	1	1	1	1	1

Table 13.1 Table of Minterms representing Odd-Prime Numbers

The table of minterms is reorganized in terms of groups of minterms having 0, 1, 2, 3 and 4 1s. Table 13.2

- Minterms 1 has a single 1s
- Minterms 3, 5 and 17 have two 1s each
- Minterms 7, 11, 13 and 19 have three 1s each
- Minterm 23 and 29 have 4 1s
- Minterm 31 has 5 1s

Minterm	A	B	C	D	E	used
1	0	0	0	0	1	✓
3	0	0	0	1	1	✓
5	0	0	1	0	1	✓
17	1	0	0	0	1	✓
7	0	0	1	1	1	✓
11	0	1	0	1	1	✓
13	0	1	1	0	1	✓
19	1	0	0	1	1	✓
23	1	0	1	1	1	✓
29	1	1	1	0	1	✓
31	1	1	1	1	1	✓

Table 13.2 Reorganized Minterms representing Odd-Prime Numbers

In the first step of Quine-McCluskey method minterms are compared to eliminate single variables. Minterm 1 is compared with minterms 3, 5 and 17 in the next group. Similarly, each of the 3 minterms 3, 5 and 17 are compared with each of the minterms in the next group having 3 1s, that is, minterms 7, 11, 13 and 19. Minterms 7, 11, 13 and 19 are compared with each of the minterms in the next group having 4 1s, that is, minterms 23 and 29. Finally, each of the two minterms 23 and 29 are compared with the minterm 31 in the last group having all 1s or 5 1s.

The results of the comparisons between minterms are represented in a separate table. Table13.3. The first column lists the minterms that have been compared together to eliminate common variables. Terms 1 and 3 form a single term eliminating variable D, forming the product term $\overline{A}\overline{B}\overline{C}E$. The comparison terms 1 and 3 are marked as used in table 13.2. Similarly, terms 1 and 5 from a single term eliminating variable C, forming the product term $\overline{A}\overline{B}D\overline{E}$. Both these terms are marked as used in table 13.2. Similarly, terms 1, 17 eliminate variable A, terms 3, 7 eliminate variable C, terms 3, 11 eliminate variable B and so on.

Minterms	Variable removed	used
1,3	2	✓
1,5	4	✓
1,17	16	✓
3,7	4	✓
3,11	8	
3,19	16	✓
5,7	2	✓
5,13	8	
17,19	2	✓
7,23	16	✓
13,29	16	
19, 23	4	✓
23,31	8	
29,31	2	

Table 13.3 Table of minterms, Single variable eliminated

As a result of comparison a total of 14, four variable product terms are formed, eliminating a single variable from each term. All the 14 terms are represented in table 13.3. The exhaustive search for finding Prime Implicants has not completed. The results of the comparisons between two terms in table 13.3 are represented in a separate table. Table 13.4.

Minterms	Variable removed	used
1,3,5,7	2,4	
1,3,17,19	2,16	
3,7,19,23	4,16	

Table 13.4 Table of minterms, Two variable eliminated

The first column lists the terms that have been compared together to eliminate common variables. So terms, 1, 3, 5 and 7 form a single term eliminating variables C and D, forming the product term $\overline{A}\overline{B}E$. The comparison terms 1,3 and 5,7 are marked as used in table 13.3. Similarly terms, 1, 3, 17 and 19 from a single term eliminating variable A and D, forming the product term $\overline{B}\overline{C}E$. Both these terms are marked as used in table 13.3. All the product terms in table 13.3 are compared to eliminate common variables. No more comparisons of terms and elimination of variables take place, thus the Prime Implicants have been found.

There are 3 Prime Implicants in table 13.4 and 5 Prime Implicant in table 13.3. The eight Prime Implicants are represented by product terms $\bar{A}\bar{C}DE$, $\bar{A}C\bar{D}\bar{E}$, $BC\bar{D}\bar{E}$, $ACDE$, $ABCE$, $\bar{A}\bar{B}\bar{E}$, $\bar{B}\bar{C}\bar{E}$ and $\bar{B}\bar{D}E$.

In the second step of Quine-McCluskey method the essential and minimal Prime Implicants are found. The Prime Implicants found in the first step are listed in left most column of the table. 13.5. All the original minterms are listed in the top row. In each cell an x is marked indicating that the Prime Implicant listed in the left column covers the minterm mentioned in the top row. Thus the Prime Implicant $\bar{A}\bar{C}DE$ covers the minterms 3 and 11. In other words minterms 3 and 11 all have the product terms $\bar{A}\bar{C}DE$. The table 13.5 can be directly implemented from table 13.3 and 13.4.

	1	3	5	7	11	13	17	19	23	29	31
$\bar{A}\bar{C}DE$		x			(x)						
$\bar{A}C\bar{D}\bar{E}$			x			x					
$BC\bar{D}\bar{E}$						x				x	
$ACDE$									x		x
$ABCE$										x	x
$\bar{A}\bar{B}\bar{E}$	x	x	x	x							
$\bar{B}\bar{C}\bar{E}$	x	x					(x)	x			
$\bar{B}\bar{D}E$		x		x				x	x		

Table 13.5 Table of Prime Implicants

Circles are marked in cells having x, which represent minterms covered by only a single prime impicant. Thus the minterms 11 and 17 are covered by only the Prime Implicants $\bar{A}\bar{C}DE$ and $\bar{B}\bar{C}\bar{E}$ respectively. These implicants do not cover all the minterms. The other essential implicants that have minimum number of variables and which cover all the remaining minterms are $BC\bar{D}\bar{E}$, $ACDE$ and $\bar{A}\bar{B}\bar{E}$. The simplified expression is $\bar{A}\bar{C}DE + BC\bar{D}\bar{E} + ACDE + \bar{A}\bar{B}\bar{E} + \bar{B}\bar{C}\bar{E}$. The function can also be represented by the expression $\bar{A}\bar{C}DE + \bar{A}C\bar{D}\bar{E} + ABCE + \bar{B}\bar{C}\bar{E} + \bar{B}\bar{D}E$. In both cases the number of product terms is the same with same number of variables.

Combinational Logic

Individual gates AND, OR and NOT, NAND and NOR Universal Gates and XOR and XNOR gates perform unique functions. These gates in their individual capacity can not perform any useful function. The Logic Gates have to be connected together in different combinations to form Logic Circuits that are able to perform some useful operation like addition , comparison etc. These combinations of gates which results in a circuit used to perform some function are known as Combinational Logic.

The function of any Digital Logic circuit is represented by Boolean expressions. In the examples discussed earlier, Boolean expressions for various functions have been determined. Two forms of representing functions through Boolean expressions are the SOP and POS

expressions. These two types of Boolean expressions are implemented using a combination of gates to form Combinational Logic Circuits.

Combinational Circuit Implementation based on SOP form

A standard way to express a Boolean expression is the SOP form. The expression has several product terms which are summed together through a single OR gate. The product terms can have variables and their complemented form. A SOP expression is implemented by using a combinational circuit made up of many AND gates and a single OR gate (AND-OR gate combination). The inputs to the AND gates can be in the complemented form or the un-complemented form, requiring the use of NOT gates.

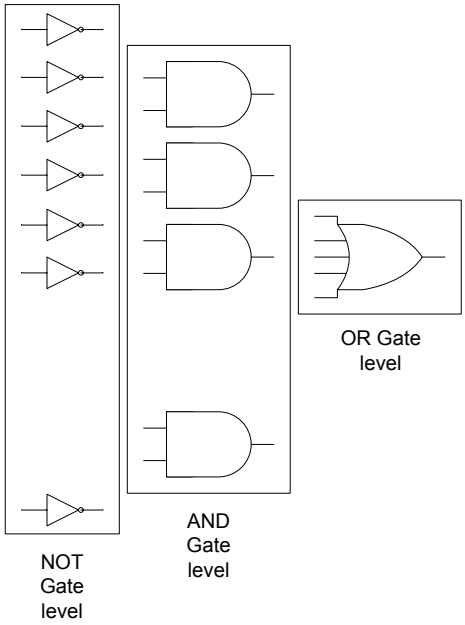


Figure 13.1 General, Combination al Logic Circuit based on SOP form

The diagram shows the general architecture of the SOP Implementation. The implementation is based on three levels of gates. SOP expression is implemented by the AND-OR combination of gates. The AND gates produce the product terms. Outputs of all the AND gates are connected to a single multiple input OR gate for sum of products. The product terms comprise of literals in their complemented form and un-complemented form which are implemented by NOT gates connected to the inputs of the AND gates.

Combinational Circuit Implementation based on POS form

A standard way to express a Boolean expression is the POS form. The expression has several sum terms which are multiplied together through a single AND gate. The sum terms can have variables and their complemented form. A POS expression is implemented by using a combinational circuit made up of many OR gates and a single AND gate (OR-AND gate combination). The inputs to the OR gates can be in the complemented form or the un-complemented form, requiring the use of NOT gates.

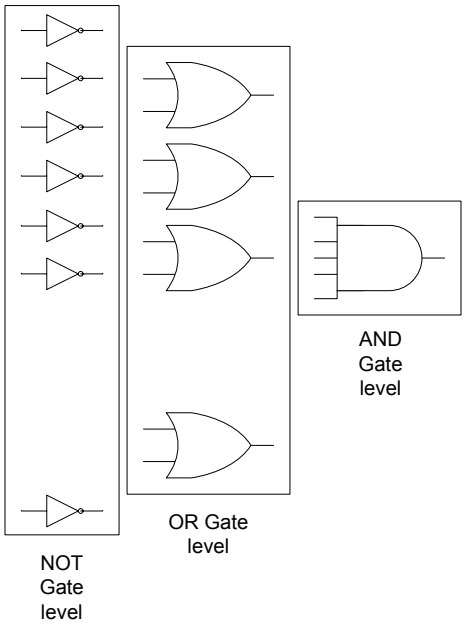


Figure 13.2 General, Combination al Logic Circuit based on POS form

The diagram shows the general architecture of the POS Implementation. The implementation is based on three levels of gates. POS expression is implemented by the OR-AND combination of gates. The OR gates produce the sum terms. Outputs of all the OR gates are connected to a single multiple input AND gate for product of sum terms. The sum terms comprise of literals in their complemented form and un-complemented form which are implemented by NOT gates connected to the inputs of the OR gates.

Design and Implementation of Combinational Circuits

The design and implementation of a combinational circuit starts by defining the function of the Combinational circuit. The function of a combinational circuit is defined by a truth table or a function table. Once the function table is defined the combinational circuit can be directly implemented from the function table.

Direct implementation of a combinational circuit from the function table results in a circuit which uses maximum number of gates organized at three levels. This increases the cost, the size of the circuit and the power requirement of the Combinational circuit. The propagation delay of the circuit is of the order of three gates. Therefore, before implementing the circuit the expression is simplified using the manual method by applying rules, laws and theorems of Boolean Algebra or by the Karnaugh map method or the Quine-McCluskey method if the number of variables exceeds 4.

Implementation of an Adjacent 1s Detector Circuit

A circuit that checks an input number and determines if it has two adjacent 1s is considered to explain the entire process of design and implementation of a typical Combinational Logic Circuit. The Adjacent 1s detector circuit is implemented using the standard SOP and POS forms of Boolean expressions. The circuit is also implemented using the simplified Boolean expressions. The alternate form of implementing the circuit using only NAND or NOR gates is also discussed.

© Virtual University of Pakistan

1. SOP based Implementation of the Adjacent 1s Detector Circuit

The Adjacent 1s Detector accepts 4-bit inputs. If two adjacent 1s are detected in the input, the output is set to high. The operation of the Adjacent 1s Detector is represented by the function table. Table 13.6. In the function table, for the input combinations 0011, 0110, 0111, 1011, 1100, 1101, 1110 and 1111 the output function is a 1.

Input				Output	Input				Output
A	B	C	D	F	A	B	C	D	F
0	0	0	0	0	1	0	0	0	0
0	0	0	1	0	1	0	0	1	0
0	0	1	0	0	1	0	1	0	0
0	0	1	1	1	1	0	1	1	1
0	1	0	0	0	1	1	0	0	1
0	1	0	1	0	1	1	0	1	1
0	1	1	0	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1

Table 13.6 Function Table of Adjacent 1s Detector

Implementing the circuit directly from the function table based on the SOP form requires 8 AND gates for the 8 product terms (minterms) with an 8-input OR gate. Figure 13.3. The total gate count is

- One 8 input OR gate
- Eight 4 input AND gates
- Ten NOT gates

The expression can be simplified using a Karnaugh map, figure 13.4, and then the simplified expression can be implemented to reduce the gate count. The simplified expression is $AB + CD + BC$. The circuit implemented using the expression $AB + CD + BC$ has reduced to 3 input OR gate and 2 input AND gates. Figure 13.5

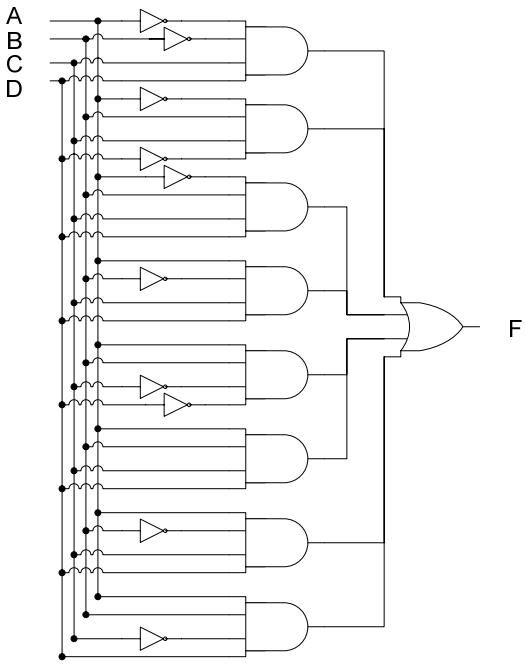


Figure 13.3 SOP Implementation of Adjacent 1s Detector

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

Figure 13.4 Simplification of Adjacent 1s Detector SOP Boolean Expression

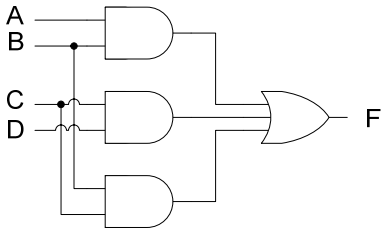


Figure 13.5 Simplified SOP based Adjacent 1s Detector

The simplified Adjacent 1s Detector circuit uses only four gates reducing the cost, the size of the circuit and the power requirement. The propagation delay of the circuit is of the order of two gates.

The simplified Adjacent 1s Detector circuit can be implemented using only NAND Gates. The AND-OR combinational circuit can be easily replaced by a NAND based implementation without changing the number of gates. Figure 13.6.



Figure 13.6 NAND based Adjacent 1s Detector

Bubbles representing NOT gates are placed at the output of the three AND gates. Converting the three AND gates to NAND gates. To balance out the three NOT gates added at the outputs of the three AND gates, three bubbles representing three NOT gates are also placed at the three inputs of the OR gate. The Resulting OR gate symbol with three bubbles at the three inputs is an alternate symbol for a three input NAND gate.

Implementing Combinational Logic Circuits using only NAND gates helps in reducing the circuit size and cost as the Integrated Circuit packages multiple gates in a single package. If, for example, the 3-input NAND gate in the circuit had been a 2-input NAND gate, only a single IC package (74LS00) would have been required. For the circuit shown in figure 13.5 two separate IC packages (74LS08 and 74LS32) are required.

2. POS based Implementation of the Adjacent 1s Detector Circuit

A combinational Adjacent 1s Detector circuit can be implemented, based on the POS form. It was discussed earlier that it is very easy to switch between SOP and the POS representations using the information in a function table or the information mapped to a Karnaugh Map. Referring to the Function Table for the Adjacent 1s Detector. Table 13.6 a POS based Adjacent 1s Detector circuit can be easily implemented by using the Sum terms (Maxterms). The POS based circuit for this particular case has 8 sum terms which require 8 OR gates and a single 8-input AND gate. Figure 13.7. The total gate count is

- One 8 input AND gate
- Eight 4 input OR gates
- Ten NOT gates

Both, the SOP based circuit discussed earlier and the POS based circuit give identical outputs for identical set of input combinations. One practical purpose of using either the SOP or the POS based implementation is to reduce the size of the circuit and have a simpler circuit. In the example of Adjacent 1s Detector circuit both the SOP and POS based implementations have equal number of minterms (8) and maxterms (8) thus both implementation use exactly the same number of gates (19). In many cases, the function describing the operation of a combinational circuit has minterms which are either less than or more than the number of maxterms. Thus it is wiser to choose the implementation form that uses the least number of minterms or maxterms to achieve a combinational circuit that uses the least number of gates.

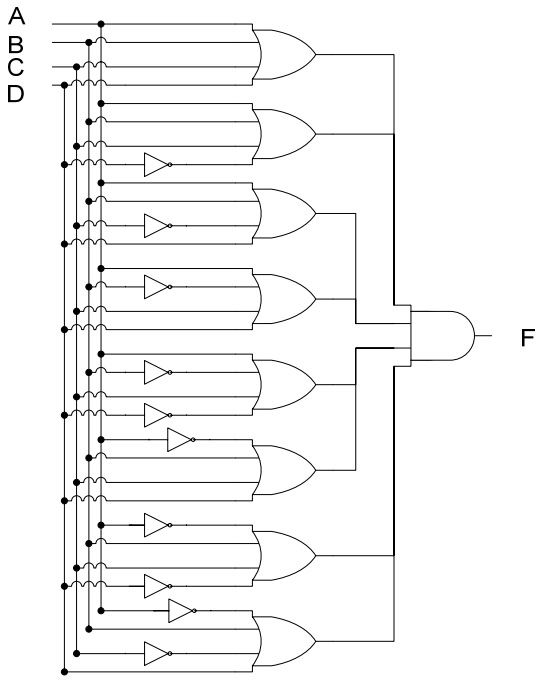


Figure 13.7 POS Implementation of Adjacent 1s Detector

The POS expression can be simplified using a Karnaugh map. Figure 13.8, the simplified expression can be implemented to reduce the gate count. The simplified expression is

$$(A + C)(B + C)(B + D)$$

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

Figure 13.8 Simplification of Adjacent 1s Detector POS Boolean Expression

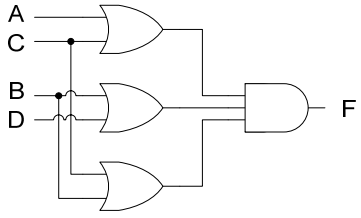


Figure 13.9 Simplified POS based Adjacent 1s Detector

The simplified Adjacent 1s Detector circuit uses only four gates reducing the cost, the size of the circuit and the power requirement. The propagation delay of the circuit is of the order of two gates.

The simplified Adjacent 1s Detector circuit can be implemented using only NOR Gates. The OR-AND combinational circuit can be easily replaced by a NOR based implementation without changing the number of gates. Figure 13.10.

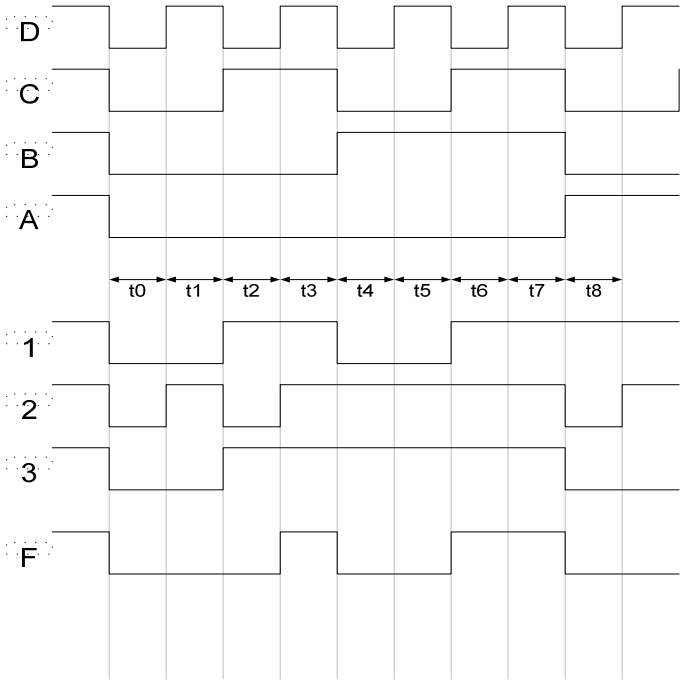


Figure 13.10 NOR based Adjacent 1s Detector

Bubbles representing NOT gates are placed at the output of the three OR gates, converting the three OR gates to NOR gates. To balance out the three NOT gates added at the outputs of the three OR gates, three bubbles representing three NOT gates are also placed at the three inputs of the AND gate. The Resulting AND gate symbol with three bubbles at the three inputs is an alternate symbol for a three input NOR gate.

Operation of Adjacent 1s detector Circuit

The operation of a Combinational Logic Circuit can be verified by applying varying set of signals at the input of the circuit and comparing the output of the combinational circuit with the corresponding outputs in the Function Table. If the varying set of inputs and the corresponding outputs are plotted over a period of time, the timing diagram thus obtained, describes the operation of the circuit. Figure 13.11



To prove that the SOP and POS based Adjacent 1s Detector combinational circuits synthesized from the Function table. Table 13.6 are identical, the timing diagram, figure 13.11 is based on the operation of the POS based simplified circuit. Figure 13.9

The timing diagram is for time intervals t0 to t8. A, B, C and D are the inputs to the circuit which are shown changing with time. The timing signals 1, 2 and 3 represent the outputs of the OR gates 1, 2 and 3. The timing signal F represents the output of the circuit.

At interval t0 the input ABCD to the circuit is 0000, the outputs of the three OR gates is 0, 0 and 0 and the circuit output is also 0. At the interval t3 the input ABCD to the circuit is 0011, the outputs of OR gates 1, 2 and 3 are 111. The output F is also a 1, which indicates adjacent 1s. At interval t6 the input ABCD to the circuit is 0110, the outputs of OR gates 1, 2 and 3 are 111. The output F is again 1 indicating adjacent 1s.

The operation of the circuit which is based on the POS simplified expression also proves that a POS based expression determined from the truth table and K-map results in a circuit which operates in an identical manner to that of a SOP based circuit.

Active low/high Inputs and Outputs

The circuits discussed so far have their output set to when to indicate an active state. For example, the output of the BCD to 7-Segment Decoder circuit has its seven segment outputs set to 1 to indicate a segment that has been selected. Similarly, the Comparator circuit's three outputs are normally at binary 0. The appropriate output is set to 1 to indicate the relationship between the two numbers. The Odd-Prime Number detector circuit output normally is set at 0. It is activated to 1 to indicate an Odd-Prime number. The Adjacent 1s Detector circuit also sets its output to active 1 to indicate detection of Adjacent 1s. All the four circuits have an active-high output. That is, normally the output is at logic 0. The output is set to 1 to indicate an active state.

Combinational circuits can have an active-high output or an active-low output. An active-high or active-low output doesn't effect the operation of the combinational circuit in any manner. To convert a circuit having an active-high output to active low-output requires the inversion of the circuit output by connecting a NOT gate. Symbolically, a bubble is added to the circuit output. Thus, circuits having a bubble at their outputs are considered to have an active-low output.

Circuits can also have active-high or active-low inputs. The operation of the circuits having an active-high input is not any different from that of an active-low input circuit. Active-low input circuits are activated on a logic 0 input. Circuits having an active-low input have bubbles connected to circuits inputs. The four circuits discussed so far have active-high inputs.

The four logic gates AND, OR, NAND and NOR can be described in terms of their input and output logic levels. The AND gate doesn't have any bubbles at its inputs or output. The AND gate performs AND operation on two active high inputs to result in an active high output. The OR gate also doesn't have any bubbles at its inputs and output. OR gate performs OR operation on two active high inputs to result in an active high output. The NAND and NOR gates have a bubble at their outputs. Their operation can be described in terms of AND and OR gates. NAND gate performs AND operation on two active high inputs resulting in an active low output. The NOR gate performs OR operation on two active high inputs to result in an

To help understand active-low input, consider the active-high input and active-high output SOP circuit. Fig. 13.5 which is converted into an active-low input and output circuit by connecting NOT gates at the circuit inputs and outputs. Figure 13.12. The circuit operation is verified with the help of a timing diagram. Figure 13.13.

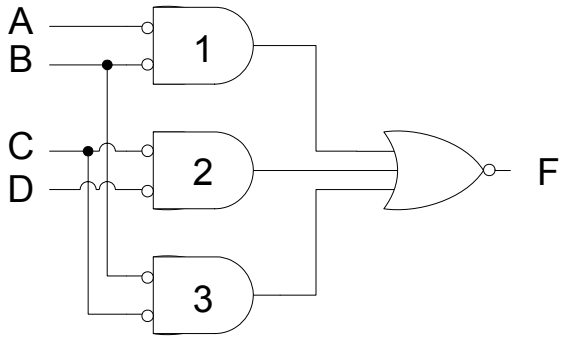


Figure 13.12 SOP based active-low input and output Adjacent 1s Detector

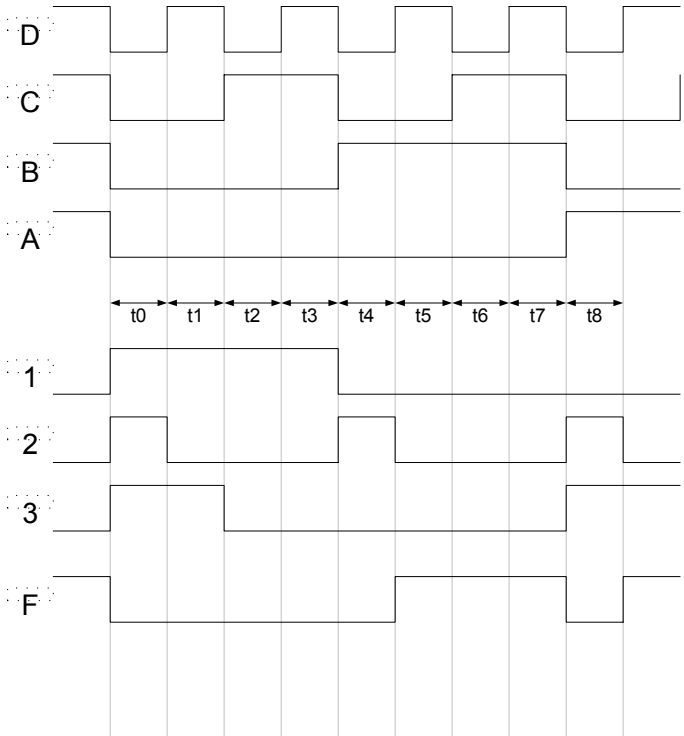


Figure 13.13 Timing Diagram of the active-low input/output Adjacent 1s Detector

The timing diagram describes the operation of the circuit for the intervals t0 to t8. The timing signals A, B, C and D represent the active-low inputs applied at the inputs. The timing signals 1, 2 and 3 represent the outputs of the NOR gates 1, 2 and 3 respectively, shown in their alternate symbolic form. The timing signal F represents the active-low output.

At interval t0 the active-low input at inputs ABCD is 0000 which actually represents 1111. The active-low output F is 0 which indicates that adjacent 1s have been detected. Similarly at intervals t1 to t4, the active-low inputs ABCD 0001, 0010, 0011 and 0100 actually represent the numbers 1110, 1101, 1100 and 1011, the output is 0 indicating that adjacent 1s have been detected.

Implementation of an Odd-Parity Generator Circuit

Consider the second example of a circuit to generate odd parity. The circuit checks an 8-bit number and generates a parity bit to fulfil the Odd-Parity condition. The 8-bit data and the parity bit are communicated to the receiver circuit. The receiver circuit checks the 8-bit data and the parity bit to determine if an error has occurred.

The first step in implementing any circuit is to represent its operation in terms of a Truth or Function table. The function table for an 8-bit data as input has 2^8 has 256 input combinations, which becomes unmanageable. Therefore, for the sake of simplicity a 4-bit data with odd parity is assumed. The receiver circuit is also based on the 4-bit data.

Lesson No. 14

IMPLEMENTATION OF AN ODD-PARITY GENERATOR CIRCUIT

The first step in implementing any circuit is to represent its operation in terms of a Truth or Function table. The function table for an 8-bit data as input has 2^8 has 256 input combinations, which becomes unmanageable. Therefore, for the sake of simplicity a 4-bit data with odd parity is assumed. The receiver circuit is also based on the 4-bit data. The function table for the 4-bit data is shown. Figure 14.1

Input				Output	Input				Output
D_3	D_2	D_1	D_0	P	D_3	D_2	D_1	D_0	P
0	0	0	0	1	1	0	0	0	0
0	0	0	1	0	1	0	0	1	1
0	0	1	0	0	1	0	1	0	1
0	0	1	1	1	1	0	1	1	0
0	1	0	0	0	1	1	0	0	1
0	1	0	1	1	1	1	0	1	0
0	1	1	0	1	1	1	1	0	0
0	1	1	1	0	1	1	1	1	1

Table 14.1 Function Table of an Odd-Parity Generator Circuit

The function table represents the 16 possible combinations of 4 data bits. The 4 data bits are represented by variables D3, D2, D1 and D0. The output P represents the state of the Parity bit. Since Odd-Parity is being used therefore the 4-bit data and the parity bit should add up to give odd number of 1s. The function table shows the Parity bit set to 1 when the 16, 4-bit data input combinations have no 1s or an even number of 1s.

The information in the function table is mapped directly to a four variable K-map to simplify the Boolean expression represented by the Odd-Parity generator function. None of the 1s mapped in the K-map are adjacent to each other thus the function mapped to the K-map can not be simplified. Figure 14.1

$D_3D_2\backslash D_1D_0$	00	01	11	10
00	1	0	1	0
01	0	1	0	1
11	1	0	1	0
10	0	1	0	1

Figure 14.1 Karnaugh map of the Odd-Parity Generator Function

However, using the Rules of Boolean algebra, applying Demorgan's theorems and knowing the function table of XOR and XNOR gates the Boolean expression can be simplified. Simplifying the expression based on SOP form results in

$$\overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}CD + \overline{A}B\overline{C}D + \overline{A}BC\overline{D} + AB\overline{C}\overline{D} + ABCD + A\overline{B}C\overline{D} + A\overline{B}\overline{C}D$$
$$= \overline{A}\overline{B}(\overline{C}\overline{D} + CD) + \overline{A}B(\overline{C}D + C\overline{D}) + AB(\overline{C}\overline{D} + CD) + A\overline{B}(\overline{C}D + C\overline{D})$$
$$= \overline{A}\overline{B}(\overline{C}\overline{D} + CD) + AB(\overline{C}\overline{D} + CD) + \overline{A}B(\overline{C}D + C\overline{D}) + A\overline{B}(\overline{C}D + C\overline{D})$$
$$= (\overline{C}\overline{D} + CD)(\overline{A}\overline{B} + AB) + (\overline{C}D + C\overline{D})(\overline{A}B + A\overline{B})$$
$$= (\overline{C \oplus D})(\overline{A \oplus B}) + (C \oplus D)(A \oplus B)$$

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

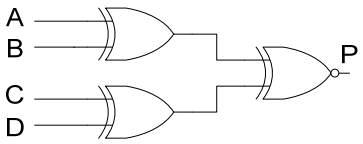


Figure 14.2 Odd-Parity Generator Circuit

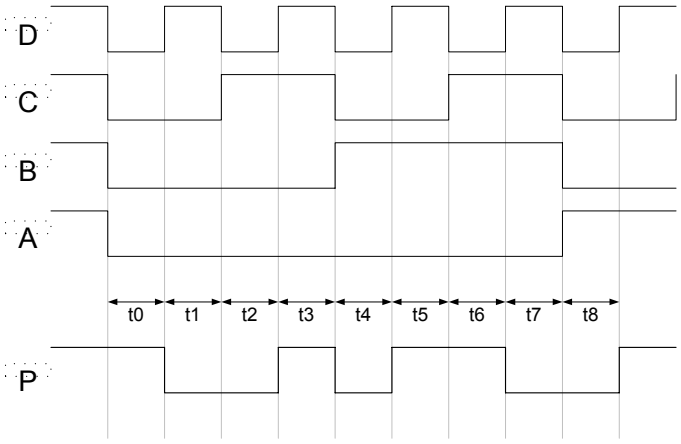
The parity generator circuit shown checks the 4-bit number, generates a parity bit which along with the 4-bit data is transmitted. The receiver calculates the parity bit of the received 4-bit data and compares it with the parity sent. If the received and calculated parity bits are the same, then no error has occurred. An XOR gate is used to detect parity errors. Table 14.2

Input		Output
Received Parity Bit	Calculated Parity Bit	Error Output
0	0	0
0	1	1
1	0	1
1	1	0

Table 14.2 Detecting Error at Receiver End

Operation of Odd-Parity Generator Circuit

The timing diagram shows the operation of the Odd-Parity generator circuit. Figure 14.3. The A, B, C and D timing diagrams represent the changing 4-bit data values. During time interval t0 the 4-bit data value is 0000, during time interval t1, the data value changes to 0001. Similarly during time intervals t2, t3, t4 up to t8 the data values change to 0010, 0011, 0100 and 1000 respectively. During interval t0 the output of the two XOR gates is 0 and 0, therefore the output of the XNOR gate is 1. At interval t1, the outputs of the two XOR gates is 1 and 0, therefore the output of the XNOR gate is 0. The output P can similarly be traced for intervals t2 to t8.



XOR and XNOR Gates

XOR and XNOR gates are used to implement the Odd-Parity Generator Circuit. An XNOR is also used to check for single bit errors at the Receiver end. Both, the XOR and XNOR gates perform simple comparison functions. The XOR gate detects dissimilar inputs, where as the XNOR gate looks for similar inputs. Both, the gates can be considered as functional devices as each gate performs a simple specific function.

The XOR and XNOR gates are implemented using a combination of NOT, AND and OR gates. Since the function performed by the XOR and XNOR gate is commonly used in digital circuits therefore XOR and XNOR gates are available in Integrated circuit form which can be readily used instead of implementing an XOR and XNOR circuit based on NOT-AND-OR combination of gates.

The function table for the Parity Error detector circuit is identical to the truth table of an XOR gate. Boolean expression representing the function of an XOR gate is $\overline{A}B + A\overline{B}$ which is implemented using a combination NOT, AND and OR gates.

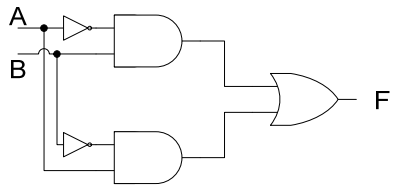


Figure 14.3 Implementation of XOR Gate

The XNOR gate is also implemented using a combination of NOT, AND and OR gates. The function of the XNOR gate is represented in term of Boolean expression as $\overline{A}\overline{B} + AB$. Figure 14.4

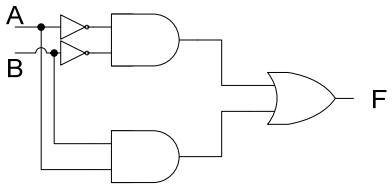


Figure 14.4 Implementation of XNOR Gate

Combinational Function Devices

Digital circuits are formed by the combination of Logic Gates. Most Combinational circuits perform standard and useful functions such as addition, comparison, decoding and encoding, multiplexing and de-multiplexing, selection and enabling of devices and many more operations. Implementation of these standard functional devices through combination of gates takes up considerable space, therefore these functional devices are implemented as MSI or Medium Scale Integrated Chips.

The simplest of these functional devices can be considered to be the NAND and NOR gates which perform the AND-NOT and OR-NOT functions. The XOR and XNOR Gates are also a combination of NOT-AND-OR gates which perform functions to detect dissimilar and similar inputs.

Half Adder and Full Adder

A single bit binary adder circuit basically adds two bits and a carry bit, generated by the addition of the least significant bits. The output of the single bit adder circuit generates a sum bit and a carry bit. A single digit binary adder circuit therefore has three inputs, one representing single bit number A, the other representing the single bit number B and the third bit represents the single bit carry. The single bit binary adder has two bit output. One bit represents the Sum between numbers A and B. The other bit represents the carry bit generated due to addition.

In Digital logic terminology the adder which has been described is known as a full adder. An adder circuit that only has two bit input representing the two single bit numbers A and B and does not have the carry bit input from the least significant digit is regarded as a half-adder. The block diagrams represent a Half-Adder and a Full-Adder. Figure 14.5.

1. Half-Adder

A Half-Adder can be fully described in terms of its Function table, its Sum and Carry Out Boolean Expressions and the circuit Implementation.

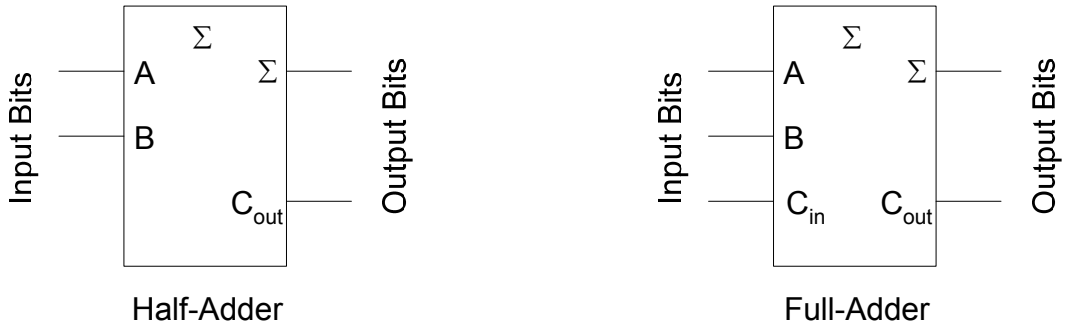


Figure 14.5 Block diagrams of Half-Adder and Full-Adder

Half-Adder Function Table

The Half-Adder has a 2-bit input and a 2-bit output. The function table of the Half-Adder has two input columns representing the two single bit numbers A and B. The function table also has two output columns representing the Sum bit and Carry Out bit. Table 14.3

Input		Output	
A	B	Sum	Carry Out
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Table 14.3 Half-Adder Function Table

Half-Adder Sum & Carry Out Boolean Expressions

The Sum and Carry Out expressions of the Half-Adder can be determined from the function table. The Half-Adder Sum and Carry Out outputs are defined by the expressions

$$Sum = \overline{A}B + A\overline{B} = A \oplus B$$
$$CarryOut = AB$$

Half-Adder Logic Circuit

The Half-Adder Logic Circuit can be directly implemented from the Sum and Carry Out Boolean expressions. Figure 14.6

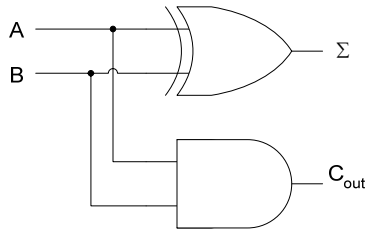


Figure 14.6 Half-Adder Logic Circuit

2. Full-Adder

A Full-Adder can be fully described in terms of its Function table, its Sum and Carry Out Boolean Expressions and the circuit Implementation.

Full-Adder Function Table

The Full-Adder has a 3-bit input and a 2-bit output. The function table of the Full-Adder has three input columns representing the two single bit numbers A, B and the Carry In bit. The function table also has two output columns representing the Sum bit and Carry Out bit. Table 14.4

Input			Output	
A	B	Carry In(C)	Sum	Carry Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 14.4 Full-Adder Function Table

Full-Adder Sum & Carry Out Boolean Expressions

The Sum and Carry Out expressions of the Full-Adder can be determined from the function table. The Full-Adder Sum and Carry Out outputs are defined by the expressions

$$Sum = \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC$$
$$Sum = \overline{A}(\overline{B}C + B\overline{C}) + A(\overline{B}\overline{C} + BC)$$
$$Sum = \overline{A}(B \oplus C) + A(\overline{B \oplus C})$$

$$Sum = A \oplus B \oplus C$$

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

Full-Adder Logic Circuit

The Full-Adder Logic Circuit can be directly implemented from the Sum and Carry Out Boolean expressions. Figure 14.7

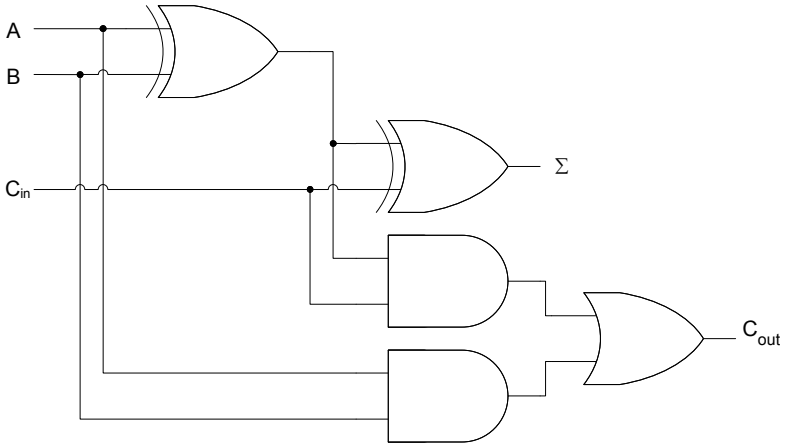


Figure 14.7 Full-Adder Logic Circuit

Forming a Full-Adder using Half-Adders

A 1-bit Full-Adder cane be implemented by combining together two Half-Adders. Figure 14.8

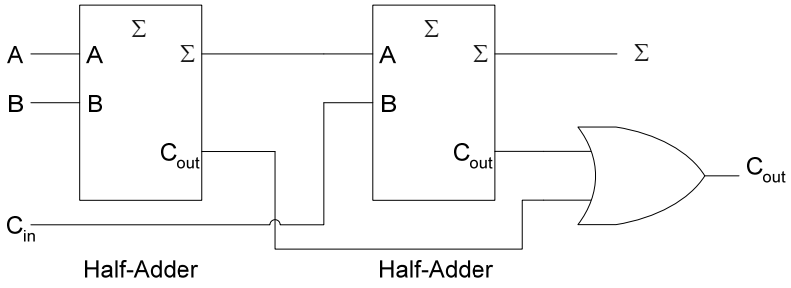


Figure14.8 Implementing a Full-Adder using two Half-Adders

- The Sum output of the first Half-Adder is $(A \oplus B)$
- The Carry Out of the first Half-Adder is AB
- The Sum output of the second Half-Adder is $(A \oplus B) \oplus C_{in} = (A \oplus B \oplus C_{in})$

- The Carry Out of the second Half-Adder is $(A \oplus B)C_{in}$
- The output of the OR gate is $AB + (A \oplus B)C_{in}$

Parallel Binary Adders

Single bit Full or Half Adders do not perform any useful function. To add two 4-bit numbers a 4-bit adder is required. Four single bit Full-Adders are connected together to form a 4-bit Parallel Adder capable of adding two 4-bit binary numbers. Figure 14.9.

The two 4-bit numbers A and B are applied at the circuit inputs A_{0-3} and B_{0-3} respectively. The 4-bit Sum output of the Parallel Adder is available at outputs S_{0-3}. The Carry In to the circuit is set to 0. (C_{in}=0). The Carry is available at C_{out}.

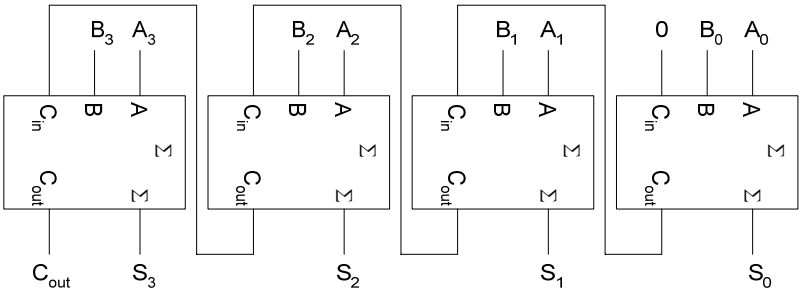


Figure 14.9 4-bit Parallel Binary Adder

Carry Propagation

Parallel Binary Adders can be implemented by connecting the required number of 1-bit full adders in a configuration represented in figure 14.9. However, there is a practical limitation to the number of 1-bit Full-Adders that can be connected in parallel. In the 4-bit Parallel Adder, the Most significant bit adder which adds bits A_3, B_3 and the Carry bit C_3, can not proceed until it receives the Carry from the next least significant 1-bit adder which adds bits A_2, B_2. The A_2 B_2 bit adder can not proceed unless it receives the carry input C_2 from the A_1, B_1 adder. The A_1, B_1 adder in tern depends on A_0, B_0 adder to provide the carry input. Thus the carry has to propagate through each Full-adder before it reaches the last or most significant full adder.

Assume that each gate has a propagation delay of 10 nsec. A 1-bit Full Adder generates a Carry out after 30 nsec. For a 4-bit Parallel Adder Full-adder the Carry out from the most significant adder would be after 120 nsec. The delay can increase to prohibitive levels if 8-bit, 16-bit or 64-bit parallel adders are implemented. 64-bit parallel adders are used by computers.

Look-Ahead Carry Circuits

To overcome the problem of carry propagation or carry ripple, Look-Ahead carry generator circuits are used. These circuits look at the bits to be added and decide if a higher order carry is to be generated. The Look-Ahead Carry Circuits although increase the circuitry but they provide a practical solution to the prohibitive delays that are caused by the ripple carry in parallel adders.

Consider the Full-Adder Circuit. 14.10. The output $(A \oplus B)$ at output P of the XOR gate and the output AB at output G of the AND gate is available simultaneously after one gate delay. If the G output of the AND gate is 1, the Carry Out has to be a 1 no matter what is the

other input of the Carry Out OR gate. The Sum and Carry Out can be expressed in terms of P and G gate outputs.

- The P output is called the Carry Propagate.
- The G output is called the Carry Generate

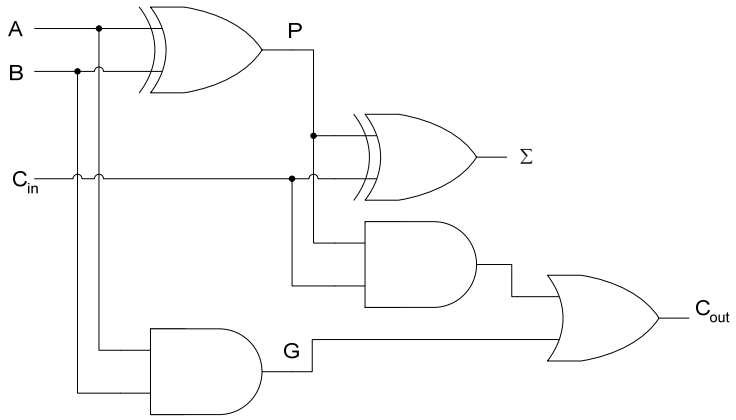


Figure 14.10 Full-Adder with Carry Generate and Carry Propagate

Carry Outputs in terms of Carry Propagate and Carry Generate

The Sum and Carry Out Boolean expressions can be rewritten in terms of P, Carry Propagate and G, Carry Generate terms.

$$\text{Sum} = P \oplus C$$
$$\text{CarryOut} = CP + G$$

Writing the expressions for the four Carry Out terms C_1, C_2, C_3 and C_4 in terms of Carry Propagate P and Carry Generate G.

- $C_1 = C_0P_0 + G_0$
- $C_2 = C_1P_1 + G_1 = P_1(C_0P_0 + G_0) + G_1 = G_1 + P_1G_0 + P_0P_1C_0$
- $C_3 = G_2 + P_2G_1 + P_1P_2G_0 + P_0P_1P_2C_0$
- $C_3 = C_2P_2 + G_2 = P_2(G_1 + P_1G_0 + P_0P_1C_0) + G_2 = G_2 + P_2G_1 + P_1P_2G_0 + P_0P_1P_2C_0$
- $C_4 = G_3 + P_3G_2 + P_2P_3G_1 + P_1P_2P_3G_0 + P_0P_1P_2P_3C_0$

where $P_n = A_n \oplus B_n$ and $G_n = A_nB_n$

The Look-Ahead Carry Generator Circuit is shown. Figure 14.11. The inputs to the Look-Ahead Carry Generator Circuit are the Carry Propagate terms P_0, P_1, P_2 and P_3 and Carry Generate terms G_0, G_1, G_2 and G_3.

The Carry Propagate P_0, P_1, P_2 and P_3 and Carry Generate terms G_1, G_2, G_3 and G_4 are generated by the XOR and AND gates after one gate delay.

The Outputs of the Look-Ahead Carry Generator Circuit are C_1, C_2, C_3 and C_4. The output C_1 is generated by the circuit represented by the expression $C_1 = C_0P_0 + G_0$ which

requires an AND gate to generate the product term C_0P_0 and a second level two input OR gate to sum the terms C_0P_0 and G_0. Thus C_1 is available after two gate delays.

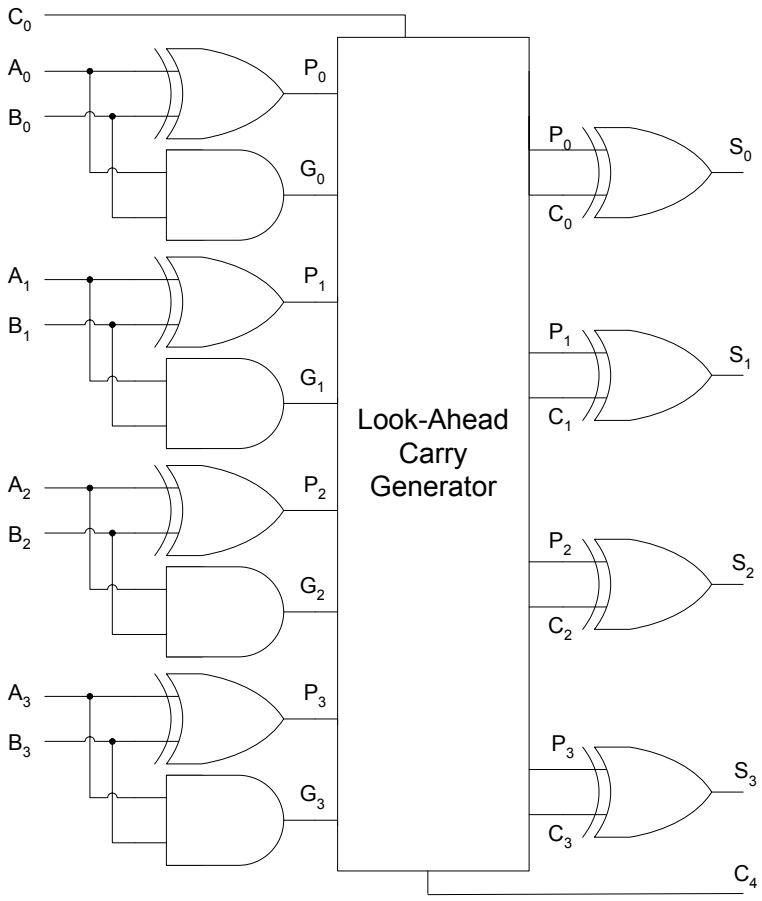


Figure 14.11 Look-Ahead Carry Generator

Similarly, the output C_2 is generated by the circuit represented by the expression $C_2 = G_1 + P_1G_0 + P_0P_1C_0$ which requires a 2-input and 3-input AND gates to generate the product terms P_1G_0 and $P_0P_1C_0$ respectively. A second level three input OR gate is required to sum the three terms. Thus C_2 is also available after two gate delays.

The output C_3 is generated by the circuit represented by the expression $C_3 = G_2 + P_2G_1 + P_1P_2G_0 + P_0P_1P_2C_0$. The expression is implemented by a combination of three AND gates having 2, 3 and 4 inputs respectively and a single 4-input OR gate. Again two levels of gates is used, C_3 is available after a gate delay of two.

Finally, the output C_4 is generated by the circuit represented by the expression $C_4 = G_3 + P_3G_2 + P_2P_3G_1 + P_1P_2P_3G_0 + P_0P_1P_2P_3C_0$. To implement the expression two levels of 2, 3, 4 and 5 input AND gates and a single 5 input OR gate is used. C_4 is available after a gate delay of two.

Thus for Carry outputs C_1, C_2, C_3 and C_4 the delay is of the order of two after the

MSI Adders

4-bit parallel Adders are available as Medium Scale Integrated Circuits. These circuits use the Look-Ahead Carry Circuitry to remove the carry ripple. The two ICS are 74LS83A and 74LS283. Both the devices are functionally identical, however they are not pin compatible. These devices are packaged as 16-pin devices. The division of the 16 pins is

- 4 pins for the 4-bit input A
- 4 pins for the 4-bit input B
- 4 pins for the 4-bit output Sum
- 1 pin for Carry In
- 1 pin for Carry Out
- 1 pin for Circuit Power Supply
- 1 pin for Circuit GND

The 74LS83A or the 74LS283 can be cascaded together to form 8-bit, 12-bit or 16-bit Parallel Adders. Figure 14.12 The Carry Out pin of one IC is connected to the Carry In pin of the other IC.

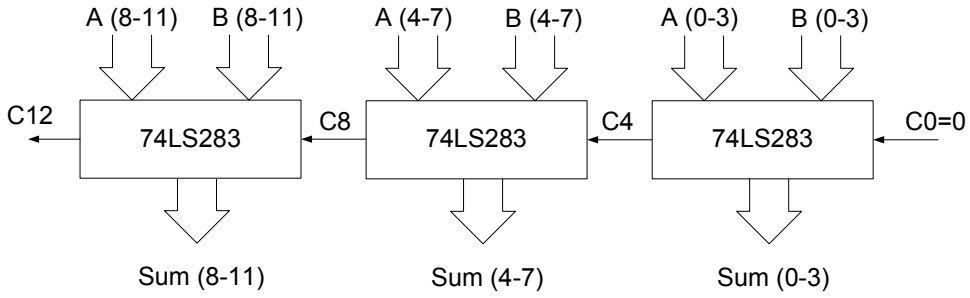


Figure 14.12 12-bit Parallel Adder using three 74LS283 ICs

Lesson No. 15

BCD ADDER

BCD binary numbers represent Decimal digits 0 to 9. A 4-bit BCD code is used to represent the ten numbers 0 to 9. Since the 4-bit Code allows 16 possibilities, therefore the first 10 4-bit combinations are considered to be valid BCD combinations. The latter six combinations are invalid and do not occur.

BCD Code has applications in Decimal Number display Systems such as Counters and Digital Clocks. BCD Numbers can be added together using BCD Addition. BCD Addition is similar to normal Binary Addition except for the case when sum of two BCD digits exceeds 9 or a Carry is generated. When the Sum of two BCD numbers exceeds 9 or a Carry is generated a 6 is added to convert the invalid number into a valid number. The carry generated by adding a 6 to the invalid BDC digit is passed on to the next BCD digit.

Addition of two BCD digits requires two 4-bit Parallel Adder Circuits. One 4-bit Parallel Adder adds the two BCD digits. A BCD Adder uses a circuit which checks the result at the output of the first adder circuit to determine if the result has exceeded 9 or a Carry has been generated. If the circuit determines any of the two error conditions the circuit adds a 6 to the original result using the second Adder circuit. The output of the second Adder gives the correct BCD output. If the circuit finds the result of the first Adder circuit to be a valid BCD number (between 0 and 9 and no Carry has been generated), the circuit adds a zero to the valid BCD result using the second Adder. The output of the second Adder gives the same result. Figure 15.1

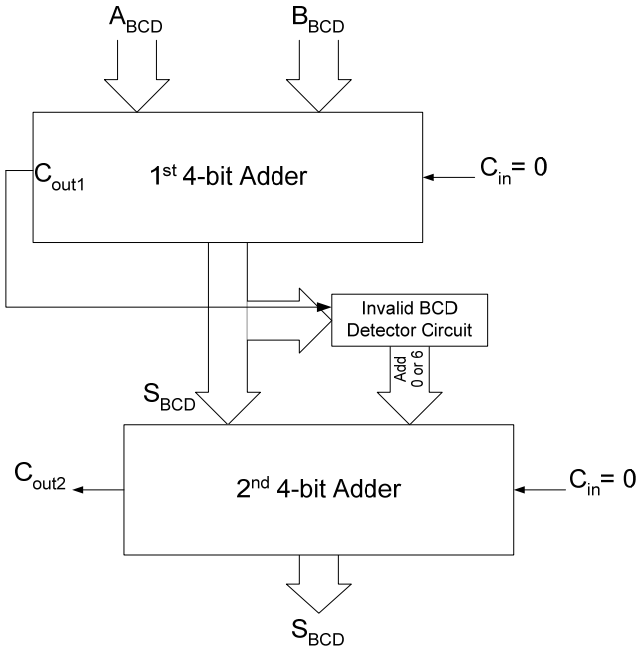


Figure 15.1 4-Bit BCD Adder

The circuit that checks if the output of the first Adder has exceeded 9 is a simple combinational circuit with the function table specified. Table 15.1

Input				Output	Input				Output
S_3	S_2	S_1	S_0	F	S_3	S_2	S_1	S_0	F
0	0	0	0	0	1	0	0	0	0
0	0	0	1	0	1	0	0	1	0
0	0	1	0	0	1	0	1	0	1
0	0	1	1	0	1	0	1	1	1
0	1	0	0	0	1	1	0	0	1
0	1	0	1	0	1	1	0	1	1
0	1	1	0	0	1	1	1	0	1
0	1	1	1	0	1	1	1	1	1

Table 15.1 Function Table of Invalid BCD Number detector

$S_3S_2 \backslash S_1S_0$	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

Figure 15.2 Mapping of Invalid BCD Number detector function

The Boolean expression for the Invalid BCD Number Detector obtained from the Karnaugh Map which maps the function table is $S_3S_2 + S_3S_1 = S_3(S_2 + S_1)$

The Invalid BCD Number is represented by two error conditions, either the BCD number is one of the invalid numbers or a Carry out has been generated. Therefore the complete expression for determining an incorrect BCD output is $C_{out1} + S_3(S_2 + S_1)$. Figure 15.3

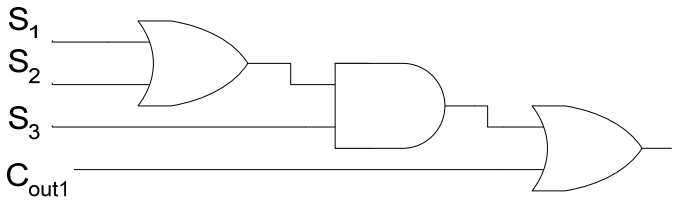


Figure 15.3 The Invalid BCD Detector Circuit

Connection of Invalid BCD Detector Circuit to second Adder

Adding of 6 when error conditions are detected and adding a zero when error conditions are not detected is implemented by connecting the output of the Invalid BCD Number Detector circuit to bits B_1 and B_2 of the Adder. Bits B_0 and B_3 are permanently connected to 0. Figure 15.4. When an error condition is detected the output of the circuit is set to logic 1, setting bits B_1 and B_2 to 1 and the 2nd Adder input B to 0110. When the error condition is not detected the circuit output is 0 and the 2nd Adder input B is set to 0000.

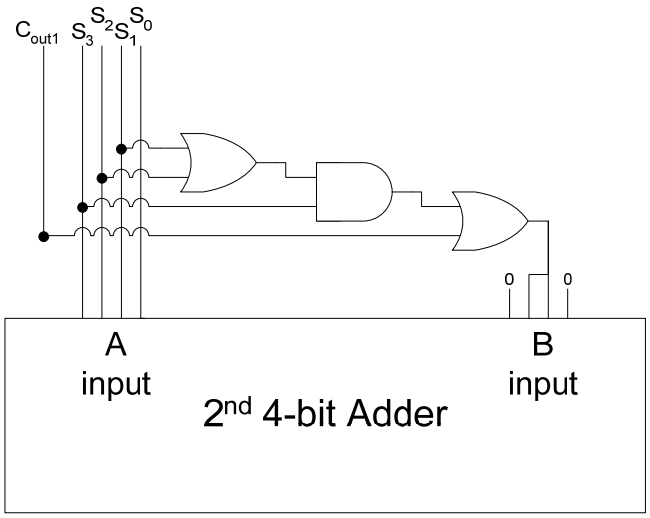


Figure 15.4 Using the Second Adder to Add 6 or 0

2-digit BCD Adder

Two singe digit BCD Adders can be cascaded together to form a 2-digit BCD Adder. Four, 4-bit 74LS283 MSI chips are used. Two 74LS283s are required to directly add the two 2-digit BCD numbers and the remaining two 74LS283s are required to add a six to the result if any of the two digits add up to invalid BCD digits or generate a Carry. Two invalid BCD detector circuits are used. Figure 15.5

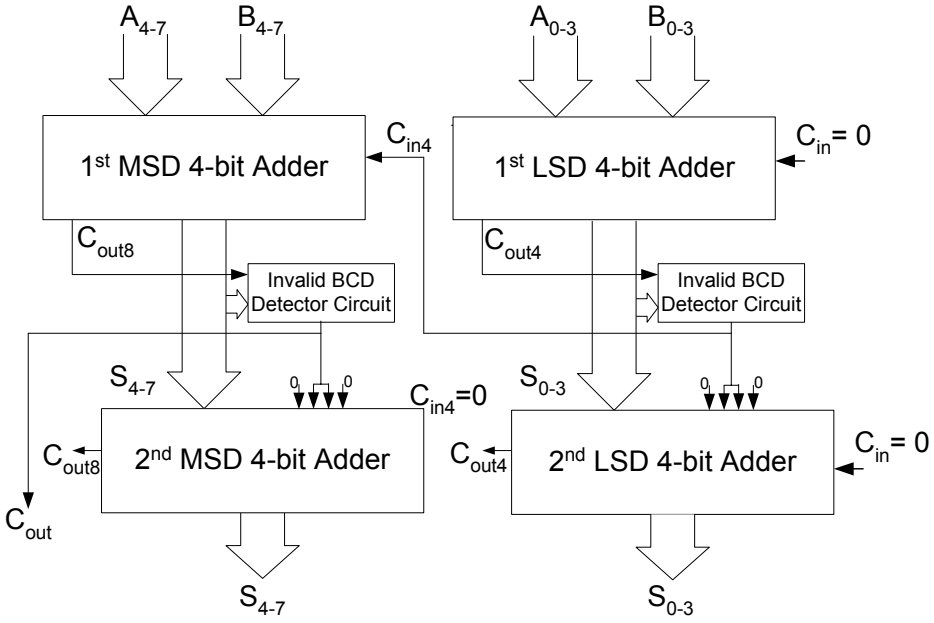


Figure 15.5 2-Digit BCD Adder

Consider two examples. In the first example, 2-digit BCD number 99 is added with another 2-digit BCD number 99. The answer should be 198 a 3-digit BCD number. Table 15.2. In the second example, 2-digit BCD number 99 is added with another 2-digit BCD number 66. The answer should be 165. Table 15.3

Carry	1st MSD Adder		1st LSD Adder	
	A(0-3)	1001	A(0-3)	1001
	B(0-3)	1001	B(0-3)	1001
	Cin4	1	Cin	0
	S(0-3)	0011	S(0-3)	0010
	Cout8	1	Cout4	1
	Ckt. o/p	1	Ckt. o/p	1
	2nd LSD Adder		2nd LSD Adder	
	A(0-3)	0011	A(0-3)	0010
	B(0-3)	0110	B(0-3)	0110
	Cin	0	Cin	0
1	S(0-3)	1001	S(0-3)	1000

Table 15.2 Adding BCD numbers 99 and 99

Carry	1st MSD Adder		1st LSD Adder	
	A(0-3)	1001	A(0-3)	1001
	B(0-3)	0110	B(0-3)	0110
	Cin4	1	Cin	0
	S(0-3)	0000	S(0-3)	1111
	Cout8	1	Cout4	0
	Ckt. o/p	1	Ckt. o/p	1
	2nd LSD Adder		2nd LSD Adder	
	A(0-3)	0000	A(0-3)	1111
	B(0-3)	0110	B(0-3)	0110
	Cin	0	Cin	0
1	S(0-3)	0110	S(0-3)	0101

Table 15.3 Adding BCD numbers 99 and 66

Subtraction

Subtraction in Digital Systems is performed by taking the 2's complement of the number to be subtracted (subtrahend) and adding it to the minuend. The example shows the subtraction of 6 represented in 2's complement form from nine also represented in its 2's complement form. Since 9 is a positive number therefore its 2's complement representation is the same. Neglecting the carry bit, the 4-bit number represents decimal 4.

```
    9          1001
-   5          1011
    4       1 0100
```

The 2's complement of any number is obtained by taking the 1's complement of a number and then adding a 1 to the 1's complement. The two step process to represent a negative number in its 2's complement form is shown

Invert all bits to result in 1's complement

1's complement of 5 is	1010
	+ 1
2's complement of 5 is	1011

An Adder can be used to perform subtraction operations if the minuend is presented in its 1's complemented form at the input of the adder circuit. The binary 1 that is added to the 1's complement of a number to convert it into 2's complement is applied at the Carry In of the Adder Circuit. Figure 15.6

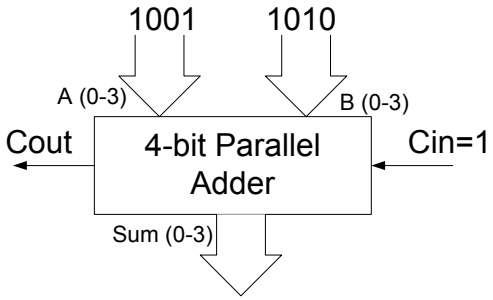


Figure 15.6 4-bit Subtraction Circuit

The Adder circuit adds the number 9 (1001), 1's complement of 5 (1010) and the Carry In which is set to 1.

A 4-bit Adder/Subtracter Unit

An Adder can be connected to perform Additions and Subtractions by applying the un-complemented and complemented data at one of the two inputs of the Adder respectively. The Carry In input has also to be connected to 0 or 1 respectively. Figure 15.7

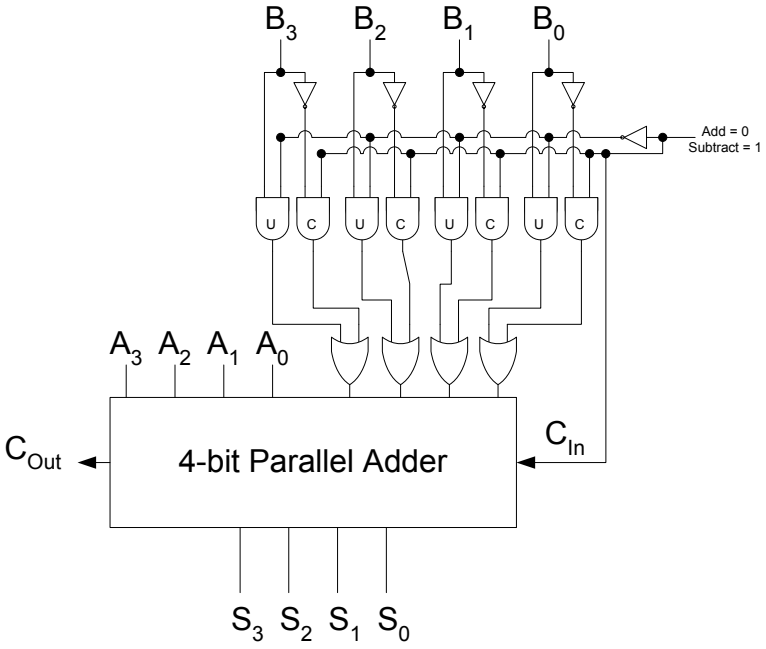


Figure 25.7 4-bit Adder/Subtracter Unit

The AND gate and OR gate implementation connected at the B input of the 4-bit Adder is used to allow Complemented or Un-Complemented B input to be connected to the Adder input. Adding of two 4-bit numbers A and B can be performed by selecting the Add/Subtract = 0. The AND gates marked U (un-complemented) are enabled allowing B_{0-3} to be passed on to the OR gates and the B input of the Adder. Subtraction is performed by selecting the Add/Subtract = 1. The AND gates marked C (complemented) are enabled allowing complemented B_{0-3} to be passed on to the OR gates and the B input of the Adder. The Carry In is also set to 1 when Add/Subtract is set to 1.

An 8-bit Adder/Subtracter Unit

Two 4-bit 74LS283 chips can be cascaded together to form an 8-bit Parallel Adder Unit. Each of the two 74LS283 ICs is connected to the 1's Complement circuitry that allows either the un-complemented form for addition or the complemented form for subtraction to be applied at the B inputs of the two 74LS283s. Figure 15.8

The 8-bit Adder/Subtracter Circuit is similar to the 4-bit Adder/Subtracter Circuit. Two sets of AND-OR based circuit that allows complemented and un-complemented B input to be applied at the B inputs of the two 4-bit Adders. The Add/Subtract function select input are tied together. The Carry In of the 1st 4-bit Adder circuit is connected to the Add/Subtract function select input. The Carry Out of the 1st 4-bit Adder circuit is connected to the Carry In of the 2nd 4-bit Adder circuit.

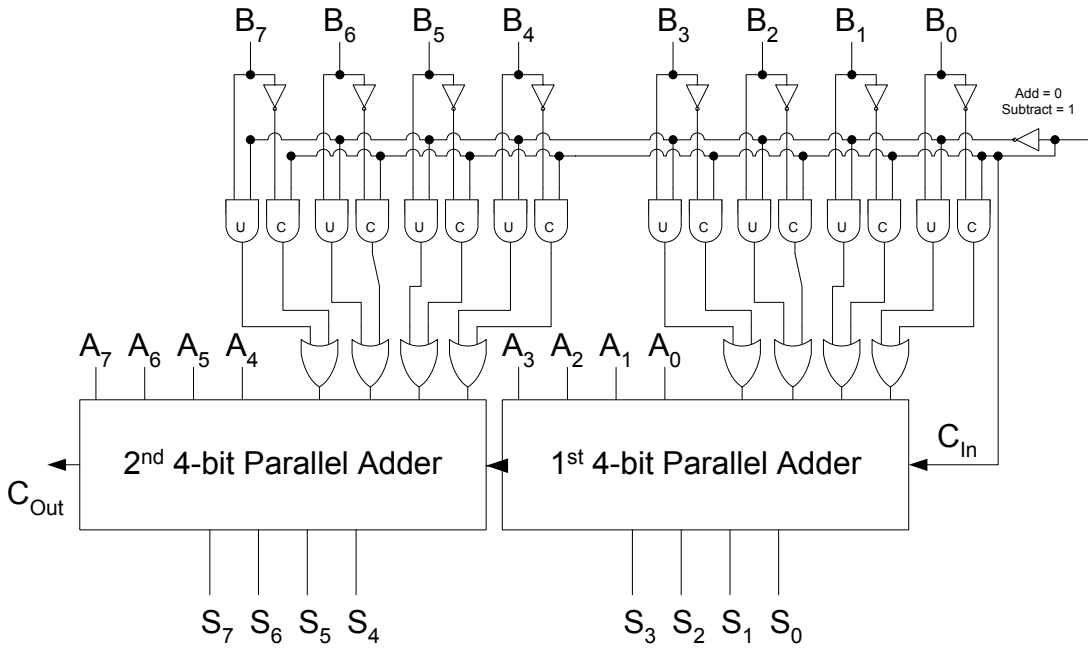


Figure 15.8 8-bit Adder/Subtracter Circuit

Consider two number A=103 and B=67 which are first added and then subtracted using the 8-bit Adder/Subtracter Circuit. Table 15.4 and Table 15.5

Adding 103 and 67				
Carry	2nd MS Adder		1st LS Adder	
	A(4-7)	0110	A(0-3)	0111
	B(4-7)	0100	B(0-3)	0011
	Cin	0	Cin	0
0	S(4-7)	1010	S(0-3)	1010

Table 15.4 Adding 103 and 67

Subtracting 103 and 67				
Carry	2nd MS Adder		1st LS Adder	
	A(4-7)	0110	A(0-3)	0111
	B(4-7)	1011	B(0-3)	1100
	Cin	1	Cin	1
1	S(4-7)	0010	S(0-3)	0100

Table 15.5 Subtracting 103 and 67

Arithmetic and Logic Unit (ALU)

Microprocessors have Arithmetic and Logic Units, a combinational circuit that can perform any of the arithmetic operations and logic operations on two input values. The operation to be performed is selected by set of inputs known as function select inputs.

There are different MSI ALUs available that have two 4-bit inputs a 4-bit output and three to five function select inputs that allows up to 32 different functions to be performed. Three commercially available 4-bit ALUS are

- 74XX181: The 4-bit ALU has five function select inputs allowing it to perform 32 different Arithmetic and Logic operations.
- 74XX381: The 4-bit ALU only has three function select inputs allowing only 8 different arithmetic and logic functions. Table 15.6
- 74XX382: The 4-bit ALU is similar to the 74XX381, the only difference is that 74XX 381 provides group-carry look-ahead outputs and 74XX382 provides ripple carry and overflow outputs

Input			
S2	S1	S0	Function
0	0	0	F=0000
0	0	1	$F=B-A-1+C_{in}$
0	1	0	$F=A-B-1+C_{in}$
0	1	1	$F=A+B+C_{in}$
1	0	0	$F = A \oplus B$
1	0	1	F=A+B
1	1	0	F=A.B

Table 15.6 Function Table of 74XX381 4-bit ALU

Implementing 16-bit ALU

16-bit ALU can be implemented by cascading together four 4-bit ALUs. These 4-bit ALUs have built in Look-Ahead Carry Generator circuits that eliminate the delay caused by carry bit propagating through the Parallel Adder circuit within the 4-bit ALU circut. However, when a number of such units are cascaded together to implement large 16-bit and 32-bit ALU, the carry propagating between one unit to the next gets delayed due to the Carry rippling through multiple 4-bit units. For large 32-bit ALUs, the Carry propagates through 8, 4-bit units delaying the Carry out from the last most significant unit by a factor of 8.The 74XX181 and 74XX381 circumvent the problem by having Group-Carry Look-Ahead.

Group-Carry Look-Ahead

The Look-Ahead Carry Generator discussed earlier and used by the 74LS283 Adder provides Carry's C_1, C_2, C_3 and C_4 simultaneously after a gate delay of two. Carry's C_1, C_2 and C_3 are used internally, where as C_4 provides the C_{out} from the 74LS283. Referring to the Look-Ahead Carry Generator Circuit the C_1, C_2, C_3 and C_4 terms are generated on the basis of P_0, P_1, P_2 and P_3 the four Carry Propagate terms and G_0, G_1, G_2 and G_3 the four Carry Generate terms. Figure 15.9

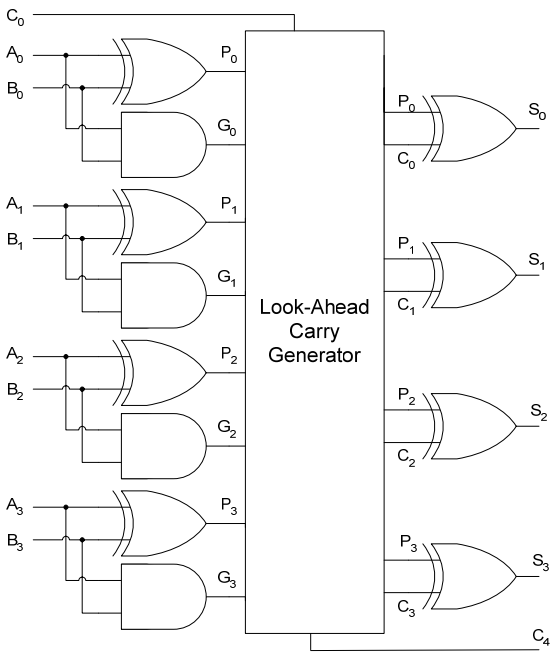
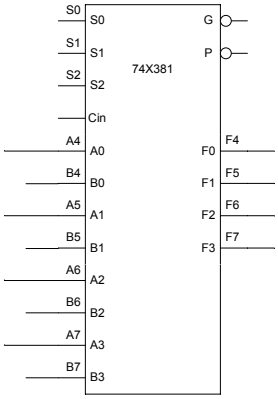


Figure 15.9 Look-Ahead Carry Generator

These terms are used to generate Group-Carry Look-Ahead outputs that can be used to cascade together multiple units eliminating the problem of rippling carry. The G and P output pins of the 74XX381 provide the group-carry look-ahead outputs that allow multiple ALUs to be cascaded together. The active-low outputs G and P are represented by the Boolean expressions. Figure 15.10



$$G = \overline{G_3 + P_3G_2 + P_2P_3G_1 + P_1P_2P_3G_0} \quad P = \overline{P_oP_1P_2P_3}$$

Figure 15.10 74X381 ALU with Group-Carry Look-Ahead outputs

Lesson No. 16

16-BIT ALU

Consider the four ALUs connected to form a 16-bit ALU without the Look-Ahead Carry circuit. Figure 16.1. The ALU1 will only generate an output and a Carry Out 8 when it has received an input at Carry In 4. Similarly, ALU2 will only generate an output and a Carry Out 12 when it has received Carry In 8. Finally, the Carry Out 16 is generated only when ALU3 has received Carry In 12. Thus the Carry instead of rippling through the 4-bits of the individual ALU circuit has to propagate through four ALU units. The last ALU unit has to wait until it receives the Carry propagating through each of the three units.

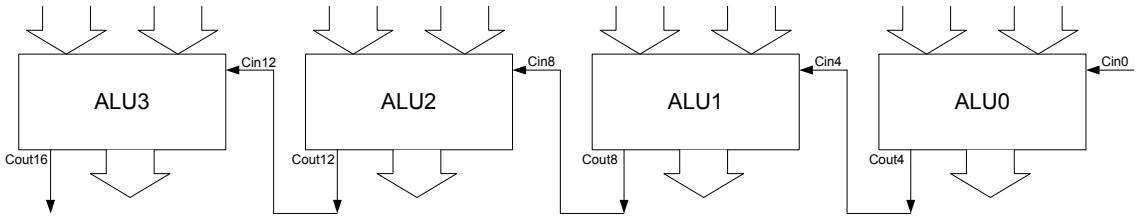


Figure 16.1 Carry Propagation Delay between 4-bit ALU units

The delay caused by the Carry Propagating through the four units is eliminated by the Group Carry terms used by the 381 ALUs. Figure 16.2. Instead of the Carry Out each ALU generates Group-Carry Generate and Propagate terms, which indicate if the most significant Carry is generated by the 4-bit ALU or otherwise. The Group Carry terms are connected to the Look-Ahead Carry Generator which generates the Cary bits C1, C2 and C3 which are connected to Cin4, Cin8 and Cin12 respectively. Thus Carry no longer propagates through the ALU units.

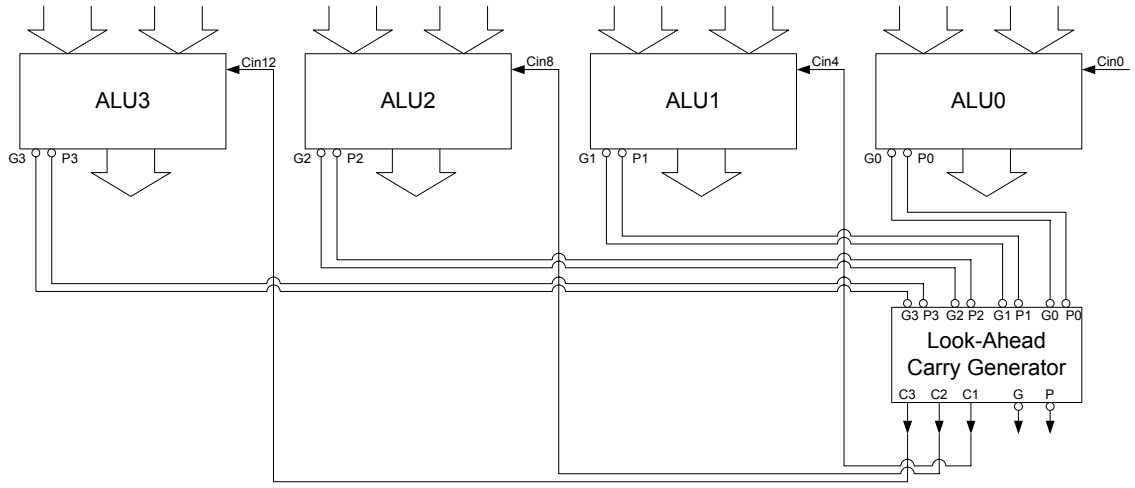


Figure 16.2 Carry Propagation Delay eliminated by using Group Carry

The G output is activated if the 4-bit unit generates a Carry Out irrespective of Carry In. The P output is activated if the 4-bit unit generates a Carry Out if the Carry In is active. The Look-Ahead circuit implemented earlier is based on Logic Gates, where the Look-Ahead Carry

Generator circuit has P_0, P_1, P_2 and P_3 Carry Propagate and G_0, G_1, G_2 and G_3 Carry Propagate Inputs and C_1, C_2, C_3 and C_4 Carry Out outputs. The 74XX182 is the MSI version of the Look-Ahead Carry Generator, which provides identical inputs and outputs except for the C_4 output which is available in the form of P and G output pins to allow a second level Cascading. The connection of four 74XX381 4-bit ALUs and a 74XX182 to implement a 16-bit ALU is shown. Figure 16.3

The inputs A, B and the output F of the four, 4-bit ALUs 0, 1, 2 and 3 are connected to appropriate bits of the 16-bit inputs A, B and output F respectively. Thus bits A(0-3), B(0-3) and F(0-3) are connected to inputs and output of ALU0, bits A(4-7), B(4-7) and F(4-7) are connected to inputs and output of ALU1, bits A(8-11), B(8-11) and F(8-11) are connected to inputs and output of ALU2 and bits A(12-15), B(12-15) and F(12-15) are connected to inputs and output of ALU3. The Group-Carry Generate and Propagate outputs of the four ALUs are connected to the inputs of Look-Ahead Carry generator 74X182 respectively. The Carry outputs C1, C2 and C3 from the Look-Ahead Carry generator circuit are generated after a gate delay of 2 and are connected to the Carry in pins of ALUS 1, 2 and 3 respectively.

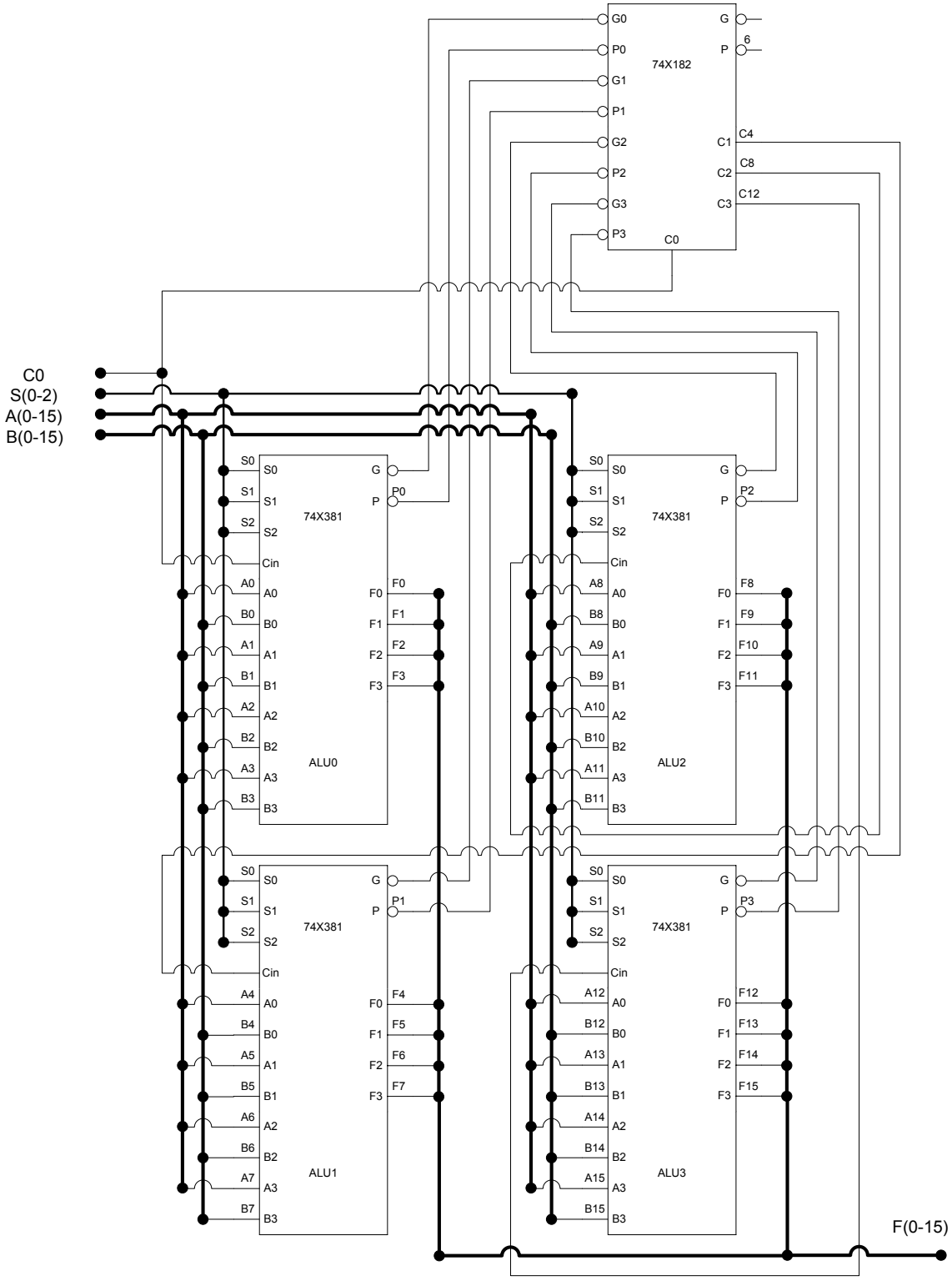


Figure 16.3 16-bit ALU

Comparators

The basic function of a Comparator is to compare two binary quantities and to determine if the two quantities are equal. If the quantities are not equal then it has to determine which of the two quantities is greater than the other. Many Integrated Circuit Comparators have three outputs to indicate A=B, A>B and A<B.

Earlier, simplified Boolean expressions for a 2-bit Comparator circuit were determined that compares two 2-bit numbers and sets one of its three outputs to indicate A=B, A>B or A<B. The Booleans expressions representing the three outputs are presented. The three Combinational Circuits implementing the three outputs are also shown. Figure 16.4

$$(A > B) = A_1\overline{B_1} + A_0\overline{B_1}\,\overline{B_0} + A_1A_0\overline{B_0}$$

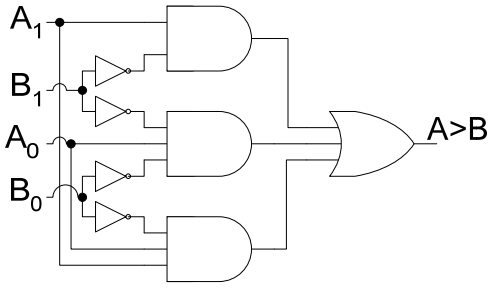


Figure 16.4a Implementation of A>B

$$(A = B) = \overline{A_1}\,\overline{A_0}\,\overline{B_1}\,\overline{B_0} + \overline{A_1}A_0\overline{B_1}B_0 + A_1A_0B_1B_0 + A_1\overline{A_0}B_1\overline{B_0}$$

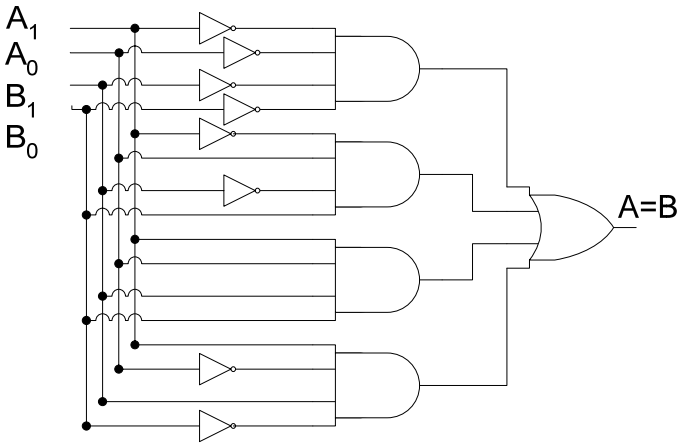


Figure 16.4b Implementation of A=B

$$(A < B) = \overline{A_1}B_1 + \overline{A_1}\,\overline{A_0}B_0 + \overline{A_0}B_1B_0$$

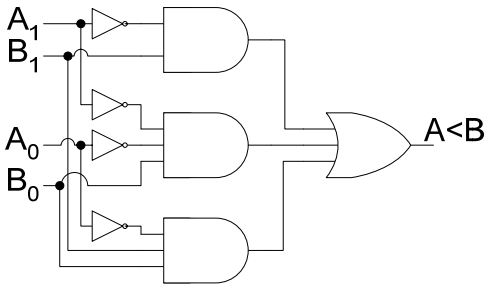


Figure 16.4c Implementation of A<B

The 2-bit Comparator discussed earlier is considered to be a Parallel Comparator as all the bits are compared simultaneously. External Logic has to be used to Cascade together two such Comparators to form a 4-bit Comparator.

The 4-bit numbers compared by the Cascaded implementation are represented in table 16.1.

A	B	Comparator M	Comparator L
1101	0111	A>B	
0110	1011	A<B	
0011	0010	A=B	A>B
0100	0101	A=B	A<B
1001	1001	A=B	A=B

Table 16.1 Comparison of numbers by Cascaded 4-bit Comparator

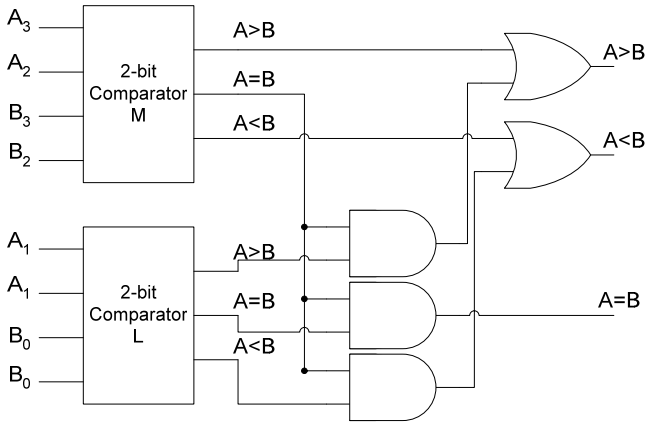


Figure 36.5 Implementation of 4-bit Comparator by Cascading two 2-bit Comparators

The two most significant bits of 4-bit numbers A and B are compared by the Most Significant 2-bit Comparator M and the least significant two bits are compared by the Least Significant 2-bit Comparator L. Figure 16.5 If the two most significant bits of number A are greater than the two most significant bits of number B, (A=1101 and B =0111) the Most Significant Comparator indicates A>B and there is no need to compare the remaining two least significant bits. Similarly, if the two most significant bits of numbers A and B (A=0110 and

then there is no need to compare the remaining two least significant bits. However, if the two most significant bits of numbers A and B indicates A=B then least significant two bits have to be compared to determine if A>B (A=0011 B=0010), A<B (A=0100 B=0101) or A=B (A=1001 B=1001). Thus the A=B output of the Most Significant 2-bit Comparator is used to enable three AND gates. The output of only one AND gate is set to 1 depending upon the output of the Least Significant 2-bit Comparator.

An alternate method of implementing Comparators which allows the Comparators to be easily cascaded without the need for extra logic gates by Iterative Circuit based Comparators.

Iterative Circuit based Comparator

An Iterative circuit is implemented using identical modules each of which has Primary Inputs and Outputs and Cascading Inputs and Outputs. The Cascading inputs of the least significant module are connected to fixed logic inputs and the Cascading outputs are connected to the Cascading inputs of the next significant module. A 2-bit Iterative Circuit based Comparator is shown. Figure 16.6.

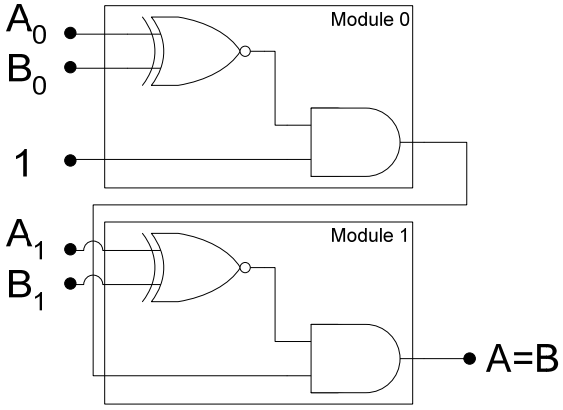


Figure 16.6a Iterative Circuit Implementation of A=B function

The Cascading input of Module 0 is connected to logic 1. If input A_0 is equal to input B_0, the XNOR gate output in Module 0 is a 1 which is passed on to Module 1 through its Cascading input. The output A=B is 1 when input A_1 is equal to B_1. If either $A_0 \neq B_0$ or $A_1 \neq B_1$ the output A=B is 0. The Equality Comparing circuit can be expanded to 4-bits by Cascading two Modules connecting their respective Cascading inputs and outputs.

In the Iterative Circuit for A>B, the Cascading input of Module 0 is connected to Logic 0. The output of Module 0 is 1 when $A_0>B_0$. The Cascading output of Module 0 is connected to the Cascading input of Module 1. The output A>B of Module 1 is 1 if $A_1=B_1$ and Cascading input is 1, or if $A_1>B_1$.

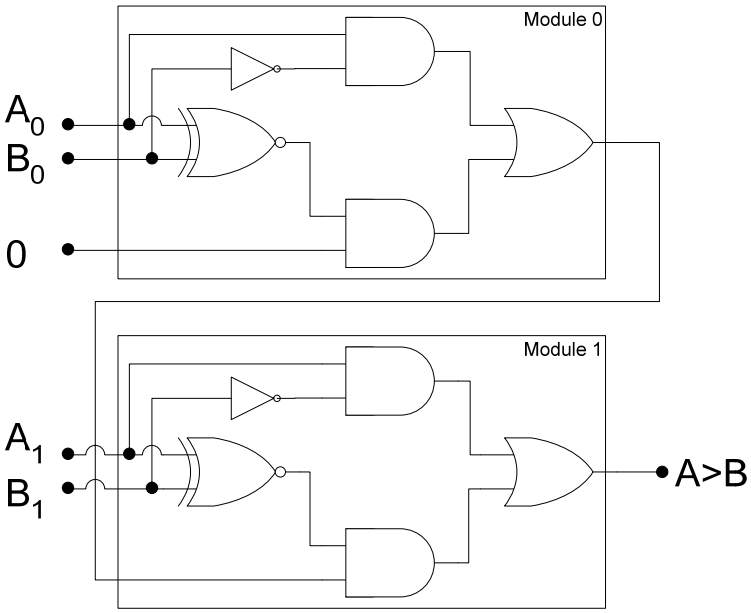


Figure 16.6b Iterative Circuit Implementation of A>B function

Similar Iterative Circuit for A<B, allows multiple modules to be Cascaded together to form multi-bit A<B unit.

MSI 4-bit Comparator

MSI 74HC85 4-bit Iterative Circuit based Comparator allows multiple 74HC85s to be cascaded together to form Comparators N x 4-bit Comparators. Three 74HC85s cascaded together forms a 12-bit Comparator circuit. Figure 16.7.

Three Comparators are cascaded together. Comparator 1 compares the least significant bits 0 to 3, Comparator 2 compares bits 4 to 7 and Comparator 3 compares the most significant bits 8 to 11. The respective input bits are shown connected to the three comparators through thick lines. The Cascading inputs of Comparator 1 are permanently connected to Ground and +5 volts. A<B in and A>B in are connected to ground and A=B in is connected to +5 Volts. The cascading outputs of Comparator 1 are connected to the respective cascading inputs of comparator 2. Similarly, the cascading outputs of Comparator 1 are connected to the cascading inputs of Comparator 3. The final output of the 12-bit Comparator circuit is available at the cascading outputs of Comparator 3.

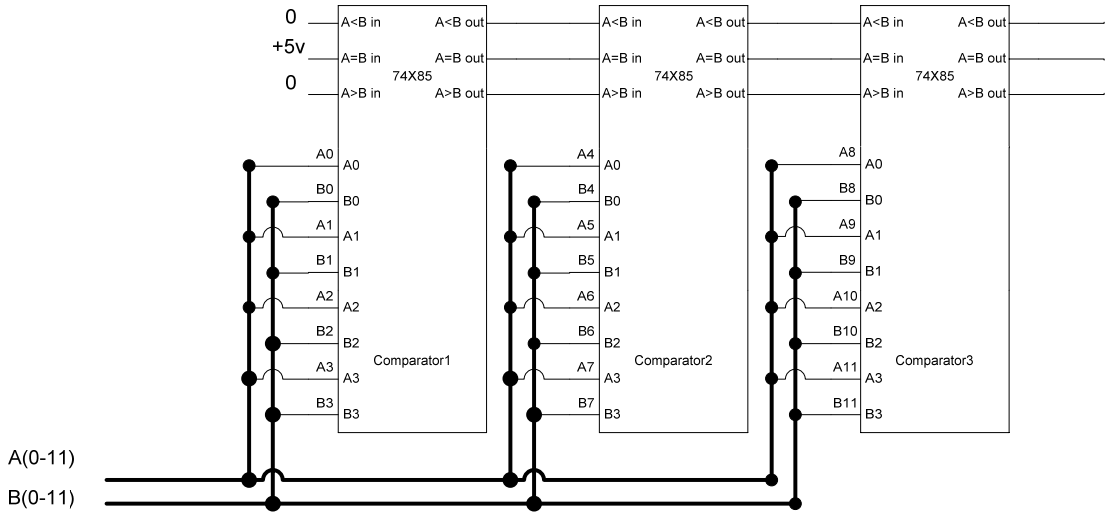


Figure 16.7 12-bit Comparator

Decoders

A Decoder has multiple inputs and multiple outputs. The Decoder device accepts as an input a multi-bit code and activates one or more of its outputs to indicate the presence of the multi-bit code. There are different variations of Decoder devices.

Basic Decoder

Consider an electronic door lock which unlocks the door when a 4-bit code 1011 is entered. The door is locked when another 4-bit combination 1001 is entered. The lock and unlock circuit is implemented using a combination of NOT and AND gates. Figure 16.8

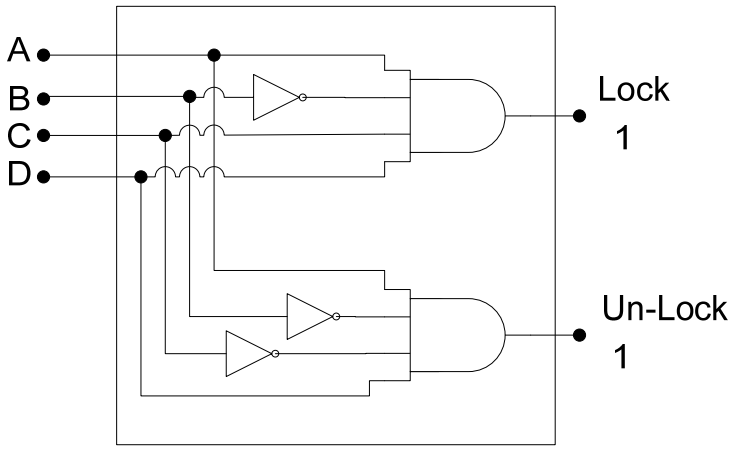


Figure 16.8 Electronic Door Lock

The circuit is configured to activate the Lock output when the Door Lock code 1011 is applied at inputs ABCD. The Un-Lock output is activated when the Door Un-Lock code 1001 is applied at the inputs ABCD. The circuit is a Decoder circuit. It detects the Code 1011 and activates the Lock output. Similarly, it detects the 1001 code and activates the Un-Lock output.

The decoder circuit can be expanded to have more Lock and Un-Lock outputs to Lock and Un-Lock different doors in a building.

Applications of decoders

Decoders have two major uses in Computer Systems.

1. Selection of Peripheral Devices

Computers have different internal and external devices like the Hard Disk, CD Drive, Modem, Printer etc. Each of these different devices is selected by specifying different codes. A decoder similar to the Electronic Door Lock/Unlock circuit is used to uniquely select or deselect the appropriate devices.

2. Instruction Decoder

Computer programs are based on instructions which are decode by the Computer Hardware and implemented. The codes 1100010, 1100011, 1110000 and 1000101 represent Add two numbers, Subtract two numbers, Clear the result and Store the result instructions. These instruction codes are decoded by an Instruction Decoder to generate signals that control different logic circuits like the ALU and memory to perform these operations.

Binary Decoder

The simplest and most commonly used Decoders are the n-to-2^n Decoders. These Decoders have n inputs and 2^n outputs Each, n-bit input selects 1 out of 2^n output code.

A 2-to-4 Decoder is represented by the function table. Table16.2. The 2-to-4 Binary Decoder circuit is shown. Figure 16.9

Input		Output			
I_1	I_0	O_0	O_1	O_2	O_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Table 16.2 Function Table of a 2-to-4 Binary Decoder

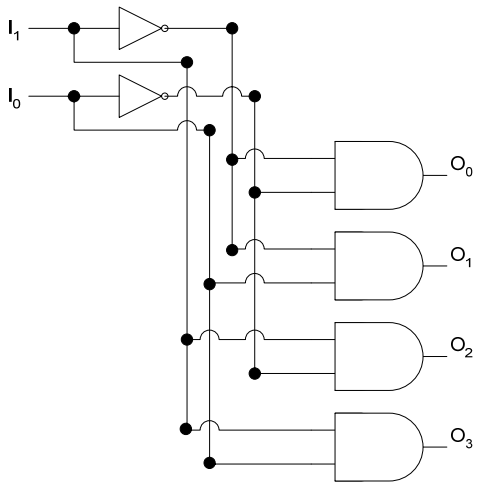


Figure 16.9 2-to-4 Decoder

The 2 to 4 Decoder output O_0 is activated to Logic 1 when the input is 00. Similarly for inputs 01, 10 and 11 the outputs O_1, O_2 and O_3 are respectively activated.

MSI Decoder

The 74LS139 MSI chip has dual 2-to-4 Decoders. The function table, table 16.3, and the gate–level circuit diagram for the 2-to-4 Decoder is shown. Figure16.10.The circuit diagram is slightly different form the one described in figure 16.9.

Input			Output			
$\overline{G}$	B	A	$\overline{Y3}$	$\overline{Y2}$	$\overline{Y1}$	$\overline{Y0}$
1	X	X	1	1	1	1
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1

Table 16.3 Function Table of 74LS139, 2-to-4 Decoder

The 74LS139 has active-low outputs, thus the output which is activated is at logic 0 where as the outputs that ate are not selected are at logic 1. A third active-low input G is the enable input, which when set to 0 enables all NAND gates. Setting the G input to 1 disables all NAND gates and all four outputs are at logic 1 the in-active state.

Extra NOT gates are placed at the inputs A and B. Without the two extra NOT gates at Inputs A and B, each of the two inputs present a unit load of three (a NOT gate and two NAND gates). By having the extra NOT gates each input presents a single unit load.

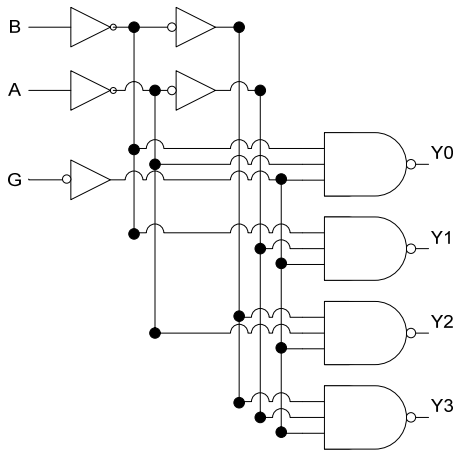


Figure 16.10 74LS139, 2-to-4 Decoder

Lesson No. 17

THE 74XX138 3-TO-8 DECODER

The 3-to-8, 74XX138 Decoder is also commonly used in logical circuits. Similar, to the 2-to-4 Decoder, the 3-to-8 Decoder has active-low outputs and three extra NOT gates connected at the three inputs to reduce the four unit load to a single unit load. The 3-to-8 Decoder has three enable inputs, one of the three enable inputs is active-high and the remaining two are active-low. All three enable inputs have to be activated for the Decoder to work. The function table of the 3-to-8 decoder is presented. Table 17.1

Inputs						Outputs							
G1	$\overline{G2A}$	$\overline{G2B}$	C	B	A	$\overline{Y7}$	$\overline{Y6}$	$\overline{Y5}$	$\overline{Y4}$	$\overline{Y3}$	$\overline{Y2}$	$\overline{Y1}$	$\overline{Y0}$
0	X	X	X	X	X	1	1	1	1	1	1	1	1
X	1	X	X	X	X	1	1	1	1	1	1	1	1
X	X	1	X	X	X	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	1	1	1	1	1	0
1	0	0	0	0	1	1	1	1	1	1	1	0	1
1	0	0	0	1	0	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	0	0	1	1	1	0	1	1	1	1
1	0	0	1	0	1	1	1	0	1	1	1	1	1
1	0	0	1	1	0	1	0	1	1	1	1	1	1
1	0	0	1	1	1	0	1	1	1	1	1	1	1

Table 17.1 Function Table of 74LS138, 3-to-8 Decoder

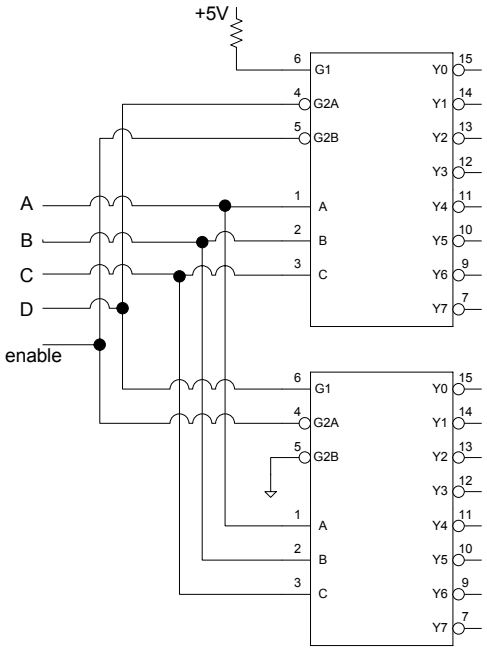


Figure 17.1 4-to-16 Decoder using two 74LS139, 3-to-8 Decoder

The three enable inputs serve to implement to larger Decoders such as 4-to-16 and 5-to-32 by cascading two or four 3-to-8 Decoders respectively. The connection of two 3-to-8 Decoders is shown. Figure 17.1

The A, B and C inputs are connected directly to the A, B and C inputs of the two 3-to-8 Decoders. The D input is connected to the active-low and active-high enable inputs G2A and G1 of the two decoders respectively. The enable input selects/deselects bothe the decoders simultaneously. G1 and G2B enable inputs of the two Decoders are connected to +5v and Ground respectively. When the D input is 0, the upper decoder is selected and when D input is 1, the lower decoder is selected. The A, B and C inputs serve to select the appropriate output of either the upper or lower decoder.

Implementing Standard SOP and POS Boolean expressions

The function table of 3-to-8 Decoder is a table of maxterms. For example, when the input A, B, C is 0, 0 and 0 the Y0 output is activated indicating the sum term or maxterm $A+B+C$. Similarly, the A, B and C inputs 1, 0 and 1 activate the Y5 output indicating the presence of $\overline{A}+B+\overline{C}$ sum term. The POS Boolean expression represented by the 3-variable Karnaugh Map, figure 17.2, can be implemented by the 3-to-8 Decoder which uses an AND gate to implement the product of sum terms. Figure 17.3

A\BC	00	01	11	10
0	0	0	0	1
1	1	0	0	1

Figure 17.2 Karnaugh Map of Boolean expression $\prod_{ABC}(0,1,3,5,7)$

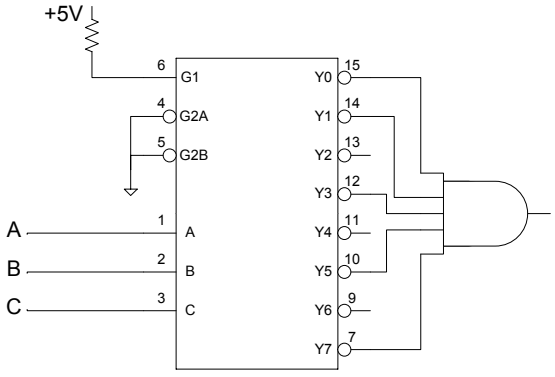


Figure 17.3 Implementation of Boolean expression $\prod_{ABC}(0,1,3,5,7)$

The 3-to-8 Decoder can also be used to Implement SOP expression by connecting the outputs of the Decoder to the input of a NAND gate. Figure 17.4. The alternate symbol for the three input NAND gate is the three input OR gate with bubbles at the inputs. The three bubbles cancel out the three bubbles connected at the outputs Y2, Y4 and Y6 representing the three minterms or product terms.

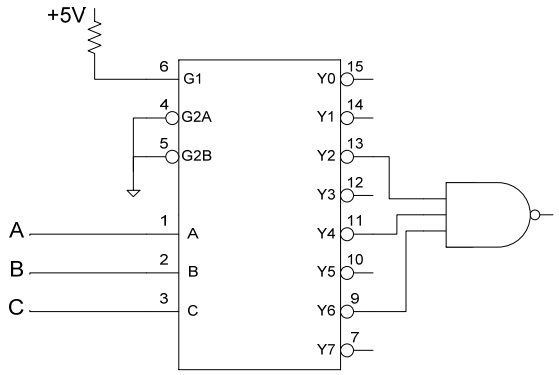
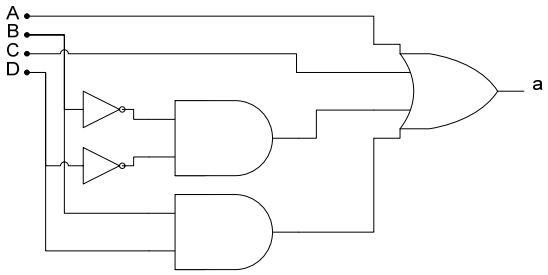


Figure 17.4 Implementation of Boolean expression $\sum_{ABC}(2,4,6)$

BCD to 7-Segment Decoder

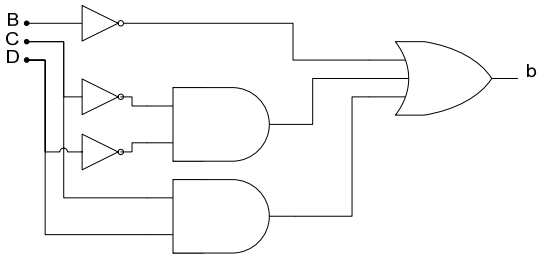
BCD to 7-Segmnet Decoder is a specific type of decoder that is used to convert a 4-bit BCD Code to a 7-Segment Code. The BCD to 7-Segment Decoder unlike the Binary Decoders activates multiple but unique set of outputs for each 4-bit BCD input combination.

Earlier, the seven expressions for activating each of the seven segments were defined. Each of the seven Boolean expressions can be implemented using a combination of NOT-AND-OR gates. The implementations for segments a, b and g are shown. Figure 17.5a-c



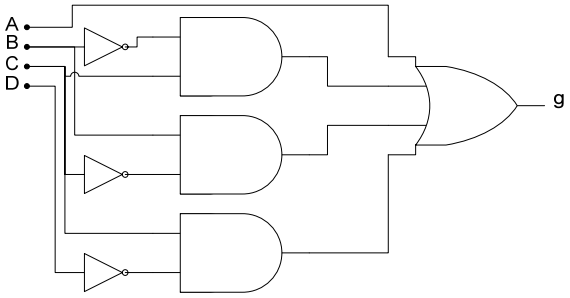
$$a = A + C + BD + \overline{B}\,\overline{D}$$

Figure 17.5a Implementation of Segment a output



$$b = \overline{B} + \overline{C}\,\overline{D} + CD$$

Figure 17.5b Implementation of Segment b output



$$g = A + B\overline{C} + C\overline{D} + \overline{B}C$$

Figure 17.5c Implementation of Segment g output

MSI Seven-Segment Decoder

The 7-Segment Decoder circuit is available in MSI form, 74LS47. The IC has 4-bit BCD input ABCD and 7-bit active-low outputs for segments a, b, c, d, e, f and g. The Decoder also has three extra active-low inputs.

- LT: Lamp test
- RBI: Ripple Blanking Input
- BI/RBO: Blanking Input/Ripple Blanking Output

When a low is applied to the LT input and the BI/RBO is high, all of the seven segments in the display are turned on to test that no segments are burned out. The Ripple Blanking Input and The Blanking Input/Ripple Blanking Outputs are used to prevent display of leading and trailing zeros.

BCD-to-Decimal Decoder

The operation of the BCD-to-Decimal Decoder is the same as a Binary 4-to-16 decoder, the only difference being that the BCD-to-Decimal Decoder has ten output pins instead of sixteen and the input is a valid BCD number. Thus invalid BCD codes 1010, 1011, 1100, 1101, 1110 and 1111 applied at the input of the Decoder do not activate any of the ten outputs. The commercially available MSI, BCD-to-Decimal Decoder is the 74LS42, which has active-high inputs and active-low outputs.

Encoder

An Encoder functional device performs an operation which is the opposite of the Decoder function. The Encoder accepts an active level at one of its inputs and at its output generates a BCD or Binary output representing the selected input. There are various types of Encoders that are used in Combinational Logic Circuits.

Binary Encoder

The simplest of the Encoders are the 2^n-to-n Encoders. The functional table and the circuit diagram of an 8-to-3 Binary Encoder are shown in table 17.2 and figure 17.6 respectively.

Input								Output		
I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	O_2	O_1	O_0
X	0	0	0	0	0	0	0	0	0	0
X	1	0	0	0	0	0	0	0	0	1
X	0	1	0	0	0	0	0	0	1	0
X	0	0	1	0	0	0	0	0	1	1

X	0	0	0	1	0	0	0	1	0	0
X	0	0	0	0	1	0	0	1	0	1
X	0	0	0	0	0	1	0	1	1	0
X	0	0	0	0	0	0	1	1	1	1

Table 17.2 Function Table of an 8-to-3 Encoder

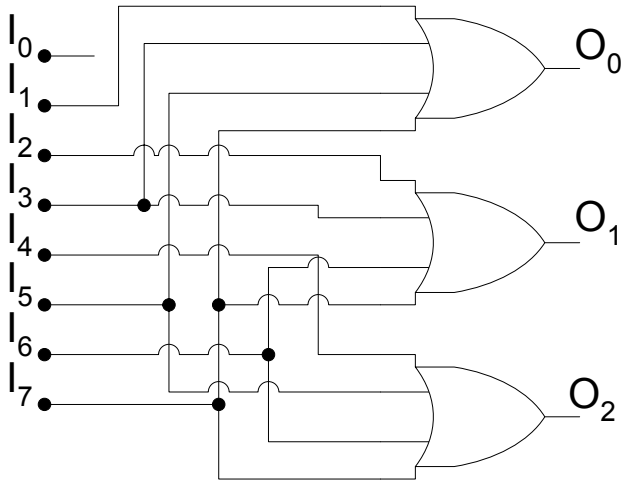


Figure 17.6 8-to-3 Encoder

The inputs and the outputs of the 8-to-3 Encoder are shown to be active-high. The I_0 is shown to be unconnected to any gate or output. Thus, if all inputs are inactive low, or the I_0 input is high the output is 000. The appropriate 3-bit output combination is activated for every input that is asserted by connecting it to logic high. The Binary encoder has a drawback when more than one input is activated. Consider that the inputs I_3 and I_6 are activated simultaneously by applying logic 1 at the two inputs. This results in the outputs 011 and 110 for the two inputs respectively. Thus all three output pins are at logic 1.

Priority Encoders

Priority Encoders remove the problem highlighted earlier with simple Binary Encoders. Priority Encoders have necessary logic to activate the outputs corresponding to the highest Priority input when multiple inputs are asserted simultaneously.

Boolean expressions for the three outputs O_2, O_1 and O_0 of an 8-to-3 Priority Encoder are can be written in terms of variables.

$$O_2 = A_7 + A_6 + A_5 + A_4$$
$$O_1 = A_7 + A_6 + A_3 + A_2$$
$$O_0 = A_7 + A_5 + A_3 + A_1$$

where

$$A_7 = I_7$$

$$A_6 = \overline{I_7} I_6$$
$$A_5 = \overline{I_7} \overline{I_6} I_5$$
$$A_4 = \overline{I_7} \overline{I_6} \overline{I_5} I_4$$

$$A_0 = \overline{I_7} \overline{I_6} \overline{I_5} \overline{I_4} \overline{I_3} \overline{I_2} \overline{I_1} I_0$$

$$O_2 = I_7 + \overline{I_7} I_6 + \overline{I_7} \overline{I_6} I_5 + \overline{I_7} \overline{I_6} \overline{I_5} I_4$$
$$= I_7 + \overline{I_7}(I_6 + \overline{I_6} I_5 + \overline{I_6} \overline{I_5} I_4) = I_7 + I_6 + \overline{I_6} I_5 + \overline{I_6} \overline{I_5} I_4 = I_7 + I_6 + \overline{I_6}(I_5 + \overline{I_5} I_4) = I_7 + I_6 + I_5 + \overline{I_5} I_4$$
$$= I_7 + I_6 + I_5 + I_4$$

$$O_1 = I_7 + \overline{I_7} I_6 + \overline{I_7} \overline{I_6} \overline{I_5} \overline{I_4} I_3 + \overline{I_7} \overline{I_6} \overline{I_5} \overline{I_4} \overline{I_3} I_2$$
$$= I_7 + \overline{I_7}(I_6 + \overline{I_6} \overline{I_5} \overline{I_4} I_3 + \overline{I_6} \overline{I_5} \overline{I_4} \overline{I_3} I_2) = I_7 + I_6 + \overline{I_6} \overline{I_5} \overline{I_4} I_3 + \overline{I_6} \overline{I_5} \overline{I_4} \overline{I_3} I_2 = I_7 + I_6 + \overline{I_6}(\overline{I_5} \overline{I_4} I_3 + \overline{I_5} \overline{I_4} \overline{I_3} I_2)$$
$$= I_7 + I_6 + \overline{I_5} \overline{I_4} I_3 + \overline{I_5} \overline{I_4} \overline{I_3} I_2 = I_7 + I_6 + \overline{I_5} \overline{I_4}(I_3 + \overline{I_3} I_2) = I_7 + I_6 + \overline{I_5} \overline{I_4}(I_3 + I_2) = I_7 + I_6 + \overline{I_5} \overline{I_4} I_3 + \overline{I_5} \overline{I_4} I_2$$
$$O_0 = I_7 + \overline{I_7} \overline{I_6} I_5 + \overline{I_7} \overline{I_6} \overline{I_5} \overline{I_4} I_3 + \overline{I_7} \overline{I_6} \overline{I_5} \overline{I_4} \overline{I_3} \overline{I_2} I_1$$
$$= I_7 + \overline{I_7}(\overline{I_6} I_5 + \overline{I_6} \overline{I_5} \overline{I_4} I_3 + \overline{I_6} \overline{I_5} \overline{I_4} \overline{I_3} \overline{I_2} I_1) = I_7 + \overline{I_6} I_5 + \overline{I_6} \overline{I_5} \overline{I_4} I_3 + \overline{I_6} \overline{I_5} \overline{I_4} \overline{I_3} \overline{I_2} I_1$$
$$= I_7 + \overline{I_6}(I_5 + \overline{I_5} \overline{I_4} I_3 + \overline{I_5} \overline{I_4} \overline{I_3} \overline{I_2} I_1) = I_7 + \overline{I_6}(I_5 + \overline{I_5}(\overline{I_4} I_3 + \overline{I_4} \overline{I_3} \overline{I_2} I_1)) = I_7 + \overline{I_6}(I_5 + \overline{I_4} I_3 + \overline{I_4} \overline{I_3} \overline{I_2} I_1)$$
$$= I_7 + \overline{I_6}(I_5 + \overline{I_4}(I_3 + \overline{I_3} \overline{I_2} I_1)) = I_7 + \overline{I_6}(I_5 + \overline{I_4}(I_3 + \overline{I_2} I_1)) = I_7 + \overline{I_6} I_5 + \overline{I_6} \overline{I_4} I_3 + \overline{I_6} \overline{I_4} \overline{I_2} I_1$$

The MSI, 74XX148 8-input Priority has a circuit implemented based on the Boolean expression for outputs O_0, O_1 and O_2. The function table of the 8-inpuy Priority Encoder is presented. Table 17.3

Inputs									Outputs				
$\overline{E1}$	$\overline{I0}$	$\overline{I1}$	$\overline{I2}$	$\overline{I3}$	$\overline{I4}$	$\overline{I5}$	$\overline{I6}$	$\overline{I7}$	$\overline{A2}$	$\overline{A1}$	$\overline{A0}$	$\overline{GS}$	$\overline{EO}$
1	X	X	X	X	X	X	X	X	1	1	1	1	1
0	X	X	X	X	X	X	X	0	0	0	0	0	1
0	X	X	X	X	X	X	0	1	0	0	1	0	1
0	X	X	X	X	X	0	1	1	0	1	0	0	1
0	X	X	X	X	0	1	1	1	0	1	1	0	1
0	X	X	X	0	1	1	1	1	1	0	0	0	1
0	X	X	0	1	1	1	1	1	1	0	1	0	1
0	X	0	1	1	1	1	1	1	1	1	0	0	1
0	0	1	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0

Table 17.3 Function Table of an 8-Input Priority Encoder

Cascading Priority Encoders

The 74XX148 Priority Encoder has active-low inputs and active-low outputs. The Encoder also has an active-low enable input E1 which enables or disables the outputs. The

Enable output EO signal is used to cascade multiple Encoders to form larger Encoders. The EO output is connected to the EI input of the Encoder which handles lower priority inputs. Two 8-input are shown connected together to form a 16-input Priority Encoder. Figure 17.7

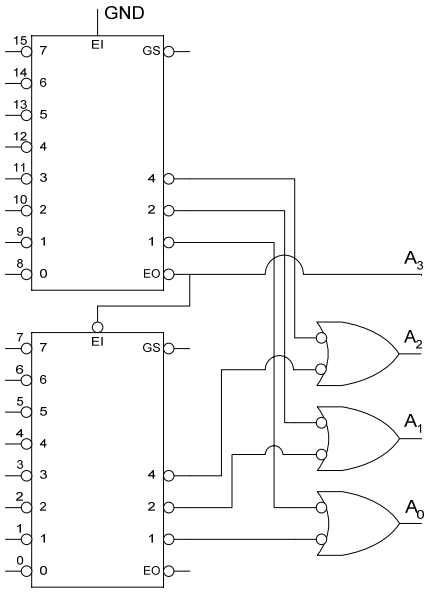


Figure 17.7 8-input Priority Encoders connected to form a 16-input Priority Encoder

Decimal-to-BCD Encoder

The Decimal-to-BCD Encoder has ten inputs, for the decimal digits 0 to 9 and four outputs corresponding to the 4-bit BCD output. The 74LS147 is a Decimal-to-BCD Priority Encoder which has active-low input and outputs. The Decimal-to-BCD Priority Encoder is used as a keypad encoder. A telephone keypad has digits 0 to 9. The keypad is connected to the encoder through pull-up resistors that ensure that the inputs to the encoder are logic high when none of the keypad keys is pressed. When ever a key is pressed the appropriate input of the encoder is connected to logic low and at the output the corresponding BCD code is generated. Figure 17.8

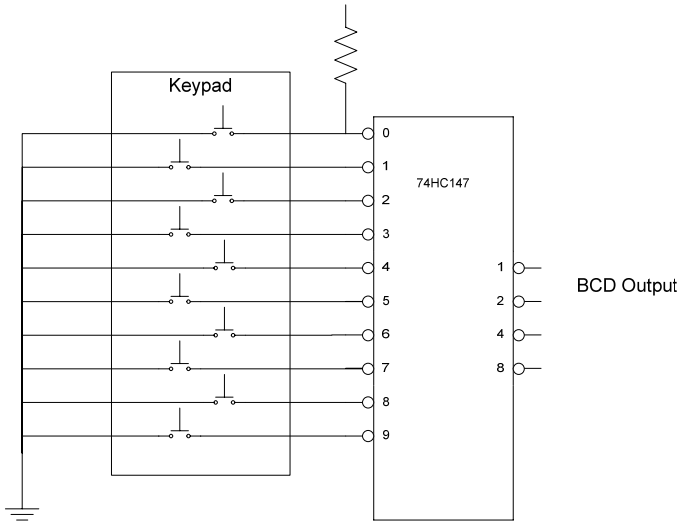


Figure 17.8 Keypad Encoder

Multiplexer

Multiplexer is a digital switch that has several inputs and a single output. The Multiplexer also has select inputs that allow any one of the multiple inputs can be selected to be connected to the output. Multiplexers are also known as Data Selectors. The main use of the Multiplexer is to select data from multiple sources and to route it to a single Destination. In a computer, the ALU combinational circuit has two inputs to allow arithmetic operations to be performed on two quantities. The two quantities are usually stored in different set of registers. The inputs of the two multiplexers are connected to the output of each of the multiple registers. The outputs of the two multiplexers are connected to the two inputs of the ALUs. The Multiplexers are used to route the contents of any two registers to the ALU inputs.

Multiplexers are available in different configurations. The 4-to-1 Multiplexer circuit is shown. Figure 17.9, the function table of the Multiplexer is presented. Table 17.4

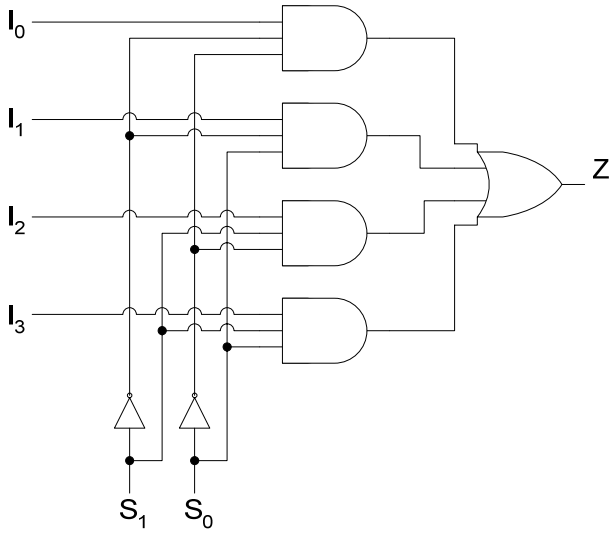


Figure 17.9 4-to-1 Multiplexer

Select Inputs		Output
S_1	S_0	Z
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Table 17.4 Function table of 1-to-4 Multiplexer

When the Select input are set to 00, the first AND gate at the top is enabled allowing the logic high or low applied at input I_0 to be routed through the OR gate to the output Z. Similarly, when the Select input is set to 10 the third gate is enabled allowing the logic value applied at the input I_2 to be routed through the OR gate to the output Z.

1. Dual 4-Input Multiplexer

Commercial available 4-input Multiplexer is the 74XX153 IC which has two 4-input multiplexers. The two select inputs of the two 4-input multiplexers are common, however each multiplexer has a separate enable input which allows the two multiplexers to be separately controlled. The circuit diagram of the dual 4-input multiplexers is shown. Figure 17.10

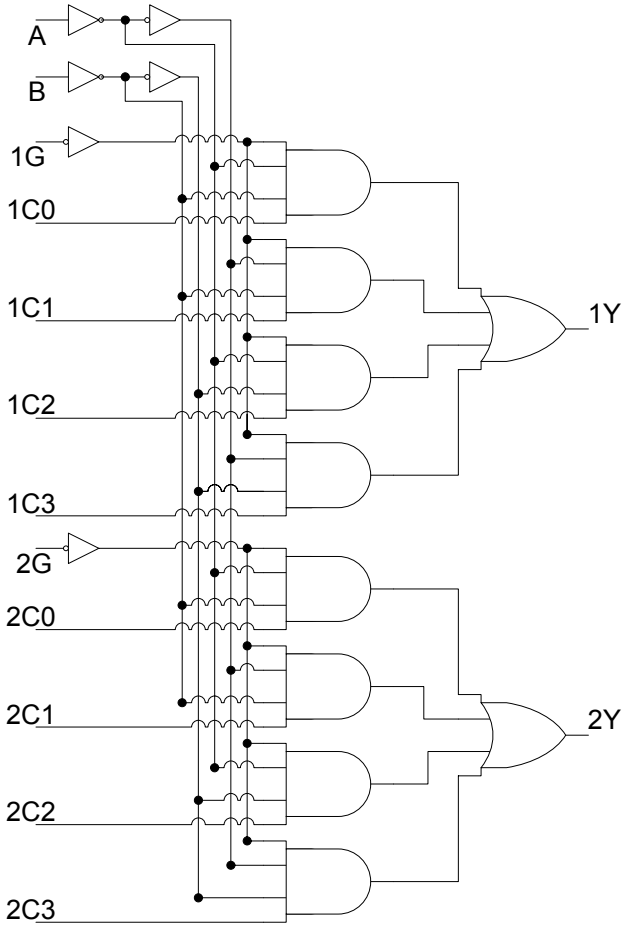


Figure 17.10 Dual, 4-input Multiplexer

Both the 4-input Multiplexers have active-high inputs and outputs. The first Multiplexer has the inputs 1C0, 1C1, 1C2 and 1C3 and the output 1Y. The multiplexer has an active-low enable signal defined by 1G. The select inputs are defined by A and B which are both active-high. Two extra NOT gates are connected at the select inputs to reduce the unit load from 5 each to one. Similar to the 4-input Multiplexer discussed earlier, the select input lines enable one of the four AND gates and allow the corresponding input logic value to be routed to the output through the OR gate. The second 4-input Multiplexer is identical it has active-high inputs defined by 2C0, 2C1, 2C2 and 2C3 and an active-high output defined by 2Y. The multiplexer has an independent active-low enable signal that enables/disables the four AND gates. The select inputs A and B controlling the first multiplexer also control the second multiplexer.

Lesson No. 18

a) 2-INPUT 4-BIT MULTIPLEXER

The MSI, 74X157 is a 2-input, 4-bit Multiplexer. This multiplexer has two sets of 4-bit inputs. It also has 4-bit outputs. The single select input line allows the first set of four inputs or the second set of 4-inputs to be connected to the output. Thus four-bits of data from two sources are routed to the output. The function table and the circuit of the multiplexer are shown. table 18.1, figure 18.1

The multiplexer has two sets of 4-bit active-high inputs 1A, 2A, 3A, 4A and 1B, 2B, 3B, 4B respectively. The multiplexer has 4-bit active-high outputs 1Y, 2Y, 3Y 4Y. The single select input allows either the 4-bit input A or the 4-bit input B to be connected to the 4-bit output Y. The G active-low pin enables or disables the Multiplexer.

Inputs		Outputs			
G	S	1Y	2Y	3Y	4Y
1	X	0	0	0	0
0	0	1A	2A	3A	4A
0	1	1B	2B	3B	4B

Table 18.1 Function table of 2-Input 4-Bit Multiplexer

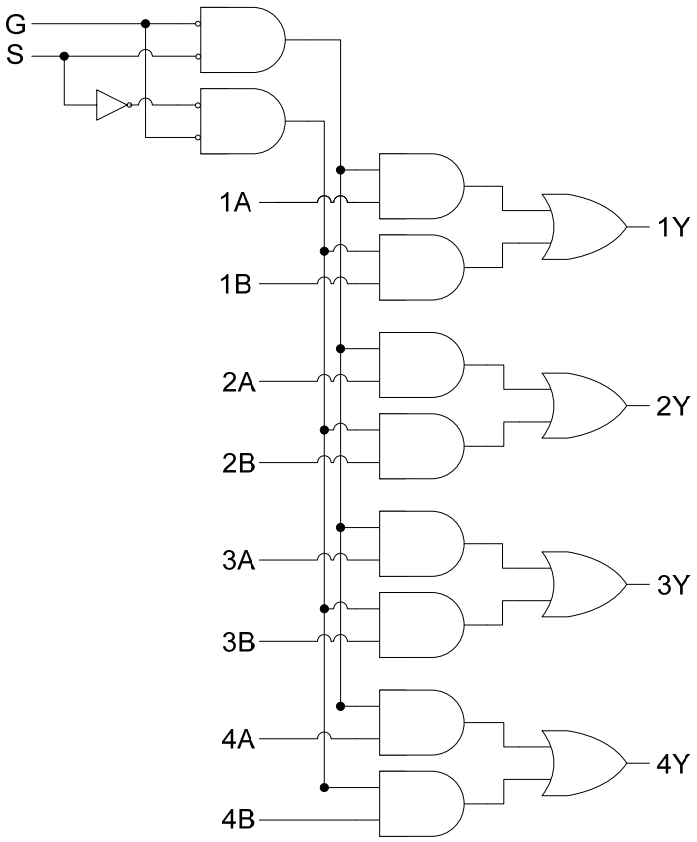


Figure 18.1 2-input 4-bit Multiplexer

Expanding Multiplexers

Multiplexers have to be connected together to form larger multiplexer to fulfil specific application requirements.

1. 8-Input Multiplexer

A single dual, 4-input multiplexer 74X153 can be connected to form an 8-input multiplexer. The circuit diagram and the function table are shown in fig. 18.2 and table 18.2 respectively. The two active-low enable inputs of the two 4-input multiplexers are connected together using a NOT gate to form the C input of the 8-input multiplexer. When C is set to 0, the first multiplexer is selected allowing its inputs 1C0, 1C1, 1C2 and 1C3 to be selected through select inputs A and B. When C is set to 1, the second multiplexer is selected allowing its inputs and outputs to be used. The two outputs are connected through an OR gate.

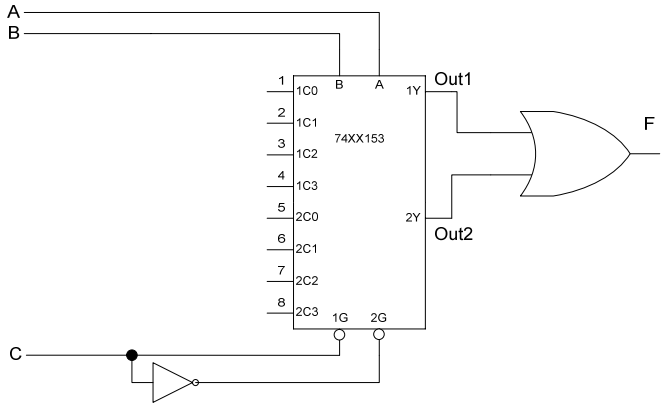


Figure 18.2 8-to-1 Multiplexer using two 4-to-1 Multiplexers

Input			Output
C	B	A	F
0	0	0	1C0
0	0	1	1C1
0	1	0	1C2
0	1	1	1C3
1	0	0	2C0
1	0	1	2C1
1	1	0	2C2
1	1	1	2C3

Table 18.2 Function Table of a 8-to-1 Multiplexer

2. 16-Input Multiplexer

Two 74XX153 Dual, 4-input multiplexer can be connected to form a 16-input multiplexer. The circuit diagram and the function table of the 16 input multiplexer are shown in Figure 18.3 and table 18.3 respectively.

The select inputs A and B of the two dual, 4-input multiplexers are connected together which allows selection of any one input out of the four set of 4-bit inputs. The four active-low multiplexer enable inputs which allow selection of any one of the four multiplexers are connected to the active-low outputs of a 2-to-4 decoder. The decoder inputs C and D enable one out of the four multiplexers. The four outputs are connected together through a 4-input OR gate. The G enable input of the decoder when set to 1 disables the decoder and the multiplexers.

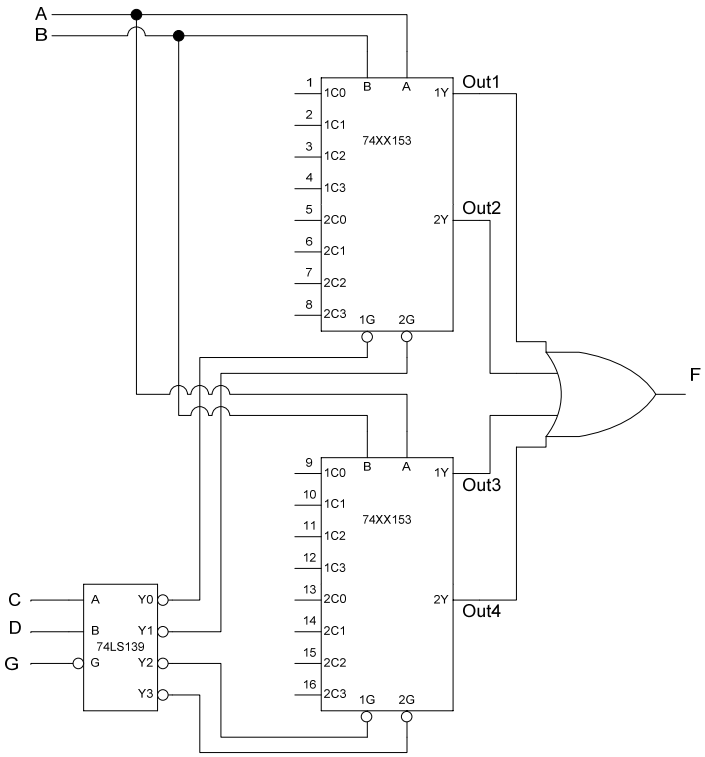


Figure 18.3 16-input Multiplexer

3. 2-Input, 8-bit Multiplexer

Two 2-input, 4-bit multiplexers 74X157 can be connected to implement a 2-input, 8-bit multiplexer. The circuit diagram is shown in figure 18.4. The select S inputs of the two multiplexers are connected together so that the 4-bit inputs A of both the multiplexers are selected simultaneously when S is set to logic low. Similarly, by setting the S input to logic-high the B inputs of both the multiplexers are selected. The active-low enable inputs G of both the multiplexers are also connected together so that both the multiplexers are enabled and disabled simultaneously by setting the G input to 0 or 1 respectively.

Inputs					Output
G	D	C	B	A	F
1	x	x	x	x	0
0	0	0	0	0	1C0 (M1)
0	0	0	0	1	1C1 (M1)
0	0	0	1	0	1C2 (M1)

0	0	1	0	0	2C0 (M1)
0	0	1	0	1	2C1 (M1)
0	0	1	1	0	2C2 (M1)
0	0	1	1	1	2C3 (M1)
0	1	0	0	0	1C0 (M2)
0	1	0	0	1	1C1 (M2)
0	1	0	1	0	1C2 (M2)
0	1	0	1	1	1C3 (M2)
0	1	1	0	0	2C0 (M2)
0	1	1	0	1	2C1 (M2)
0	1	1	1	0	2C2 (M2)
0	1	1	1	1	2C3 (M2)

Table 18.3 Function Table of 16-bit Multiplex

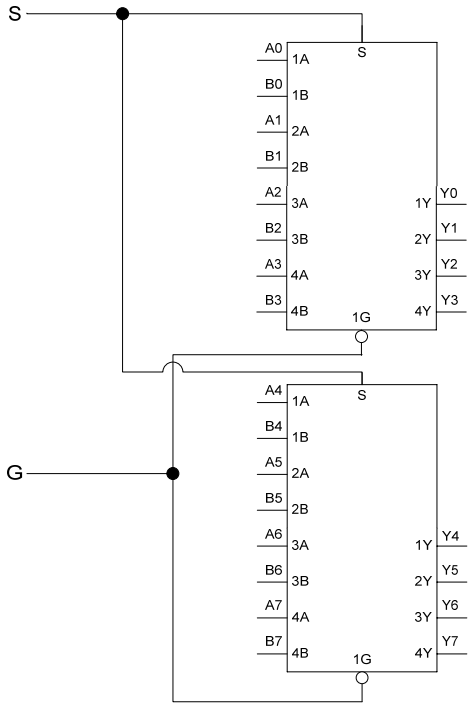


Figure 18.4 2-Input, 8-bit Multiplexer

Applications of Multiplexers

Multiplexers are used in a wide variety of applications. Their primary use is to route data from multiple sources to a single destination. Other than its use as a Data router, a parallel to serial converter, logic function generator and used for operation sequencing.

1. Data Routing

A two digit 7-Segment display uses two 7-Segments Display digits connected to two BCD to 7-Segment display circuits. To display the number 29 the BCD number 0010 representing the MSD is applied at the inputs of the BCD to 7-Segment display circuit connected to the MSD 7-Segment Display Digit. Similarly, the BCD input 1001 representing the numbers 9 is applied at the inputs of the LSD display circuit. The circuit uses two BCD to 7-Segment decoder circuits to decode each of the two BCD inputs to the respective 7-

Segment display outputs. Figure 18.5. The display circuit can be implemented using a single BCD to 7-Segment IC and a Multiplexer.

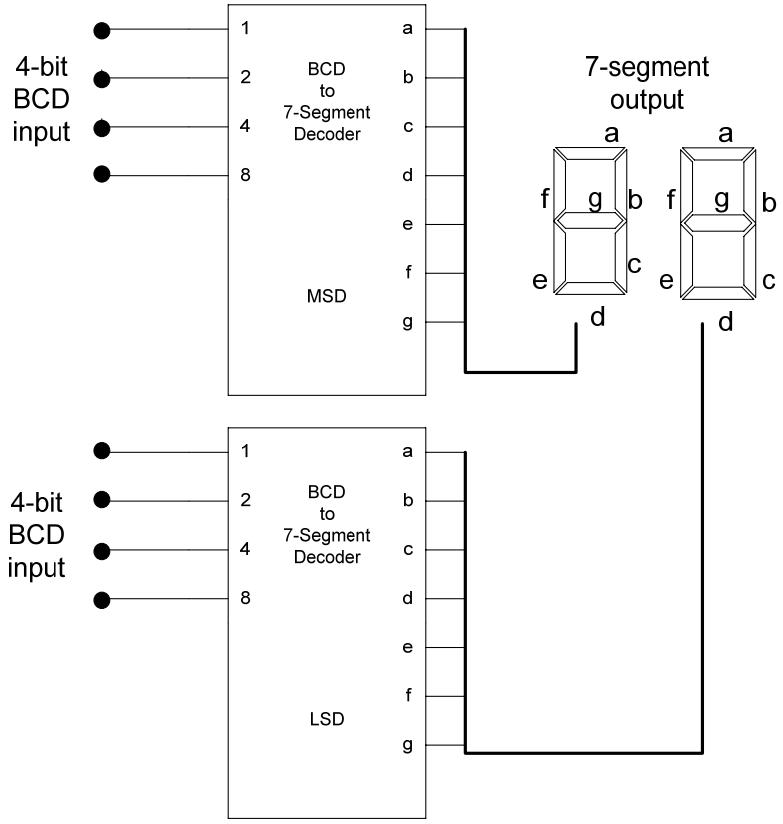


Figure 18.5 2-Digit Decimal Display Circuit

To fully understand the working of the alternate circuit it is essential to understand the working of the 7-Segment Display Digit. 7-Segment Display Digits are implemented using 7 LEDs (Light Emitting Diodes) connected in the form of number 8. To turn on a LED, its Anode is connected to +5 volts and its Cathode is connected to Ground or 0 volts. 7-Segment displays are of two types, the Common Anode type and the Common Cathode type.

a. Common Anode 7-Segment Display

The Common Anode 7-Segment Display has positive end of each of the seven display segments (LEDs) connected together. To display any segment the Common Anode of the display has to be connected to +5 volts and the other end of each segment has to be connected to 0 volts. Figure 18.6a

b. Common Cathode 7-Segment Display

The Common Cathode 7-Segment Display has negative end of each of the seven display segments (LEDs) connected together. To display any segment the Common Cathode of the display has to be connected to 0 volts and the other end of each segment has to be connected to +5 volts. Figure 18.6b.

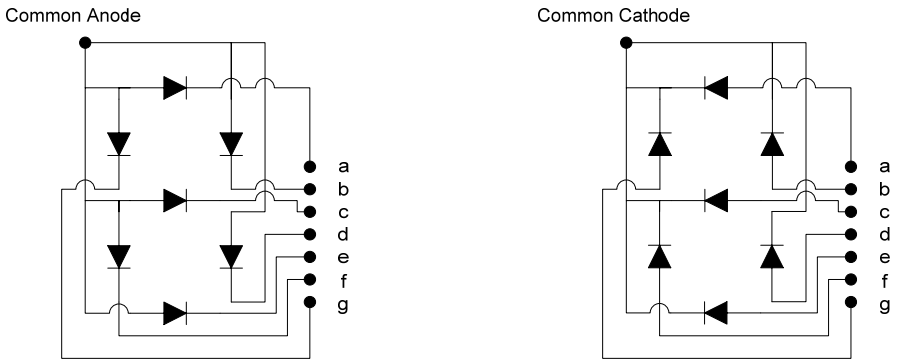


Figure 18.6 Common Anode and Common Cathode 7-Segment Displays

The alternate 2-digit display circuit based on a multiplexer and a BCD to 7-Segment Decoder is shown in figure 18.7. The BCD numbers of the two digits to be displayed are applied at the inputs A and B of the multiplexer. The 4-bit output of the Multiplexer is connected to the 4-bit input of the BCD to 7-Segment Decoder circuit. The 7-Segment output of the Decoder is connected to the 7 segments of both the Common Cathode Displays. The MSD/LSD input is connected to the select input of the Multiplexer, the Common Cathode of the MSD and the Common Cathode of the LSD through a NOT gate. The MSD is applied at Input A, and the LSD at input B. To Display the MSD the MSD/LSD input is set to 0. The BCD number at Input A of the multiplexer is selected and routed through the BCD to 7-Segment Decoder to both the two 7-Segment Displays. Since the MSD/LSD input is 0 therefore the MSD display is selected and the MSD is displayed. The MSD/LSD input is switched to 1, which selects the BCD at input B which is routed through the Multiplexer to the 7-Segment Decoder and ultimately to the 7-segment displays. Since the MSD/LSD is set to 1, the Common Cathode of the LSD is connected to zero, thus the number at input B of the multiplexer is displayed on the LSD display. The MSD/LSD input is rapidly switched between 0 and 1 to allow both the digits to be seen on the 2-digit display. This circuit can be expanded to incorporate any number of digits.

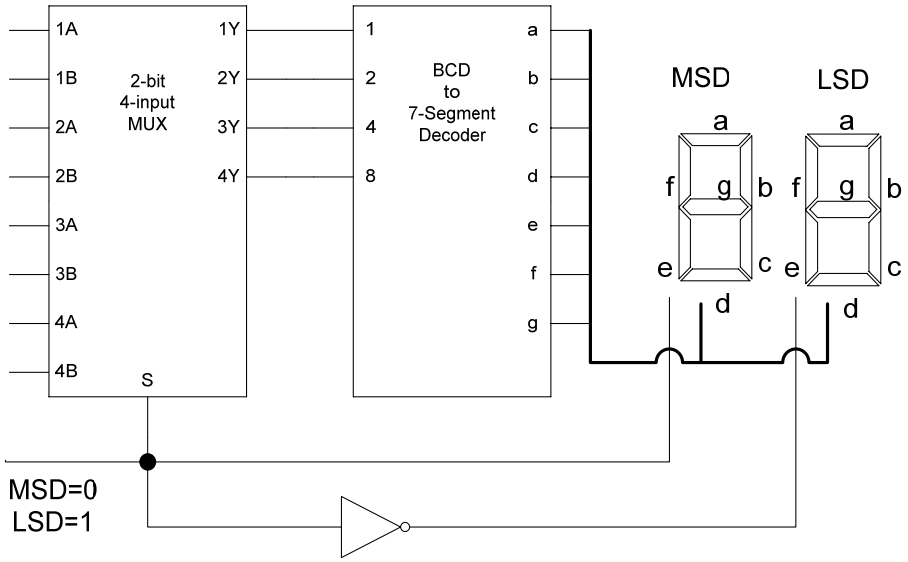


Figure 18.7 2-Digit Decimal Display using a Multiplexer

2. Parallel to Series Conversion

In a Digital System, Binary data is used and represented in parallel. Parallel data is a set of multiple bits. For example, a nibble is a parallel set of 4-bits, a byte is a parallel set of 8 bits. When two binary numbers are added, the two numbers are represented in parallel and the parallel adder works and generates a sum term which is also in parallel.

Transmission of information to remote locations through a piece of wire requires that the parallel information (data) be converted into serial form. In a serial data representation, data is represented by a sequence of single bits. An 8-bit parallel data can be transmitted through a single piece of wire 1-bit at a time. Transmitting 8-bits simultaneously (in parallel form) requires 8 separate wires for the 8-bits. Laying of 8 wires across two remote locations for data transfer is expensive and is therefore not practical. All communication systems set up across remote locations use serial transmission.

An 8-bit parallel data can be converted into serial data by using an 8-to-1 multiplexer such as 74X151 which has 8 inputs and a single output. The 8-bit data which is to be transmitted serially is applied at the 8 inputs I_{0-7} of the multiplexer. A three bit counter which counts from 0 to 7 is connected to the three select inputs S_0, S_1 and S_2. The counter is connected to a clock which sends a clock pulse to the counter every 1 millisecond. Initially, the counter is reset to 000, the I_0 input is selected and the data at input I_0 is routed to the output of the multiplexer. On receiving the clock signal after 1 millisecond the counter increments its count from 000 to 001 which selects I_1 input of the multiplexer and routes the data present at the input to the output. Similarly, at the next clock pulse the counter increments to 010, selecting I_2 input and routing the data to the output. Thus after 8 milliseconds the parallel data is routed to the output 1-bit at a time. The output of the multiplexer is connected to the wire through which the serial data is transmitted. Figure 18.8

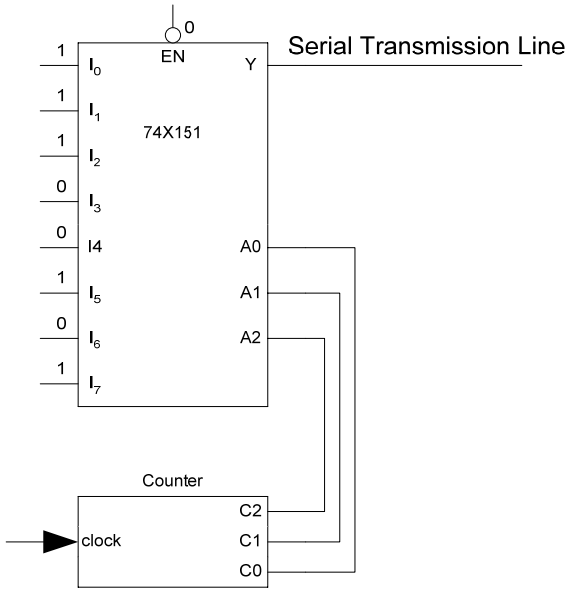


Figure 18.8a Parallel to Serial Conversion

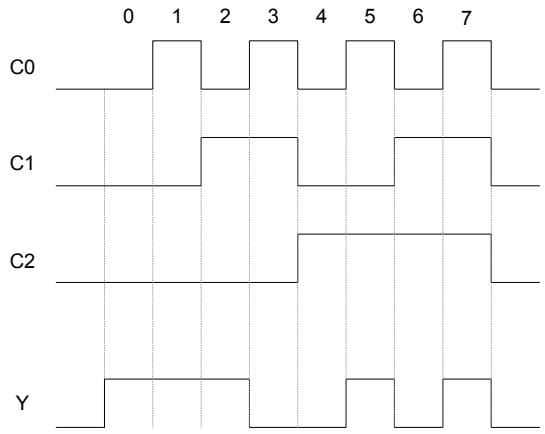
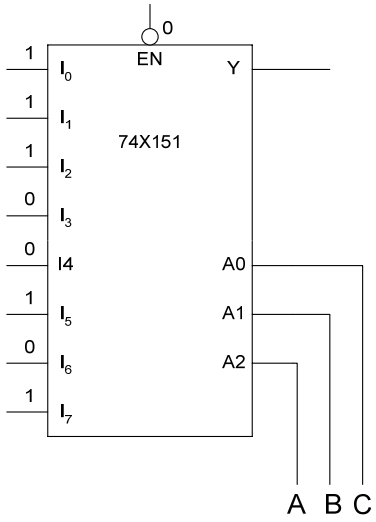


Figure 18.8b Timing diagram of the Parallel to Serial Conversion Circuit

3. Logic Function Generator

Multiplexers can be used to implement a logic function directly from the function table without the need for simplification. The select inputs of the multiplexer are used as the function variables. The inputs of the multiplexer are connected to logic 1 and 0 to represent the missing and available terms. The three variable function table and its 8-to-1 multiplexer based function implementation is shown in figure 18.9



Input			Output
A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Figure 18.9 Logic Function Generator based on 3-variable logic function table

4. Operation Sequencing

Many industrial applications have processes that run in a sequence. A paint manufacturing plant might have a four step process to manufacture paint. Each of the four steps runs in a sequence one after the other. The second step can not start before the first step has completed. Similarly, the third and fourth step of the paint manufacturing process can not proceed unless steps two and three have completed. It is not necessary that each of the

duration and can be variable depending upon the quantity of paint manufactured or other parameters. Normally, the end of each step in the manufacturing process is indicated by a signal which is actuated by some machine which has completed its part of the manufacturing process. On receiving the signal the next step of the manufacturing process is initiated.

The entire sequence of operations is controlled by a Multiplexer and a Decoder circuit. Figure 18.10. The manufacturing processes are started by resetting the 2-bit counter to 00. The counter output is connected to the select input of the Multiplexer and the inputs of the Decoder which selects the Multiplexer input I0 is and activates the Decoder output Y0. The Decoder output is connected to initiate the first process. When the process completes it indicates the completion of the process by setting its output to logic 1. The output of Process 1 is connected to I0 input of the Multiplexer. When Process 1 sets its output to 1 to indicate its completion, the logic 1 is routed by the Multiplexer to the clock input of the 2-it counter. The counter on receiving logic 1 increments its count to 01, which selects I1 input of the Multiplexer and the Y1 output of the Decoder. The input to Process 1 is deactivated and Process 2 is activated by Y1. On completion of Process 2 its output is set to logic 1, which is routed by the multiplexer to the clock input of the 2-bit counter which increments to the next count. This continues until Process 4 signals its completion after which the Decoder and the Multiplexer is deselected completing the manufacturing process.

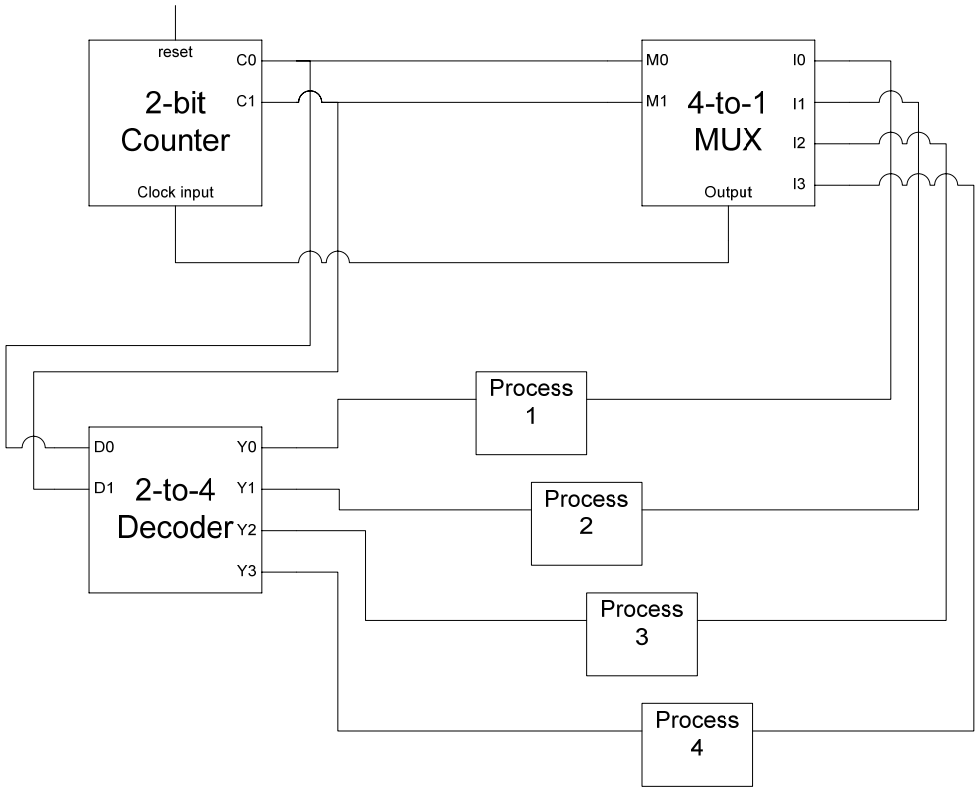


Figure 18.10 Control of Manufacturing process through Operation Sequencing

Lesson No. 19

DEMULTIPLEXER

A Multiplexer has several inputs. It selects one of the inputs and routes the data at the selected input to the single output. Demultiplexer has an opposite function to that of the Multiplexer. It has a single input and several outputs. The Demultiplexer selects one of the several outputs and routes the data at the single input to the selected output. A demultiplexer is also known as a Data Distributor.

The circuit diagram of a 1-to-4 line Demultiplexer is shown. Figure 19.1. The circuit if compared to that of the 2-to-4 Decoder. The Decoder enable input is used as the Demultiplexer data input. A Demultiplexer is not available commercially. A Demultiplexer is available as a Decoder/Demultiplexer chip which can be configured to operate as a Demultiplexer or a Decoder.

The circuit of the 1-to-4 Demultiplexer is similar to the 2-to-4 Binary Decoder described earlier figure 16.9. The only difference between the two is the addition of the Data Input line, which is used as enable line in the 2-to-4 Decoder circuit figure 16.10. Assuming the select inputs I_1 and I_0 are set to 1 and 0 respectively. The O_2 output is set to 1 if the Data input is 1 or it is set to 0 if the Data input is 0.

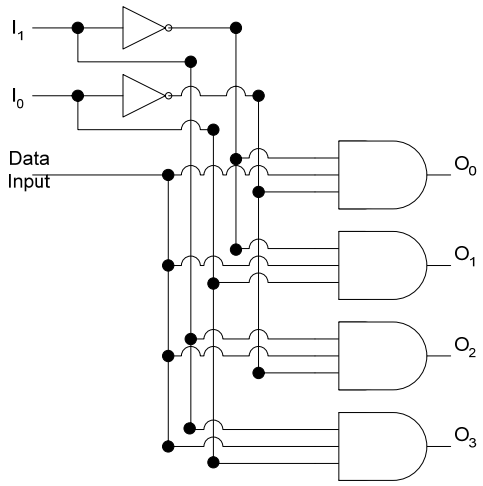


Figure 19.1 1-to-4 Demultiplexer

Applications of Demultiplexer

Demultiplexer is used to connect a single source to multiple destinations. One use of the Demultiplexer is at the output of the ALU circuit. The output of the ALU has to be stored in one of the multiple registers or storage units. The Data input of the Demultiplexer is connected to the output of the ALU. Each output of the Demultiplexer is connected to each of the multiple registers. By selecting the appropriate output data from the ALU is routed to the appropriate register for storage.

The second use of the Demultiplexer is the reconstruction of Parallel Data from the incoming serial data stream. Serial data arrives at the Data input of the Demultiplexer at fixed time intervals. A counter attached to the Select inputs of the Demultiplexer routes the incoming serial bits to successive outputs where each bit is stored. When all the bits have been stored, data can be read out in parallel. Figure 19.2

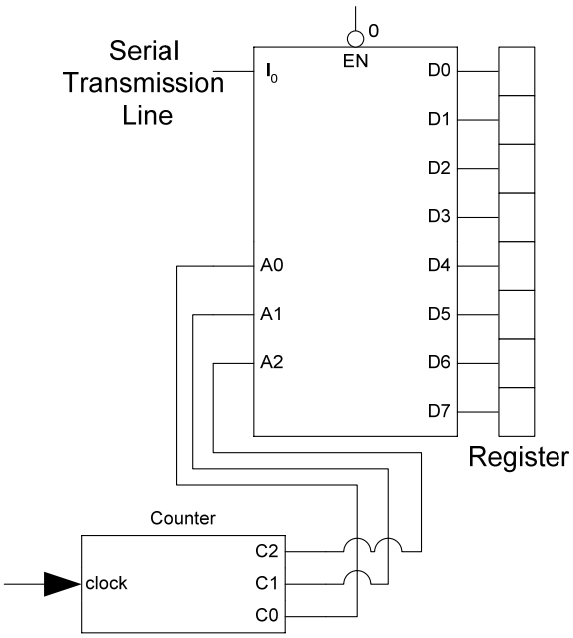


Figure 19.2 Demultiplexer as a Serial to Parallel Converter

Programmable Logic Devices

Programmable Logic Devices are used in many applications to replace the Logic gates and MSI chips. PLDs save circuit space and reduce and save the cost of components in a Digital Circuit. PLDS consists of Arrays of AND gates and OR gates that can be programmed to perform specific functions.

Programmable Arrays of AND Gates and OR Gates

The array is essentially a grid of conductors that forms rows and columns with a fuse connecting each column conductor with each row conductor. The fuses can be blown to disconnect a particular column from a particular row. The OR gate array consists of the grid and OR gates. Similarly the AND gate array consists of the grid and AND Gates. Figure 19.3

Each column conductor in the grid represents a single variable or its complement. A grid of several column conductors represents several variables and their complements. Each OR and AND gate in the array is connected to each of the variables through horizontal conductors. When all the fuses are intact, all variables are present at the inputs of all the OR and AND gates. The OR and AND gates can be configured to have specified literals connected to their inputs by blowing away appropriate fuses which are blown through programming. A programmed OR array has sum terms at the output of its OR gates. Similarly a programmed AND array has product terms at its output. Figure 19.4

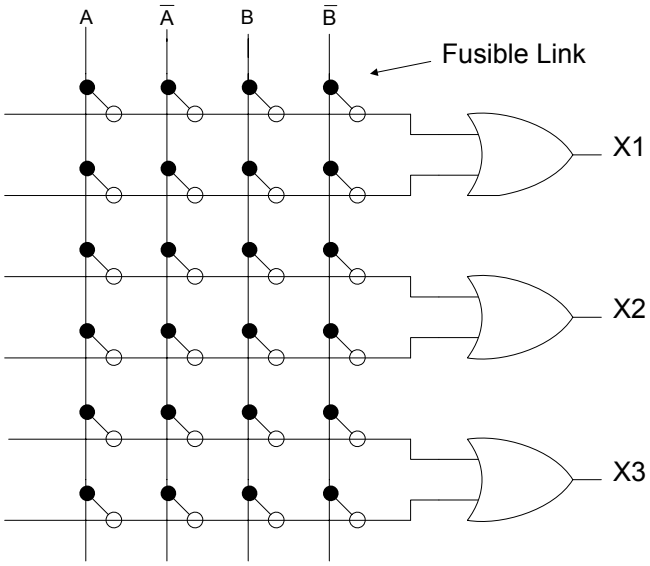


Figure 19.3a OR Gate Array

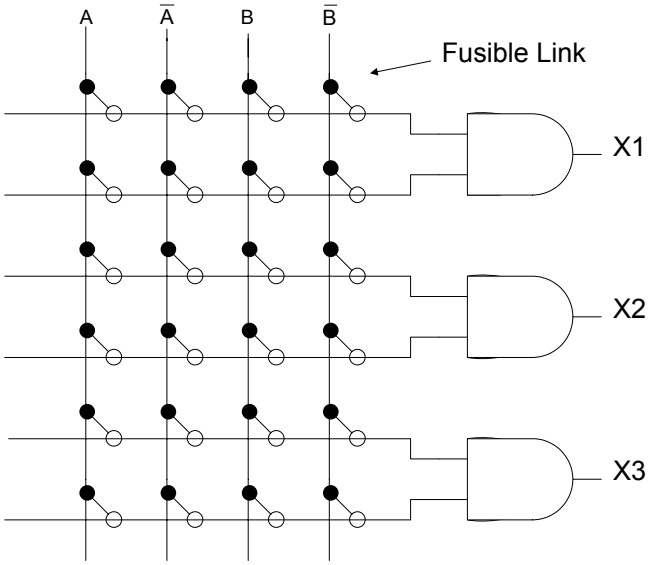


Figure 19.3b AND Gate Array

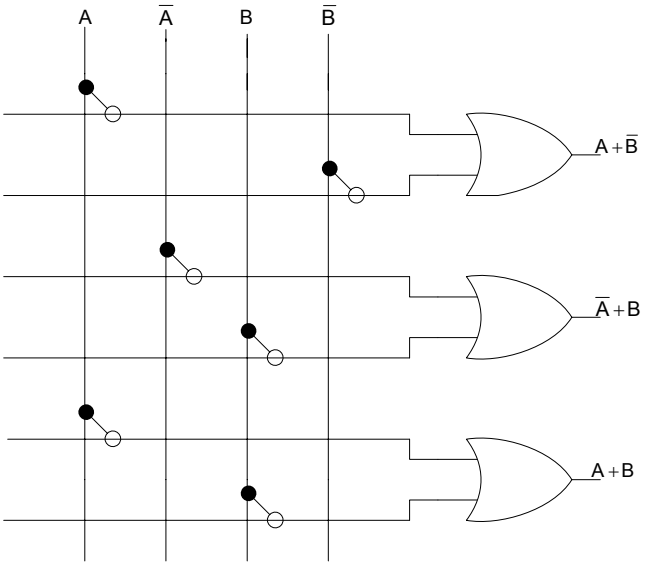


Figure 19.4a Programmed OR Gate Array

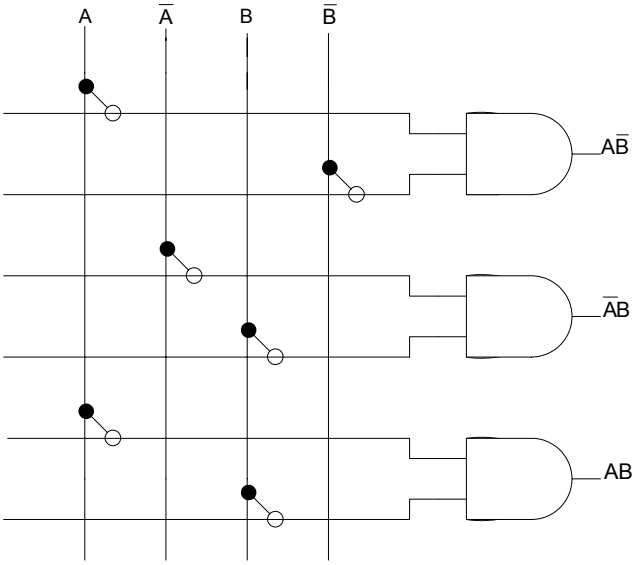


Figure 19.4b Programmed AND Gate Array

An alternate implementation of the grid is with no fuses, the grid column and row conductors are not connected to each other. A specific column conductor can be connected to a row conductor by shorting the column and row conductors. Both the methods in which a fuse is blown to disconnect a column from a row and the shorting method in which a column is connected to a row can only be done once. Thus when an array has been configured to perform a function it can not be reprogrammed.

Programmable Logic Devices have an array of AND gates and an array of OR gates

classified according to their architecture which allows either both the arrays to be programmed or only one of the two arrays.

1. Programmable Read-Only Memory (PROM)

The PROM consists of a fixed non-programmable AND array configured as a decoder and a programmable OR array. Figure 19.5. The PROM is used as a storage device which stores information at addressable locations. It has limited applications and is not used as a logic device. PROM architecture and details are discussed in latter lectures.

2. Programmable Logic Array (PLA)

The PLA consists of a programmable AND array and a programmable OR array. Figure 19.6. It has been designed to overcome the limitations of a PROM. PLA is also known as a Field-Programmable Logic Array as it can be programmed by the user and not by the manufacturer.

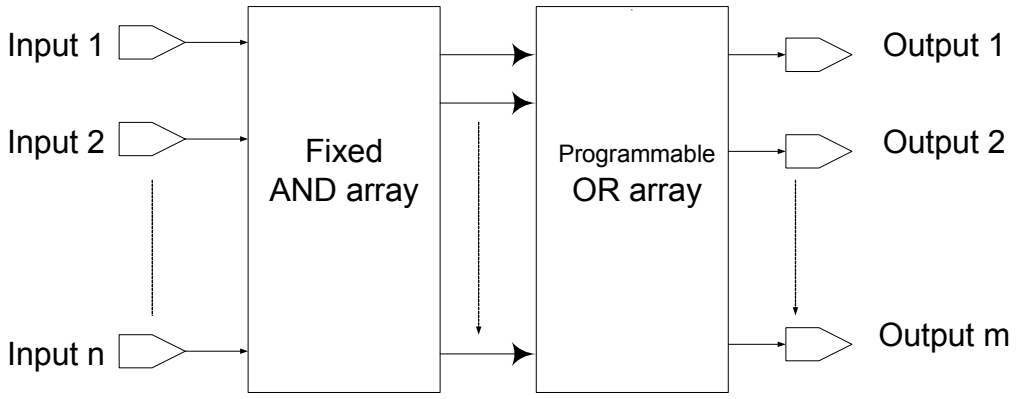


Figure 19.5 Block diagram of a PROM

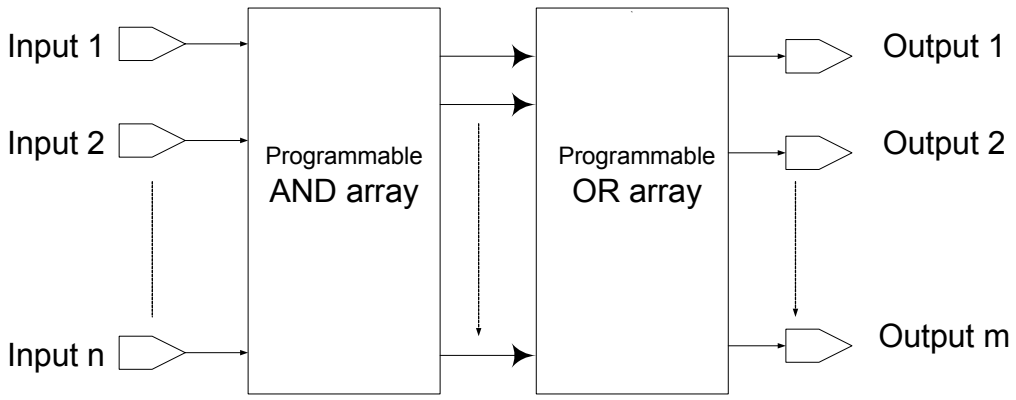


Figure 19.6 Block diagram of a PLA

3. Programmable Array Logic (PAL)

The PAL has been designed to overcome the longer delays and the complex circuitry associated with the PLA due to two programmable arrays. The PAL has programmable AND array and a fixed OR array. Figure 19.7

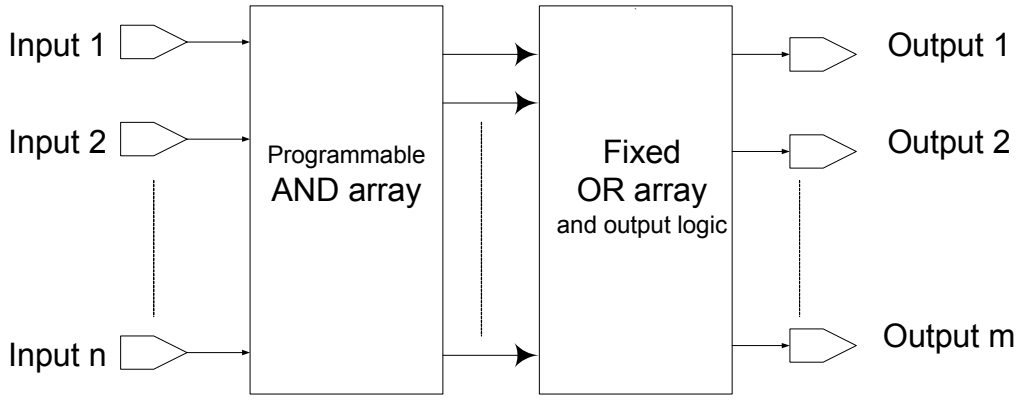


Figure 19.7 Block diagram of a PAL

4. Generic Array Logic (GAL)

The GAL has a reprogrammable AND array and a fixed OR array with programmable output logic. Figure 19.8. The main difference between GAL and PAL are the reprogrammable AND array which can be programmed again and again, unlike PAL AND array which can be programmed once. GAL uses E^2CMOS technology which is Electrically Erasable CMOS instead of Bipolar technology and fusible links. The other difference is the programmable outputs.

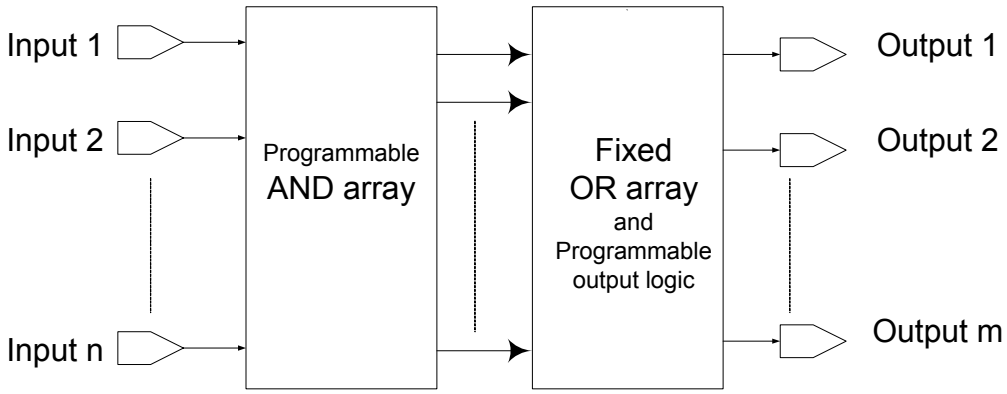


Figure 19.8 Block diagram of a GAL

All the four PLD devices use AND arrays followed by OR arrays. Therefore they all allow implementation of Sum-of-Product Boolean expressions.

PAL Circuit and Programming

A simplified PAL structure is shown where the AND array has been programmed to generate three product terms which are added together by the OR array. Figure 19.9

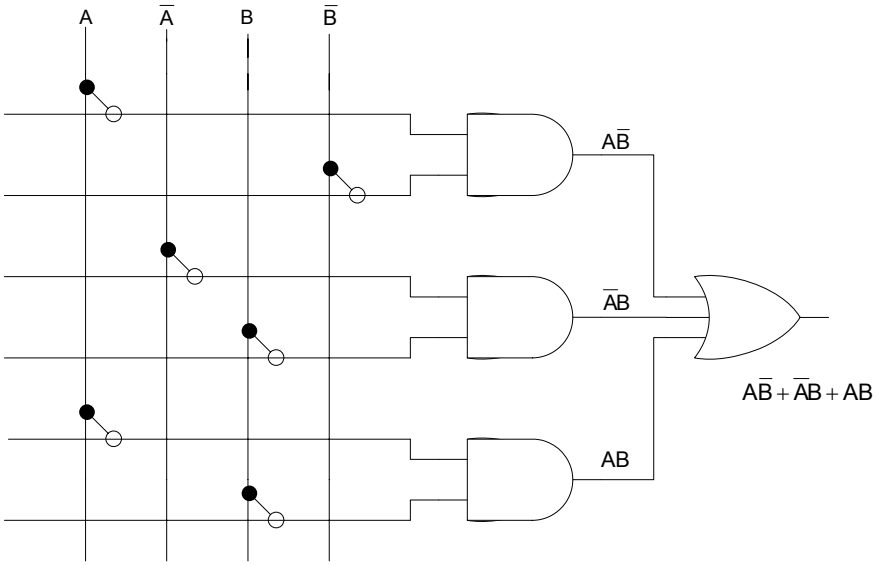


Figure 19.9 PAL programmed to implement an SOP function

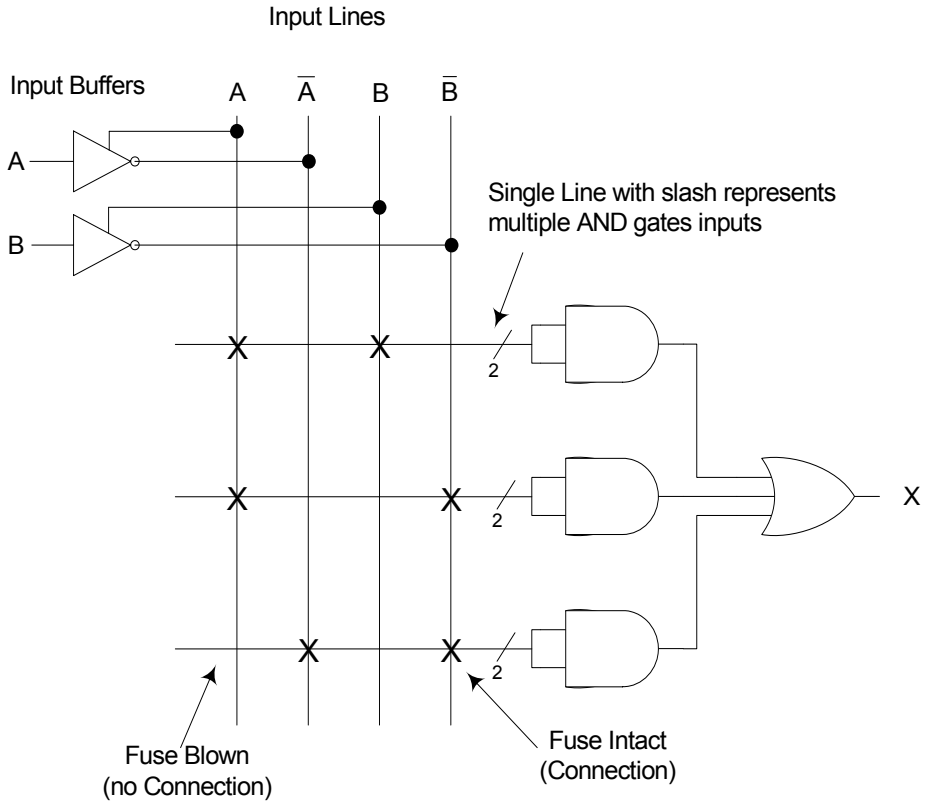


Figure 19.10 Simplified diagram of programmed PAL

PALs have many inputs and multiple outputs connected through a large number of AND gates and OR gates. Drawing the circuit diagram of a PAL having multiple gates each having multiple inputs becomes difficult. PALs have Buffers at the inputs which produce the actual variable and its complement. The multiple input lines to an AND gate array are

the fuses that are intact showing a connection between the vertical line and horizontal line of the AND array. Figure 19.10

PAL Outputs

PALs typically have 8 or more inputs to the AND array and 8 or less outputs from the fixed OR array. Some PALs have combined inputs and outputs that can be programmed as either inputs or outputs. PAL output logic can be configured according to the application of the PAL. The modified block diagram representing a PAL showing the output of the OR Array connected to output logic which allows the outputs to be configured is shown in figure 19.11. The three types of outputs are

- Combinational Output used for an SOP function and is available as an active-high or active-low output. Figure 19.12a
- Combinational Input/Output is used when the output is connected back to the input of the PAL or if the output pin is used as an input only. Figure 19.12b
- Programmable polarity output is used to either select the output function or its complement by programming an XOR gate at the output. Figure 19.12c

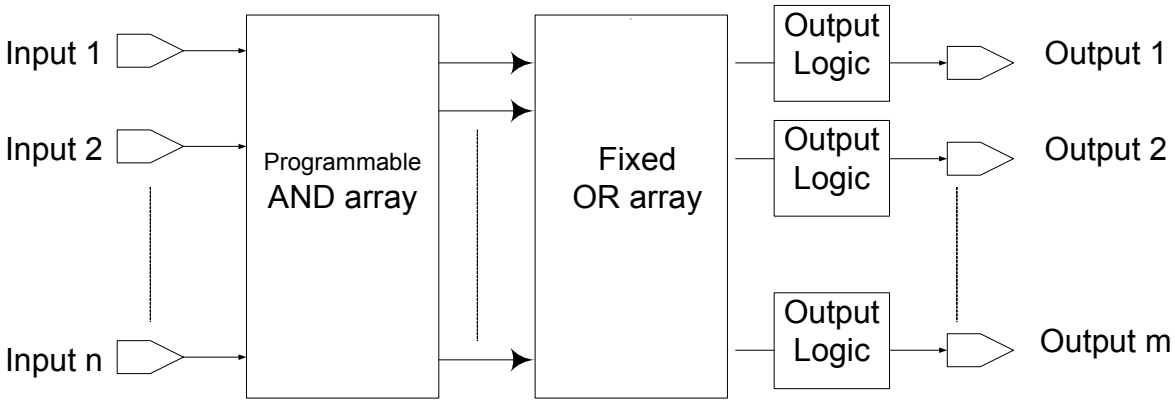


Figure 19.11 Block diagram of a PAL with programmable outputs

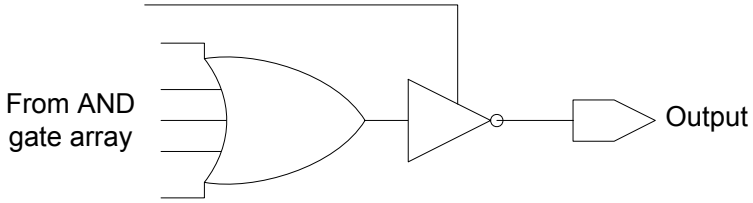


Figure 19.12a Combinational Output with active-low output

The output of the OR gate from the OR gate Array is shown to be connected to a tri-state buffer input. The tri-state buffer can be activated or deactivated through the control line shown connected to its side. The Combinational Output for an SOP function is implemented by activating the tri-state buffer which allows the output of the OR gate to be inverted by the tri-state buffer and passed to the output of the PAL device. An active-high output can be obtained if the PAL device has active-high output tri-state buffers.

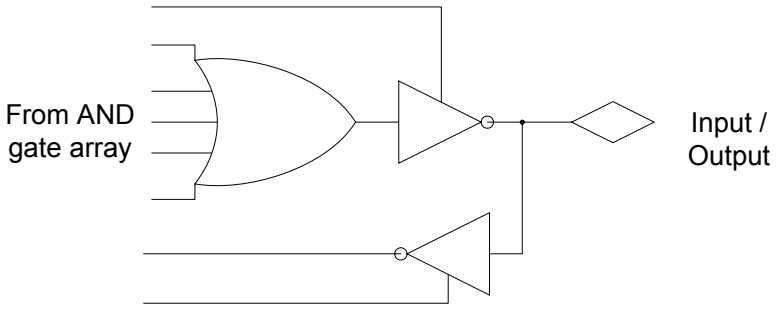


Figure 19.12b Combinational Input/Output with active-low output

The Combinational input/output function is used when the output of the OR gate has to be connected back to the input of the AND Gate. As shown in the figure the output of the tri-state buffer is connected to the input of an inverting and non-inverting buffer which allows the inverted and non-inverted outputs of the OR gate to be connected to the input of the AND gate array. Secondly, by deactivating the tri-state buffer connected at the output of the OR Gate, the output pin is configured as an input pin. External signals connected to the output pin are passed to the input of the AND array.

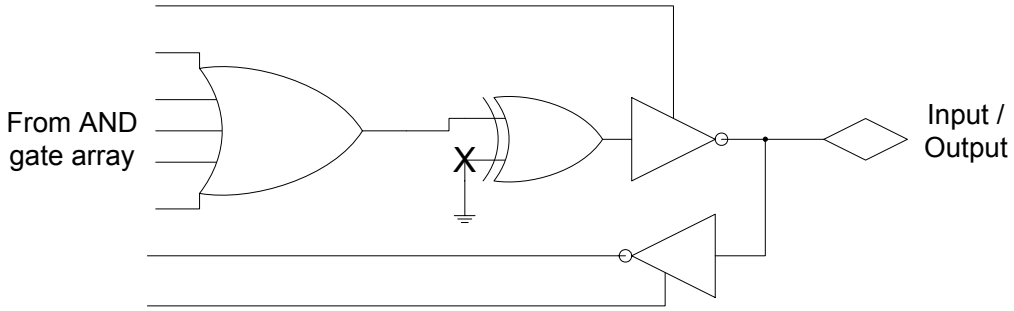


Figure 19.12c Programmed Polarity output

The Programmed Polarity output has the output of the OR gate connected through an XOR gate to the tri-state buffer. The XOR gate allows the output of the OR gate to be set to active-high or active-low. When the second input of the XOR gate is connected to ground, the output of the XOR gate is the same as the output of the OR gate. When the fuse of the XOR gate input is blown to set the input to logic high, the output of the XOR gate is opposite of the OR gate output.

PAL Identification

PALs come in different configurations they are identified by unique number. The numbers begin with the prefix PAL followed by two digits that indicate the number of inputs followed by a letter L active-low, H active-high or P programmable polarity followed by a single or two digits that indicate the number of outputs. In addition to the standard number there may be suffixes which specify the speed, package type and temperature range. Figure 19.13

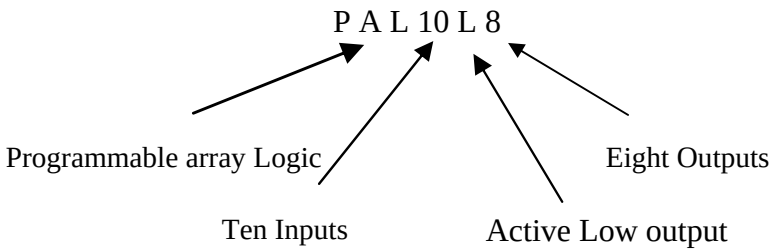


Figure 19.13 Standard PAL Numbering

PLA Circuit and Programming

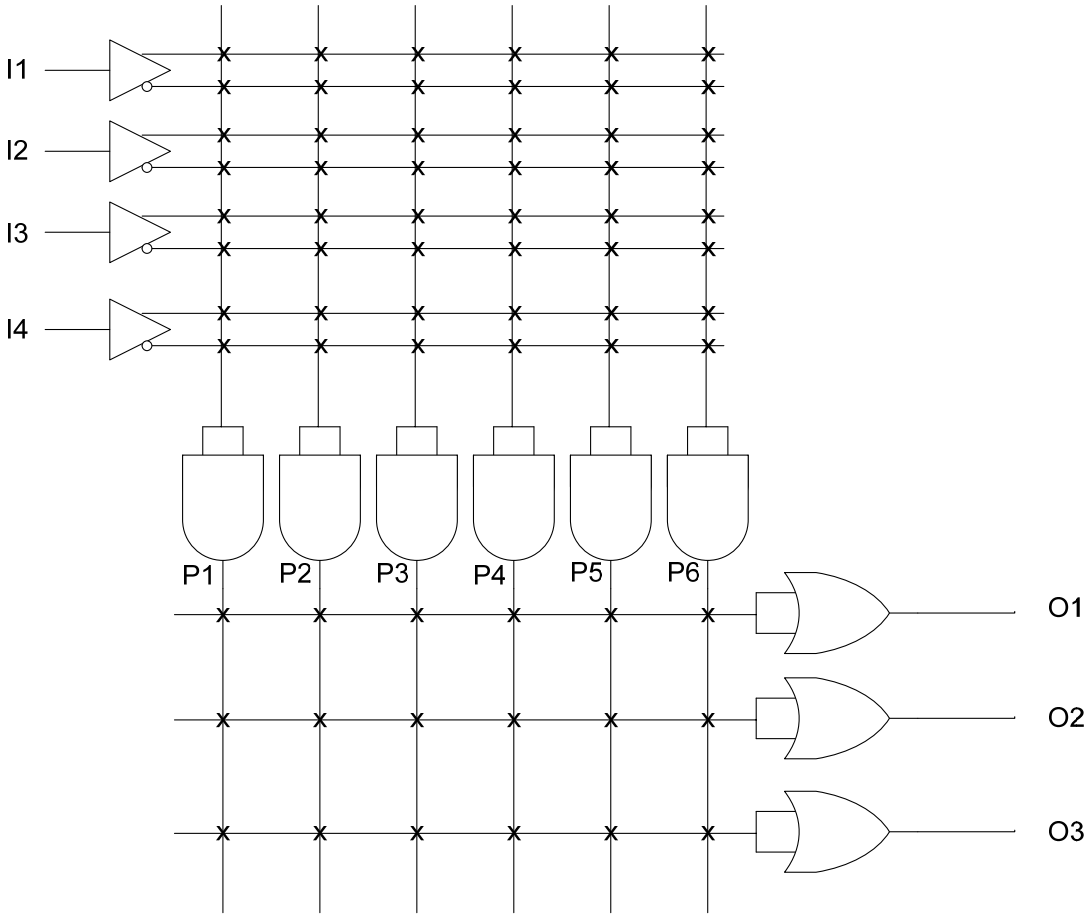


Figure 19.14 A 4 x 3 PLA Device

Programmable Logic Array as mentioned earlier has a programmable AND and OR arrays. A PLA can be programmed to implement any Sum-of-Product logic expressions, limited by the parameters of the PLA device. The limitations are

- Number of inputs (n)
- Number of outputs (m)

- Number of product terms (p)

Such a device is described as an n x m PLA device with p product terms. The simplified diagram of a PLA 4 x 3 device is shown in the figure 19.14. The four inputs I1, I2, I3 and I4 are shown connected through input buffers to the input of the AND gate array. The Input buffers provide the un-complemented and complemented input signals. Each of the 6 AND gates which provide six product terms P1, P2, P3, P4 P5 and P6 have 8 inputs. The outputs of each of the six AND gates are connected to the input of the OR gate array. Each of the three OR gates six inputs. Each OR gate can thus perform a sum operation on six product terms.

The PLA allows both its AND Gate array and the OR gate array to be programmed independently. The 4 x 3 PLA programmed with three separate functions is show in figure 19.15. The product terms generated are

$$P1 = I1.\overline{I2}.I4$$
$$P2 = \overline{I1}.\overline{I2}.I3$$
$$P3 = I1.I2.I3.\overline{I4}$$
$$P4 = I1.I3.I4$$
$$P5 = I2.I4$$
$$P6 = \overline{I1}.I2.\overline{I3}.\overline{I4}$$

The first OR gate sums product terms P1, P2, P3 and P5, the fuses for these product terms are seen to be intact. The second OR gate sums the product terms P2, P4 and P6. The third OR gate sums the product terms P1, P3, P4 and P6. The three sum-of-product terms are

$$O1 = I1.\overline{I2}.I4 + \overline{I1}.\overline{I2}.I3 + I1.I2.I3.\overline{I4} + I2.I4$$
$$O2 = \overline{I1}.\overline{I2}.I3 + I1.I3.I4 + \overline{I1}.I2.\overline{I3}.\overline{I4}$$
$$O3 = I1.\overline{I2}.I4 + I1.I2.I3.\overline{I4} + I1.I3.I4 + \overline{I1}.I2.\overline{I3}.\overline{I4}$$

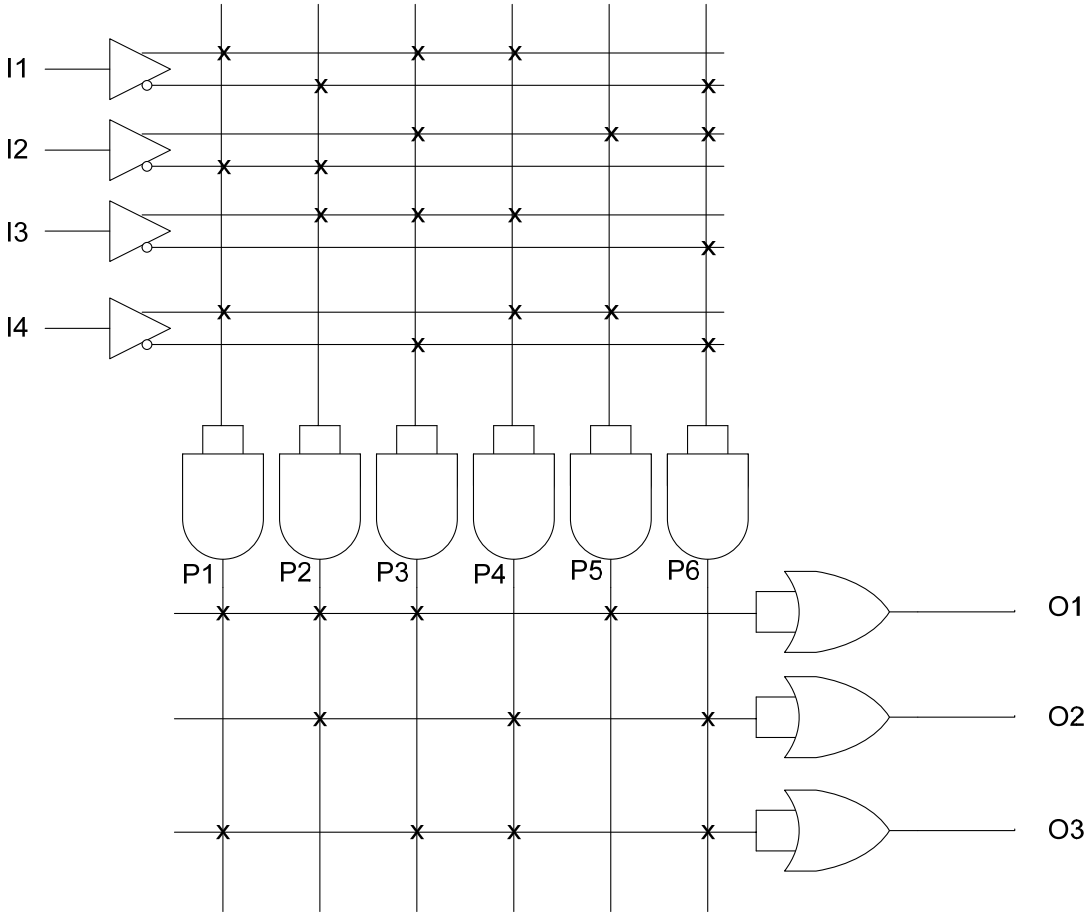


Figure 19.15 Programmed 4 x 3 PLA Device

Lesson No. 20

IMPLEMENTING CONSTANT 0S AND 1S

The PLA can be programmed to give an output of constant 0 or 1. Figure 20.1. All the four inputs and their complements are shown connected to the first AND gate. The product term generated by the AND gate is 0. $P1 = 0$. The P1 product term is connected to the input of first OR gate. Thus the output of OR gate is 0. The inputs to the second AND gate are disconnected, thus the product term generated by the AND gate is a 1. $P2 = 1$. The P2 term is connected to the input of the second OR gate, therefore the output of the second OR gate is a 1. No product term is connected to the input of the third OR gate, therefore the output of the third OR gate is 0.

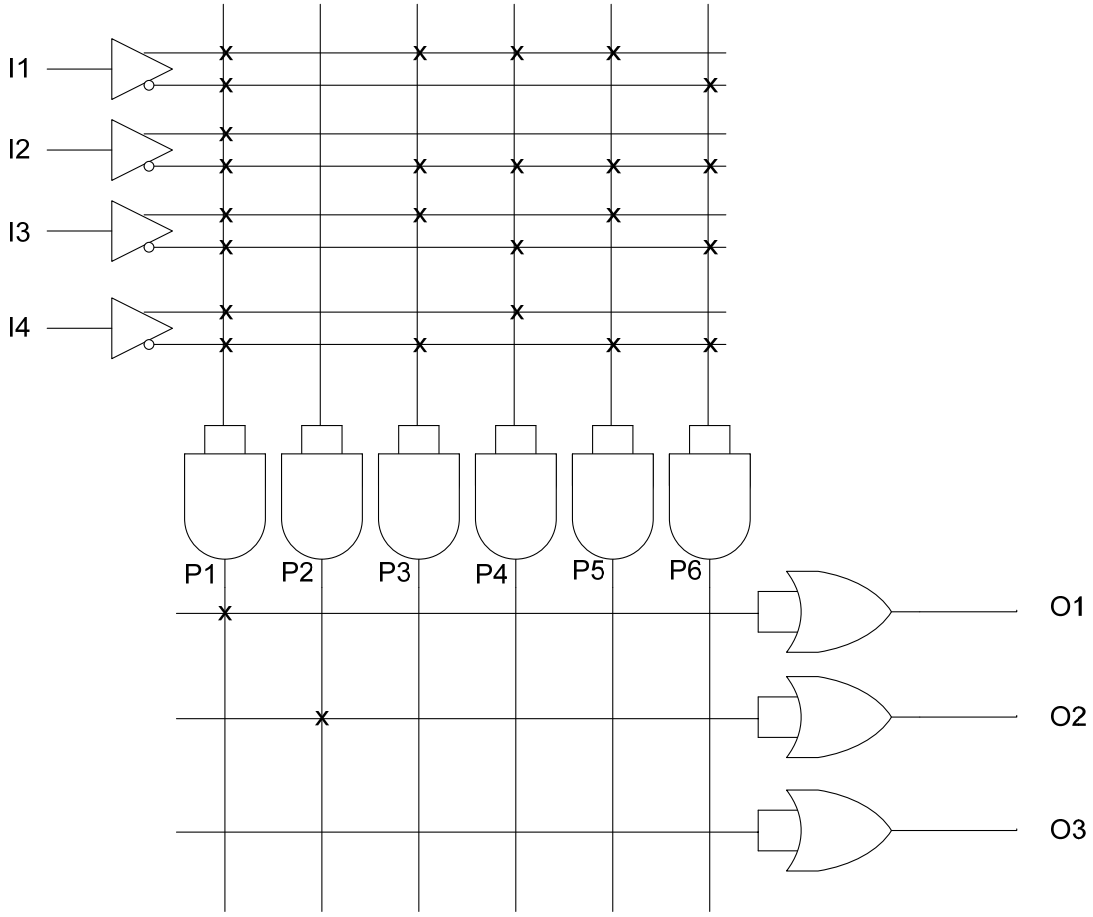


Figure 20.1 4 x 3 PLA Device programmed for 0, 1 and 0 output

Implementing Odd-Prime Number Function

The Odd-Prime Number generator can be implemented by programming the 4 x 3 PLA. Due to the limitations of the PLA which only has six product term (six AND gates), only the first six Odd-Prime numbers 1, 3, 5, 7, 11 and 13 can be detected. Additional two outputs are programmed to detect Odd-Prime multiples of 15 and 39 respectively. The six product terms represented by P1, P2, P3, P4, P5 and P6 are minterms 1, 3, 5, 7, 11 and 13. The first OR gate sums the six minterms (product terms) to give an output of 1 when any one of the first six Odd-Prime numbers is applied at the inputs I1, I2, I3 and I4 of the PLA respectively. The

when any of the three minterms is applied at the PLA inputs. Similarly, the third OR gate sums the minterms 1, 3 and 13 and the output is set to logic 1 when any one of the three inputs are detected at the input of the PLA. Figure 20.2.

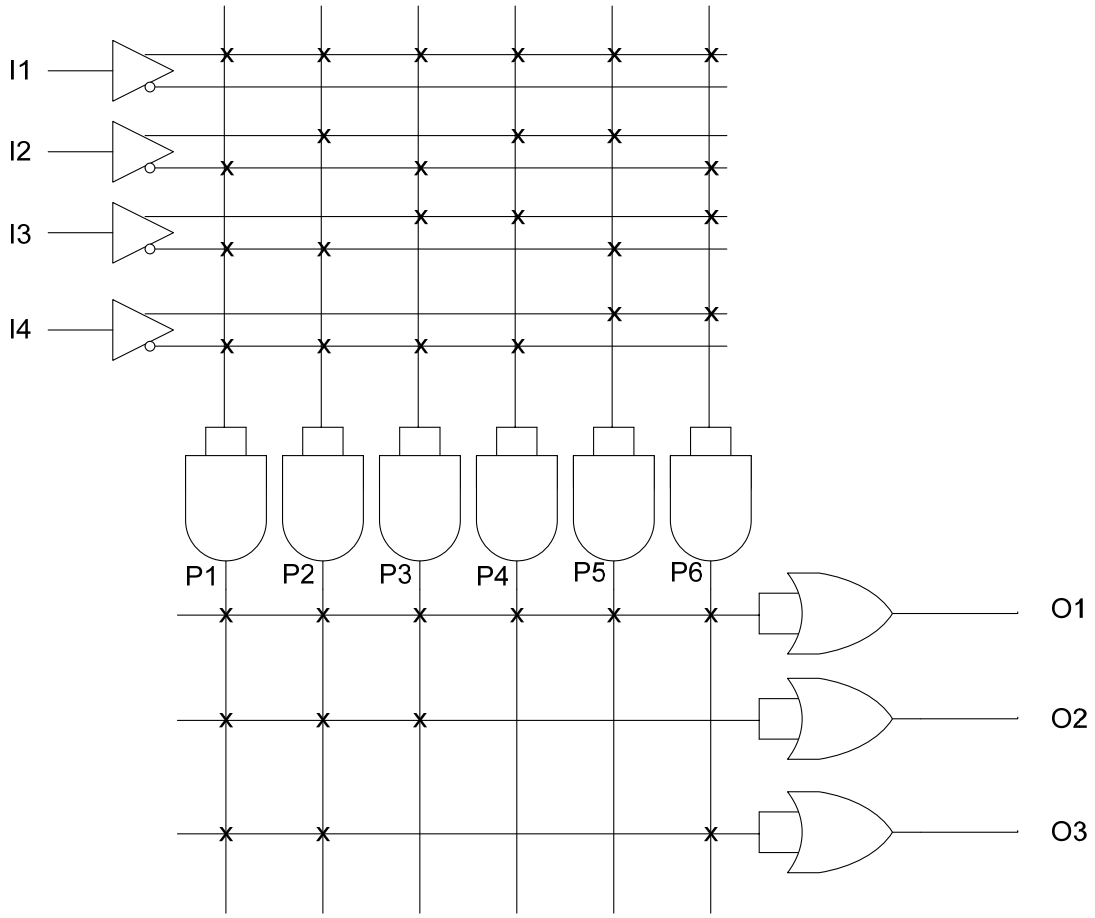


Figure 20.2 4 x 3 PLA Device programmed to Detect Odd-Prime Numbers

GAL Operation

The GAL has a reprogrammable AND gate array and a fixed OR array. GAL can be reprogrammed as instead of fuses E^2CMOS logic is used which can be programmed to connect a column with a row. The E^2CMOS logic at each column–row intersection is known as a cell. Figure 20.3. The E^2CMOS cell in the 'on' state connects the column with the row and a cell in the 'off' state disconnects the column and row. Appropriate cells are programmed to the 'on' state to allow appropriate literals to be connected to the AND gates which generate product terms. The simplified GAL structure shows the implementation of an SOP function. Figure 20.4

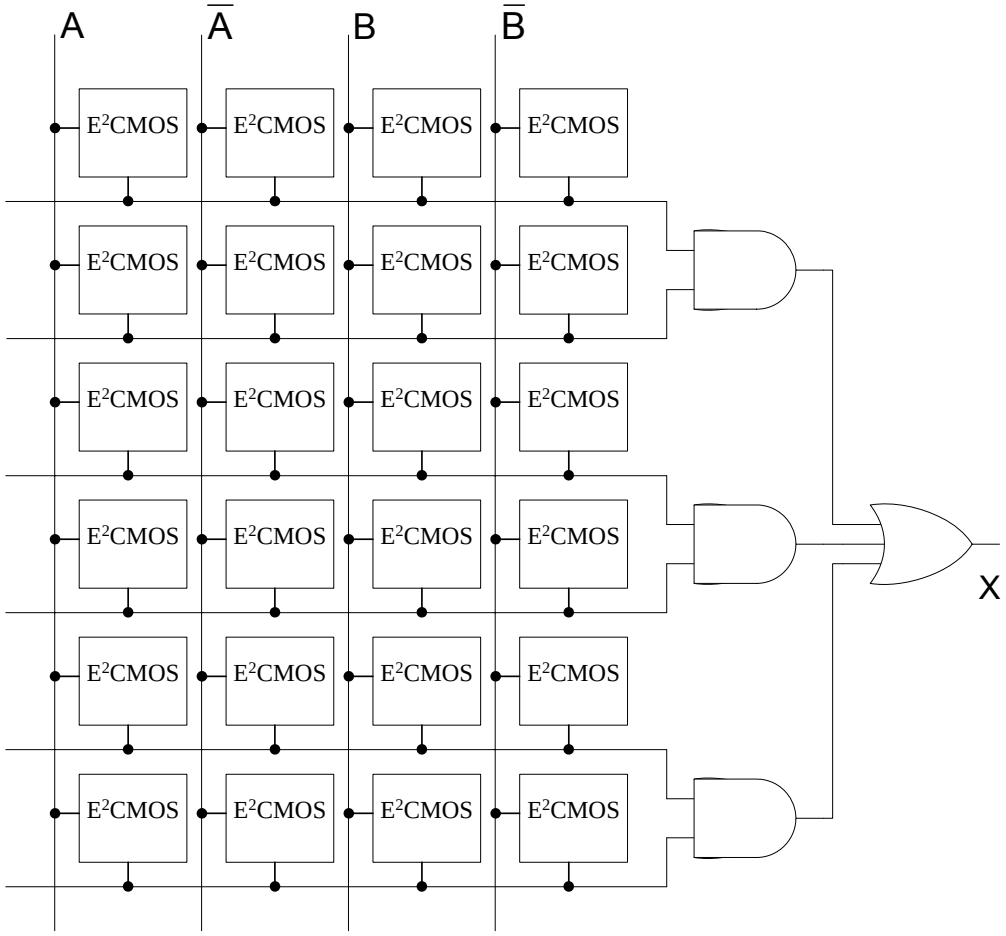


Figure 20.3 Simplified E^2CMOS array structure of GAL

A typical Gal has eight or more inputs to the reprogrammable AND array and 8 or more input/outputs from its 'Output Logic Macro Cells' OLMCs. The OLMCs can be programmed to Combinational Logic or Registered Logic. Combinational Logic is used for combinational circuits, where as Registered Logic is based on Sequential circuits. Figure 20.5

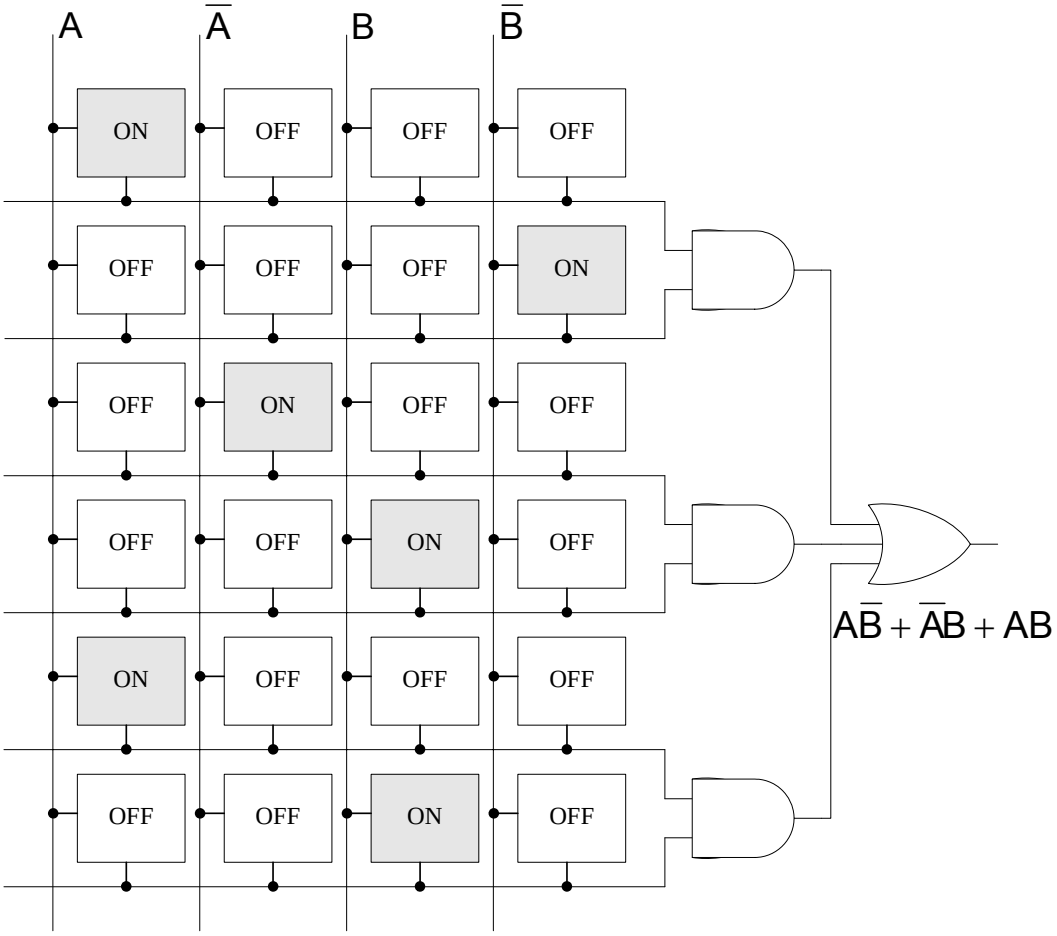


Figure 20.4 GAL implementation of an SOP function

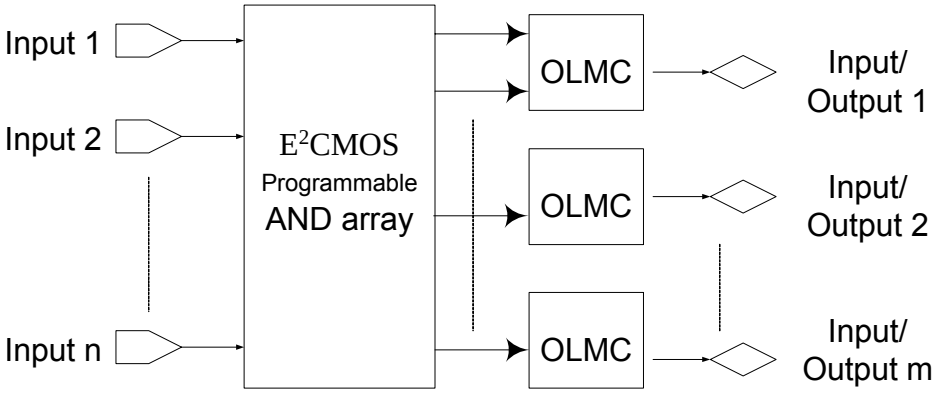


Figure 20.5 Block diagram of a GAL

GALs are also available in a variety of configurations. GALs are identified by a prefix GAL followed by a 2-digt number indicating the number of inputs which is followed by V indicating variable output configuration followed by a number which indicates the number of outputs. Figure 20.6

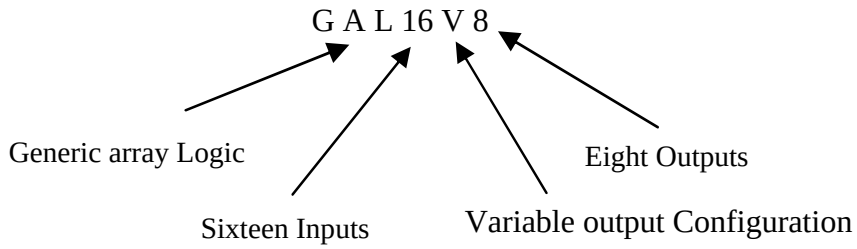


Figure 20.6 Standard GAL Numbering

Programming of PLDs

PLDs are programmed with the help of computer which runs the programming software. The computer is connected to a programmer socket in which the PLD is inserted for programming. PLDs can also be programmed when they are installed on a circuit board.

The programming of a PLD device involves entering the logic function in the form of a Boolean equation, truth table or a state diagram. Any errors during the entry process are corrected. The software compiler processes the information in the input file and translates it into a suitable format. The complier also minimizes the logic. The minimized logic is then tested by using a set of hypothetical inputs known as test vectors. The testing verifies the design of the logic circuit before committing it to the PLD. If any flaws are detected during the testing process the design must be debugged and submitted for recompilation. Once the design has been finalized a documentation file is produced along with a fuse map file which is downloaded to the programmer which programs the PLD device inserted in the programmer socket.

PLDs have In-System Programming (ISP) capability that allows the PLDs to be programmed after they have been installed on a circuit board. A standard 4-wire interface is used for programming the In-System PLD. ISP capability allows systems to be upgraded by reprogramming the PLD.

The GAL22V10

The GAL22V10 is a popular GAL device having twelve inputs and ten inputs/outputs. The device is available as low-voltage 3.3v version. It is also available as an ISP version. The device has ten OLMCs that can be programmed to different output modes. The ten OLMCs receive different number of inputs from the programmable AND gate array. Figure 20.7. Of the ten OLMCs, two have eight inputs, two have ten inputs, two have twelve inputs, two have fourteen and two have sixteen inputs. Each OLMC can be programmed for active-high, active-low output or it can be programmed as an input.

The circuit diagram of an OLMC is shown in figure 20.8. The OLMC consists of a flip-flop which is a sequential logic device which stores the information at the output of the OR gate. Flip-flops will be discussed latter. The output and the complemented output of the flip-flop are connected to the two inputs of the 4-to-1 MUX. The remaining two inputs of the MUX are connected to the OR gate output and its complemented output. The output of the MUX is

a 2-to-1 MUX. The other input of the 2-to-1 MUX is connected to the complemented output of the flip-flop. The output of the 2-to-1 MUX and its complemented output is connected to the input of the AND array. The select inputs S_0 and S_1 select the appropriate 4-to-1 MUX input to be routed to the output or the input. The S_1 select input of the 2-to-1 MUX is used to route the appropriate input to the input of the AND array. The select bits S_0 and S_1 are programmed in a dedicated group of cells in the array which are separate from the logic array cells.

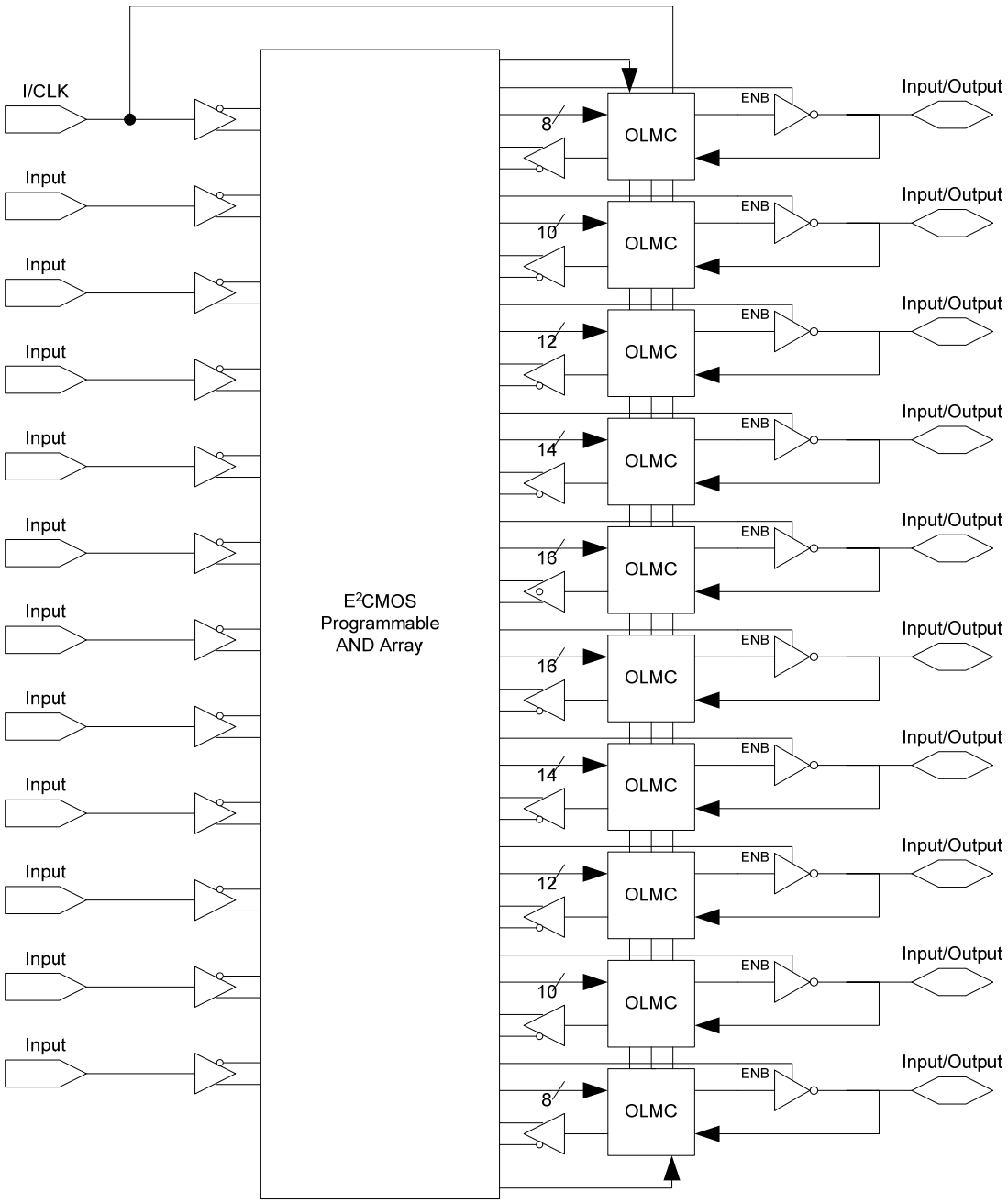


Figure 20.7 Block diagram of the GAL22V10

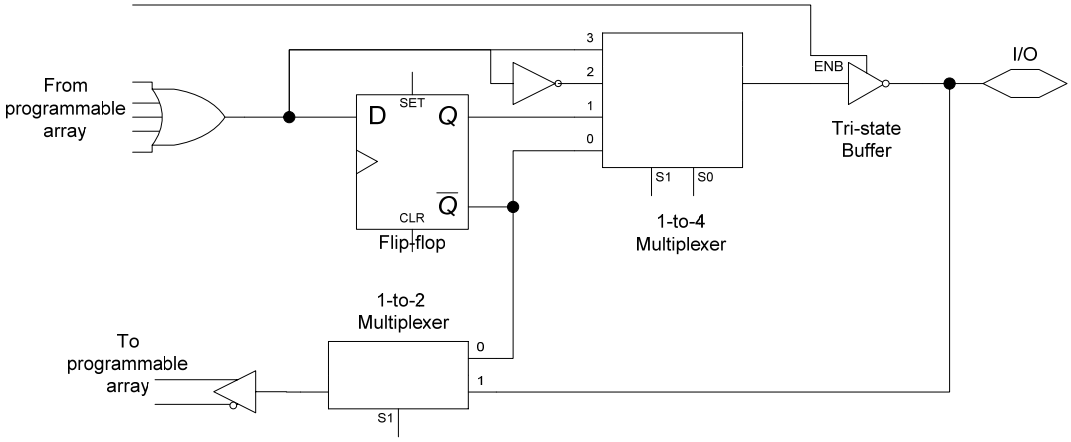


Figure 20.8 Circuit Diagram of OLMC

The four OLMC configurations are

- Combination Mode with active-low output
- Combinational Mode with active-high output
- Registered Mode with active-low output
- Registered Mode with active-high output

OLMC Combinational Mode

When the select inputs S_0 and S_1 are set to 0 and 1 respectively, the 4-to-1 MUX selects the OR gate output and the output is active-low because of the inversion by the tri-state buffer. When the select inputs are set to 1 and 1 respectively, the MUX selects the complement of the OR gate output. The output of the OLMC is active-high due to double inversion.

Tri-State Buffers

Tri-State Buffer is a NOT gate with a control line that disconnects the output from the input. When the control line is high the buffer operates like a NOT gate and when the control line is low the output is disconnected from the output and high impedance is seen at the output. Tri-state buffers are used to disconnect the outputs of devices which are connected or share a common output line. Figure 20.9

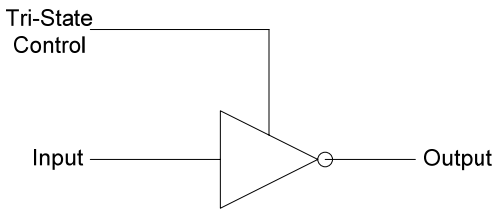


Figure 20.9a Tri-State Buffer

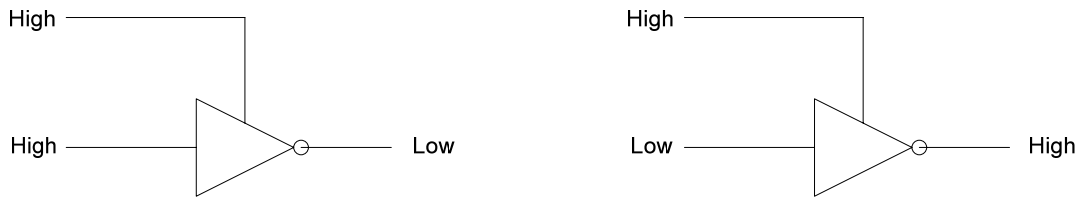


Figure 20.9b Tri-State Buffer operating as a NOT gate

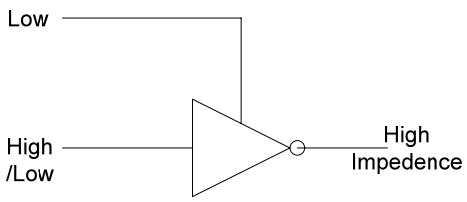


Figure 20.9c Tri-State Buffer in High-Impedence State

Referring to the OLMC logic circuit. Figure 20.8. When the control input to the tri-state buffer is set to low, the output of the buffer is set to high impedance disconnecting the OLMC from the output pin. The output pin is used as an input pin.

The GAL22V10 Array

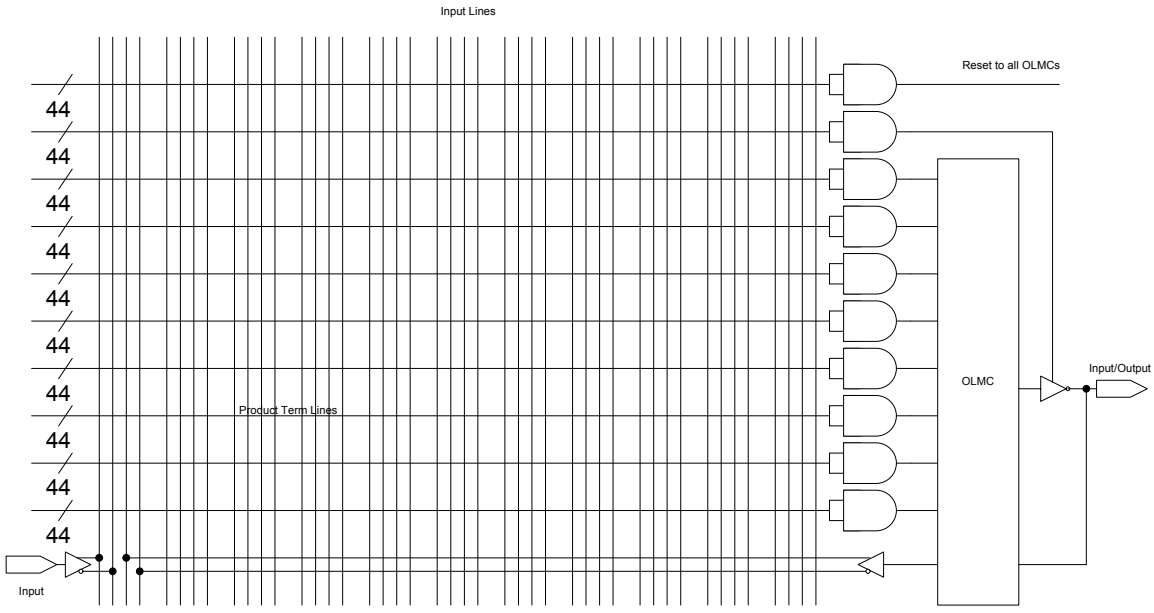


Figure 20.10 Detailed Connection to the first OLMC of GAL22V10

The GAL22V10 has 22 inputs organized as 44 lines, one for each input and its

connection of the first OLMC to the AND array is shown in figure 20.8. The vertical lines in groups of four represent the inputs. Thus the first group of four vertical lines represents the input from the GAL input pin and the input from the OLMC. The horizontal lines represent the product terms. The first OLMC has ten input product terms. Out of the ten product terms, eight product terms are connected to the OR gate in the first OLMC. Out of the remaining two product terms, the first product term is used to control the tri-state buffer and the other is used for reset in the Registered mode for all OLMCs.

Each OLMC ORs the product term to give a single sum of product term. The GAL has ten such OLMCs therefore a total of ten Sum-of-Product terms can be implemented.

Programming the GAL22V10

Figure 20.11 shown the programmed GAL for the Boolean expression

$$X = ABCDEF + A\overline{B}\overline{C}\overline{D}\overline{E}\overline{F} + \overline{A}B\overline{C}\overline{D}\overline{E}\overline{F} + \overline{A}\overline{B}CDE\overline{F} + A\overline{B}CD\overline{E}F + \overline{A}B\overline{C}DEF + \overline{A}\overline{B}\overline{C}DEF$$

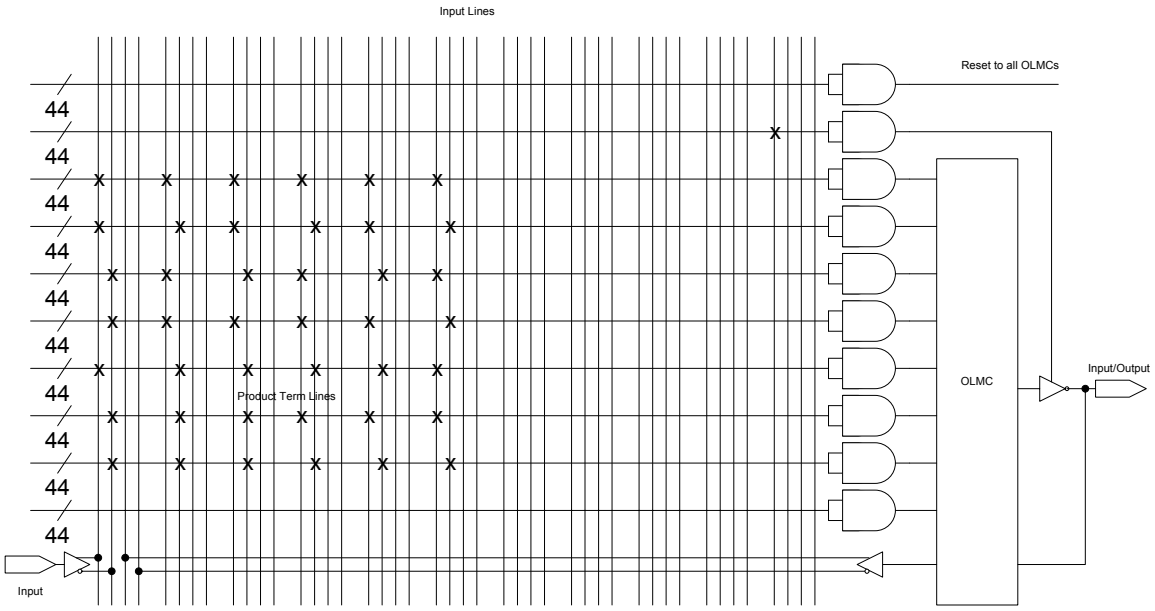


Figure 20.11 GAL22V10 programmed for Boolean Function

In the figure 20.9 the GAL has been programmed for a six variable Boolean function. The six variables are connected at the six inputs of the GA device. The figure shows the connection detail for the first variable A. The first group of four vertical lines represents the variable A and its complement $\overline{A}$. The remaining two lines in the group are not used receive the un-complemented and complemented output from the OLMC. Similarly, the second group of four vertical lines are connected to the second input pin of the GAL which is connected to a signal representing variable B. The next four sets of four vertical lines represent input pins 3, 4, 5 and 6 which are connected to variables C, D, E and F. The Boolean expression that is implemented has seven product terms. The first OLMC has eight input product terms, thus it can be used to program the Boolean expression. The output of the first AND gate generates the first product term of the Boolean expression. Similarly, the 2^{nd} to 7^{th} AND gates generate the remaining six product terms respectively. The eight input OR gate (not shown) in the OLMC block generates the sum of product terms. The last group of vertical lines is used to control the tri-state buffer connected at the output of the OLMC. The diagram shows that it has

been set to high to allow the tri-state buffer connect the OLMC output to the output pin of the GAL.

Introduction to ABEL

ABEL which is an acronym for Advanced Boolean Expression Language is a hardware description language used for implementing logic designs using PLDs. ABEL is a device-independent language and can be used to program any type of PLD.

ABEL is run on a computer connected to a PLD programmer which programs the PLD.
ABEL provides three different text-based methods for describing and entering a logic design.
The three methods are
ÅBoolean Equations
ÅTruth Tables
ÅState Diagrams
The Boolean Equations and the Truth Table method are used for Combinational Logic Circuits. The State Diagram is used specifically for Sequential Logic circuits. The Boolean Equations and the Truth Table method can also be used for describing and entering Sequential Logic Circuits.

Lesson No. 21

THE GAL16V8

This device has eight inputs, two special function input pins and eight pins that can be used as inputs or output. The architecture of the GAL16V8 is similar to that of a PAL and it is designed to be programmed in one of the three available modes to emulate most of the existing PALs, thus replacing the PAL. The three modes in which PALs are programmed are

- Simple
- Complex
- Registered

The simple and complex modes are associated with the Combinational Logic whereas the Registered mode is associated with Sequential Logic.

The GAL16V8 has eight OLMCs each connected to eight product terms. Each product term is implemented using a 32-bit input AND gate. The 32 inputs comprise of the 16 complemented and un-complemented inputs of the 8 input pins and 16 complemented and un-complemented inputs of the 8 input/output pins that can be configured as input pins.

OLMC for GAL16V8

The OLMC of the GAL16V8 is similar to the OLMC of the GAL22V10 with some enhancements. The main aspects of the GAL16V8 OLMC are

Tri-state Buffer and OLMC output pin

The tri-state buffer connecting the output of the OLMC circuit to the output pin is controlled through four different sources. The tri-state buffer control input can be connected in four different ways.

1. Connected to V_{cc}. The output is always enabled.
2. Connected to GND. The output is disabled and the output pin is configured as an input pin.
3. Connected to the external pin (11) which can be connected to V_{cc} or GND. The tri-state buffer is therefore controlled externally by applying an appropriate signal at the pin.
4. Connected to the output of one of the eight AND gates connected to the OLMC. Thus the tri-state buffer is controlled by a logical expression.

The feedback from the OLMC to the AND Gate array input

The OLMC can be configured to provide a feedback input signal to the AND gate array input. There are three possibilities.

1. Connecting the feedback signal line to the output of the OLMC. This allows the output of the OLMC to be connected back to the AND gate array input. This allows implementation of Sequential Logic circuits.
2. Connecting the feedback signal line to the output of the adjacent OLMC. This also allows implementation of Sequential Logic circuits.
3. Connecting the feedback signal line to a flip-flop. This allows implementation of synchronized Sequential circuits.

The output of the Sum of Product term

The OR gate used to implement the Sum-of-Product term has its output connected to the output pin thorough the tri-state buffer. The tri-state buffer is also connected to the output from the flip-flop. Thus either of the two inputs to the tri-state buffer can be selected. The output of the OR gate can also be programmed for output polarity by configuring the XOR gate connected at the output of the OR gate.

Simple Mode

In the Simple Mode the OLMC is configured as dedicated active combinational output or as dedicated input (limited to six). Three possible combinations of the Simple Mode are

- Combinational Output. Figure 21.1
- Combinational Output with feedback to AND Array. Fig 21.2
- Dedicated input. Fig 21.3

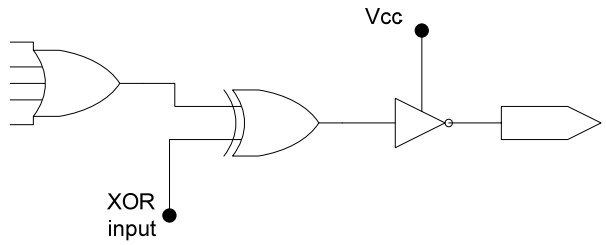


Figure 21.1 Combinational Output

In the Combinational Output the OLMC is configured to give an output which is either active-low or active-high. The active-state of the output is determined by the XOR input. The tri-state buffer control pin is set to logic high by connecting it to V_{cc}. The Sum-of-Product term generated by the OR gate has eight product terms.

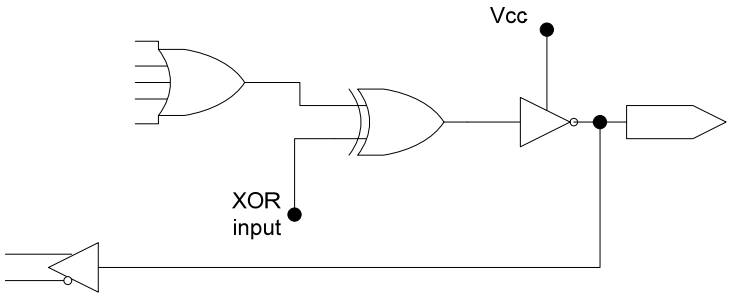


Figure 21.2 Combinational Output with feedback to AND array

The Combinational Output with feedback to AND array is similar. The tri-state control pin is set to logic high (V_{cc}), the XOR gate input determines the active-state of the output. The signal at the output is also connected to the input of the AND array through the buffer which provides inverting and non-inverting outputs. The feedback capability is limited to six OLMCs as they have a physical connection from the tri-state buffer output to the AND gate array input. OLMCs connected to input/output pins 15 and 16 do not have the feedback path therefore they can not be programmed with Combinational output with feedback.

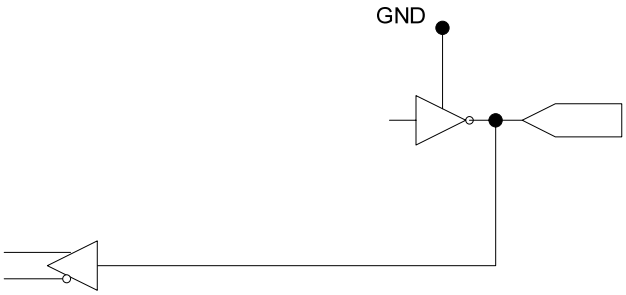


Figure 21.3 Dedicated Input

In the Dedicated Input configuration the tri-state buffer is configured in the high impedance state by setting the control pin of the tri-state buffer to low (GND). Thus the output pin is connected to an input signal which is passed to the input of the AND Array in its complemented and un-complemented form by the buffer.

Complex Mode

In this mode the OLMCs can be configured in two ways. In the complex Mode the tri-state control is formed by a logical expression, this leaves seven product terms that can be used to form a sum-of product expression. Two possible combinations of the Complex Mode are

- Combinational Output. Fig. 21.4
- Combinational Input/Output. Fig. 21.5

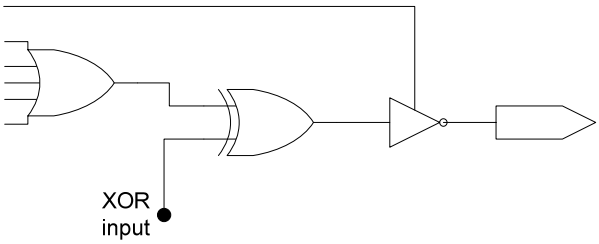
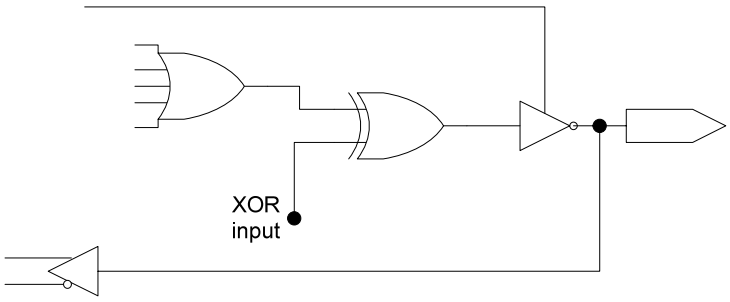


Figure 21.4 Combinational Output

The tri-sate buffer is enabled by connecting the control input of the buffer to the output of one of the AND gate. Thus the tri-state buffer is controlled by programming a product term. Similarly, the Combinational Input/Output Mode is also implemented by connecting the tri-state buffer control input to the output of the AND gate. OLMCs which have the feedback path connecting the output to the input of the AND gate array can be used in this mode.



Introduction to ABEL

ABEL which is an acronym for Advanced Boolean Expression Language is a hardware description language used for implementing logic designs using PLDs. ABEL is a device-independent language and can be used to program any type of PLD. ABEL is run on a computer connected to a PLD programmer which programs the PLD.

ABEL provides three different text-based methods for describing and entering a logic design. The three methods are

- Boolean Equations
- Truth Tables
- State Diagrams

The Boolean Equations and the Truth Table method are used for Combinational Logic Circuits. The State Diagram is used specifically for Sequential Logic circuits. The Boolean Equations and the Truth Table method can also be used for describing and entering Sequential Logic Circuits.

Boolean Operations and Boolean Notations

The NOT, AND, OR and XOR operations have special symbols in ABEL as shown in table 21.1

Logic Operation	ABEL Symbol
NOT	!
AND	&
OR	#
XOR	$

Table 21.1 ABEL Symbols for logic operations

The standard Boolean notations in terms of ABEL notations are defined in table 20.2. The operators !, &, # and $ have precedence in the order given in table.

Boolean Notation	ABEL Notation
$\overline{A}$	!A
A.B	A&B
$A + B$	A#B
$A \oplus B$	A$B

Table 21.2 Boolean and equivalent ABEL Notations

3. Boolean Equations

One of the ABEL entry methods uses logic equations. In ABEL any letter or combination of letters and numbers can be used to identify variables. ABEL however is case-sensitive, thus variable ‘A’ is treated separately from variable ‘a’. All ABEL equations must end with ‘;’. Figure 21.6

Boolean expression $F = A\overline{B} + AC + \overline{B}\overline{D}$

ABEL expression F = A & !B # A & C # !B & !D;

Multiple Inputs and Outputs

In some cases, multiple input and output variables can be grouped as a set to simplify an equation. Fig 21.7. Thus D_0, D_1 and D_2 input or output variables can be defined by a single variable D using the ABEL notation D = [D0, D1, D2];

A 4-input 4-bit Multiplexer is represented by the function table 21.3. The Boolean expressions representing the operation of the MUX are shown in figure 21.7.

Select Inputs		Outputs			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	A_3	A_2	A_1	A_0
0	1	B_3	B_2	B_1	B_0
1	0	C_3	C_2	C_1	C_0
1	1	D_3	D_2	D_1	D_0

Table 21.3 Truth Table of 4-input 4-bit MUX

$$Y_3 = A_3\overline{S_1}\,\overline{S_0} + B_3\overline{S_1}S_0 + C_3S_1\overline{S_0} + D_3S_1S_0$$
$$Y_2 = A_2\overline{S_1}\,\overline{S_0} + B_2\overline{S_1}S_0 + C_2S_1\overline{S_0} + D_2S_1S_0$$
$$Y_1 = A_1\overline{S_1}\,\overline{S_0} + B_1\overline{S_1}S_0 + C_1S_1\overline{S_0} + D_1S_1S_0$$
$$Y_0 = A_0\overline{S_1}\,\overline{S_0} + B_0\overline{S_1}S_0 + C_0S_1\overline{S_0} + D_0S_1S_0$$

Figure 21.7 Boolean expressions representing a 4-input 4-bit MUX

The ABEL notations representing the operation of the MUX are shown in figure 21.8.

```
Y3 = A3 & !S1 & !S0 # B3 & !S1 & S0 # C3 & S1 & !S0 # D3 & S1 & S0;
Y2 = A2 & !S1 & !S0 # B2 & !S1 & S0 # C2 & S1 & !S0 # D2 & S1 & S0;
Y1 = A1 & !S1 & !S0 # B1 & !S1 & S0 # C1 & S1 & !S0 # D1 & S1 & S0;
Y0 = A0 & !S1 & !S0 # B0 & !S1 & S0 # C0 & S1 & !S0 # D0 & S1 & S0;
```

Figure 21.8 ABEL notations representing a 4-input 4-bit MUX

The four ABEL notations can be represented by a single notation if variables A3, A2, A1 and A0 are defined as a set A. Similarly, sets B, C and D can be defined. Figure 21.9

```
A = [A3, A2, A1, A0];
B = [B3, B2, B1, B0];
C = [C3, C2, C1, C0];
D = [D3, D2, D1, D0];
Y = [Y3, Y2, Y1, Y0];
S = [S1, S0];
```

The ABEL notation representing the MUX is

```
Y = (S = = 0) & A # (S = = 1) & B # (S = = 2) & C # (S = = 3) & D;
```

The '= =' is a relational operator

Figure 21.9 ABEL representation of multiple inputs and outputs

4. Truth Table

ABEL accepts a logical design described in the form of a Truth Table. Truth Tables are sometimes more convenient in describing certain logic circuits. The ABEL Truth Table format includes a header and the truth table entries.

```
TRUTH_TABLE     ( [ A, B, C, D] → [ X1, X2])
```

A, B, C and D are the inputs and XI and X2 are the outputs.

The truth table of an XOR gate is represented by the ABEL Truth Table notation. Figure 21.10.

```
TRUTH_TABLE     ( [A, B] → [X])
                    [0, 0] → [0];
                    [0, 1] → [1];
                    [1, 0] → [1];
                    [1, 1] → [0];
```

Figure 21.10 ABEL representation of the Truth table of an XOR gate

The 2-bit Comparator logic circuit can be described in terms of the truth table using ABEL notations. Fig 21.11

```
TRUTH_TABLE     (  [A1, A0, B1, B0] → [G, E, L] )
                   [0, 0, 0, 0] → [0, 1, 0];
                   [0, 0, 0, 1] → [0, 0, 1];
                   [0, 0, 1, 0] → [0, 0, 1];
                   [0, 0, 1, 1] → [0, 0, 1];
                   [0, 1, 0, 0] → [1, 0, 0];
                   [0, 1, 0, 1] → [0, 1, 0];
                   [0, 1, 1, 0] → [0, 0, 1];
                   [0, 1, 1, 1] → [0, 0, 1];
                   [1, 0, 0, 0] → [1, 0, 0];
                   [1, 0, 0, 1] → [1, 0, 0];
                   [1, 0, 1, 0] → [0, 1, 0];
                   [1, 0, 1, 1] → [0, 0, 1];
                   [1, 1, 0, 0] → [1, 0, 0];
                   [1, 1, 0, 1] → [1, 0, 0];
                   [1, 1, 1, 0] → [1, 0, 0];
                   [1, 1, 1, 1] → [0, 1, 0];
```

Figure 21.11 ABEL representation of the Truth table of a 2-bit Comparator

The ABEL notation can be rewritten by defining a set. Fig 21.12

```
INPUT = [A1, A0, B1, B0];

TRUTH_TABLE     (  INPUT → [G, E, L] )
                   0   → [0, 1, 0];
                   1   → [0, 0, 1];
                   2   → [0, 0, 1];
```

```
4  → [1, 0, 0];
5  → [0, 1, 0];
6  → [0, 0, 1];
7  → [0, 0, 1];
8  → [1, 0, 0];
9  → [1, 0, 0];
10 → [0, 1, 0];
11 → [0, 0, 1];
12 → [1, 0, 0];
13 → [1, 0, 0];
14 → [1, 0, 0];
15 → [0, 1, 0];
```

Figure 21.12 ABEL representation of a Truth Table of a 2-bit Comparator using a set

Test Vectors

Once the Logic circuit design has been entered its operation is verified by using 'test vectors'. A 'test vector' specifies the inputs and the corresponding outputs. The software simulates the operation of the logic circuit by applying the test vector and checking the outputs. Test vectors are essentially the same as Truth Tables. Figure 21.13

```
TEST_VECTORS  ( [A1, A0, B1, B0] → [G, E, L] )
                [0, 0, 0, 0] → [0, 1, 0];
                [0, 0, 0, 1] → [0, 0, 1];
                [0, 0, 1, 0] → [0, 0, 1];
                [0, 0, 1, 1] → [0, 0, 1];
                [0, 1, 0, 0] → [1, 0, 0];
                [0, 1, 0, 1] → [0, 1, 0];
                [0, 1, 1, 0] → [0, 0, 1];
                [0, 1, 1, 1] → [0, 0, 1];
                [1, 0, 0, 0] → [1, 0, 0];
                [1, 0, 0, 1] → [1, 0, 0];
                [1, 0, 1, 0] → [0, 1, 0];
                [1, 0, 1, 1] → [0, 0, 1];
                [1, 1, 0, 0] → [1, 0, 0];
                [1, 1, 0, 1] → [1, 0, 0];
                [1, 1, 1, 0] → [1, 0, 0];
                [1, 1, 1, 1] → [0, 1, 0];
```

Figure 21.13 Test Vector of a 2-bit Comparator

```
INPUT = [A1, A0, B1, B0];
TEST_VECTORS  ( INPUT → [G, E, L] )
                0  → [0, 1, 0];
                1  → [0, 0, 1];
                2  → [0, 0, 1];
                3  → [0, 0, 1];
                4  → [1, 0, 0];
                5  → [0, 1, 0];
                6  → [0, 0, 1];
```

8 → [1, 0, 0];
9 → [1, 0, 0];
10 → [0, 1, 0];
11 → [0, 0, 1];
12 → [1, 0, 0];
13 → [1, 0, 0];
14 → [1, 0, 0];
15 → [0, 1, 0];

Figure 21.14 Test Vector of a 2-bit Comparator using a set

The ABEL Input File

When an Input (source) file is created in ABEL a module is created which has three sections. The three sections are

4. Declarations

The declaration section generally includes the device declaration, pin declarations and set declarations. Figure 21.15. Device declaration is used to specify the PLD device that is to be programmed. The device is referred to as the target device.

```
Decoder device 'P22V10';

A0, A1, A2, A3, PIN 1, 2, 3, 4;

INPUT = [A1, A0, B1, B0];
```

Figure 21.15 ABEL Input declarations

The 'Decoder' is a description which can be anything defined by the user
The 'device' is a reserved keyword which can be in lower or upper case.
The 'P22V10' is the device name. It should be in the format shown.

'PIN" is a keyword which can be in lower or upper case.
Pin declaration defines the relationship between the variables and the corresponding pin numbers of the PLD.

'INPUT' defines a set made up of set elements A1, A0, B1 and B0. In subsequent ABEL notations the set 'INPUT' can be used instead of set variables.

5. Logic Descriptions

Logic descriptions include the three methods of describing a logic circuit. Two methods the Boolean equation and the Truth Table method already have been discussed.

6. Test Vectors

The Test Vector format has been described. The Test vector description is used to simulate the logic circuit and verify its operation.

The Documentation file

After an input file is processed by ABEL a documentation file is generated which provides a hardcopy of the final reduced equations, a JEDEC file and a device pin diagram.

The JEDEC file

The JEDEC file is downloaded to the PLD programmer to program the appropriate PLD device.

Lesson No. 22

ABEL INPUT FILE OF A QUAD 1-OF-4 MUX

A Quad 1-of-4 MUX has four Multiplexers, each Multiplexer has four inputs and a single output. Each multiplexer has two select inputs to select one of the four inputs. The two select inputs are common to all the four multiplexers. The function table of the Quad 1-of-4 MUX is shown in table 22.1.

Select Inputs		Outputs			
S_1	S_0	D_{out}	C_{out}	B_{out}	A_{out}
0	0	D_0	C_0	B_0	A_0
0	1	D_1	C_1	B_1	A_1
1	0	D_2	C_2	B_2	A_2
1	1	D_3	C_3	B_3	A_3

Table 22.1 Truth table of a Quad 1-of-4 Multiplexer

```
Module quad_1of4_mux

Title          'Quad 1 of 4 multiplexer in a GAL20V8'

      mux    device          'P20V8';

      A0, A1, A2, A3                pin 1, 2, 3, 4;
      B0, B1, B2, B3                pin 5, 6, 7, 8;
      C0, C1, C2, C3                pin 9, 10, 11, 13;
      D0, D1, D2, D3                pin 14, 15, 16, 17;
      Aout, Bout, Cout, Dout        pin 21, 20, 19, 18;
      S0, S1                        pin 22, 23;

Equations
      Aout = !S1 & !S0 & A0 # !S1 & S0 & A1 # S1& !S0 & A2 # S1 & S0 & A3;
      Bout = !S1 & !S0 & B0 # !S1 & S0 & B1 # S1& !S0 & B2 # S1 & S0 & B3;
      Cout = !S1 & !S0 & C0 # !S1 & S0 & C1 # S1& !S0 & C2 # S1 & S0 & C3;
      Dout = !S1 & !S0 & D0 # !S1 & S0 & D1 # S1& !S0 & D2 # S1 & S0 & D3;

Test_vectors
([S1, S0, A0, A1, A2, A3, B0, B1, B2, B3, C0, C1, C2, C3, D0, D1, D2, D3] →
[Aout, Bout, Cout, Dout])

"S  S  A  A  A  A  B  B  B  B  C  C  C  C  D  D  D  D  outputs
"0  1  0  1  2  3  0  1  2  3  0  1  2  3  0  1  2  3              A  B  C  D

[0, 0,  1,  0,  0,  0, 0,  1,  0, 0,  0,  0,  1, 0,  0,  0, 0,  1]  →   [1,  0,  0,  0];
[0, 1,  1,  0,  0,  0, 0,  1,  0, 0,  0,  0,  1, 0,  0,  0, 0,  1]  →   [0,  1,  0,  0];
[1, 0,  1,  0,  0,  0, 0,  1,  0, 0,  0,  0,  1, 0,  0,  0, 0,  1]  →   [0,  0,  1,  0];
[1, 1,  1,  0,  0,  0, 0,  1,  0, 0,  0,  0,  1, 0,  0,  0, 0,  1]  →   [0,  0,  0,  1];
[0, 0,  1,  1,  1,  0, 1,  1,  0, 1,  1,  0,  1, 1,  0,  1, 1,  1]  →   [1,  1,  1,  0];
[0, 1,  1,  1,  1,  0, 1,  1,  0, 1,  1,  0,  1, 1,  0,  1, 1,  1]  →   [1,  1,  0,  1];
[1, 0,  1,  1,  1,  0, 1,  1,  0, 1,  1,  0,  1, 1,  0,  1, 1,  1]  →   [1,  0,  1,  1];
[1, 1,  1,  1,  1,  0, 1,  1,  0, 1,  1,  0,  1, 1,  0,  1, 1,  1]  →   [0,  1,  1,  1];
END
```

Implementation of Quad MUX

The Quad Multiplexer has 16 inputs, 4 inputs for each Multiplexer. Each multiplexer has a single output, therefore a total of 4 outputs are required. To select an appropriate multiplexer input there are two select input lines connected to all the four multiplexers. The Quad Multiplexer has a total of 22 pins through which the device is operated. The GAL16V8 device can not be used as it does not enough pins to implement the quad multiplexer. The GAL20V8 PLD is used for the implementation of the Quad 1-of-4 Multiplexer. The device has 12 inputs, 2 special function inputs and 8 input/output pins. Four input/output pins of the GAL device are configured as inputs to support the fourth multiplexer inputs D1, D2 and D3 and the select input S0.

Each Multiplexer output Aout , Bout, Cout and Dout is represented by a Sum-of-product Boolean expression, each having four product terms. Refer to figure 21.16. Thus each of the four OLMCs which are connected to the four output pins have four product terms connected to the inputs of the OR gates. The implementation of the multiplexer function Aout is shown in figure 22.2.

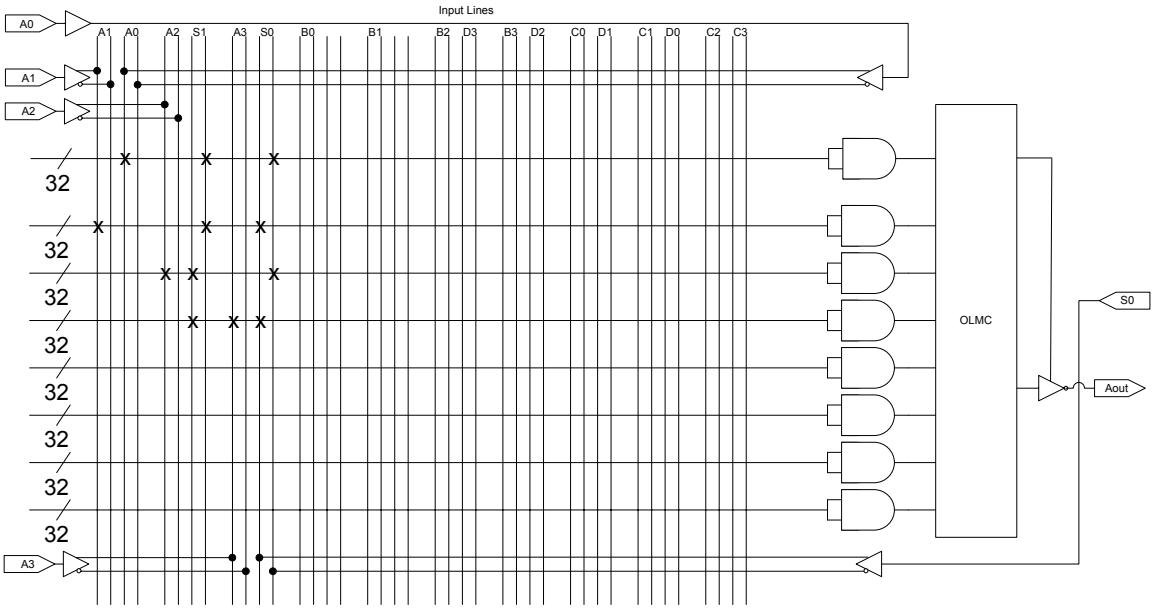


Figure 22.2 Implementation of 1-of-4 Multiplexer

Sequential Circuits

The combinational digital circuits have no storage element; therefore combinational circuits handle only instantaneous inputs. The outputs of the combinational circuits also can not be stored. The absence of a memory element restricts the use of digital combinational circuits to certain application areas. The use of a memory element which is capable of storing digital inputs and outputs is an important part of all practical digital circuits.

Consider an ALU which performs Arithmetic and Logical operations. An ALU can not perform its operations unless it is connected to memory elements that store the inputs applied at the inputs of the ALU and outputs from the ALU. Consider an ALU that performs addition

therefore the ALU has to add the four numbers two at a time. The four numbers have to be stored temporarily, the partial results after adding two numbers also need to be stored. To add the four numbers, the first two numbers 2 and 3 stored in two separate memory elements are added together, the result (5) has to be added to the next number 4. The result (5) is temporarily stored in one of the two memory elements used to store the numbers 2 and 3. The result (5) is added to the third number 4 to provide another partial sum result 9 which has to be stored and then added with the fourth number 5.

In a parallel-to-serial conversion of byte data using a multiplexer and the conversion from serial-to-parallel using a demultiplexer, memory elements are required that store the byte data at the input of the multiplexer for conversion into serial information and another memory element at the output of the demultiplexer for conversion back to parallel.

The counter circuit used in digital circuits count to the next value because of the memory element which stores and remembers the previous count value. A counter can not operate without a memory element.

Digital circuits that use memory elements for their operation are known as Sequential circuits. Thus Sequential circuits are implemented by combining combinational circuits with memory elements.

Latches and Flip-Flops

A latch is a temporary storage device that has two stable states. A latch output can change from one state to the other by applying appropriate inputs. A latch normally has two inputs, the binary input combinations at the latch input allows the latch to change its state. A latch has two outputs Q and its complement $\overline{Q}$. The latch is said to be in logic high state when Q=1 and $\overline{Q}$=0 and it is in the logic low state when Q=0 and $\overline{Q}$=1. When the latch is set to a certain state it retains its state unless the inputs are changed to set the latch to a new state. Thus a latch is a memory element which is able to retain the information stored in it.

The NAND gate based S-R (Set-Reset) Latch

An S-R Latch is implemented by connecting two NAND gates together. The output of each NAND gate is connected to the input of the other NAND gate. The unconnected inputs of the two NAND gates are the Set S and Reset R inputs. The outputs of the two NAND gates are the Q and its complement $\overline{Q}$. The circuit diagram of the NAND based S-R latch is shown in figure 22.3

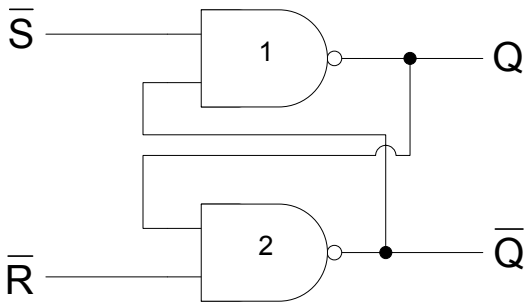


Figure 22.3 NAND based S-R Latch

The S-R latch has two inputs, therefore four different combinations of inputs can be applied to control the operation of the S-R latch. The four possible input combinations are

1. Inputs S=0 & R=0

a. Assume that the outputs Q and $\overline{Q}$ are set at logic 1 and logic 0 respectively. Since both the inputs S and R are logic low, therefore both the Q and $\overline{Q}$ outputs are set to 1. The inputs S = 0 and R = 0 are never applied as these inputs result in invalid output states as Q and $\overline{Q}$ should be complements of each other.

b. Assume that the outputs Q and $\overline{Q}$ are set at logic 0 and logic 1 respectively. Since both the inputs S and R are logic low, therefore both the Q and $\overline{Q}$ outputs are set to 1. The inputs S = 0 and R = 0 are never applied as these inputs result in invalid output states as Q and $\overline{Q}$ should be complements of each other.

The input combination S=0 and R=0 is considered to be invalid as it results in an invalid output of Q=1 and $\overline{Q}$=1.

2. Inputs S=0 & R=1

a. Consider that the outputs Q and $\overline{Q}$ have 1 and 0 logic states. The Set input S = 0 sets the output Q to 1. The Q input and the R inputs to gate 2 are both at logic 1, therefore the output $\overline{Q}$ is set to 0.

b. Consider that initially the Q and $\overline{Q}$ outputs are at logic state 0 and 1 respectively. The Set input S = 0 sets the output Q to 1. The Q input and the R inputs to gate 2 are both at logic 1, therefore the output $\overline{Q}$ is set to 0.

Thus what ever the initial outputs, setting S to 0 and R to 1 sets the Q and $\overline{Q}$ outputs to 1 and 0 respectively.

3. Inputs S=1 & R=0

a. Initially, the Q and $\overline{Q}$ outputs are at 1 and 0 respectively. The Reset input R=0 sets the output $\overline{Q}$ to 1. The inputs of gate 1, $\overline{Q}$ and S are both at logic 1, therefore the output Q is set to 0.

b. Initially, if the Q and $\overline{Q}$ outputs are at logic 0 and 1 respectively, setting R to 0 sets $\overline{Q}$ to 1. The inputs of gate 1, $\overline{Q}$ and S are both at logic 1, therefore the output Q is set to 0.

Thus, what ever the outputs, setting S to 1 and R to 0 sets the Q and $\overline{Q}$ outputs to 0 and 1 respectively.

4. Inputs S=1 & R=1

a. Initially, the Q and $\overline{Q}$ outputs are at 1 and 0 respectively. The inputs of gate 2 , Q and R are both at logic 1, therefore the output $\overline{Q}$ is set to 0. The inputs of gate 1, $\overline{Q}$ and S are 0 and 1 respectively, therefore the output is set to 1.
b. Initially, the Q and $\overline{Q}$ outputs are at 0 and 1 respectively. The inputs of gate 2 , Q and R are at logic 0 and 1 respectively, therefore the output $\overline{Q}$ is set to 1. The inputs of gate 1, $\overline{Q}$ and S are both at logic 1 respectively, therefore the output is set to 0.

Thus, with S and R inputs both set to logic 1, the previous output state is maintained. If initially, the Q and $\overline{Q}$ are at logic 1 and 0 respectively, setting S=1 and R=1 maintains the same outputs. Similarly, if initially Q and $\overline{Q}$ are at logic 0 and 1 respectively, setting S=1 and R=1 maintains the same outputs.

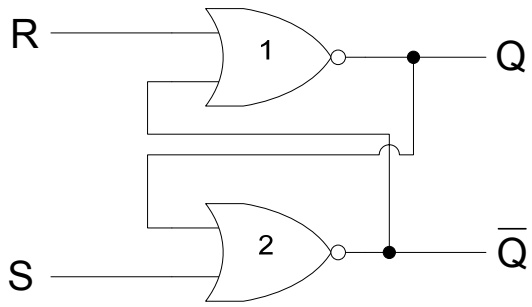
A truth-table shows the operation of the S-R NAND based latch. Table 22.2. The Output Q_{t+1} represents the Q output of NAND gate 1 at time interval t+1.When inputs are S = 1 and R = 1 the next state output Q_{t+1} remains the same as the previous state output Q_t. When inputs are S = 0 and R = 1 the output Q is set to 1. When inputs are S = 1 and R = 0 the output Q is set to 0. Inputs S = 0 and R = 0 are not applied as they place the latch in an invalid state. The NAND gate based S-R latch has active-low inputs.

Input		Output
S	R	Q_{t+1}
0	0	invalid
0	1	1
1	0	0
1	1	Q_t

Table 22.2 Truth-Table of NAND based S-R Latch

The NOR gate based S-R (Set-Reset) Latch

A NOR based S-R latch is implemented using NOR gates instead of NAND gates. Connections are identical to that of the NAND based latch. The S and R inputs have been switched. Figure 22.2.



The S-R NOR based latch has two inputs, therefore four different combinations of inputs can be applied to control the operation of the S-R latch. The four possible input combinations are

1. Inputs S=0 & R=0

a. Assume that the outputs Q and $\overline{Q}$ are set at logic 1 and logic 0 respectively. The R and $\overline{Q}$ inputs at gate 1 are both at logic 0, therefore the Q output is set to logic 1. The S and Q inputs at gate 2 are at logic 0 and 1 respectively, therefore the output $\overline{Q}$ is set to logic 0.

b. Assume that the outputs Q and $\overline{Q}$ are set at logic 0 and logic 1 respectively. The S and Q inputs at gate 2 are both at logic 0, therefore the $\overline{Q}$ output is set to logic 1. The R and $\overline{Q}$ inputs at gate 1 are at logic 0 and 1 respectively, therefore the output Q is set to logic 0.

Thus, with S and R inputs both set to logic 0, the previous output state is maintained. If initially, the Q and $\overline{Q}$ are at logic 1 and 0 respectively, setting S=0 and R=0 maintains the same outputs. Similarly, if initially Q and $\overline{Q}$ are at logic 0 and 1 respectively, setting S=0 and R=0 maintains the same outputs.

2. Inputs S=0 & R=1

a. Consider that the outputs Q and $\overline{Q}$ have 1 and 0 logic states. The Reset input R = 1 sets the output Q to 0. The Q input and the S inputs to gate 2 are both at logic 0, therefore the output $\overline{Q}$ is set to 1.

b. Consider that initially the Q and $\overline{Q}$ outputs are at logic state 0 and 1 respectively. The Reset input R = 1 sets the output Q to 0. The Q input and the S inputs to gate 2 are both at logic 0, therefore the output $\overline{Q}$ is set to 1.

Thus what ever the initial outputs, setting S to 0 and R to 1 sets the Q and $\overline{Q}$ outputs to 0 and 1 respectively.

3. Inputs S=1 & R=0

a. Initially, the Q and $\overline{Q}$ outputs are at 1 and 0 respectively. The Set input S=1 sets the output $\overline{Q}$ to 0. The inputs of gate 1, $\overline{Q}$ and R are both at logic 0, therefore the output Q is set to 1.

b. Initially, if the Q and $\overline{Q}$ outputs are at logic 0 and 1 respectively, setting S to 1 sets $\overline{Q}$ to 0. The inputs of gate 1, $\overline{Q}$ and R are both at logic 0, therefore the output Q is set to 1.

Thus, what ever the outputs, setting S to 1 and R to 0 sets the Q and $\overline{Q}$ outputs to 1 and 0 respectively.

4. Inputs S=1 & R=1

a. Initially, the Q and $\overline{Q}$ outputs are at 1 and 0 respectively. Since both the inputs S and R

are never applied as these inputs result in invalid output states as Q and $\overline{Q}$ should be complements of each other.

b. Initially, the Q and $\overline{Q}$ outputs are at 0 and 1 respectively. Since both the inputs S and R are logic 1, therefore both the Q and $\overline{Q}$ outputs are set to 0. The inputs S = 1 and R = 1 are never applied as these inputs result in invalid output states as Q and $\overline{Q}$ should be complements of each other.

The input combination S=1 and R=1 is considered to be invalid as it results in an invalid output of Q=0 and $\overline{Q}$=0.

The truth table of the NOR gate based latch is shown. Table 22.3. When inputs are S = 0 and R = 0 the next state output Q_{t+1} remains the same as the previous state output Q_t. When inputs are S = 0 and R = 1 the output Q is set to 0. When inputs are S = 1 and R = 0 the output Q is set to 1. Inputs S = 1 and R = 1 are not applied as they place the latch in an invalid state. The NOR gate based S-R latch has active-high inputs.

Input		Output
S	R	Q_{t+1}
0	0	Q_t
0	1	0
1	0	1
1	1	invalid

Table 22.3 Truth-Table of NOR based S-R Latch

Comparing the operation of the NOR based and NAND based S-R latches. The NAND based latch has active-low inputs, where as NOR based latch has active-high inputs. Both the S-R latches are set to logic 1 when the set input is activated and the reset input is inactive. Both the latches are set to logic 0 when the reset input is activated and the set input is inactive. The latches maintain the output state when both the set and reset inputs are inactive. For both the latches both the set and reset inputs can not be activated simultaneously as this leads to invalid output states. The Logic symbols of the two latches are shown in figure 22.5.



Figure 22.5 NOR based Active-High and NAND based Active-Low S-R Latches

S-R Latch Timing Diagrams

The operation of the active-high and active-low input latches can be understood with the help of timing diagrams. Figure 22.6 shows the timing diagrams of the active high and

the inputs S=0 and R=0 are not applied as it results in an invalid output state. Similarly, in the timing diagram of the NOR based S-R flip-flop, the inputs S=1 and R=1 are not applied as it results in an invalid output state.

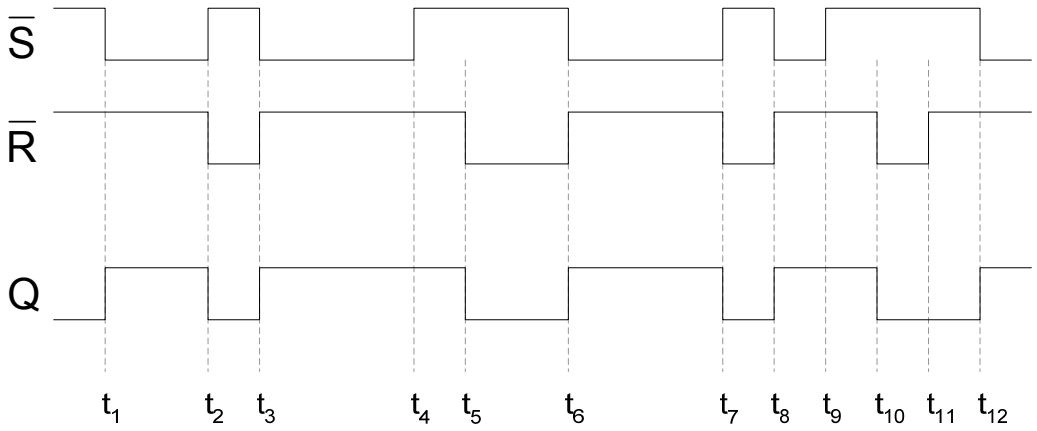


Figure 22.6a Timing diagram of an active-low input S-R latch

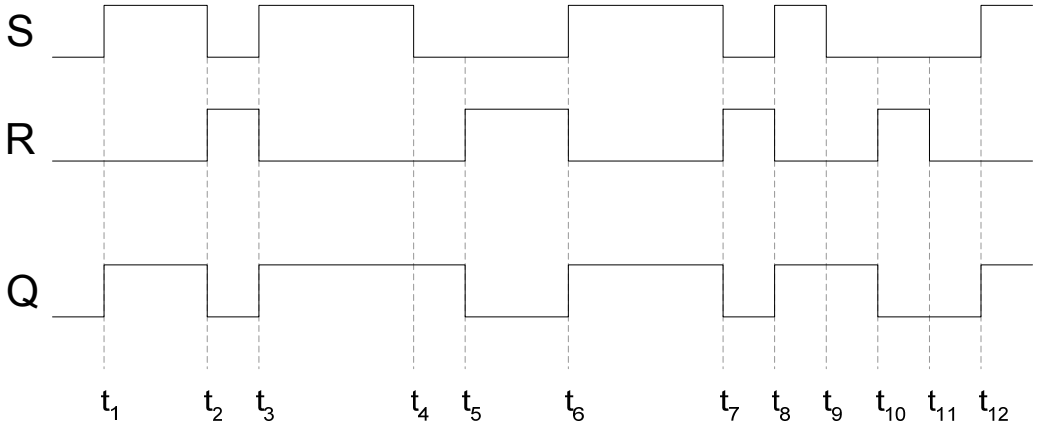


Figure 22.6b Timing diagram of an active-high input S-R latch

Lesson No. 23

APPLICATION OF S-R LATCH

Digital systems use switches to input values and to control the output. For example, a keypad uses 10 switches to enter decimal numbers 0 to 9. When a switch is closed the switch contacts physically vibrate or 'bounce' before making a solid contact. The switch bounce causes the voltage at the output of the switch to vary between logic low and high for a very short duration before it settles to a steady state. Figure 23.1a. The variation in the voltage causes the digital circuit to operate in an erratic manner. An S-R latch connected between the switch and the digital circuit prevents the varying switch output from reaching the digital circuit. Figure 23.1b.

In the figure 23.1a when the switch is moved up to connect the resistor to the ground, the output voltage fluctuates between logic 1 and 0 for a very brief period of time when the switch vibrates before making a solid contact. The output voltage settles to logic 0 when a solid contact is made. The active-low input S-R latch shown in figure 23.1b prevents the output signal from varying between logic 1 and 0. When the switch is moved from down position to up position, the R input is set to 1 and S input is set to 0, which sets the Q output of the S-R latch to 1. The S input varies between 0 and 1 due to switch 'bounce', however the S-R latch doesn't change its output state Q when S = 1 and R = 1.

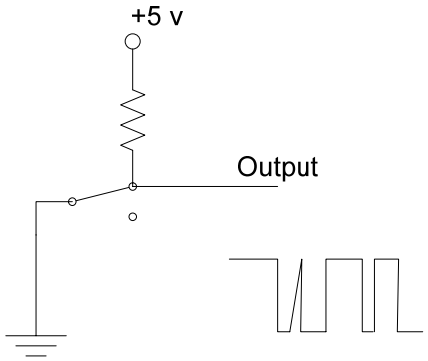


Figure 23.1a The output of a switch connected to Logic High

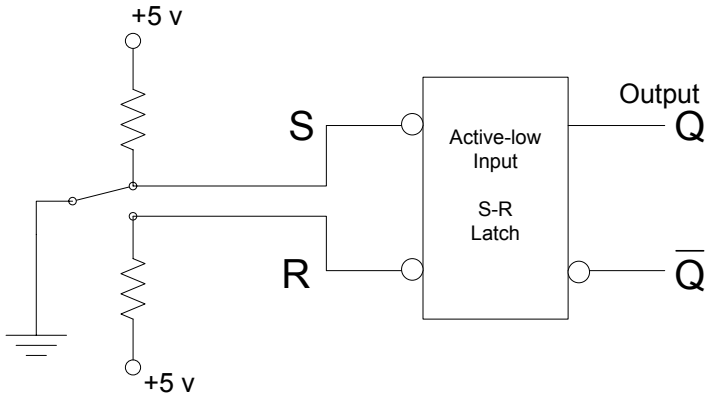


Figure 23.1b The switch connected through an S-R latch

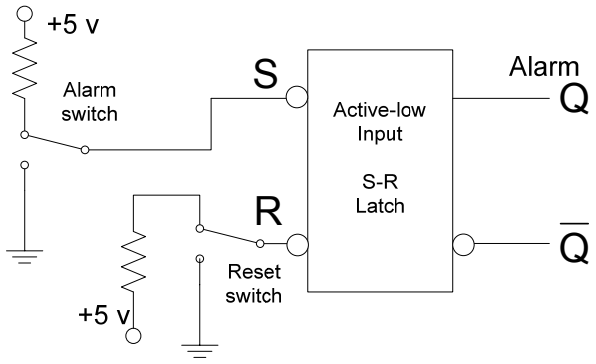


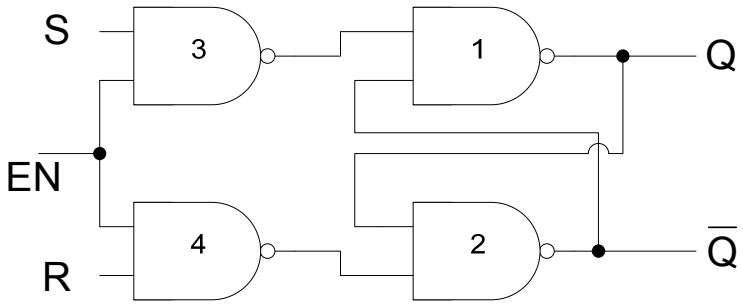
Figure 23.1c The switch connected through an S-R latch

The circuit diagram in figure 23.1c shows a burglar alarm circuit. The alarm switch is connected to logic high connecting the S input to logic high. The alarm is activated by setting the reset switch to ground connecting the R input to 0 volts. This sets the Q output of the latch to 0. The switch is reset to logic high. When an intruder opens a door the alarm switch is connected to ground or logic 0. The set input is set to logic 0, setting the Q output to logic 1 and activating the alarm. If the door is closed the alarm switch is reconnected to logic 1, however the Q output is maintained at logic 1 and the alarm continues to sound as S=1 and R=1 which maintains the output. The alarm can only be disabled by reconnecting the reset switch to ground.

The S-R NAND gate based latch is available in the form of an Integrated Circuit. The 74LS279 IC has four S-R latches which can be used independently.

The Gated S-R Latch

The gated S-R latch has an enable input which has to be activated to operate the latch. The circuit diagram of the gated S-R latch is shown. Figure 23.2. In the gated S-R circuit, the S and R inputs are applied at the inputs of the NAND gates 1 and 2 when the enable input is set to active-high. If the enable input is disabled by setting it to logic low the output of NAND gates 3 and 4 remains logic 1, what ever the state of S and R inputs. Thus logic 1 applied at the inputs of NAND gates 1 and 2 keeps the Q and $\overline{Q}$ outputs to the previous state. The logic symbol of a gated S-R latch is shown in figure 23.3. The Truth Table of the gated S-R latch is shown in table 23.1. The timing diagram showing the operation of the gated S-R latch is shown in figure 23.4



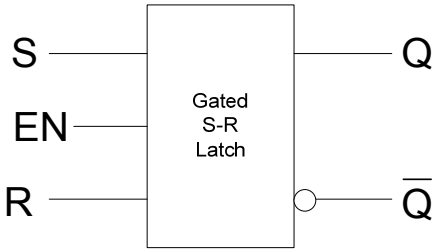


Figure 23.3 Logic Symbol of a Gated S-R Latch

Input			Output
EN	S	R	Q_{t+1}
0	x	x	Q_t
1	0	0	Q_t
1	0	1	0
1	1	0	1
1	1	1	invalid

Table 23.1 Truth-Table of a gated S-R Latch

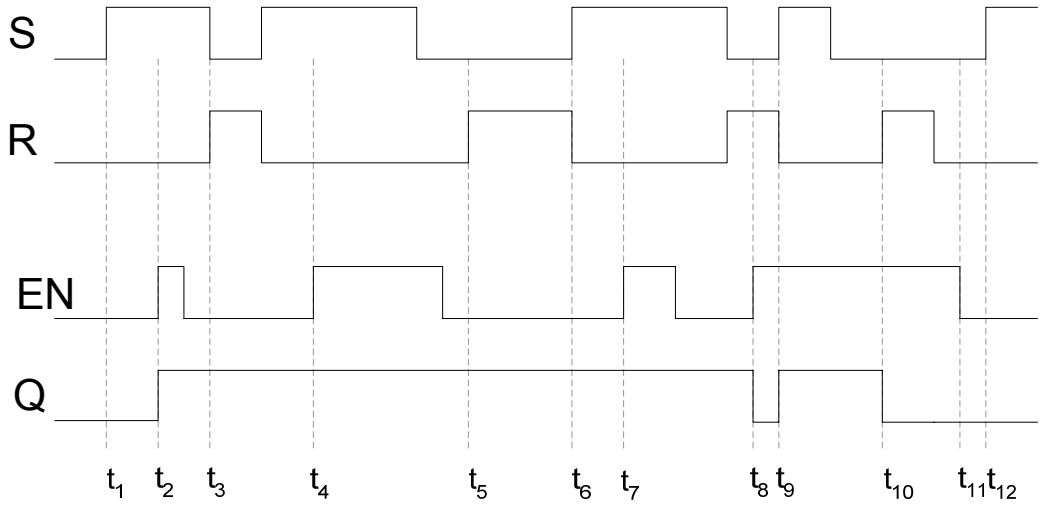


Figure 23.4 Timing diagram of a gated S-R latch

The Gated D Latch

If the S and R inputs of the gated S-R latch are connected together using a NOT gate then there is only a single input to the latch. The input is represented by D instead of S or R. Figure 23.5. The gated D-latch can either have D set to 0 or 1, thus the four input combinations applied at the S-R inputs of an S-R latch reduce to only two input combinations. Table 23.2. The logic symbol of a gated D-latch is shown in figure 23.6. The timing diagram of the operation of a D-latch is shown in figure 23.7. The Q output of the D latch is seen to be following the D input.

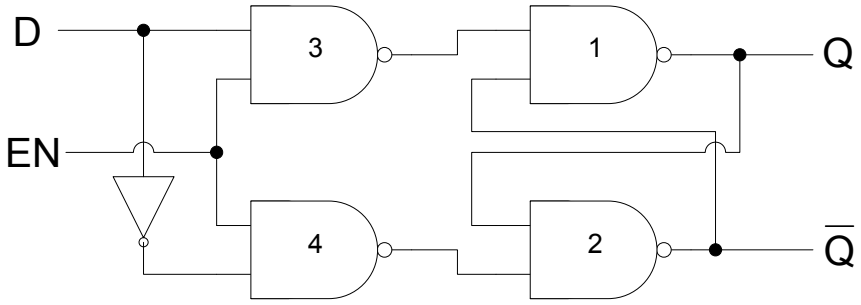


Figure 23.5 Gated D Latch

Input			Output
EN	S (D)	R	Q_{t+1}
0	x	X	Q_t
1	0	0	Q_t
1	0	1	0
1	1	0	1
1	1	1	Invalid

Input		Output
EN	D	Q_{t+1}
0	x	Q_t
1	0	0
1	1	1

Table 23.2 Truth-Table of a gated D Latch

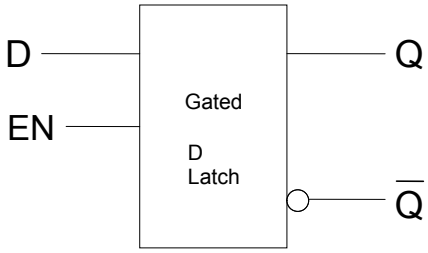


Figure 23.6 Logic Symbol of a Gated D Latch

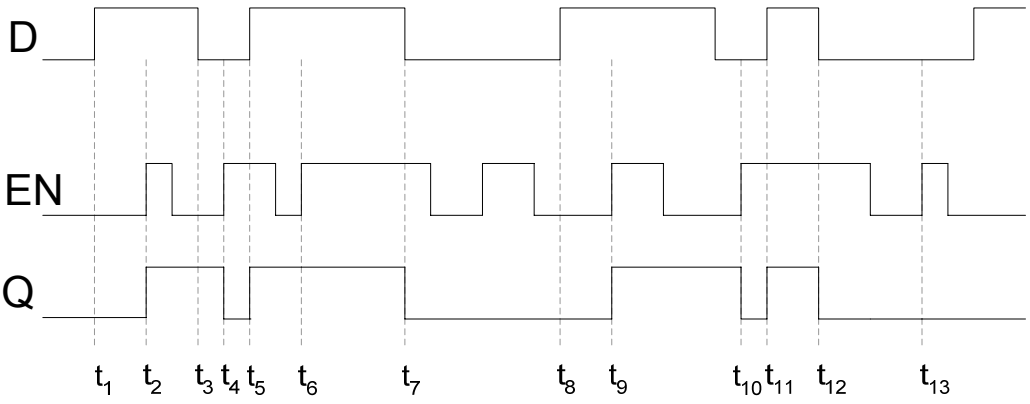


Figure 23.7 Timing diagram of a gated D latch

Application of Gated D Latch

The D latch is available in the form of an Integrated Circuit. The 74LS75 has four D latches which can be used independently. The gated D latch can be used to store binary

of 1-of-8 multiplexer to store a byte value (parallel). The multiplexer accesses each bit value stored in the D-latch and routes it to the output. Thus the 8-bit (byte) parallel data is converted into serial data.

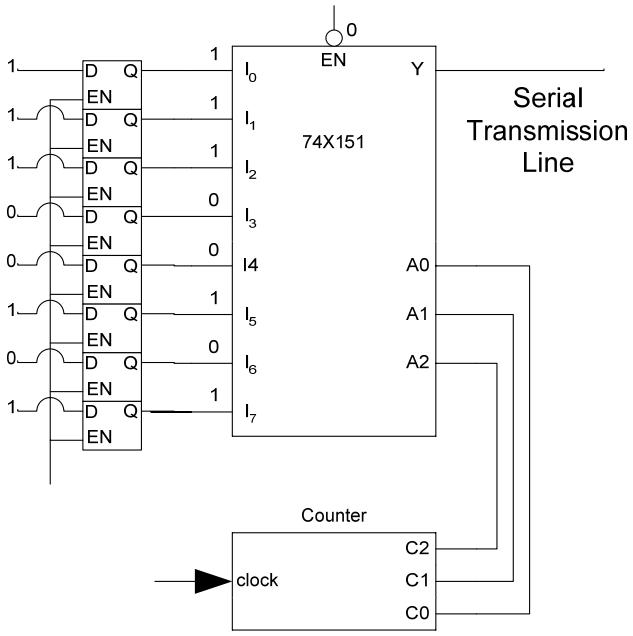


Figure 23.8 Gated D-latch used to store parallel data

Edge-Triggered Flip-Flop

Flip-Flops are synchronous bi-stable devices, known as bi-stable multivibrators. Flip-flops have a clock input instead of a simple enable input as discussed earlier. The output of the flip-flop can only change when appropriate inputs are applied at the S and R inputs and a clock signal is applied at the clock input. Flip-flops with enable inputs can change their state at any instant when the enable input is active. Digital circuits that change their outputs when the enable input is active are difficult to design and debug as different parts of the digital circuit operate at different times.

In Synchronous systems, the output of all the digital circuits changes when a clock signal is applied instead of the enable signal. The change in the state of the digital circuit occurs either at the low-to-high or high-to-low transition of the clock signal. Since the transition of the clock signal is for a very short a precise time intervals thus all digital parts of a Digital system change their states simultaneously. The low to high or high to low transition of the clock is considered to be an edge. Three different types of edge-triggered flip-flops are generally used in digital logic circuits.

- S-R edge-triggered flip-flop
- D edge-triggered flip-flop
- J-K edge-triggered flip-flop

Each flip-flop has two variations, that is, it is either positive edge-triggered or negative edge triggered. A positive edge-triggered flip-flop changes its state on a low-to-high transition of the clock and a negative edge-triggered flip-flop changes its state on a high-to-low transition of the clock. The edge-detection circuit which allows a flip-flop to change its state on either the

circuit. The edge detection circuit that detects the positive and the negative clock transition are shown in figure 23.9.

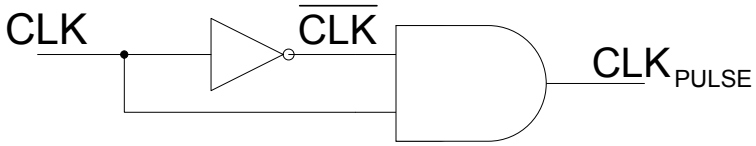


Figure 23.9a Positive clock edge detection circuit

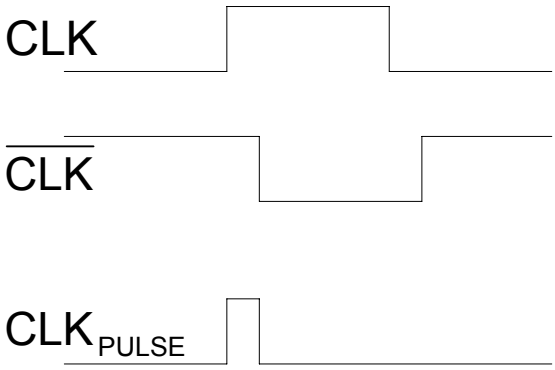


Figure 23.9b Timing diagram of the Positive clock edge detection circuit

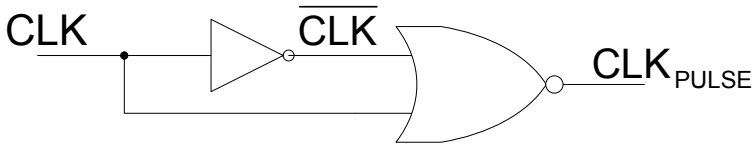


Figure 23.9c Negative clock edge detection circuit

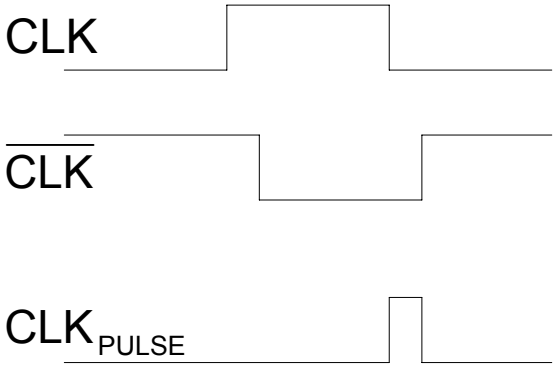


Figure 23.9d Timing diagram of the Negative clock edge detection circuit

Edge-Triggered S-R Flip-flop

The Logic symbols of a positive edge and a negative edge triggered S-R flip-flops are shown in figure 23.10. The truth table of the two S-R flip-flops are shown. Table 23.3. The timing diagrams of the two S-R flip-flops are shown in figure 23.11.

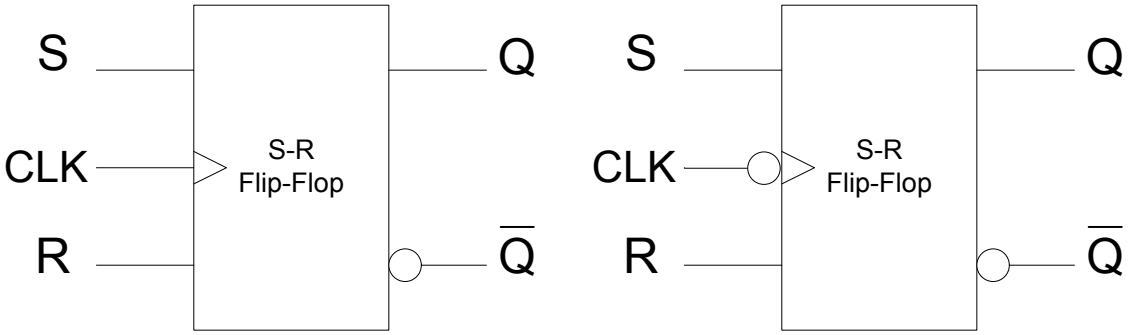


Figure 23.10 Logic Symbol of Positive and Negative edge triggered S-R flip-flops

Input			Output
CLK	S	R	Q_{t+1}
0	X	X	Q_t
1	X	X	Q_t
↑	0	0	Q_t
↑	0	1	0
↑	1	0	1
↑	1	1	invalid

Input			Output
CLK	S	R	Q_{t+1}
0	x	x	Q_t
1	x	x	Q_t
↓	0	0	Q_t
↓	0	1	0
↓	1	0	1
↓	1	1	invalid

Table 23.3 Truth-Table of Positive and Negative Edge triggered S-R flip-flops

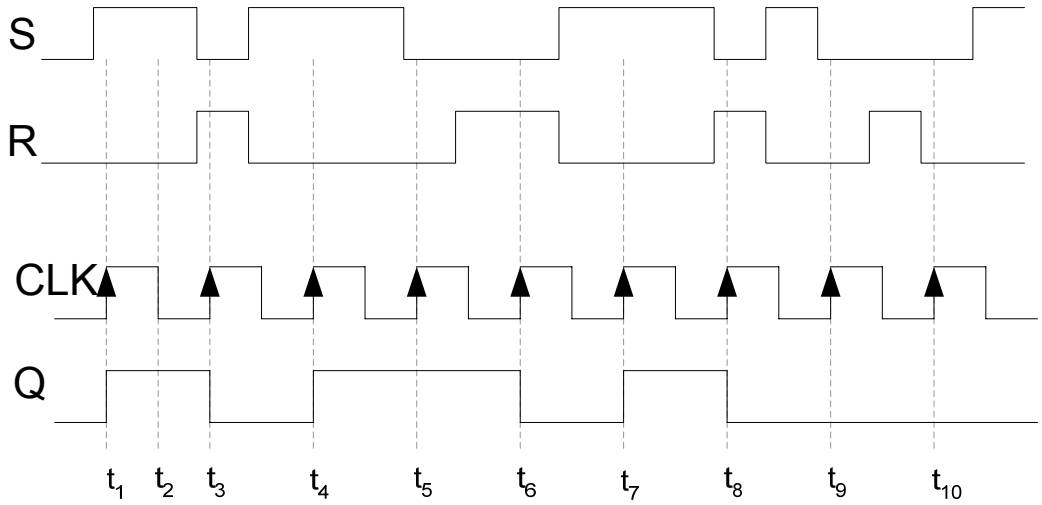


Figure 23.11a Timing diagram of a Positive Edge triggered S-R flip-flop

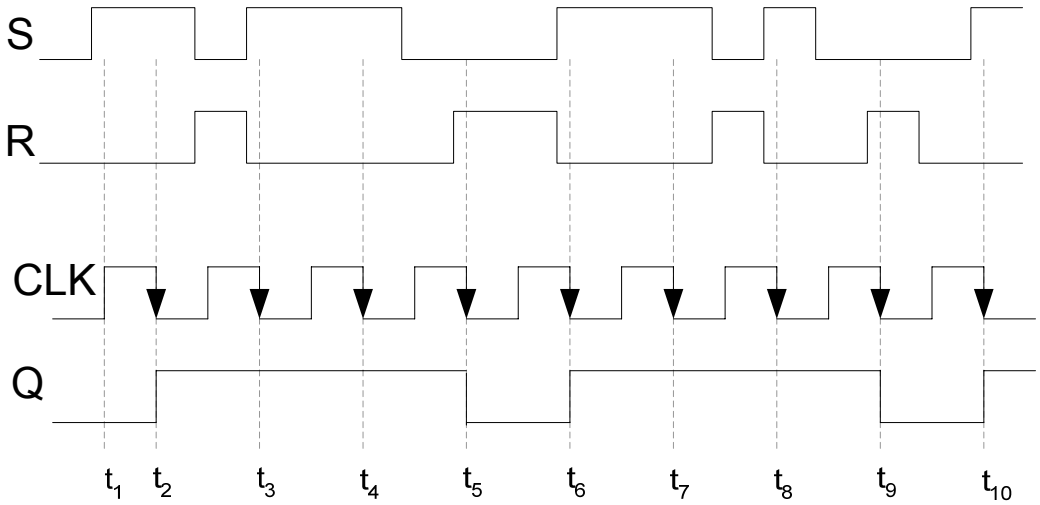


Figure 23.11b Timing diagram of a Negative Edge triggered S-R flip-flop

Edge-Triggered D Flip-flop

The Logic symbols of a positive edge and a negative edge triggered D flip-flops are shown in figure 23.12. The truth table of the two D flip-flops are shown. Table 23.4. The timing diagrams of the two D flip-flops are shown in figure 23.13.

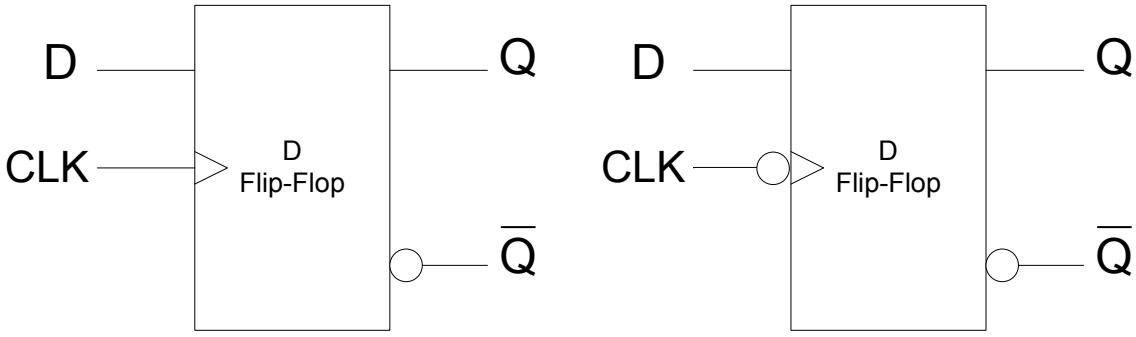


Figure 23.12 Logic Symbol of Positive and Negative edge triggered D flip-flops

Input		Output
CLK	D	Q_{t+1}
0	X	Q_t
1	X	Q_t
↑	0	0
↑	1	1

Input		Output
CLK	D	Q_{t+1}
0	X	Q_t
1	X	Q_t
↓	0	0
↓	1	1

Table 23.4 Truth-Table of Positive and Negative Edge triggered D flip-flops

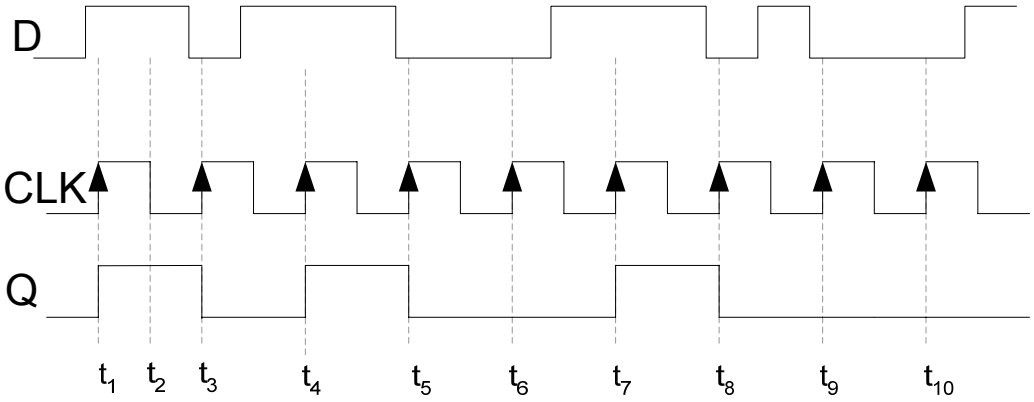


Figure 23.13a Timing diagram of a Positive Edge triggered D flip-flop

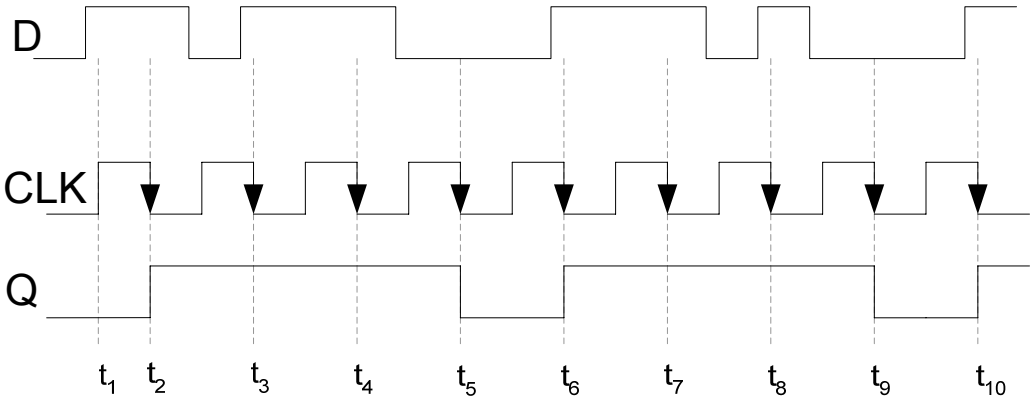


Figure 23.13b Timing diagram of a Negative Edge triggered D flip-flop

Edge-Triggered J-K Flip-flop

The J-K flip-flop is widely used in digital circuits. Its operation is similar to that of the S-R flip-flop except that the J-K flip-flop doesn't have an invalid state, instead it toggles its state. The circuit diagram of a J-K edge-triggered flip-flop is shown in figure 23.14

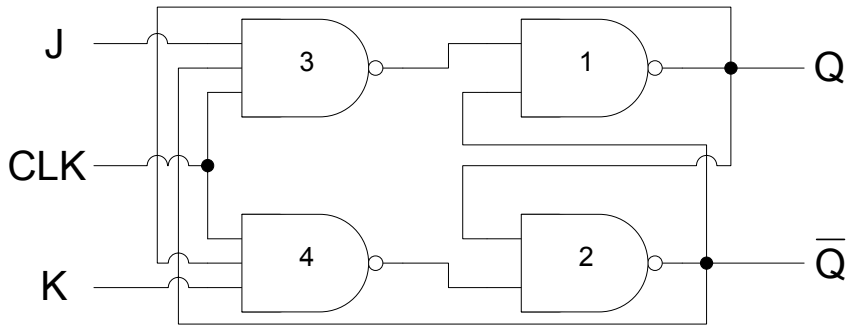


Figure 23.14 Edge-triggered J-K flip-flop

Consider the Q and $\overline{Q}$ output of the J-K flip-flop set to 1 and 0 respectively and 0 and 1

J = 0 and K =0

With Q=1 and $\overline{Q}$=0, on a clock transition the outputs of NAND gates 3 and 4 are set to logic 1. With logic 1 value at the inputs of NAND gates 1 and 2 the output Q and $\overline{Q}$ remains unchanged. With Q=0 and $\overline{Q}$=1, on a clock transition the outputs of the NAND gates 3 and 4 are set to logic 1. With logic 1 value at the inputs of NAND gates 1 and 2 the output Q and $\overline{Q}$ remains unchanged.

Thus when J=0 and K=0 the previous state is maintained and there is no change in the output.

J = 0 and K =1

With Q=1 and $\overline{Q}$=0, on a clock transition the output of NAND gate 3 is set to logic 1. The output of the NAND gate 4 is set to 0 as all three of its inputs are at logic 1. The logic 1 and 0 at the inputs of the NAND gates 3 and 4 respectively resets the Q output to 0 and $\overline{Q}$ to 1. With Q=0 and $\overline{Q}$=1, on a clock transition the output of NAND gate 3 is set to logic 1. The output of the NAND gate 4 is also set to 1 as the input of the NAND gate 4 is connected to Q=0. The logic 1 and 1 at the inputs of the NAND gates 3 and 4 respectively retains the Q and $\overline{Q}$ to 0 and 1 respectively.

Thus when J=0 and K=1 the J-K flip-flop irrespective of its earlier state is rest to state Q=0 and $\overline{Q}$=1.

J = 1 and K =0

With Q=1 and $\overline{Q}$=0, on a clock transition the output of NAND gate 4 is set to logic 1. The output of the NAND gate 3 is also set to 1 as its input connected to $\overline{Q}$ is at logic 0. Thus inputs 1 and 1 at inputs of NAND gates 1 and 2 retain the Q and $\overline{Q}$ output to 1 and 0 respectively. With Q=0 and $\overline{Q}$=1, on a clock transition the output of NAND gate 4 is set to logic 1. The output of the NAND gate 3 is set to 0 as all its input are at logic 1. Thus inputs 0 and 1 at inputs of NAND gates 1 and 2 sets the flip-flop to Q=1 and $\overline{Q}$=0.

Thus when J=1 and K=0 the J-K flip-flop irrespective of its output state is set to state Q=1 and $\overline{Q}$=0.

J = 1 and K =1

With Q=1 and $\overline{Q}$=0, on a clock transition the output of the NAND gates 3 and 4 depend on the outputs Q and $\overline{Q}$. The output of NAND gate 3 is set to 1 as $\overline{Q}$ is connected to its input. The output of NAND gate 4 is set to 0 as all its inputs including Q is at logic 1. A logic 1 and 0

respectively. With Q=0 and $\overline{Q}$=1, on a clock transition the output of NAND gate 3 is set to 0 as $\overline{Q}$ and the output of NAND gate 4 is set to 1. A logic 0 and 1 at the input toggles the outputs Q and $\overline{Q}$ from logic 0 and 1 to 1 and 0 respectively.

In summary when J-K inputs are both set to logic 0, the output remains unchanged. At J=0 and K=1 the J-K flip-flop is reset to Q=0 and $\overline{Q}$=1. At J=1 and K=0 the flip-flop is set to Q=1 and $\overline{Q}$=0. With J=1 and K=1 the output toggles from the previous state. The truth tables of the positive and negative edge triggered J-K flip-flops are shown in table 23.5. The logic symbols of the J-K flip-flops are shown in figure 23.15. The timing diagrams of the J-K flip-flops are shown in figure 23.16.

Input			Output
CLK	J	K	Q_{t+1}
0	x	X	Q_t
1	x	X	Q_t
↑	0	0	Q_t
↑	0	1	0
↑	1	0	1
↑	1	1	$\overline{Q}_t$

Input			Output
CLK	J	K	Q_{t+1}
0	x	x	Q_t
1	x	x	Q_t
↓	0	0	Q_t
↓	0	1	0
↓	1	0	1
↓	1	1	$\overline{Q}_t$

Table 23.5 Truth-Table of Positive and Negative Edge triggered J-K flip-flops

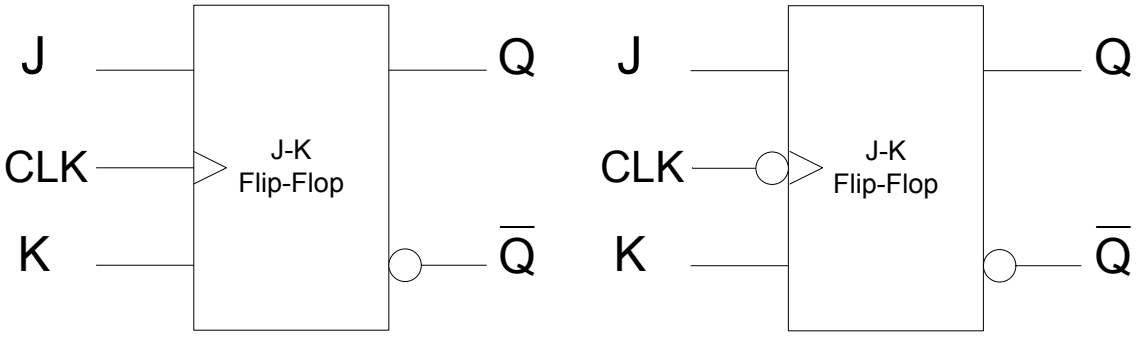


Figure 23.15 Logic Symbol of Positive and Negative edge triggered J-K flip-flops

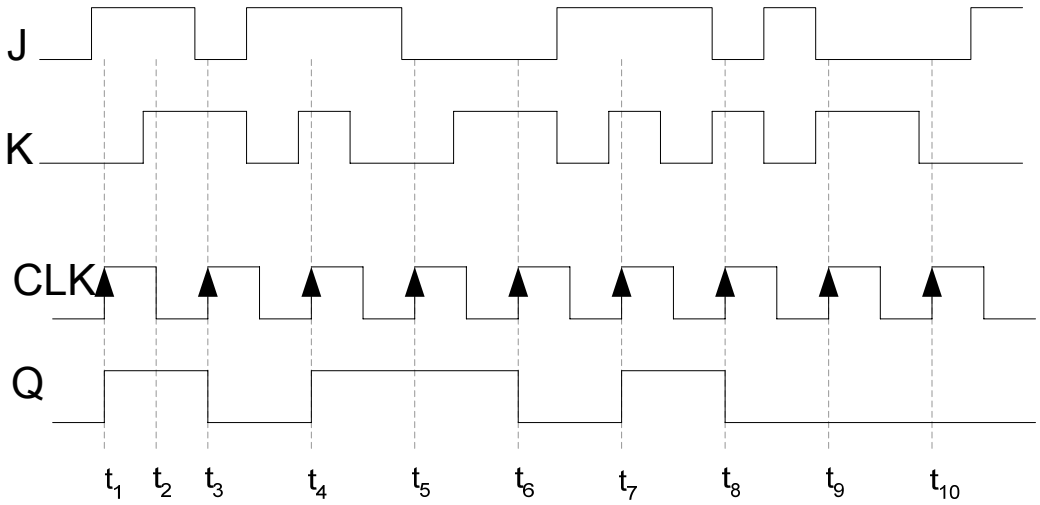


Figure 23.16a Timing diagram of a Positive Edge triggered J-K flip-flop

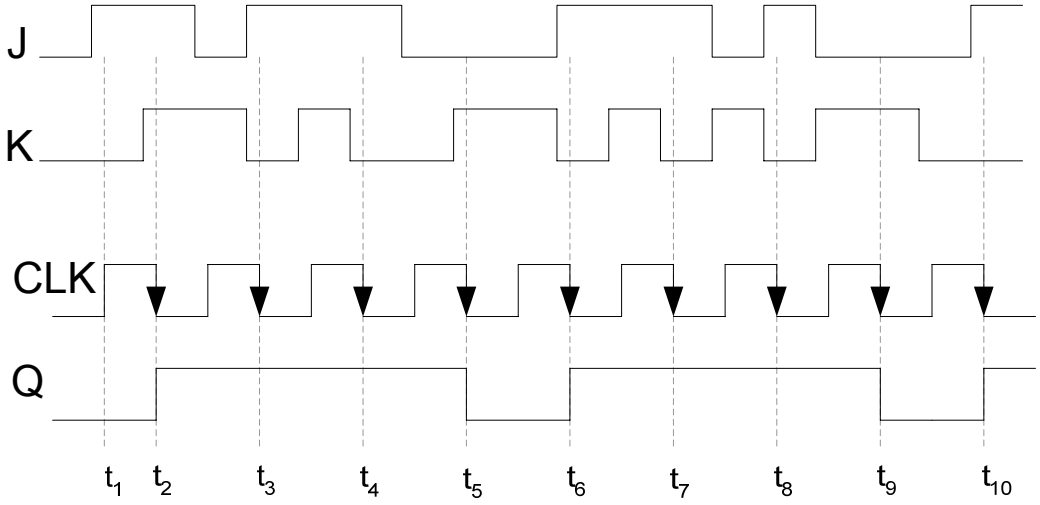


Figure 23.16b Timing diagram of a Negative Edge triggered J-K flip-flop

Asynchronous Preset and Clear Inputs

The S-R, J-K and D inputs are known as synchronous inputs because the outputs change when appropriate input values are applied at the inputs and a clock signal is applied at the clock input. If there is no clock transition then the inputs have no effect on the output. Digital circuits require that the flip-flops be set or reset to some initial state before a new set of inputs is applied for changing the output. The flip-flops are set-reset to some initial state by using asynchronous inputs known as Preset and Clear inputs. Since these inputs change the output to a known logic level independently of the clock signal therefore these inputs are known as asynchronous inputs. The circuit diagram of a J-K flip-flop with Preset and Set Asynchronous inputs is shown in figure 23.17. The asynchronous inputs override the synchronous inputs thus to operate the flip-flop in the synchronous mode the asynchronous inputs have to be disabled. To preset the flip-flop to Q=1 and $\overline{Q}$=0 the $\overline{PRE}$ input is set to 0 which sets the Q output to 1 and the output of NAND gate 4 to 1. The $\overline{CLR}$ input is set to 1

which sets the $\overline{Q}$ output to 0 as all three inputs of the NAND gate 2 are set to 1. The flip-flop is cleared to Q=0 and $\overline{Q}$=1 by setting the $\overline{PRE}$ input is set to 1 and the $\overline{CLR}$ input is to 0. The $\overline{CLR}$ input set to 0 sets $\overline{Q}$=1 it also sets the output of NAND gate 3 to 1. The $\overline{PRE}$ input set to 1 sets the output Q to 0. When the $\overline{PRE}$ and the $\overline{CLR}$ inputs are used inputs J and K have no effect on the operation of the flip-flop. To use the flip-flop with synchronous inputs J-K, the $\overline{PRE}$ and the $\overline{CLR}$ inputs are set to logic 1. Setting $\overline{PRE}$ and the $\overline{CLR}$ to logic 0 is not allowed.

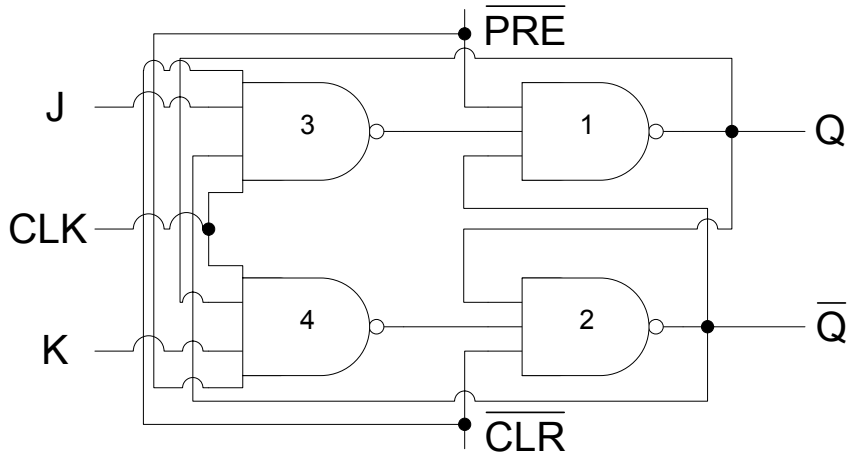


Figure 23.17 J-K flip-flop with Asynchronous Preset and Clear inputs

Figure 23.18 shows the logic symbol of a J-K edge-triggered flip-flop with synchronous and asynchronous inputs.

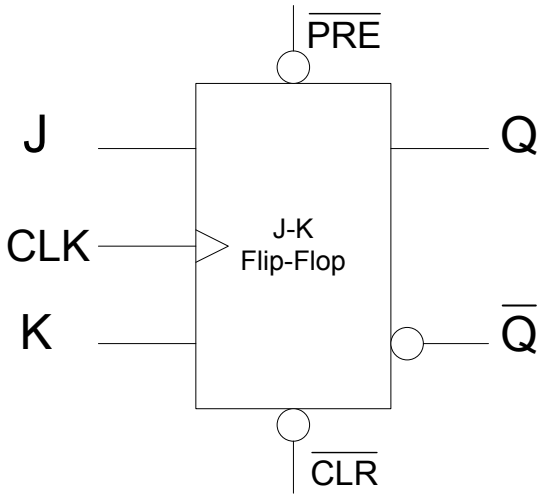


Figure 23.18 Logic Symbol of a J-K flip-flop with Asynchronous inputs

The truth table of a J-K flip-flop with Asynchronous inputs is shown in table 23.4. The timing diagram describes the effect of asynchronous inputs on the operation of the flip-flop. Figure 23.10

Input		Output
$\overline{PRE}$	$\overline{CLR}$	Q_{t+1}
0	0	Invalid
0	1	1
1	0	0
1	1	Clocked operation

Table 23.4 Truth table of J-K flip-flop with Asynchronous inputs

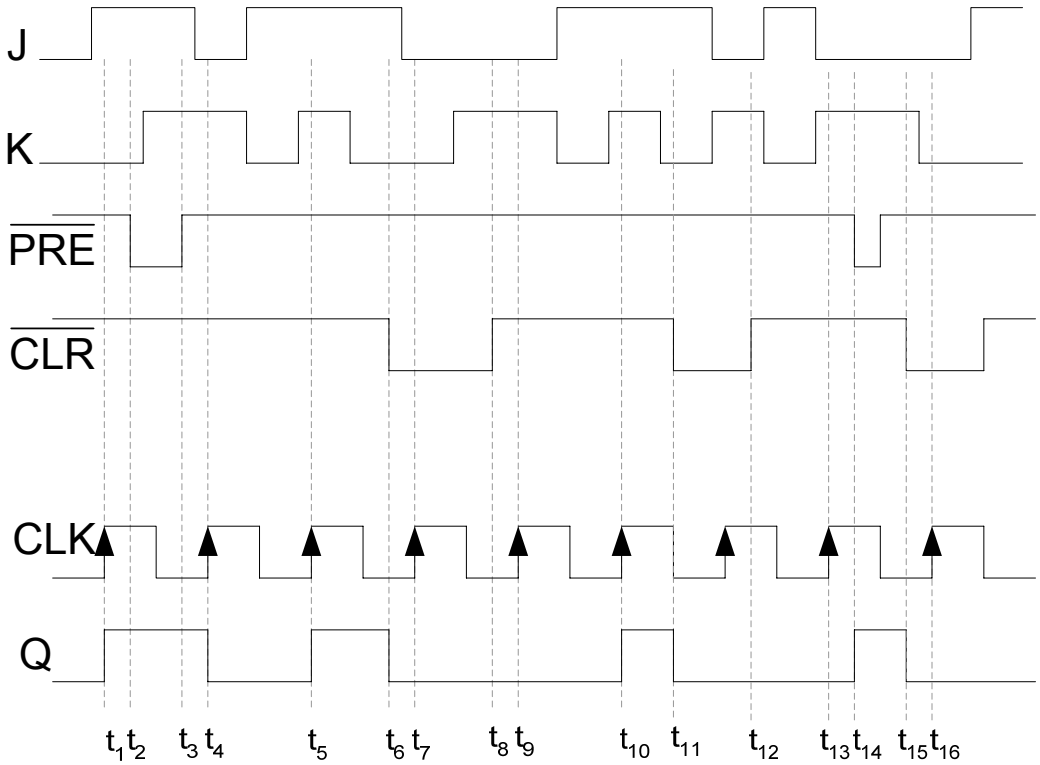


Figure 23.10 Timing diagram of a J-K flip-flop with Preset and Clear inputs

The 74HC74 Dual Positive-Edge triggered D flip-flop

The edge-triggered D flip-flop with asynchronous inputs is available as an Integrated Circuit. The 74HC74 has dual D-flip-flops with independent clock inputs, synchronous and asynchronous inputs.

The 74HC112 Dual Positive-Edge triggered J-K flip-flop

The edge-triggered D flip-flop with asynchronous inputs is available as an Integrated Circuit. The 74HC112 has dual J-K-flip-flops with independent clock inputs, synchronous and asynchronous inputs.

Master-Slave Flip-Flops

Master-Slave flip-flops have become obsolete and are being replaced by edge-

clock signal. The inputs are applied in the first half of the clock signal. The outputs do not change until the second half of the clock signal. As mentioned earlier the use of edge-triggered flip-flip is to synchronize the operation of a digital circuit with a common clock signal. The master-slave setup also allows digital circuits to operate in synchronization with a common clock signal. The circuit diagram of the master-slave J-K flip-flop is shown in figure 23.11. The Master-Slave flip-flop is composed of two parts the Master and the Slave. Both the Master and the Slave are Gated S-R flip-flops. The Master-Slave flip-flop is not synchronised with the clock positive or negative transition, rather it known as a pulse triggered flip-flop as it operates at the positive and negative clock cycles.

Consider that the J-K inputs of the flip-flop are set at 1 and 0 respectively. The outputs Q and $\overline{Q}$ are initially set at 1 and 0 respectively. During the positive half of the clock gates 3 and 4 are both enabled by the clock signal. The output of gate 3 is set to 1 due to the $\overline{Q}$ output set at 0. Similarly the output of gate 4 is also set at 1 due to the K input set at 0. The outputs of gates 1 and 2 remain unchanged as the inputs to gates 1 and 2 are both logic 1. Assume the outputs of gates 1 and 2 to be 1 and 0 respectively. During the positive half cycle, the clock input to gates 7 and 8 is inverted therefore both the gates are disabled and their output is set to logic 1. With logic 1 at the inputs of gates 5 and 6 the output Q and $\overline{Q}$ remains unchanged throughout the positive half of the clock cycle. During the negative half of the clock cycle the Master flip-flop is disabled and the output of the Master flip-flop remains fixed during the negative half cycle. The Slave flip-flop is enabled and the 1 and 0 outputs of the Master flip-flop set the Q and $\overline{Q}$ output to 1 and 0 respectively.

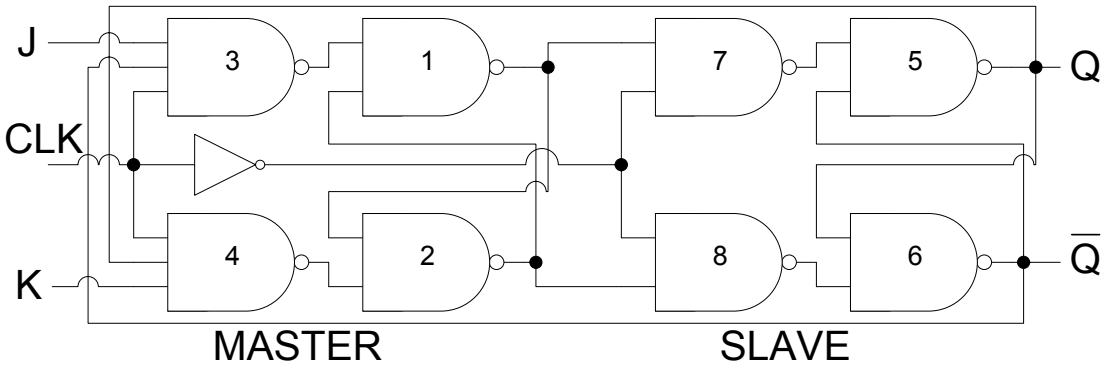


Figure 23.11 Master-Slave flip-flop

Initially, if the Q and $\overline{Q}$ outputs are 0 and 1 respectively, setting the J and K inputs to 1 and 0 respectively sets the output to 1 and 0 respectively. During the positive half of the clock the Master flip-flop is enabled, the output of gate 3 is set to 0 as the J, Q and CLK inputs are all at logic 1. The output of gate 4 is set to 1 as the K input is logic 0. These inputs set the output of the Master flip-flop at gates 1 and 2 to logic 1 and 0 respectively. During the negative half of the clock cycle the Slave flip-flop is enabled the output Q and $\overline{Q}$ are set to logic 1 and 0 respectively.

The truth-table of the master-slave flip-flop is shown in table 23.5. The timing diagram of the master-slave flip-flop is shown in figure 23.12.

Input			Output
CLK	J	K	Q_{t+1}
Pulse	0	0	Q_t
Pulse	0	1	0
Pulse	1	0	1
Pulse	1	1	$\overline{Q}_t$

Table 23.5 Truth table of the Master-Slave J-K flip-flop

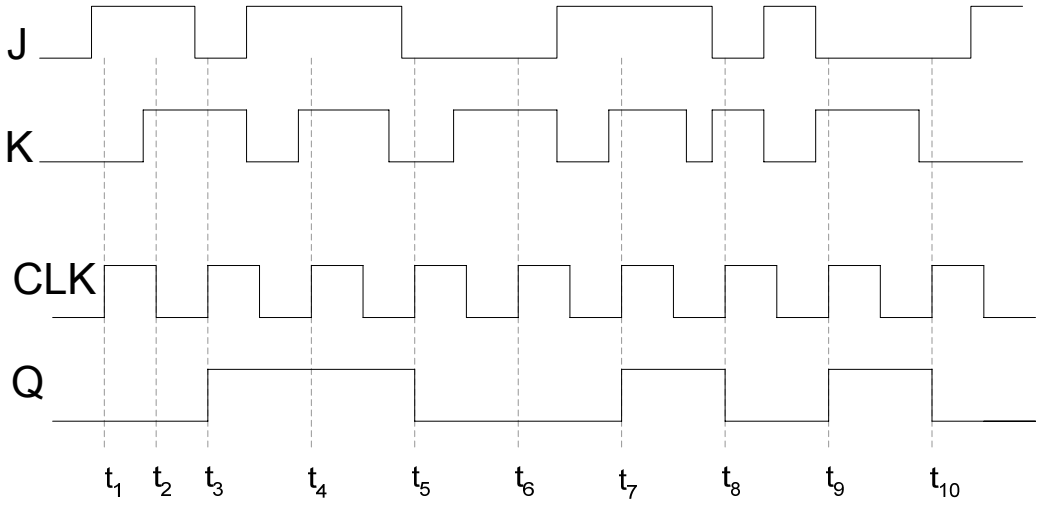


Figure 23.12 Timing diagram of a Master Slave J-K flip-flop

Flip-Flop Operating Characteristics

The performance of the flip-flop is specified by several operating characteristics mentioned in the data sheets of the flip-flops. The important operating characteristics are

- Propagation Delay
- Set-up Time
- Hold Time
- Maximum Clock frequency
- Pulse width
- Power Dissipation

Propagation Delay

The propagation delay time is the interval of time when the input is applied and the output changes. Four different types of Propagation Delays are measured.

1. Propagtaion Delay t_{PLH} measured with respect to the triggering edge of the clock to the low-to-high transition of the output. Figure 23.13. On a positive or negative clock transition the flip-flop changes its output state. The Propagation Delay is measured at 50% transition

mark on the triggering edge of the clock and the 50% mark on the low-to-high transition of the output that occurs due to the clock transition.

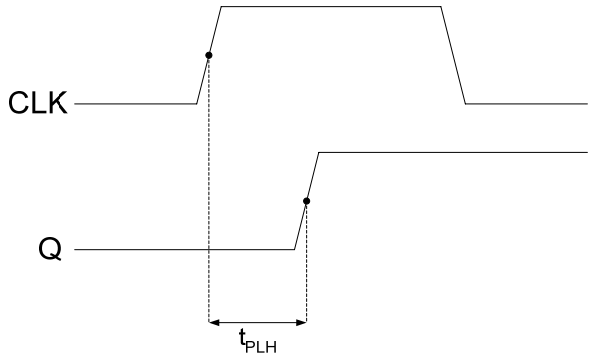


Figure 23.13 Propagation Delay, clock to low-to-high transition of the output

2. Propagtaion Delay t_{PHL} measured with respect to the triggering edge of the clock to the high-to-low transition of the output. Figure 23.14. On a positive or negative clock transition the flip-flop changes its output state. The Propagation Delay is measured at 50% transition mark on the triggering edge of the clock and the 50% mark on the high-to-low transition of the output that occurs due to the clock transition.

3. Propagtaion Delay t_{PLH} measured with respect to the leading edge of the preset input to the low-to-high transition of the output. Figure 23.15. On a high-to-low transition of the preset signal the flip-flop changes its output state to logic high. The Propagation Delay is measured at 50% transition mark on the triggering edge of the preset signal and the 50% mark on the low-to-high transition of the output that occurs due to the preset signal.

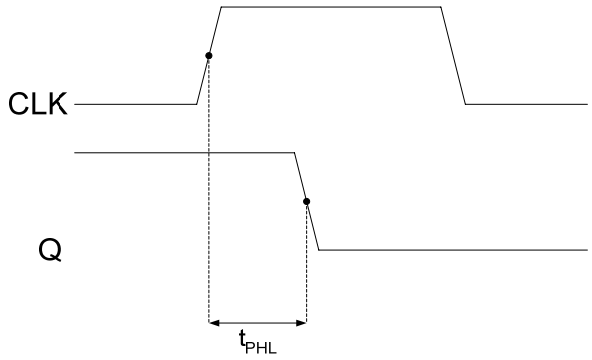


Figure 23.14 Propagation Delay, clock to high-to-low transition of the output

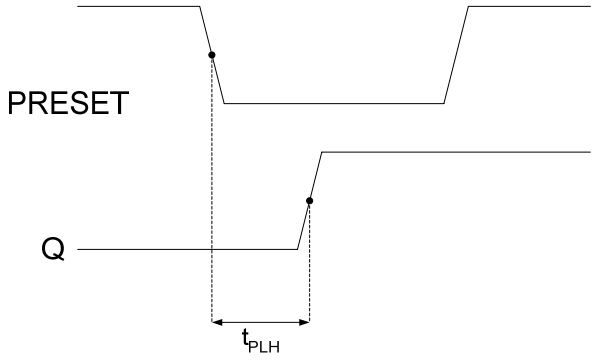


Figure 23.15 Propagation Delay, preset to low-to-high transition of the output

4. Propagtaion Delay t_{PHL} measured with respect to the leading edge of the clear input to the high-to-low transition of the output. Figure 23.16. On a high-to-low transition of the clear signal the flip-flop changes its output state to logic low. The Propagation Delay is measured at 50% transition mark on the triggering edge of the clear signal and the 50% mark on the high-to-low transition of the output that occurs due to the preset signal.

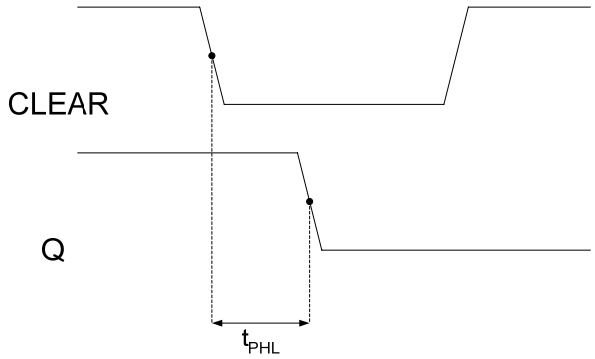


Figure 23.16 Propagation Delay, clear to high-to-low transition of the output

Set-up Time

When a clock transition occurs at the clock input of a flip-flop the output of the flip-flop is set to a new state based on the inputs. For the flip-flop to change its output to a new state at the clock transition, the input should be stable. The minimum time required for the input logic levels to remain stable before the clock transition occurs is known as the Set-up time. Figure 23.17.

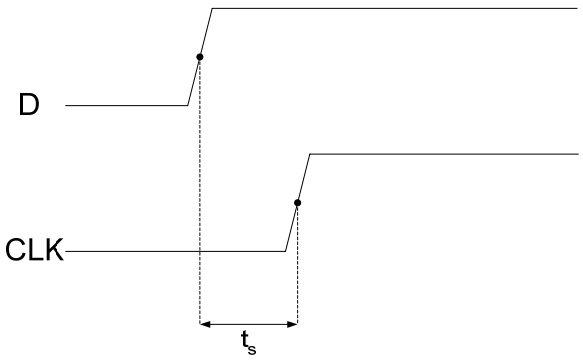


Figure 23.17 Set-up time for a D flip-flop

Hold Time

The input signal maintained at the flip-flop input has to be maintained for a minimum time after the clock transition for the flip-flop to reliably clock in the input signal. The minimum time for which the input signal has to be maintained at the input is the Hold time of the flip-flop. Figure 23.18

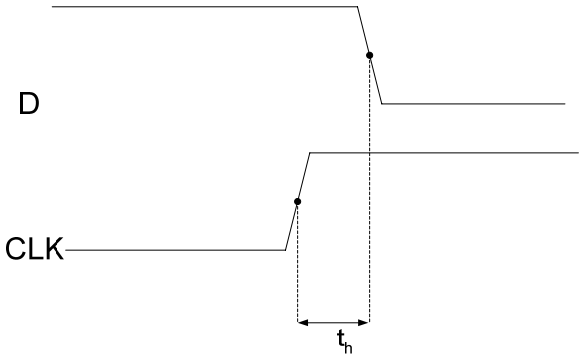


Figure 23.18 Hold time for a D flip-flop

Maximum Clock Frequency

A flip-flop can be operated at a certain clock frequency. If the clock frequency is increased beyond a certain limit the flip-flop will be unable to respond to the fast changing clock transitions, therefore the flip-flop will be unable to function. The maximum clock frequency f_{max} is the highest rate at which the flip-flop operates reliably.

Pulse Width

A flip-flop uses the clock, preset and clear inputs for its operation. Each signal has to be of a specified duration for correct operation of the flip-flop. The manufacturer specifies the minimum pulse width t_w for each of the three signals. The clock signal is specified by minimum high time and minimum low time.

Power Dissipation

A flip-flop consumes power during its operation. The power consumed by a flip-flop is defined by $P = V_{cc} \times I_{cc}$. The flip-flop is connected to +5 volts and it draws 5 mA of current during its operation, therefore the power dissipation of the flip-flop is 25 mW.

A digital circuit is made of a number of gates, functional units and flip-flops. The total power requirement of each device should be known so that an appropriate dc power source is used to supply power to the digital circuit.

© Virtual University of Pakistan

Lesson No. 24

APPLICATIONS OF EDGE-TRIGGERED D FLIP-FLOP

1. Data Storage using D-flip-flop

A Multiplexer based Parallel-to-Serial converter needs to have stable parallel data at its inputs as it converts it to serial data. Latches are used to maintain stable data at the input of the multiplexer. The time required to convert Parallel data to Serial data depends upon the number of parallel bits. A byte parallel data requires 8-bit storage and 8 clocks are required to convert it into serial data. The demerit in a gated D-latch based circuit is the extended enable time. During the time in which the D-latches are enabled data applied at the input of the latches can change. D-latch is said to work in transparent Mode when the enable signal is activated. D-latch operates in the latched mode when the enable signal is inactive. The conversion should only start when the enable signal has been deactivated and the 8-bit data has been stored in the latches. A better and a precise parallel to serial converter circuit uses Edge triggered D-flip-flops. The 8-bit data to be converted into serial data is stored precisely at the clock transition. Thus, if the data changes after the clock transition it has no effect on the data stored in the D flip-flop. Figure 24.1

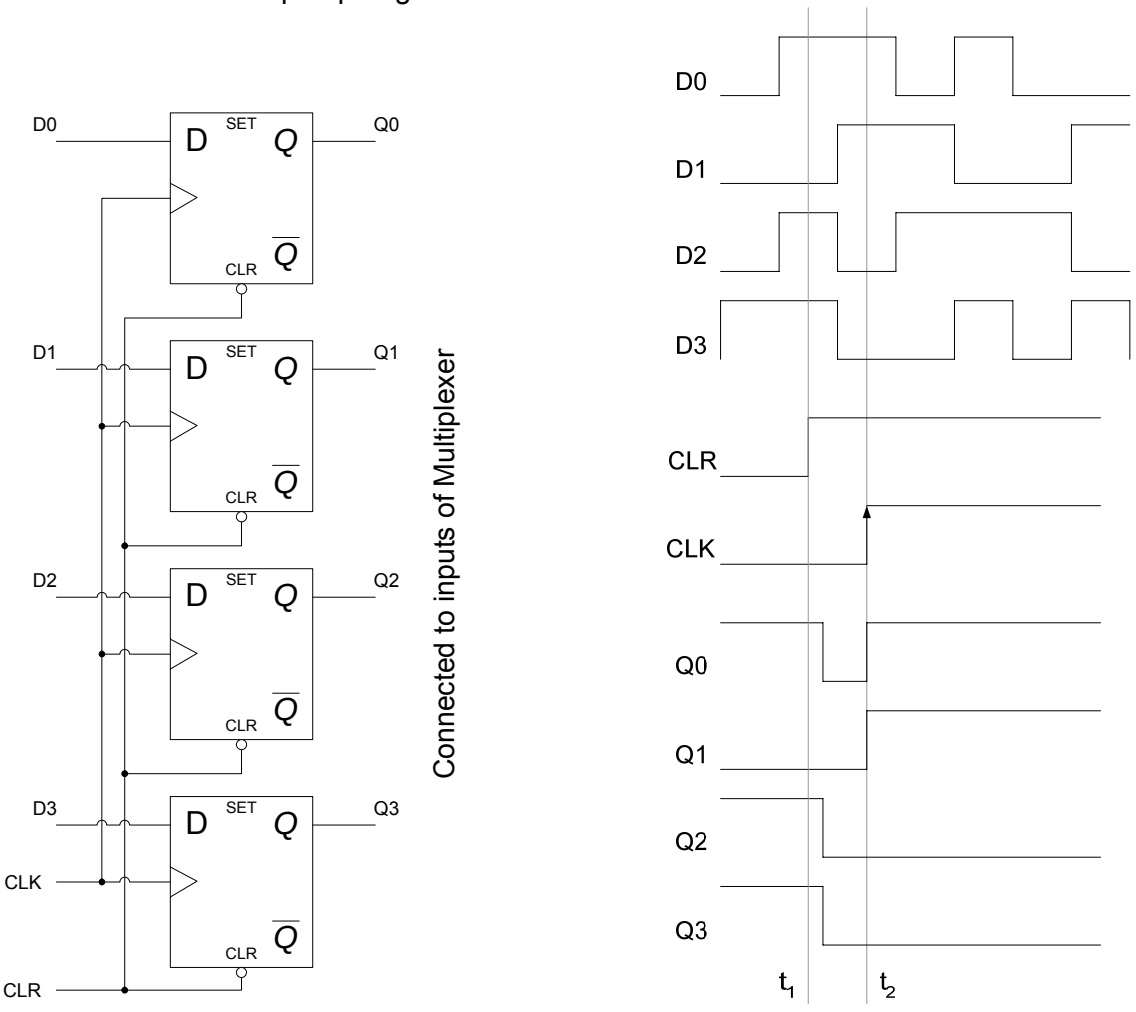


Figure 24.1 D-flip-flops used for Parallel Data Storage

In the timing diagram shown the data at inputs D0, D1, D2 and D3 is constantly changing. At interval t the four D-flip-flops are reset to 0,0,0 and 0 by activating the clear

input. In the timing diagram the outputs of the four D flip-flops are shown set to logic zero after a slight delay. At interval t_2 the clock transition from logic low to logic high latches in the in the data at the inputs of the four D flip-flops. The Q output of all the four latches remains stable after interval t_1. Changes at the D inputs of the four latches do not change the Q outputs of the flour D flip-flops.

In the transparent Mode, the changes in the data applied at the inputs of the latch are seen at the output of the latch, where as in the latched mode changes in the input data are not reflected at the output. Figure 24.2

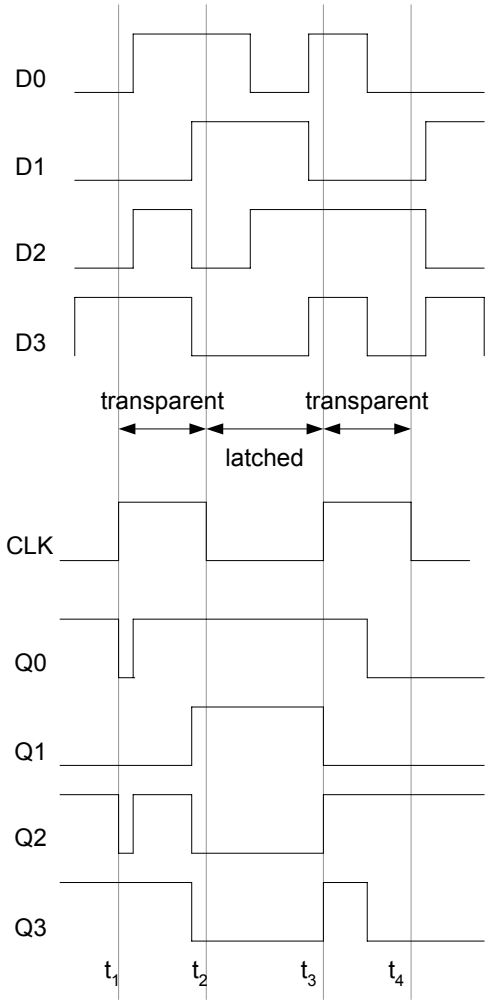


Figure 24.2 Timing diagram of D-Latch

2. Synchronizing Asynchronous inputs using D flip-flop

In synchronized digital systems all the circuits change their state with respect to a common clock and all the input and output signals are synchronized. However, external inputs that are applied to digital circuits through switches and keypads are not synchronized with the clock. The asynchronous inputs can occur at any instant of time. Consider the circuit based on a 2-input AND gate which has a clock signal connected to one of its inputs and the other input is connected to an input de-bounced switch. Figure 24.3. An asynchronous input applied through the switch can cause incomplete or partial pulses at the output of the AND gate.

Figure 24.4. A D-flip-flop synchronizes the input asynchronous signal such that the output of the AND gate has complete clock pulses. Figure 24.5. The timing diagram of the synchronized input circuit is shown in figure 24.6.

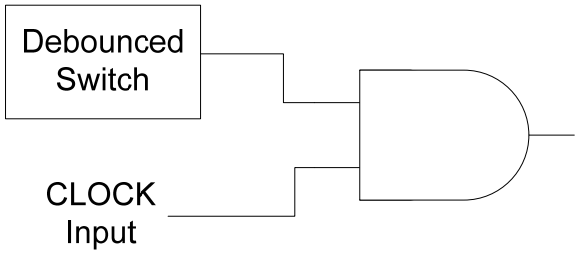


Figure 24.3 AND Gate connected to external switch and clock

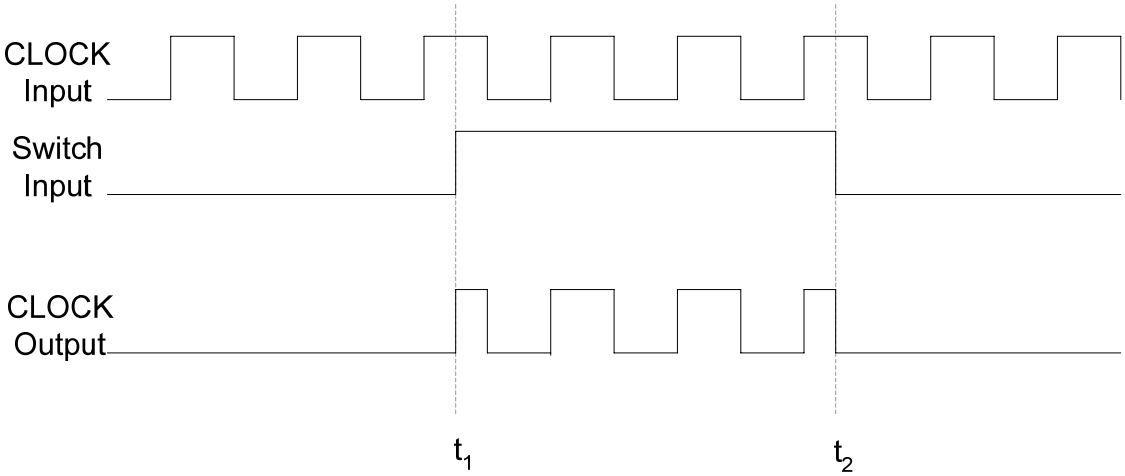


Figure 24.4 Timing Diagram of AND Gate connected to external switch and clock

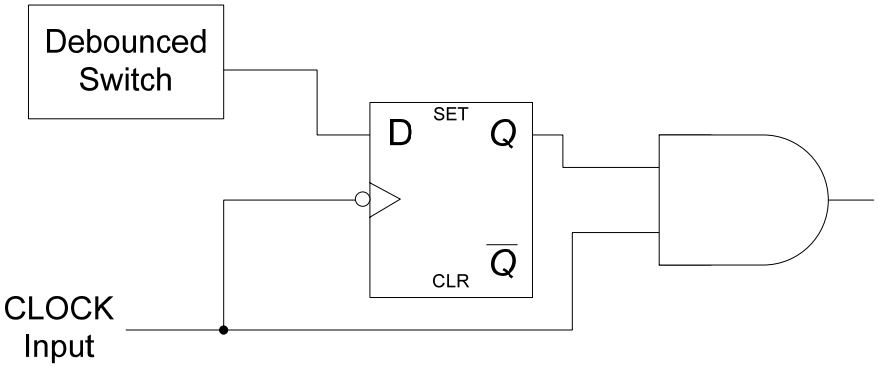


Figure 24.5 D flip-flop used to synchronize the AND Gate output

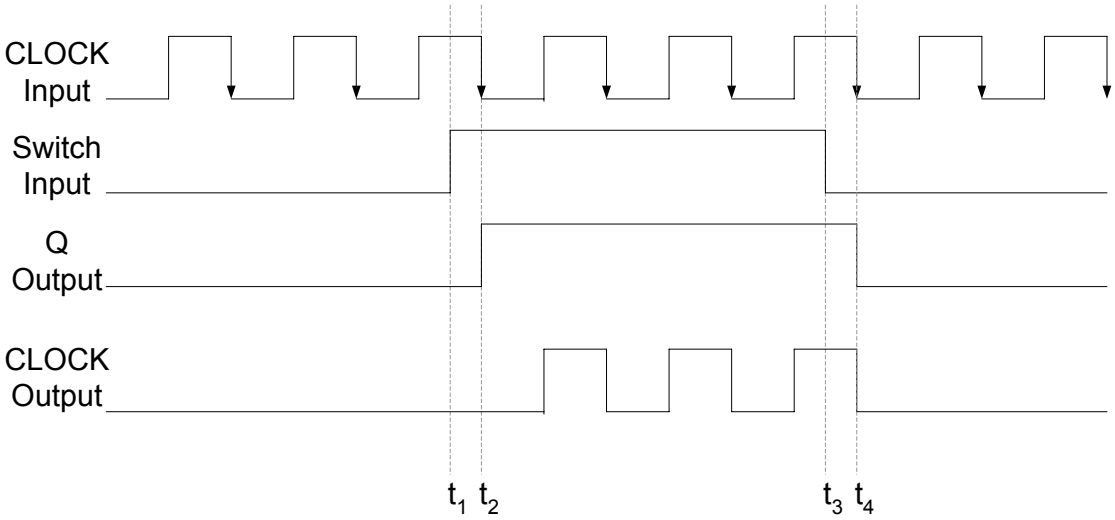


Figure 24.6 Timing Diagram of the synchronized switch input

3. Parallel Data Transfer using D flip-flop

Microprocessor use multi-bit flip-flops to store information. These multi-bit flip-flops are known as registers. These registers for example, can store data generated at the output of the ALU. The registers can also be used to exchange or copy data. Figure 24.7. A register is a set of flip-flops connected in parallel to store multi-bit binary information. The clock inputs of all the flip-flops are connected together, to allow simultaneous latching of the multi-bit input data.

Edge-Triggered J-K Flip-flop

The J-K flip-flop is widely used in digital circuits. Its operation is similar to that of the S-R flip-flop except that the J-K flip-flop doesn't have an invalid state, instead it toggles its state. The circuit diagram of a J-K edge-triggered flip-flop is similar to that of the edge-triggered S-R flip-flop except that the Q and $\overline{Q}$ output of the J-K flip-flop are connected back to the input NAND gates which have the K and J inputs respectively. Figure 24.8. The operation of the J-K flip-flop for different combinations of inputs is described below.

1. J = 0 and K =0

With Q=1 and $\overline{Q}$=0, on a clock transition the outputs of NAND gates 3 and 4 are set to logic 1. With logic 1 value at the inputs of NAND gates 1 and 2 the output Q and $\overline{Q}$ remains unchanged. Similarly, with Q=0 and $\overline{Q}$=1, on a clock transition the outputs of the NAND gates 3 and 4 are set to logic 1. With logic 1 value at the inputs of NAND gates 1 and 2 the output Q and $\overline{Q}$ remains unchanged. Thus when J=0 and K=0 the previous state is maintained and there is no change in the output.

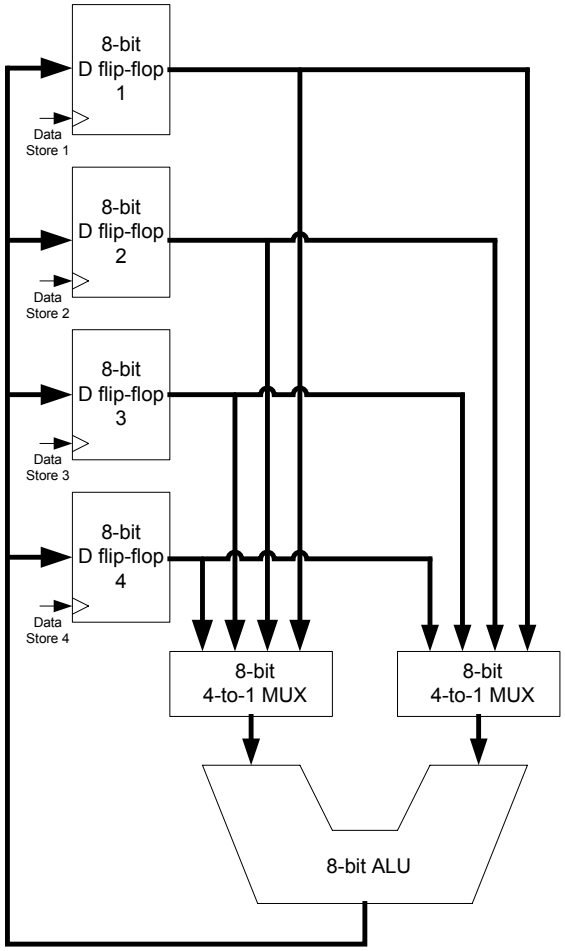


Figure 24.7 D-flip-flops used to store data

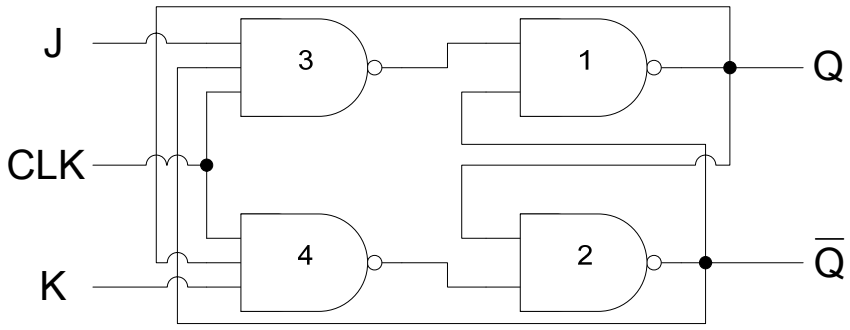


Figure 24.8 Edge-triggered J-K flip-flop

2. J = 0 and K =1

With Q=1 and $\overline{Q}$=0, on a clock transition the output of NAND gate 3 is set to logic 1. The output of the NAND gate 4 is set to 0 as all three of its inputs are at logic 1. The logic 1 and 0 at the inputs of the NAND gates 3 and 4 respectively resets the Q output to 0 and $\overline{Q}$ to

1. With Q=0 and $\overline{Q}$=1, on a clock transition the output of NAND gate 3 is set to logic 1. The output of the NAND gate 4 is also set to 1 as the input of the NAND gate 4 is connected to Q=0. Logic 1 at the inputs of the NAND gates 3 and 4 respectively retains the Q and $\overline{Q}$ to 0 and 1 respectively. Thus when J=0 and K=1 the J-K flip-flop irrespective of its earlier state is rest to state Q=0 and $\overline{Q}$=1.

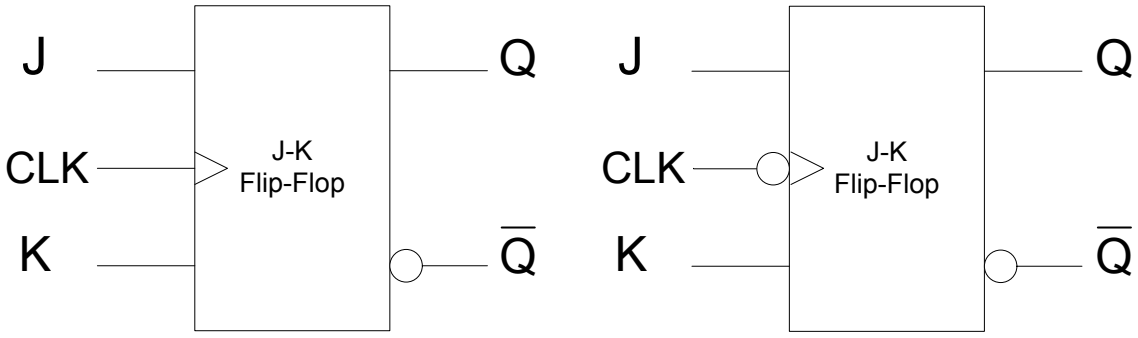
3. J = 1 and K =0

With Q=1 and $\overline{Q}$=0, on a clock transition the output of NAND gate 4 is set to logic 1. The output of the NAND gate 3 is also set to 1 as its input connected to $\overline{Q}$ is at logic 0. Thus inputs 1 and 1 at inputs of NAND gates 1 and 2 retain the Q and $\overline{Q}$ output to 1 and 0 respectively. With Q=0 and $\overline{Q}$=1, on a clock transition the output of NAND gate 4 is set to logic 1. The output of the NAND gate 3 is set to 0 as all its input are at logic 1. Thus inputs 0 and 1 at inputs of NAND gates 1 and 2 sets the flip-flop to Q=1 and $\overline{Q}$=0. Thus when J=1 and K=0 the J-K flip-flop irrespective of its output state is set to state Q=1 and $\overline{Q}$=0.

4. J = 1 and K =1

With Q=1 and $\overline{Q}$=0, on a clock transition the output of the NAND gates 3 and 4 depend on the outputs Q and $\overline{Q}$. The output of NAND gate 3 is set to 1 as $\overline{Q}$ is connected to its input. The output of NAND gate 4 is set to 0 as all its inputs including Q is at logic 1. A logic 1 and 0 at the input of gates 1 and 2 toggles the outputs Q and $\overline{Q}$ from logic 1 and 0 to 0 and 1 respectively. With Q=0 and $\overline{Q}$=1, on a clock transition the output of NAND gate 3 is set to 0 as $\overline{Q}$ and the output of NAND gate 4 is set to 1. A logic 0 and 1 at the input toggles the outputs Q and $\overline{Q}$ from logic 0 and 1 to 1 and 0 respectively.

In summary when J-K inputs are both set to logic 0, the output remains unchanged. At J=0 and K=1 the J-K flip-flop is reset to Q=0 and $\overline{Q}$=1. At J=1 and K=0 the flip-flop is set to Q=1 and $\overline{Q}$=0. With J=1 and K=1 the output toggles from the previous state. The truth tables of the positive and negative edge triggered J-K flip-flops are shown in table 24.1. The logic symbols of the J-K flip-flops are shown in figure 24.9. The timing diagrams of the J-K flip-flops are shown in figure 24.10.



Input			Output
CLK	J	K	Q_{t+1}
0	x	X	Q_t
1	x	X	Q_t
↑	0	0	Q_t
↑	0	1	0
↑	1	0	1
↑	1	1	$\overline{Q}_t$

Input			Output
CLK	J	K	Q_{t+1}
0	x	x	Q_t
1	x	x	Q_t
↓	0	0	Q_t
↓	0	1	0
↓	1	0	1
↓	1	1	$\overline{Q}_t$

Table 24.1 Truth-Table of Positive and Negative Edge triggered J-K flip-flops

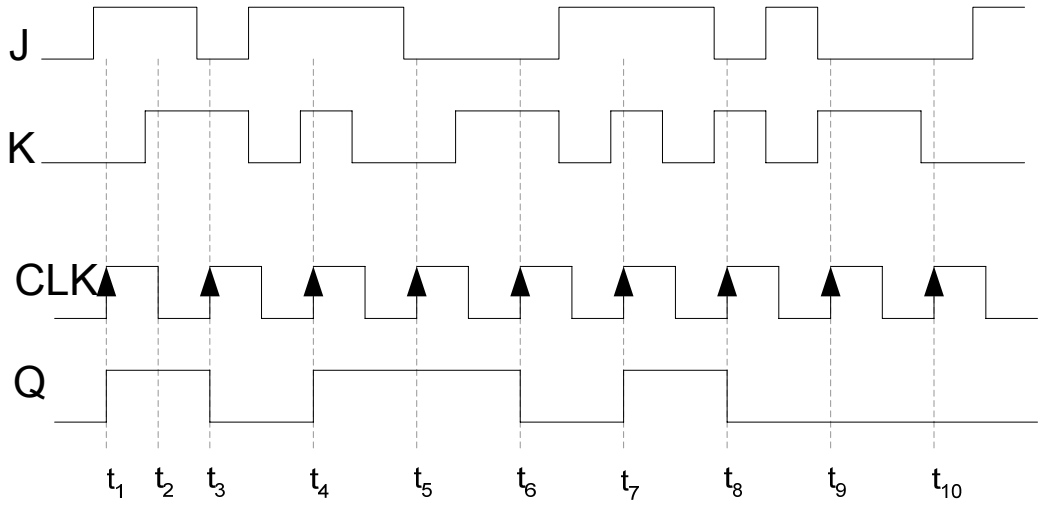


Figure 24.10a Timing diagram of a Positive Edge triggered J-K flip-flop

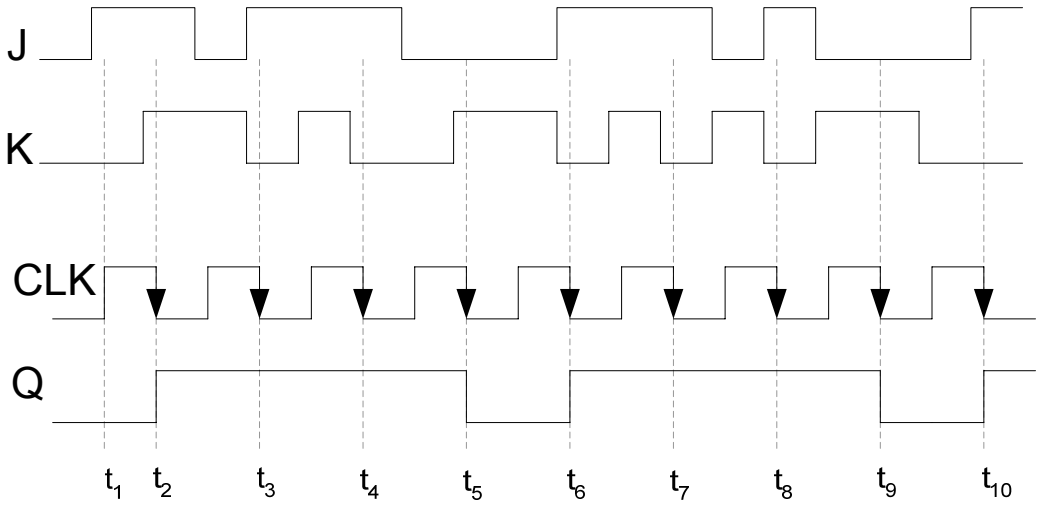


Figure 24.10b Timing diagram of a Negative Edge triggered J-K flip-flop

Applications of Edge-Triggered J-K Flip-flop

1. J-K flip-flop used as sequence detector

Some digital applications require that the inputs be applied in a certain sequence to activate an output. This is possible with J-K flip-flops. Figure 24.11

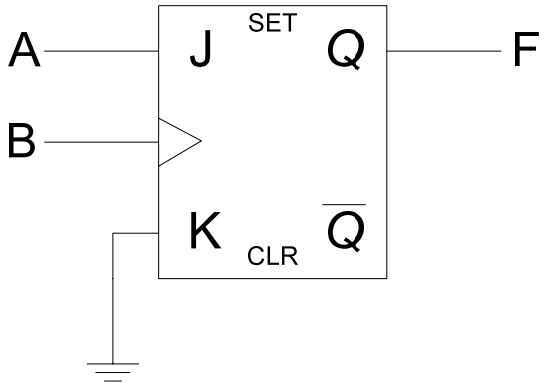
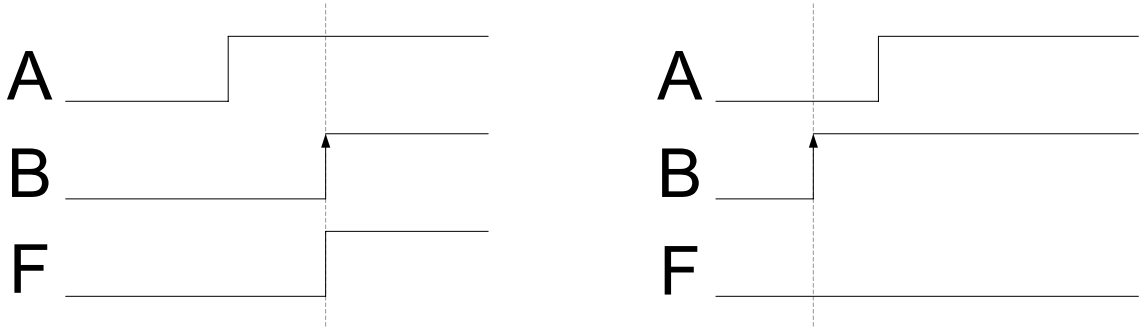


Figure 24.11a J-K flip-flop connected to respond to a particular input sequence



2. J-K flip-flop used as frequency divider

In digital circuit different parts of the circuit can operate at different frequencies obtained from the master clock frequency. For example, three different parts of a digital system might operate at 4 MHZ, 2 MHZ and 1 MHZ clock frequency respectively. Same clock source should be used (instead of three separate clock sources) to maintain synchronization between the three parts. A clock frequency can be divided by 2 using a J-K flip flop. The J-K inputs of the flip-flop are connected to logic high (1). At each clock transition the output of the flip-flop toggles to the alternate state. Figure 24.12. A 4MHz clock signal can be divided into 2 MHZ and 1 MHZ signal using two J-K flip-flops connected together. Figure 24.13.

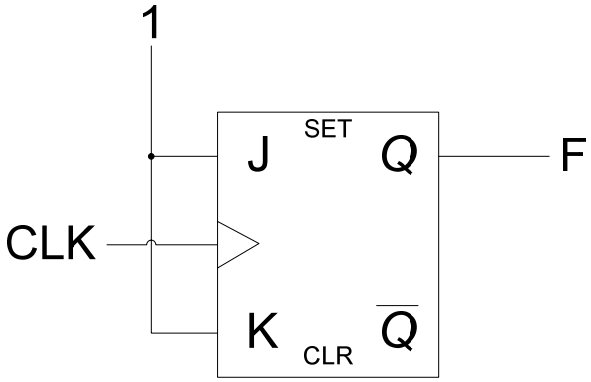


Figure 24.12a J-K flip-flop connected as frequency divider

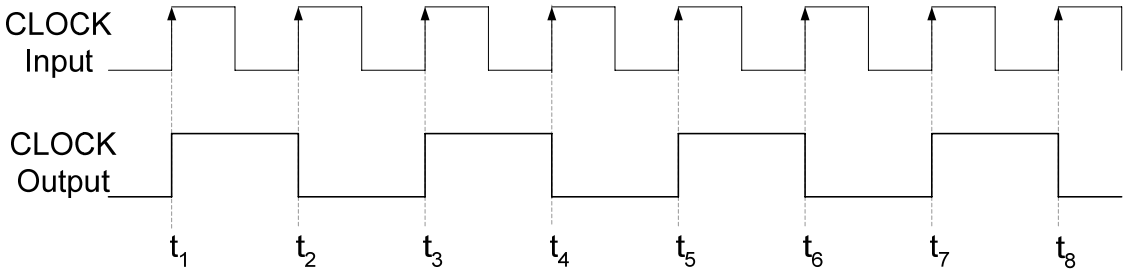


Figure 24.12b Timing diagram of J-K flip-flop frequency divider

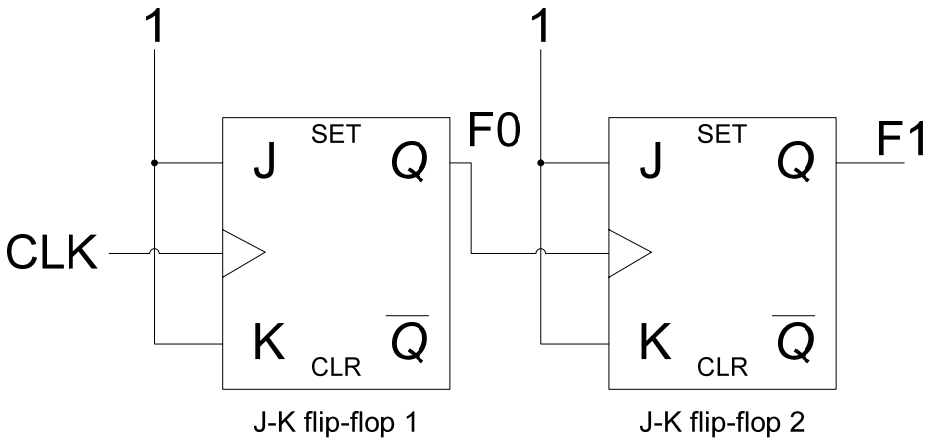


Figure 24.13a J-K flip-flop connected as divide-by-4 frequency divider

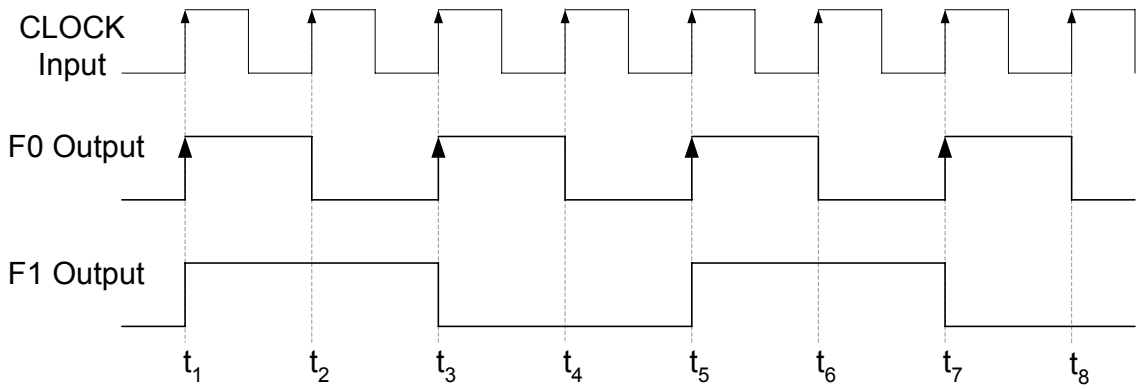


Figure 24.13b Timing diagram of J-K divide-by-4 frequency divider

3. J-K flip-flop used as a shift register

Binary numbers can be multiplied or divided by a constant 2 by shifting the binary numbers left or right by 1-bit respectively. Multiplication and Division by a factor of 2^n, (where n = 1, 2, 3, 4 ….) can be achieved by shifting the binary by n bits to the left or right respectively. Binary numbers can be easily shifted in the left or right direction by using J-K flip-flop based shift registers. figure 24.14.

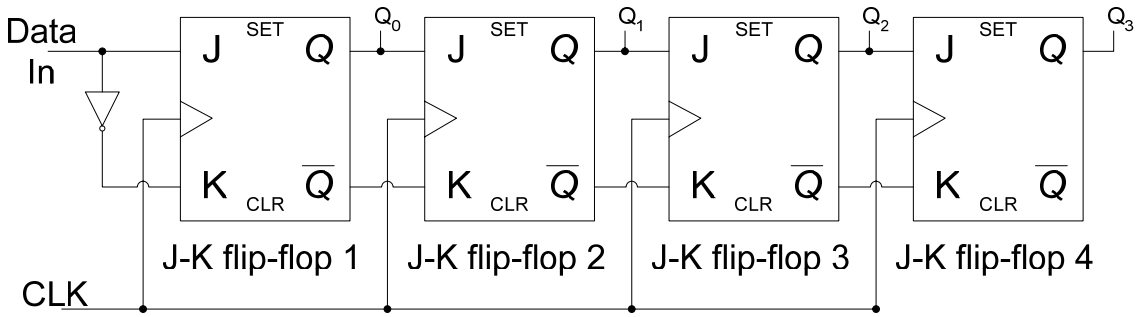


Figure 24.14a 4-bit right shift register

4. J-K flip-flop used as a counter

Counters which count up or count down are commonly used in digital circuits. An up-counter counts up from 0 to 10 increments to the next higher count value on the application of each clock signal. Similarly, a down-counter counts down to the next lower count value on the application of each clock pulse. Figure 24.15.

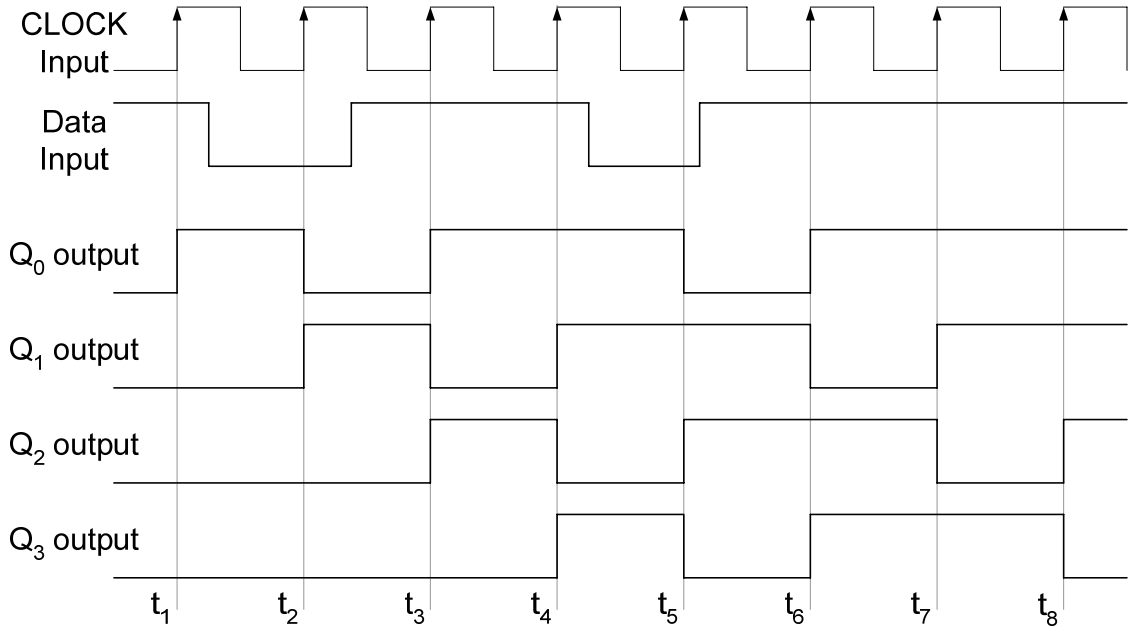


Figure 24.14b Timing diagram of a 4-bit right shift register

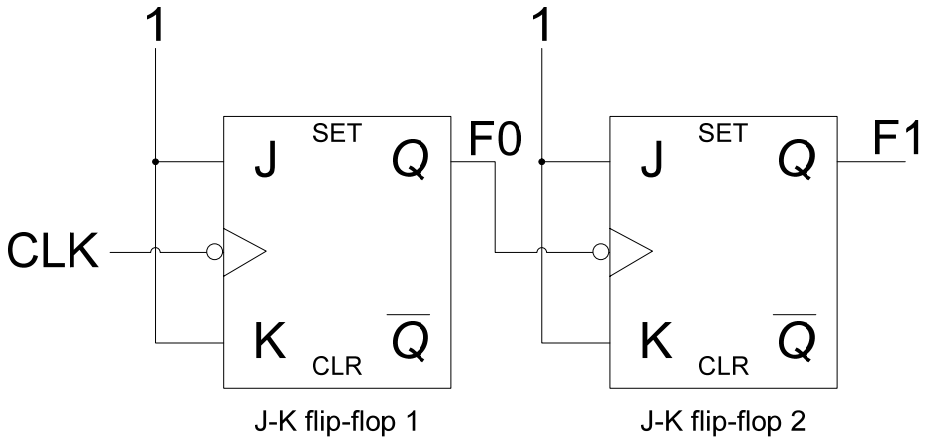


Figure 24.15a 2-bit up-counter

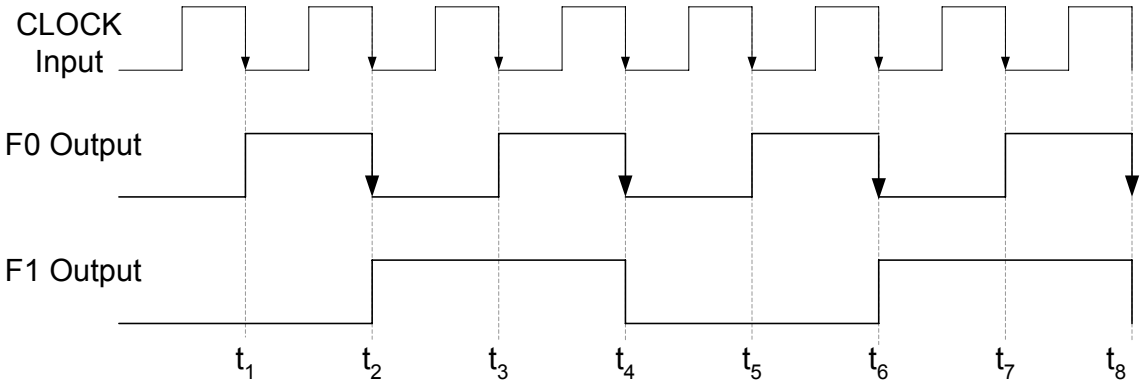


Figure 24.15b Timing diagram of a 2-bit up-counter

Lesson No. 25

ASYNCHRONOUS PRESET AND CLEAR INPUTS

The S-R, J-K and D inputs are known as synchronous inputs because the outputs change when appropriate input values are applied at the inputs and a clock signal is applied at the clock input. If there is no clock transition then the inputs have no effect on the output. Digital circuits require that the flip-flops be set or reset to some initial state before a new set of inputs is applied for changing the output. The flip-flops are set-reset to some initial state by using asynchronous inputs known as Preset and Clear inputs. Since these inputs change the output to a known logic level independently of the clock signal therefore these inputs are known as asynchronous inputs. The circuit diagram of a J-K flip-flop with Preset and Set Asynchronous inputs is shown in figure 25.1a. The asynchronous inputs override the synchronous inputs thus to operate the flip-flop in the synchronous mode the asynchronous inputs have to be disabled.

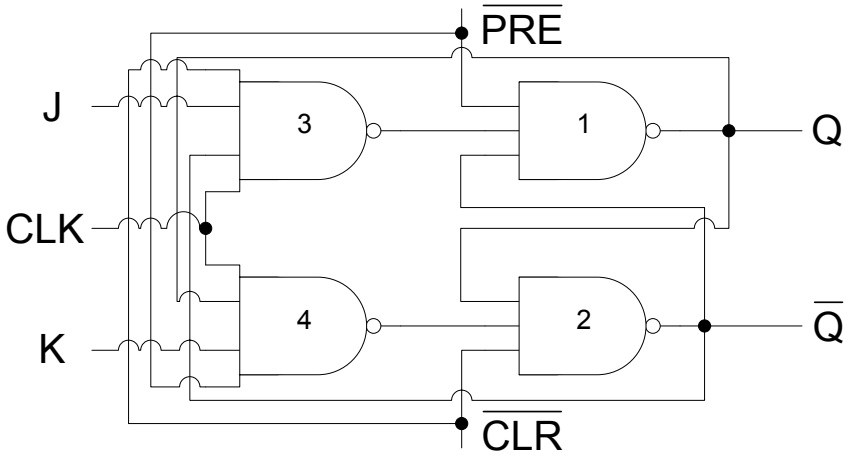


Figure 25.1a J-K flip-flop with Asynchronous Preset and Clear inputs

To preset the flip-flop to Q=1 and $\overline{Q}$=0 the $\overline{PRE}$ input is set to 0 which sets the Q output to 1 and the output of NAND gate 4 to 1. The $\overline{CLR}$ input is set to 1, the remaining two inputs (Q and output of NAND gate 4) of the NAND gate 2 are also set at logic 1, therefore $\overline{Q}$ output is set to 0. The flip-flop is cleared to Q=0 and $\overline{Q}$=1 by setting the $\overline{PRE}$ input is set to 1 and the $\overline{CLR}$ input is to 0. The $\overline{CLR}$ input set to 0 sets $\overline{Q}$=1 it also sets the output of NAND gate 3 to 1. The $\overline{PRE}$ input along with the other two inputs of NAND gate 1 are set at logic 1 which sets the output Q to 0. When the $\overline{PRE}$ and the $\overline{CLR}$ inputs are used inputs J and K have no effect on the operation of the flip-flop. To use the flip-flop with synchronous inputs J-K, the $\overline{PRE}$ and the $\overline{CLR}$ inputs are set to logic 1. Setting $\overline{PRE}$ and the $\overline{CLR}$ to logic 0 is not allowed.

Logic symbol of a J-K edge-triggered flip-flop with synchronous and asynchronous inputs is shown in figure 25.1b. The truth table of a J-K flip-flop with Asynchronous inputs is shown in table 25.1. The timing diagram describes the effect of asynchronous inputs on the operation of the flip-flop. Figure 25.1c

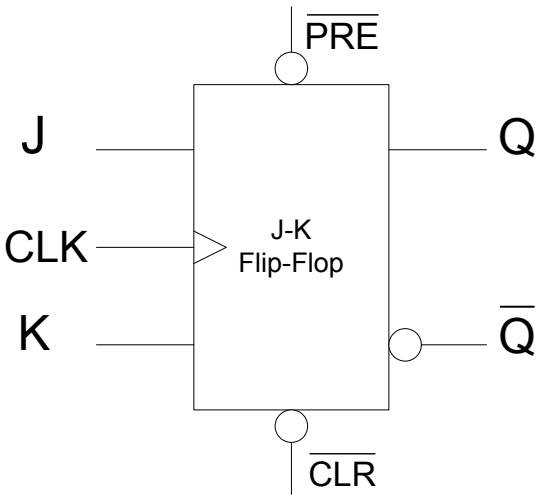
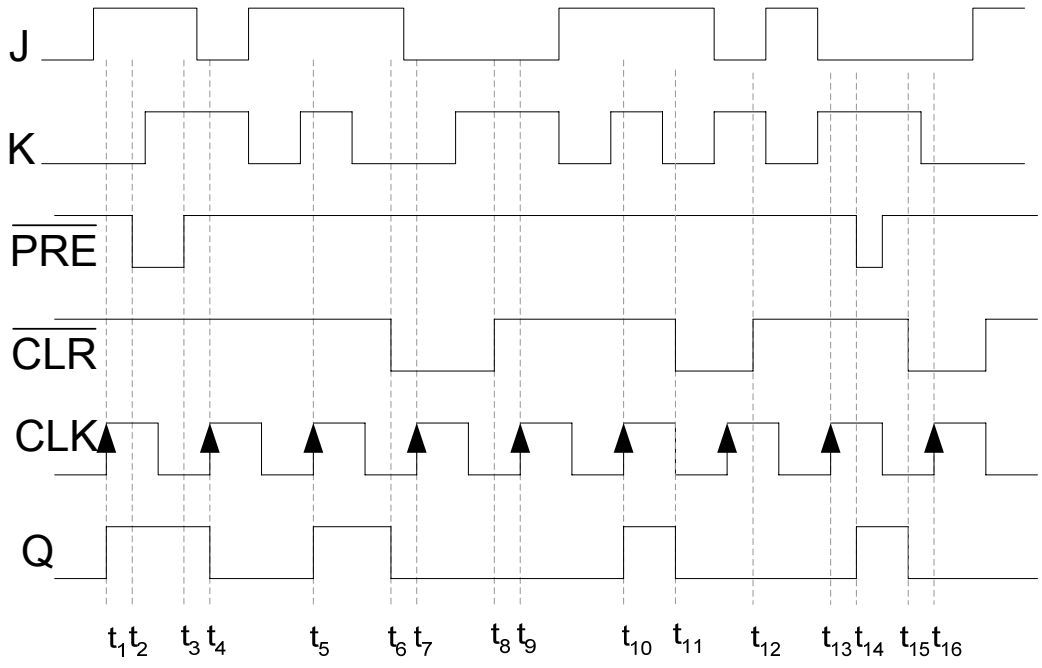


Figure 25.1b Logic Symbol of a J-K flip-flop with Asynchronous inputs

Input		Output
$\overline{PRE}$	$\overline{CLR}$	Q_{t+1}
0	0	Invalid
0	1	1
1	0	0
1	1	Clocked operation

Table 25.1 Truth table of J-K flip-flop with Asynchronous inputs



The 74HC74 Dual Positive-Edge triggered D flip-flop

The edge-triggered D flip-flop with asynchronous inputs is available as an Integrated Circuit. The 74HC74 has dual D-flip-flops with independent clock inputs, synchronous and asynchronous inputs.

The 74HC112 Dual Positive-Edge triggered J-K flip-flop

The edge-triggered D flip-flop with asynchronous inputs is available as an Integrated Circuit. The 74HC112 has dual J-K-flip-flops with independent clock inputs, synchronous and asynchronous inputs.

Master-Slave Flip-Flops

Master-Slave flip-flops have become obsolete and are replaced by edge-triggered flip-flops. Master-Slave flips have two stages each stage works in one half of the clock signal. The inputs are applied in the first half of the clock signal. The outputs do not change until the second half of the clock signal. As mentioned earlier the use of edge-triggered flip-flip is to synchronize the operation of a digital circuit with a common clock signal. The master-slave setup also allows digital circuits to operate in synchronization with a common clock signal. The circuit diagram of the master-slave J-K flip-flop is shown in figure 25.2a. The Master-Slave flip-flop is composed of two parts the Master and the Slave. Both the Master and the Slave are Gated S-R flip-flops. The Master-Slave flip-flop is not synchronised with the clock positive or negative transition, rather it known as a pulse triggered flip-flop as it operates at the positive and negative clock cycles.

Consider that the J-K inputs of the flip-flop are set at 1 and 0 respectively. The outputs Q and $\overline{Q}$ are initially set at 1 and 0 respectively. During the positive half of the clock gates 3 and 4 are both enabled by the clock signal. The output of gate 3 is set to 1 due to the $\overline{Q}$ output set at 0. Similarly the output of gate 4 is also set at 1 due to the K input set at 0. The outputs of gates 1 and 2 remain unchanged as the inputs to gates 1 and 2 are both logic 1. Assume the outputs of gates 1 and 2 to be 1 and 0 respectively. During the positive half cycle, the clock input to gates 7 and 8 is inverted therefore both the gates are disabled and their output is set to logic 1. With logic 1 at the inputs of gates 5 and 6 the output Q and $\overline{Q}$ remains unchanged throughout the positive half of the clock cycle. During the negative half of the clock cycle the Master flip-flop is disabled and the output of the Master flip-flop remains fixed during the negative half cycle. The Slave flip-flop is enabled and the 1 and 0 outputs of the Master flip-flop set the Q and $\overline{Q}$ output to 1 and 0 respectively.

Initially, if the Q and $\overline{Q}$ outputs are 0 and 1 respectively, setting the J and K inputs to 1 and 0 respectively sets the output to 1 and 0 respectively. During the positive half of the clock the Master flip-flop is enabled, the output of gate 3 is set to 0 as the J, Q and CLK inputs are all at logic 1. The output of gate 4 is set to 1 as the K input is logic 0. These inputs set the output of the Master flip-flop at gates 1 and 2 to logic 1 and 0 respectively. During the negative half of the clock cycle the Slave flip-flop is enabled the output Q and $\overline{Q}$ are set to logic 1 and 0 respectively.

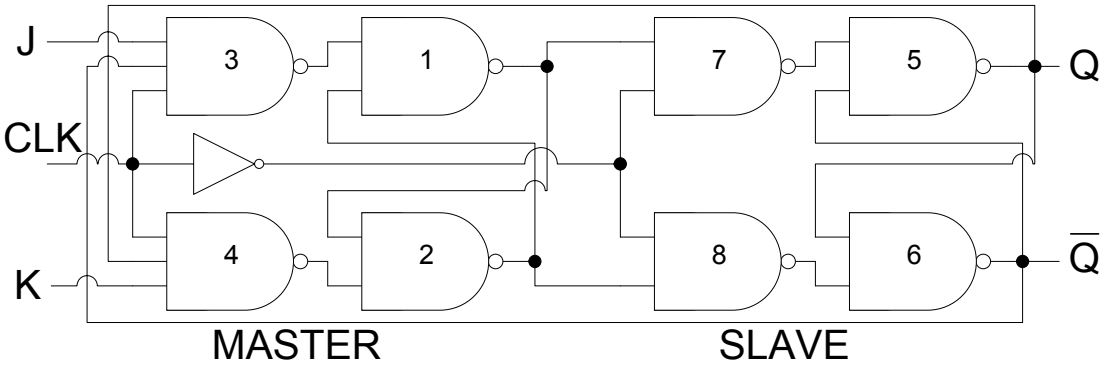


Figure 25.2a Master-Slave flip-flop

The truth-table of the master-slave flip-flop is shown in table 25.2. The timing diagram of the master-slave flip-flop is shown in figure 25.2b.

Input			Output
CLK	J	K	Q_{t+1}
Pulse	0	0	Q_t
Pulse	0	1	0
Pulse	1	0	1
Pulse	1	1	$\overline{Q}_t$

Table 25.2 Truth table of the Master-Slave J-K flip-flop

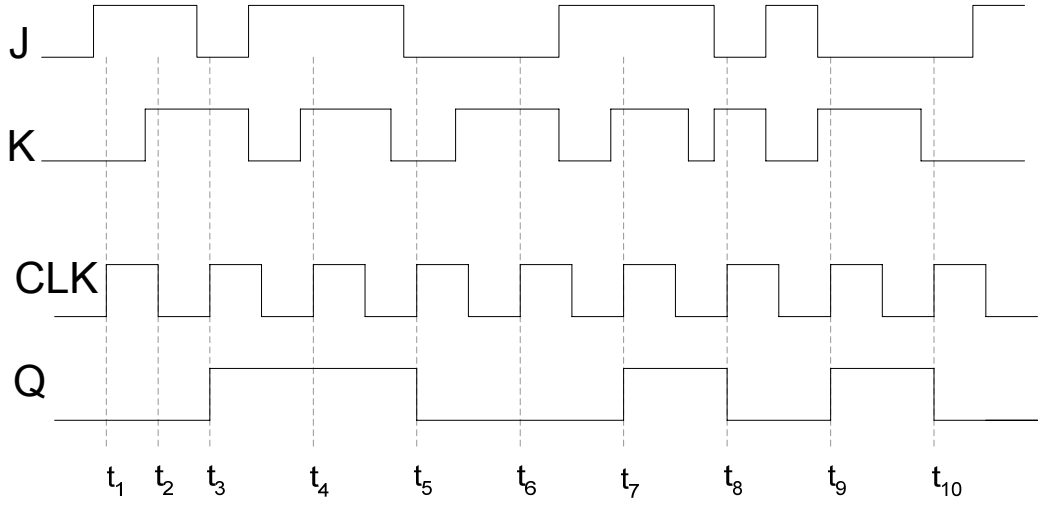


Figure 25.2b Timing diagram of a Master Slave J-K flip-flop

Flip-Flop Operating Characteristics

The performance of the flip-flop is specified by several operating characteristics

- Propagation Delay
- Set-up Time
- Hold Time
- Maximum Clock frequency
- Pulse width
- Power Dissipation

Propagation Delay

The propagation delay time is the interval of time when the input is applied and the output changes. Four different types of Propagation Delays are measured.

5. Propagtaion Delay t_{PLH} measured with respect to the triggering edge of the clock to the low-to-high transition of the output. Figure 25.3. On a positive or negative clock transition the flip-flop changes its output state. The Propagation Delay is measured at 50% transition mark on the triggering edge of the clock and the 50% mark on the low-to-high transition of the output that occurs due to the clock transition.

6. Propagtaion Delay t_{PHL} measured with respect to the triggering edge of the clock to the high-to-low transition of the output. Figure 25.4. On a positive or negative clock transition the flip-flop changes its output state. The Propagation Delay is measured at 50% transition mark on the triggering edge of the clock and the 50% mark on the high-to-low transition of the output that occurs due to the clock transition.

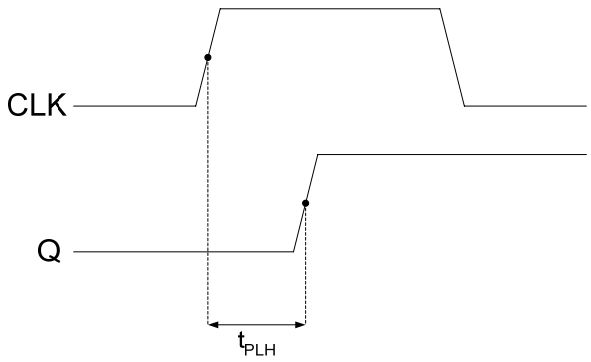


Figure 25.3 Propagation Delay, clock to low-to-high transition of the output

7. Propagtaion Delay t_{PLH} measured with respect to the leading edge of the preset input to the low-to-high transition of the output. Figure 25.5. On a high-to-low transition of the preset signal the flip-flop changes its output state to logic high. The Propagation Delay is measured at 50% transition mark on the triggering edge of the preset signal and the 50% mark on the low-to-high transition of the output that occurs due to the preset signal.

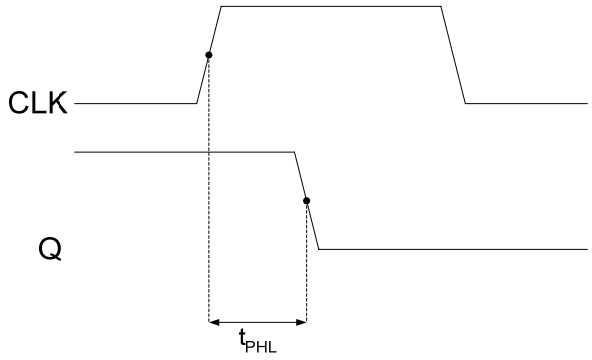


Figure 25.4 Propagation Delay, clock to high-to-low transition of the output

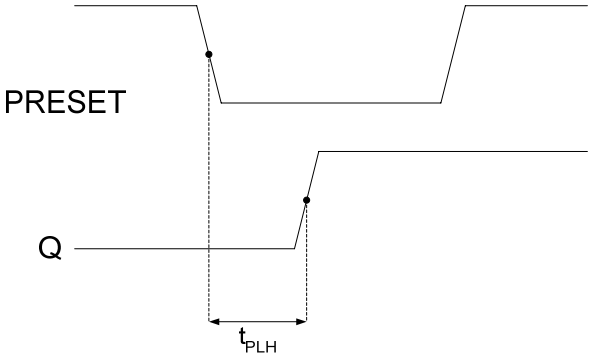


Figure 25.5 Propagation Delay, preset to low-to-high transition of the output

8. Propagtaion Delay t_{PHL} measured with respect to the leading edge of the clear input to the high-to-low transition of the output. Figure 25.6. On a high-to-low transition of the clear signal the flip-flop changes its output state to logic low. The Propagation Delay is measured at 50% transition mark on the triggering edge of the clear signal and the 50% mark on the high-to-low transition of the output that occurs due to the preset signal.

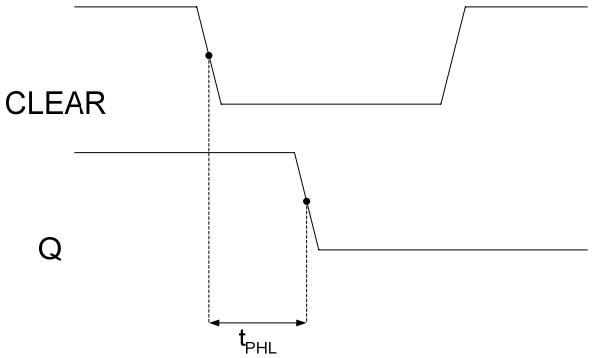


Figure 25.6 Propagation Delay, clear to high-to-low transition of the output

Set-up Time

When a clock transition occurs at the clock input of a flip-flop the output of the flip-flop is set to a new state based on the inputs. For the flip-flop to change its output to a new state at

levels to remain stable before the clock transition occurs is known as the Set-up time. Figure 25.7

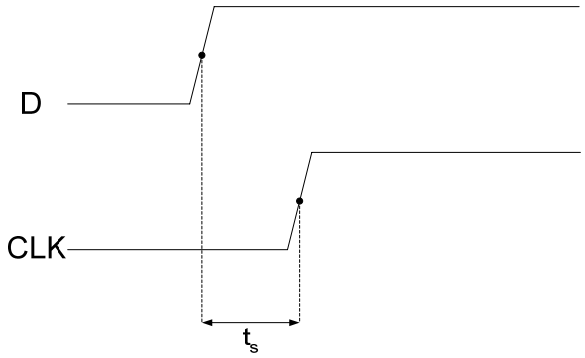


Figure 25.7 Set-up time for a D flip-flop

Hold Time

The input signal maintained at the flip-flop input has to be maintained for a minimum time after the clock transition for the flip-flop to reliably clock in the input signal. The minimum time for which the input signal has to be maintained at the input is the Hold time of the flip-flop. Figure 25.8

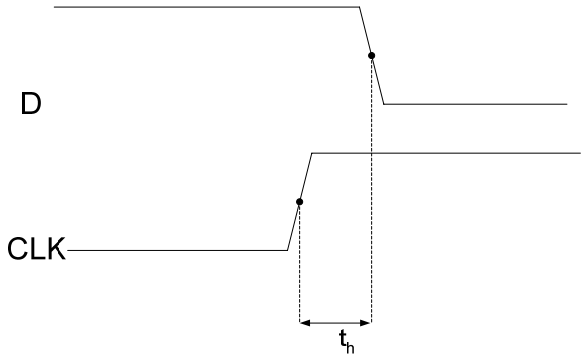


Figure 25.8 Hold time for a D flip-flop

Maximum Clock Frequency

A flip-flop can be operated at a certain clock frequency. If the clock frequency is increased beyond a certain limit the flip-flop will be unable to respond to the fast changing clock transitions, therefore the flip-flop will be unable to function. The maximum clock frequency f_{max} is the highest rate at which the flip-flop operates reliably.

Pulse Width

A flip-flop uses the clock, preset and clear inputs for its operation. Each signal has to be of a specified duration for correct operation of the flip-flop. The manufacturer specifies the minimum pulse width t_w for each of the three signals. The clock signal is specified by minimum high time and minimum low time.

Power Dissipation

A flip-flop consumes power during its operation. The power consumed by a flip-flop is defined by $P = V_{cc} \times I_{cc}$. The flip-flop is connected to +5 volts and it draws 5 mA of current during its operation, therefore the power dissipation of the flip-flop is 25 mW.

A digital circuit is made of a number of gates, functional units and flip-flops. The total power requirement of each device should be known so that an appropriate dc power source is used to supply power to the digital circuit.

One-Shot Mono-stable multi-vibrator

Bi-stable devices remain in either of their two states unless the inputs force the device to switch its state. The device remains in its alternate state unless the inputs are changed again to force the device back to its original state. A mono-stable device only has a single stable state and it remains in its stable state. It temporarily changes to its unstable state when it is triggered. It remains in its unstable state for a predetermined length of time and then it automatically switches back to its stable state. The length of time for which the device remains in the unstable state is determined by the time constant determined by the Resistor and Capacitor connected externally to the mon-stable device. The output of the device is a pulse having a time duration determined by R and C. These mono-stable devices are also known as One-Shots. Figure 25.9. One-Shots are of two types, the nonretriggerable and retriggerable.

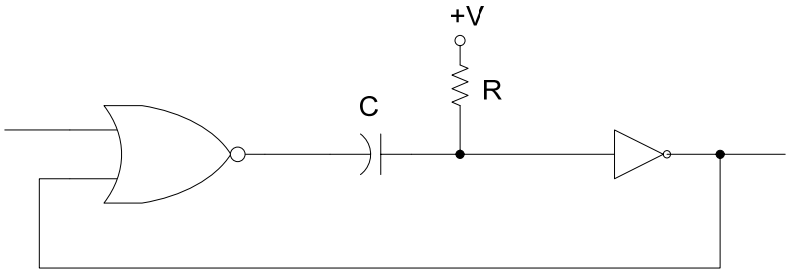


Figure 25.9a Circuit diagram of a One-Shot

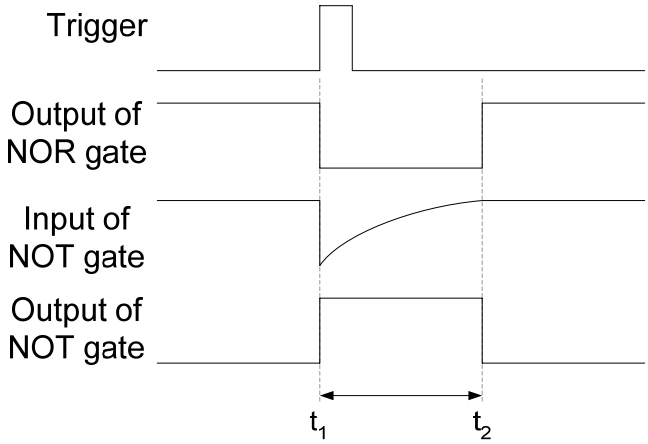


Figure 25.9b Timing diagram of a One-Shot

The One-Shot is triggered by applying a short pulse at the input of the NOR gate at time interval t_1. The One-Shot is in its stable state with output at logic zero at time interval < t_1.

logic low output of the NOR gate is inverted into logic high by the NOT gate and the One-Shot is in unstable state at the start of interval t_1. The logic high output of the NOT gate is connected back to the second input of the NOR gate, which maintains the output of the NOR gate at logic low. When the output of the NOR gate is set to logic low at interval t_1, the capacitor C begins charging through the Resistor R. The charging time (in seconds) is determined by the time constant RC. During the charging of the capacitor during interval t_1 to t_2, the input of the NOT gate remains at logic low, therefore the output of the NOT gate remains in the unstable state at logic high. When the capacitor is fully charged to potential +V (logic high) at time interval t_2, the NOT gate input also become logic high, which sets the output of the NOT gate to logic low. With the setting of the NOT gate output to logic low at interval t_2, the One-Shot id switched back to its stable state. The interval t_1 to t_2 during which the One-Shot is in its unstable state is determined by the time constant RC.

1. Nonretriggerable One-Shot

A nonretriggerable One–Shot is triggered to its unstable state.

a. The One-Shot output remains in the unstable state for a fixed period of time on each trigger input.
b. The One-Shot will have to return to its stable state before it can be triggered again. If it is already in its unstable state due to application of a trigger input, a new trigger input will have no effect.
c. The duration of trigger input pulses has no effect on the output pulse duration. The One-Shot is triggered either on the positive or the negative edge. Figure 25.10

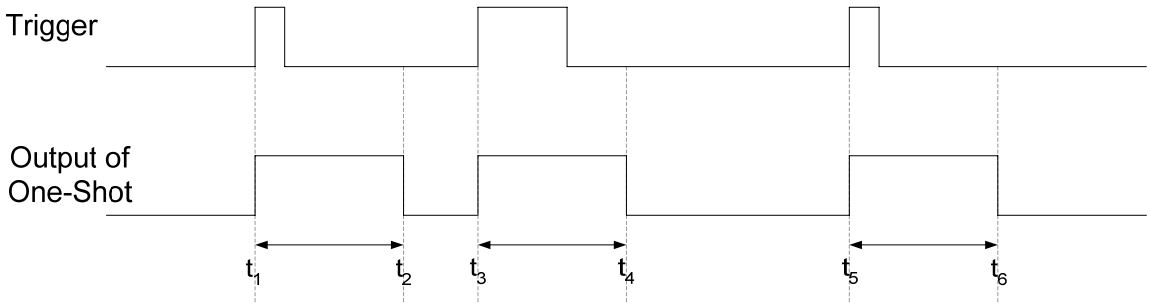


Figure 25.10a Timing diagram of a non-retriggerable One-Shot

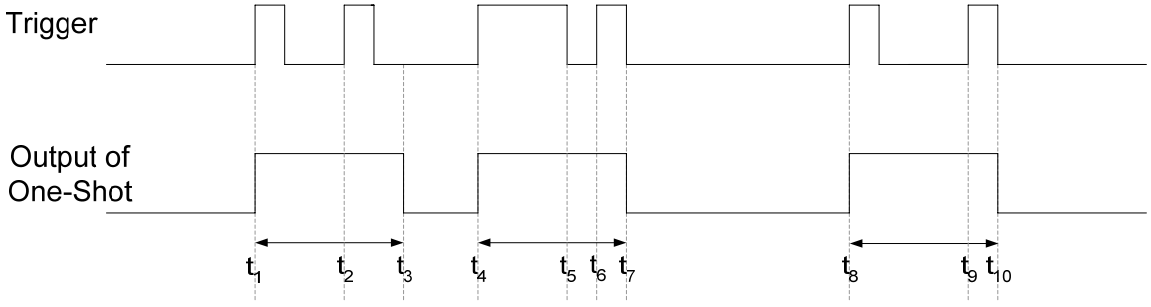


Figure 25.10b Timing diagram of a non-retriggerable One-Shot with ignored triggers

2. Retriggerable One-Shot

A retriggerabe One-Shot operation is very similar to that of the Nonretriggerbale One-Shot except that the retriggerable One-Shot will retrigger even if it is in its unstable state. Figure 25.11. The retriggerable and Nonretriggerbale are available in Integrated Circuit form.

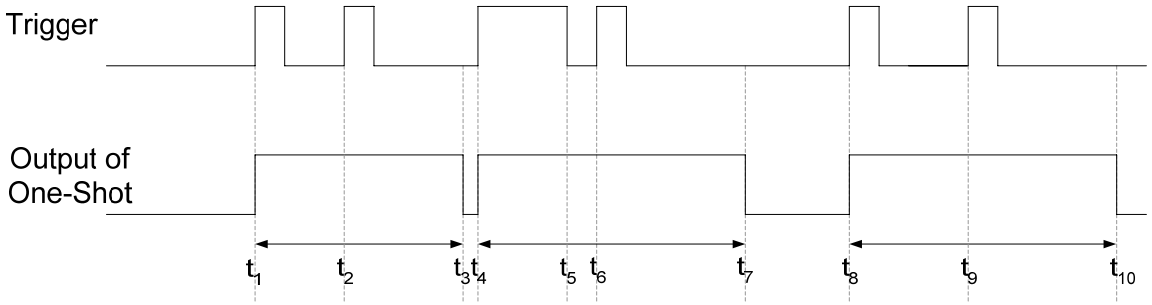


Figure 25.11 Timing diagram of a Retriggerable One-Shot

Lesson No. 26

THE 555 TIMER

The 555 Timer is a versatile and widely used device which can be configured as a mono-stable One-Shot or as an Astable multivibrator. An Astable multivibrator is known as an Oscillator which does not have any stable state. Therefore it continuously changes from one unstable state to the other without any external trigger.

Timing Problem in flip-flop circuits

In synchronous digital circuits the output of one flip-flop is connected to the input of a second flip-flop, either directly or through logic gates. Both the flip-flops are triggered through a common clock signal connected to the clock input of both the flip-flops. This leads to a potential timing problem as shown in figure 26.1.

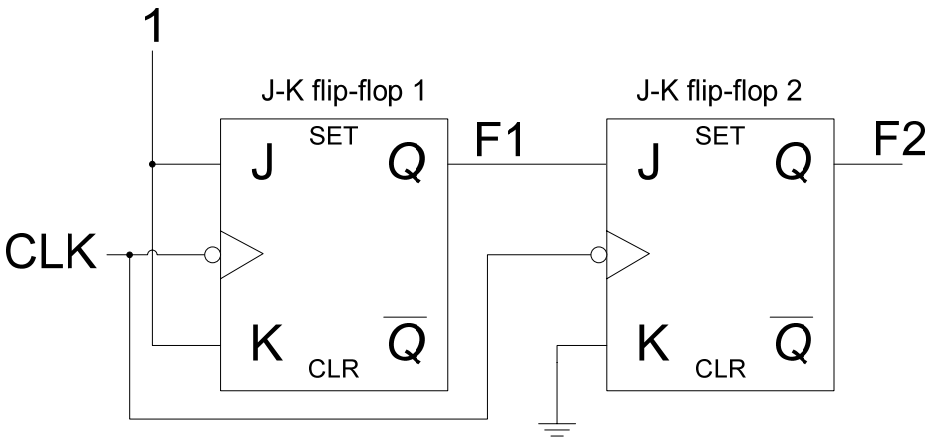


Figure 26.1a J-K flip-flop circuit with potential timing problem

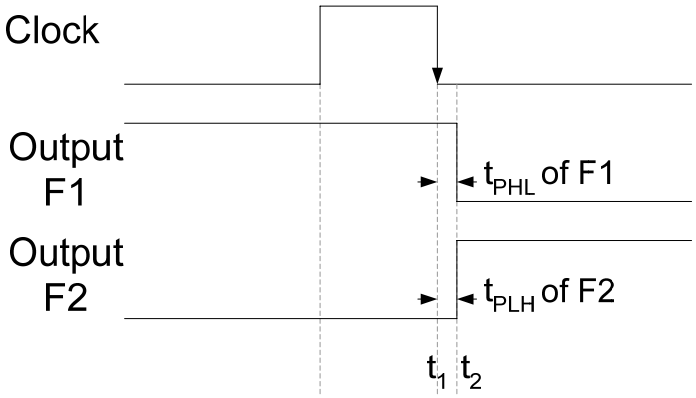


Figure 26.1b Timing diagram of J-K flip-flop circuit with potential timing problem

Assume the initial outputs of flip-flop 1 and 2 are at logic high and low respectively. When there is a high to low clock transition t_1, the output of flip-flop 1 toggles to logic low. The high to low clock transition at the clock input of Flip-flop 2 also occurs at the same instant t_1. During the interval t_1 and t_2 the output of flip-flop is changing from logic high to logic low and will go to logic low after a propagation delay t_{PHL}. The input to flip-flop 2 is changing from logic high to low during the time interval t_1 and t_2. The input to flip-flop 2 should be held stable for a minimum hold time requirement t . If the input is not held stable for t interval the output can

not be guaranteed to be logic high. Output of flip-flop 2 will be set to logic high state if t_{PHL} time of flip-flop 1 is more than the t_H of flip-flop 2. Practically flip-flops have hold times that are 5 nsec or less, most have $t_H = 0$. Therefore, flip-flop circuits such as the one shown connected in the diagram work reliably.

Clock Skew

One of the most common problems in synchronous circuits is 'Clock Skew'. One type of Clock Skew occurs when the same clock signal arrives at different times at different clock inputs to propagation delay, which causes different flip-flops to change states asynchronously leading to unpredictable outputs. Figure 26.2

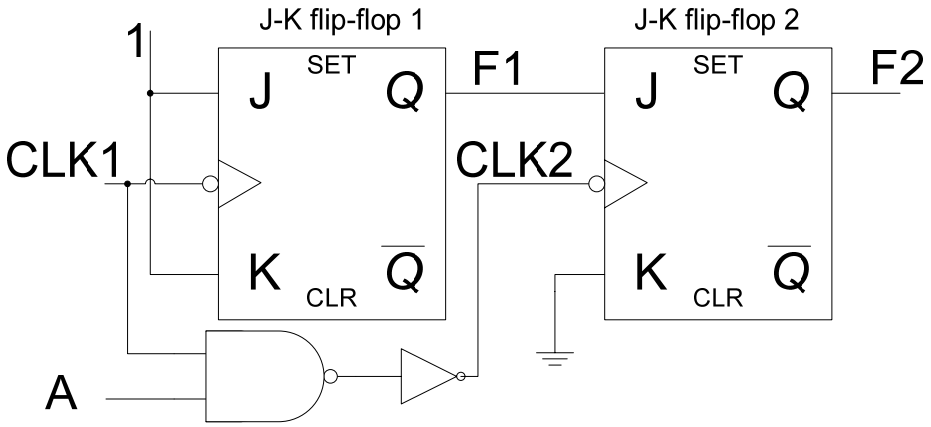


Figure 26.2a Flip-flop circuit with potential timing problem due to Clock Skew

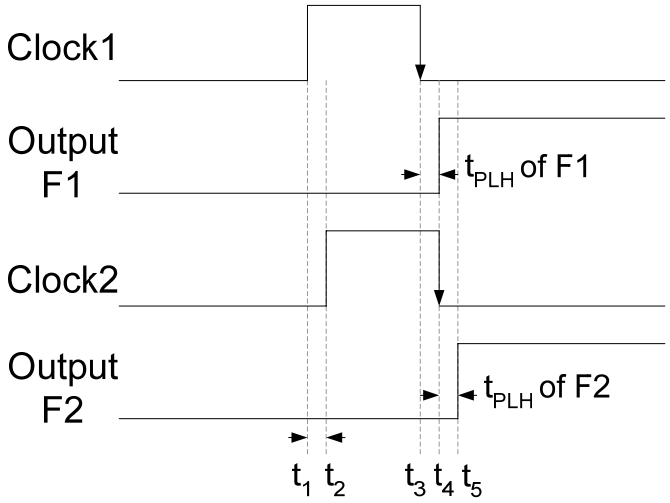


Figure 26.2b Timing diagram of J-K flip-flop circuit with Clock Skew

In the circuit diagram both the flip-flops are connected to the same clock signal. However, the clock signal to the second flip-flop is delayed by the NAND and NOT gates. On a high to low clock transition both the flip-flops change their output states assuming that the initial output state of each flip-flop is logic low. The Clock Skew is the delay in the two clock signals by a time interval t_1 t_2 or t_3 t_4. At the high to low transition of clock 1 signal the output of F1 toggles from logic low to logic high after a propagation delay of t_{PLH}. If the propagation

input of flip-flop is set to logic high and at the clock transition the output F2 is set to logic high. If the propagation delay t_{PLH} of F1 is of a longer duration than the Clock Skew, the J input of the flip-flop is at logic low at the high to low transition of clock 2 the output of F2 remains unchanged.

Timing problems occurring due to clock skew are intermittent in nature and therefore are difficult to detect. The clock skew can vary with changes in temperature, power supply voltages, length of connections and loading effects. Problems caused due to clock skew can be eliminated by equalizing clock delays to different parts of the circuit.

Race Conditions

Race conditions are said to occur when multiple internal variables change due to change in one input variable. Depending upon the sequence in which the internal variables change, the circuit output operates erratically. Figure 26.3. In the timing diagram shown, if the Q and $\overline{Q}$ output high to low transitions are slightly delayed, they coincide with the clock low to high transitions resulting in short duration pulses which are difficult to detect. The glitches due to race condition can be avoided by using a negative-edge triggered flip-flop instead of the positive-edge-triggered flip-flop used.

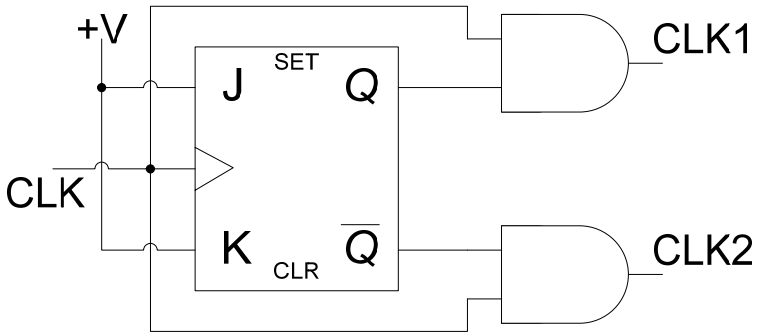


Figure 26.3a J-K flip-flop circuit with potential race condition

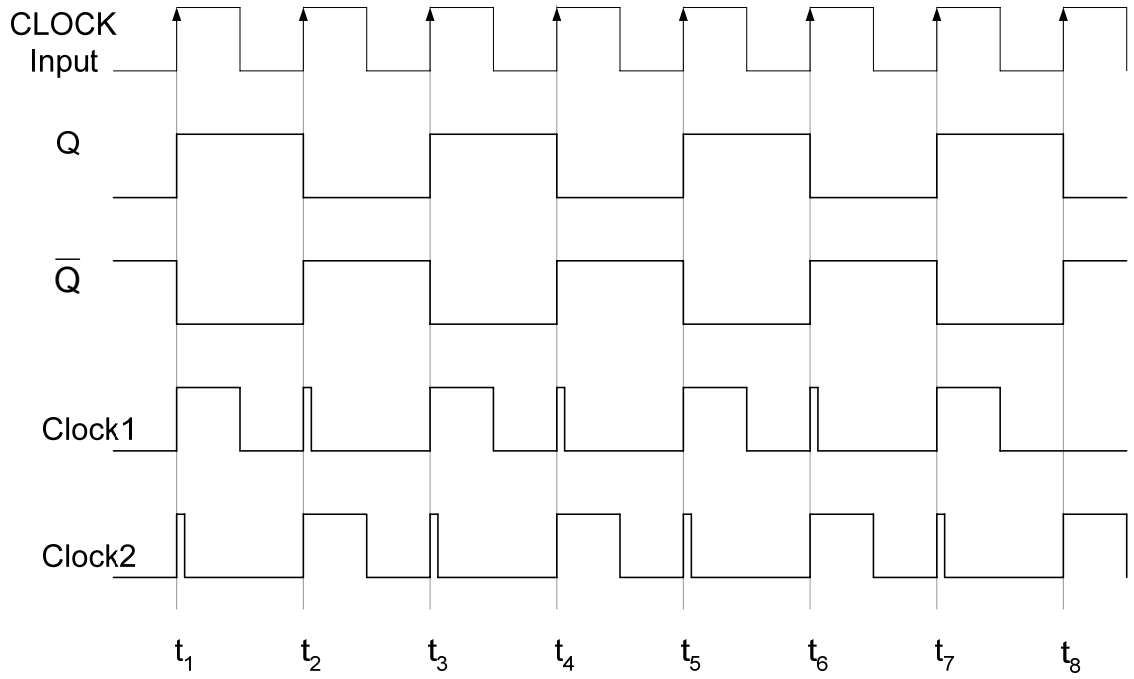


Figure 26.3b Timing Diagram showing glitches due to race conditions

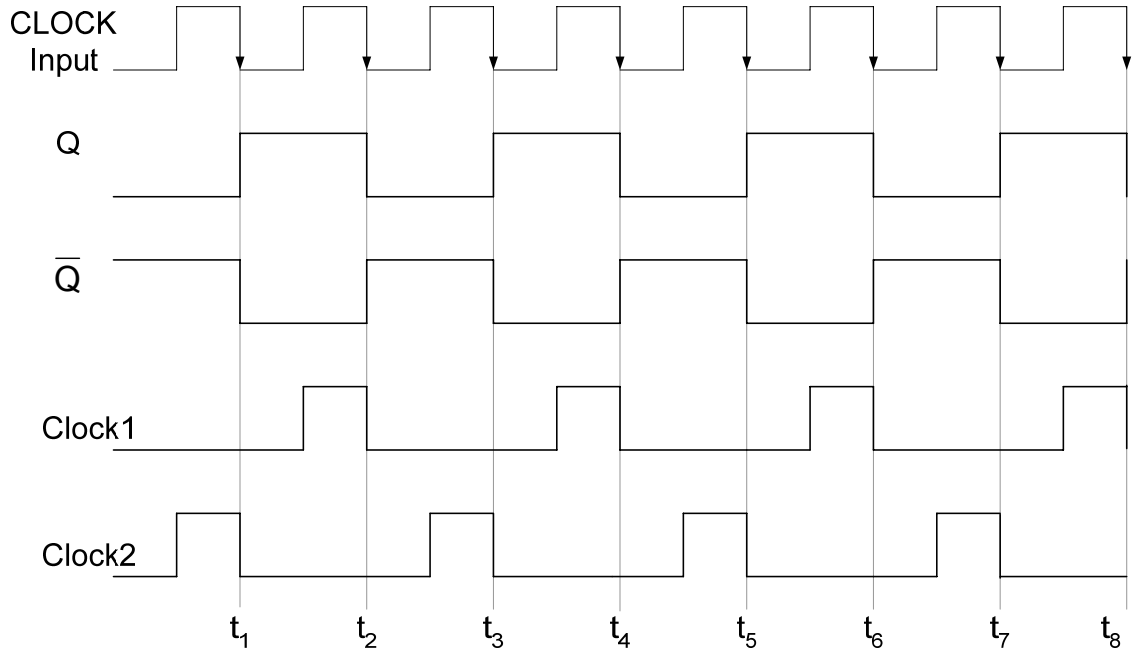


Figure 26.3c Timing Diagram of negative-edge triggered flip-flop avoiding glitches

Counters

Counter circuits based on flip-flops are widely used in Digital Systems. Besides

they are used as shift registers. Counters are classified as Asynchronous and Synchronous counters. Asynchronous counters as the name indicates are not triggered simultaneously. The multiple flip-flops that are connected together to form a counter circuit do not receive the triggering clock signal simultaneously. The flip-flop that represents the least significant count bit of the n-bit counter is connected to the clock signal, the remaining flip-flops receive their clock signals form the outputs of the preceding flip-flops connected in the counter circuit. The clock signal thus ripples through successive flip-flops. Synchronous counters on the other hand have all the clock inputs of the multiple flip-flops connected to a common clock signal. All the flip-flops in a Synchronous counter receive clock signals simultaneously.

Asynchronous and Synchronous are further classified as up counters or down counters depending upon the sequence in which they count. They are further classified in terms of the number of states or the range of numbers to which the counters can count.

Asynchronous Counters (Ripple Counters)

Asynchronous counters are implemented by connecting together multiple flip-flops together. The triggering clock signal is connected to the clock input of the first flip-flop. The clock inputs of the remaining flip-flops are connected to the Q or $\overline{Q}$ output of the previous flip-flop. On a clock transition at the clock input of the first flip-flop the output state of the flip-flop changes. With the transition in the output state of the first flip-flop, there is also a transition at the clock input to the second flip-flop as the output of the first flip-flop is connected to the clock input of the second flip-flop. Due to the clock transition the second flip-flop changes its output state. The change in the output state of the second flip-flop occurs after the first flip-flop changes its state. Similarly, the last flip-flop connected in the counter circuit changes its output state after the output of the flip-flop connected to its clock input has changed it state. The outputs of the flip-flops change in a sequence as the clock signal propagates through the flip-flops as they change their output states one after the other. The Asynchronous counters are also known as Ripple Counters due to the rippling effect of the clock signal.

A three-bit Asynchronous counter circuit is shown in Figure 26.4. In the circuit diagram shown the $\overline{Q}$ output of each is connected to the clock input of the next flip-flop. The J-K inputs of each of the three flip-flop are connected to logic high allowing the flip-flop to toggle their output state on a high to low transition at their clock input.

The output state of the first flip-flop toggles at every positive to negative clock transition in intervals t_1 to t_8. The output F_1 of the second flip-flop toggles at intervals t_2, t_4, t_6 and t_8 on every high to low transition of the output F_0. The output F_2 toggles its output state at intervals t_4 and t_8 on a high to low transition of the flip-flop output F_1.

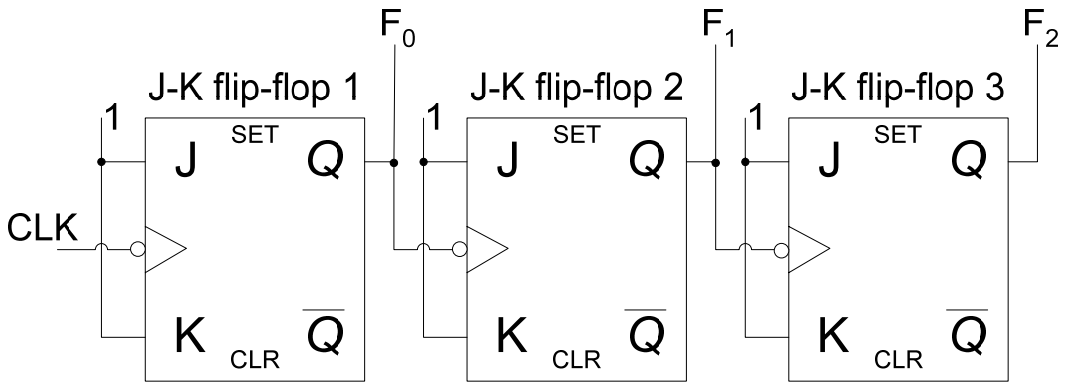


Figure 26.4a 3-bit Asynchronous Up-Counter

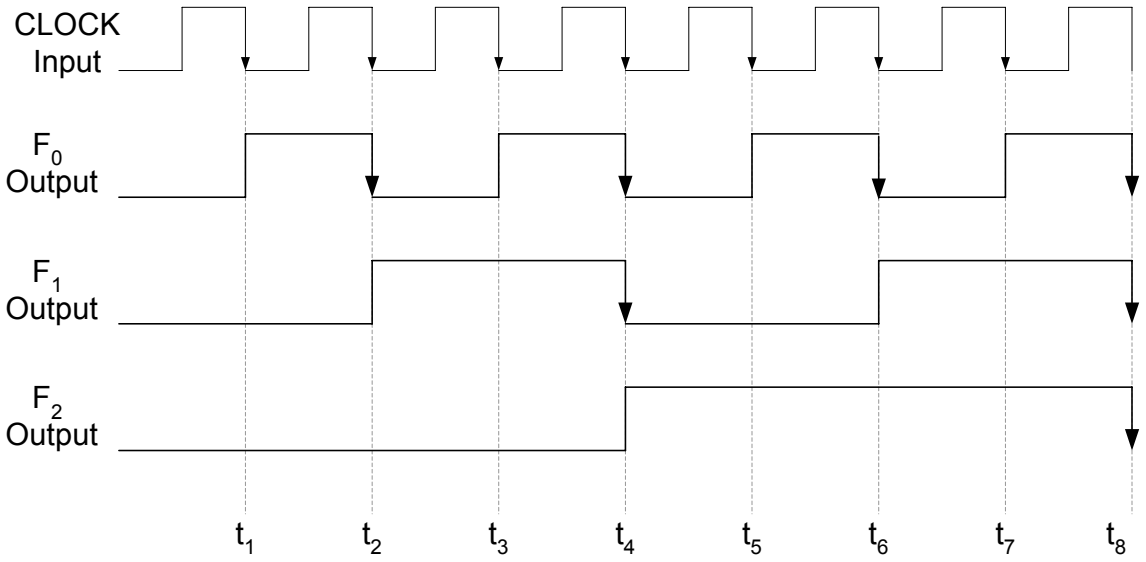


Figure 26.4b Timing Diagram of a 3-bit Asynchronous Up-Counter

Input	Output		
Clock Pulses	F_2	F_1	F_0
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1

Propagation Delay

The timing diagram shown in figure 26.4b doesn't take into account the propagation delay that occurs between each clock input and the corresponding toggling output. The timing diagram which takes into account the propagation delay is shown in figure 26.5. At time interval t_4 on a clock transition the output F_0 toggles to a new state after a delay determined by t_{PHL} propagation delay of the first flip-flop. At interval t_5 on the high to low transition of the F_0 output, the output F_1 toggles to a new state after a propagation delay t_{PHL} of the second flip-flop. Finally, at interval t_6 the transition in F_1 output toggles the output F_2 of the third flip-flop. The output F_2 becomes stable after a propagation delay t_{PLH} of the third flip-flop. The propagation delay of each of the three flip-flop adds up to delay the output F_2 by three propagation delays with respect to the clock transition at interval t_4. If the counter circuit is extended by adding more flip-flops, then the output of the last flip-flop might exceed the clock period of the clock which causes timing problems. The Asynchronous counters can not work at high clock frequencies and cause problems with decoding circuits.

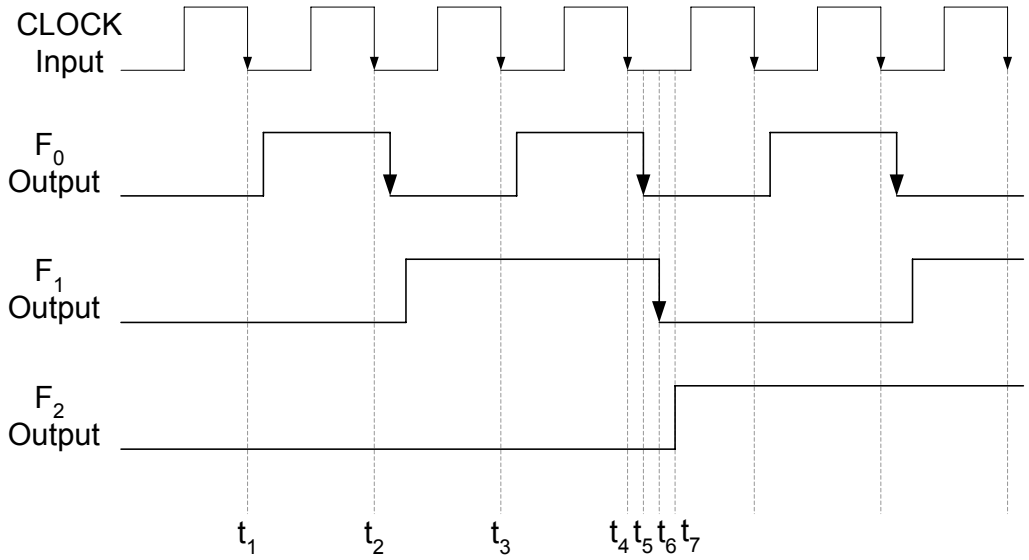


Figure 26.5 Timing Diagram of a 3-bit Asynchronous with propagation delay

The timing diagram of the 3-bit counter circuit using a clock of a higher frequency is shown in Figure 26.6a. At interval t_4, the negative clock transition toggles the F_0 output to logic low at interval t_A after a propagation delay of t_{PHL}. The negative transition of F_0 at t_A toggles the F_1 output to logic low at interval t_5 after a propagation delay of t_{PHL}. Finally, the F_2 output is toggled to logic high at interval t_B after a delay of t_{PLH} after the clock (F_1) transition at interval t_5. The output states of the counter at intervals t_1 to t_7 are shown in table 26.2. The output at interval t_5 should be 100 instead of 010.

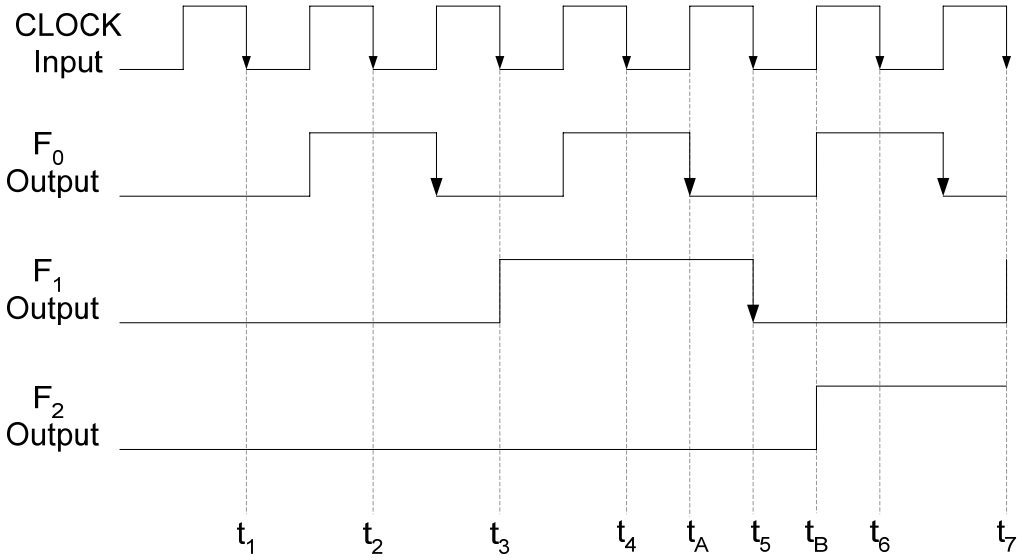


Figure 26.6a Timing Diagram of a 3-bit Asynchronous with high frequency clock

Input	Output		
Clock Pulses	F_2	F_1	F_0
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	0	1	0
6	1	0	1
7	1	1	0

Table 26.2 Output of a 3-bit Asynchronous Up-Counter with high frequency clock

Mod-n Counters

The term Mod represents the Modulus of the counter which is the total number of unique states through which the counter will sequence through. A 3-bit Asynchronous counter can count up from 0 to 7 or count down from 7 to 0. The 3-bit counter has 8 different states represented by the 8 outputs 0 to 7. The counter states or the range of numbers of a counter is determined by the formula 2^m. where m represents the number of flip-flops. Therefore, a Mod-8 counter implemented using three flip-flops 2^3 has 8 output states.

Counter can also be designed to have less number of states than 2^m. The resulting sequence is called a truncated sequence. The counter therefore counts up to the truncated sequence. Designing a truncated sequence counter is very simple. When the counter counts up to the intended sequence it is reset to the initial count value 0. The counter is reset to the initial count value by activating the Clear asynchronous inputs. The clears input is activated by the counter through a combinational circuit that activates its output when the appropriate count sequence is reached. The Mod-6 counter is shown in figure 26.7.

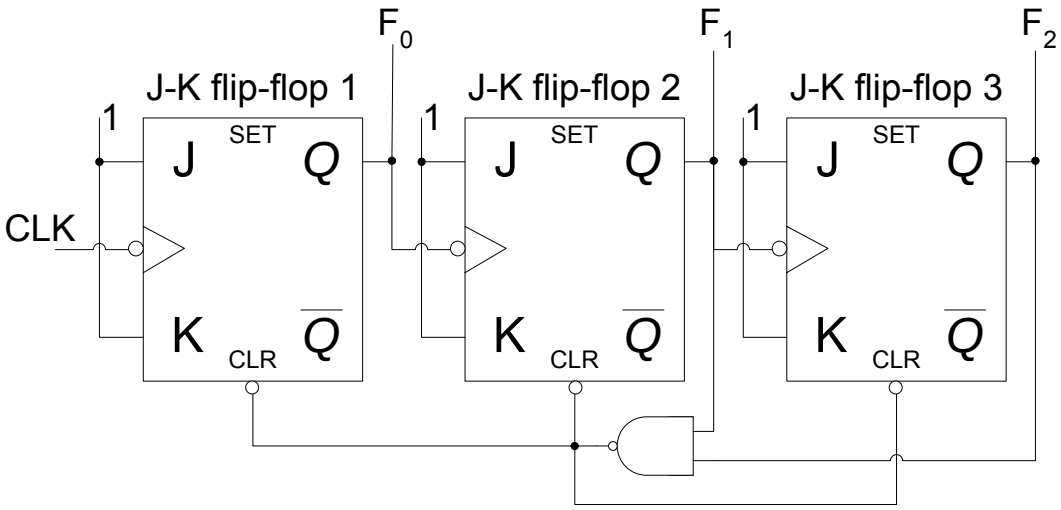


Figure 26.7a Mod-6 Counter

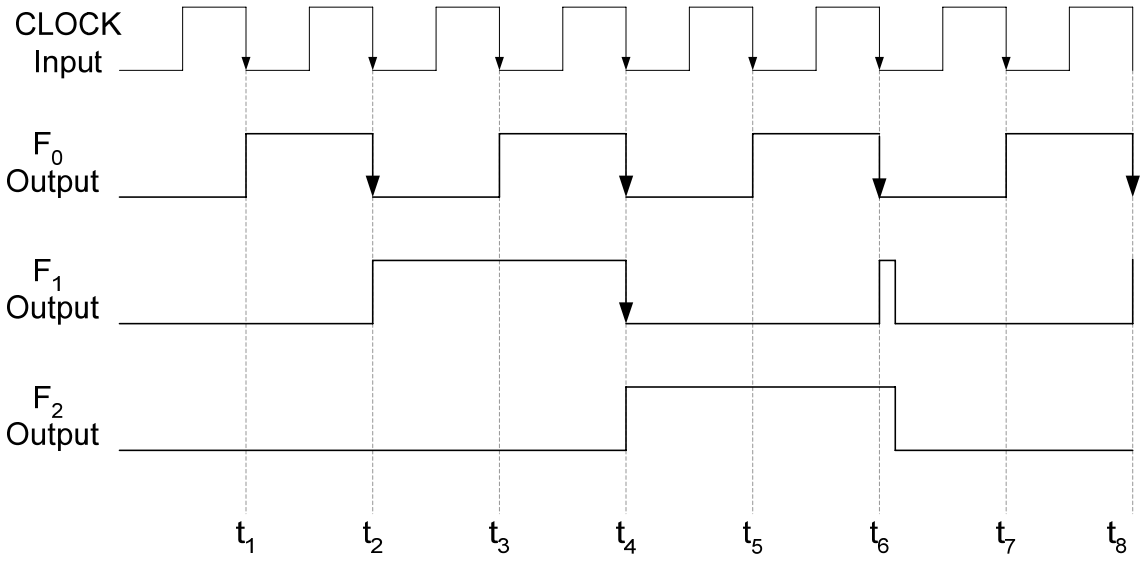


Figure 26.7b Timing diagram of a Mod-6 Counter

The counter counts from state 000 to 101. At interval t_6 the counter counts to 110. The outputs F_1 and F_2 of the counter are connected to the inputs of a 2-input NAND gate, which sets its output to logic zero when both its inputs become logic 1 at interval t_6. The output of the NAND gate is connected to the three active-low asynchronous Clear input of the three flip-flops which are set to low by the NAND gate. Therefore the counter is immediately reset to state 000 from where it proceeds to sequence through the count values. The Mod number of the counter also determines the frequency at the output of the counter. The output at F_2 has a frequency which is $1/6^{th}$ of the input clock frequency. Thus Mod-n counters can be design to generate $1/n^{th}$ frequency signal with respect to the input clock signal.

Mod-10 Counter (Decade Counter)

A decade counter uses four-flip-flops to implement the circuit which counts up to 10 unique states (0000 to 1001). The counter is reset when it counts to the next state 1010. The frequency of the output signal is 1/10th the input clock frequency. Figure 26.8.

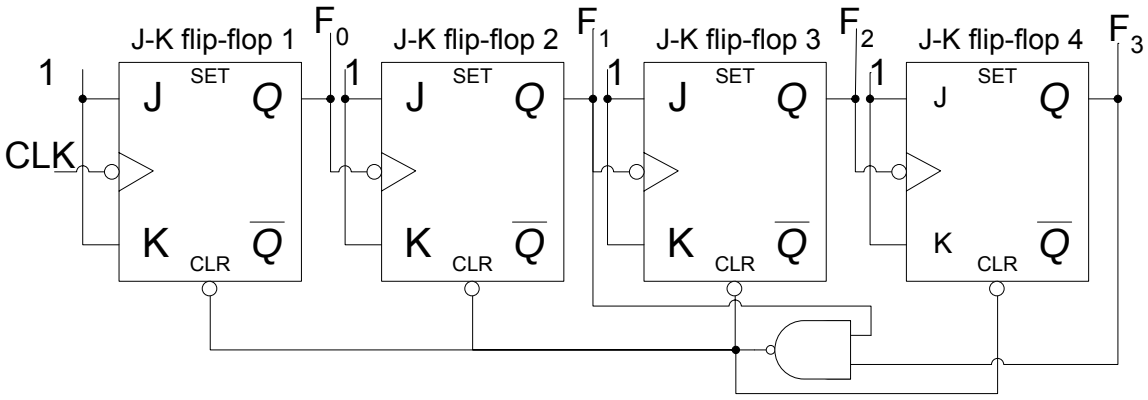


Figure 26.8a Asynchronous Decade Counter

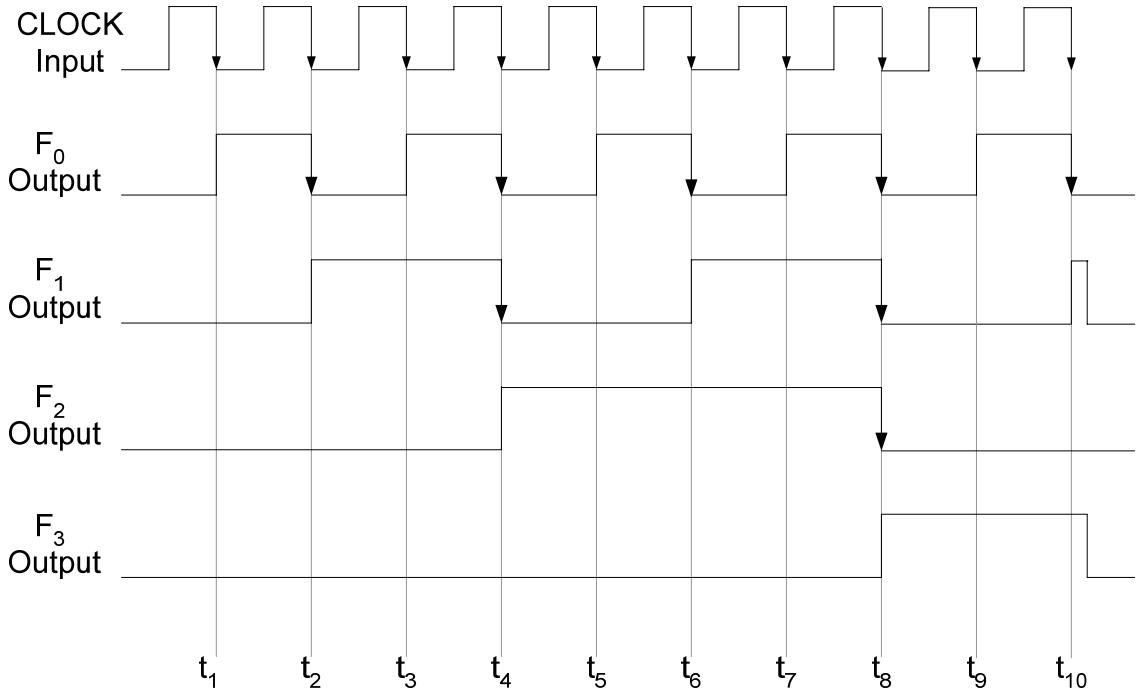


Figure 26.8b Timing diagram of a Decode Counter

The output F_1 and F_3 are connected through a NAND gate to the active-low clear inputs of all the four flip-flops. The counter counts from 0000 to 1001 (ten output states), when it counts to 1010, the output of the NAND gate is set to logic low which resets all the four flip-flops to state 0000.

Integrated Circuit Asynchronous Counters

Asynchronous Counters are available in Integrated Circuit form. The 74LS93A is a 4-bit Asynchronous Counter. The counter has two separate clock inputs CLK A and CLK B connected to the clock input of the first and second flip-flop respectively. The second, third and fourth flip-flops are internally connected as a ripple 3-bit counter. The counter also has two inputs pins connected to the inputs of a 2-input NAND (internal) gate, the output off which is connected to the clear inputs of all the four flip-flops. The counter provides four outputs, one form each flip-flop. Figure 26.9

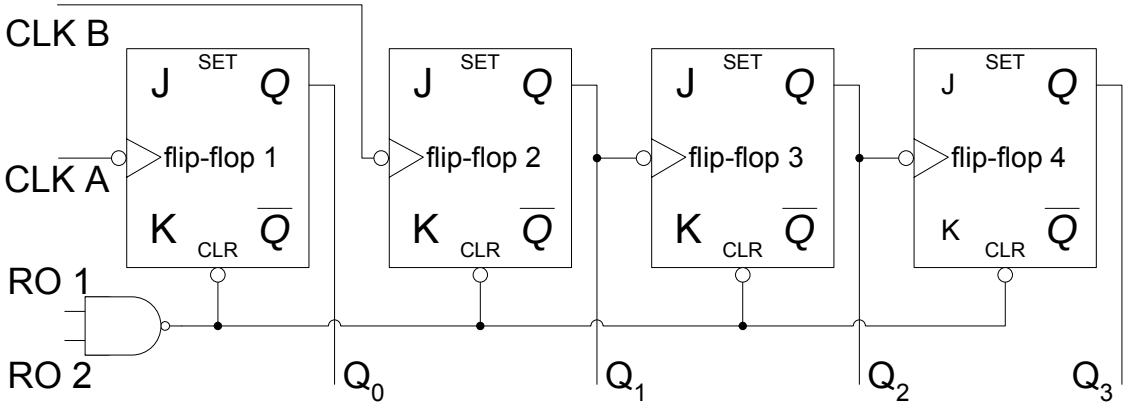


Figure 26.9 Internal circuit diagram of the 74LS93A Counter

The 74LS93A can be configured as MOD-16 counter by connecting CLK B input pin to the Q_0 output pin of the IC. RO 1 and RO 2 are connected to logic low. A Decade counter can be implemented by connecting CLK B input to the Q_0 and Q_1 and Q_3 outputs to RO 1 and RO 2 respectively. Figure 26.10 Two 74LS 93As ca be cascaded together to form a larger counter. A MOD-50 counter is implemented using two 74LS93A ICs. Figure 26.11

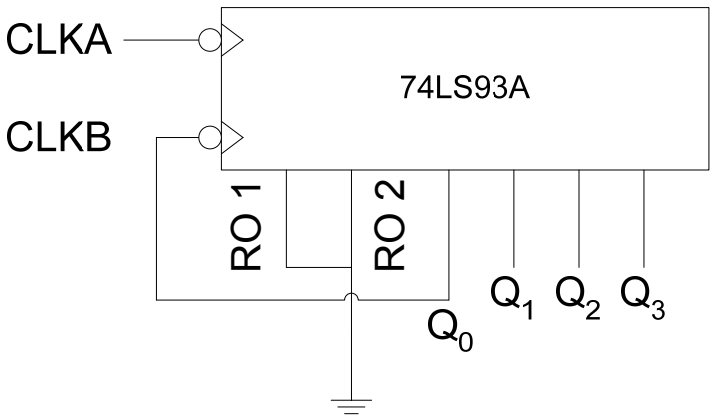


Figure 26.10a 74LS93A connected as MOD-16 Counter

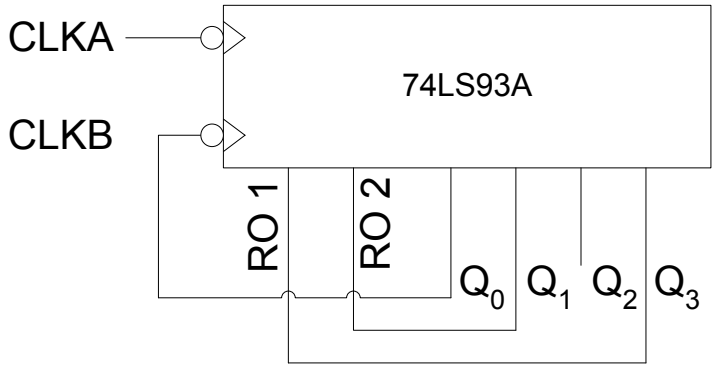


Figure 26.10b 74LS93A connected as Decade Counter

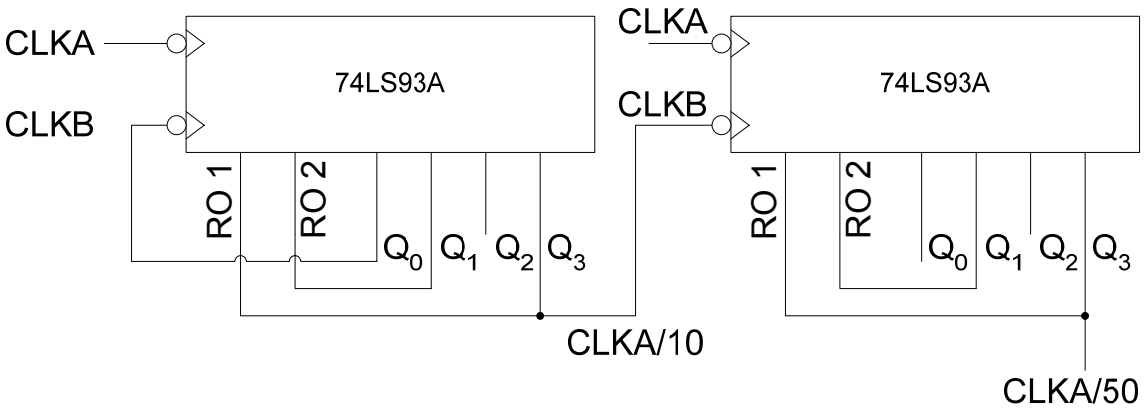


Figure 26.11 74LS93A Connected as a frequency divider (divide by 50)

In the circuit diagram two 74LS93As are connected together to form a frequency divider which divides the input frequency by 50. The first 74LS93A is connected to divide the input frequency by 10. The Q_3 output of the first 74LS93A is connected to the CLKB input of the second 74LS93A. The second 74LS93A is connected to divide the input frequency at CLKB by 5. The Q_3 output of the second 74LS93A therefore provides an output which is $1/50^{th}$ of the clock applied at the CLKA input of the first 74LS93A. The second 74LS93A requires the use of only three flip-flops, therefore the first flip-flop with clock input CLKA is left unconnected.

Lesson No. 27

DOWN COUNTERS

All the examples considered so far have used counters that count up from binary zero to some maximum count value depending upon the Modulus value of the counter. When the counter reaches its maximum count value it is reset to binary zero and continues with the counting sequence. A down counter counts in a sequence which starts with some maximum count value and counts down to binary zero. It is then reset to the maximum count value and repeats the counting sequence. A Down-counter is implemented by connecting the $\overline{Q}$ output instead of the Q output of all the flip-flops to the clock inputs of the next flip-flops. Figure 27.1

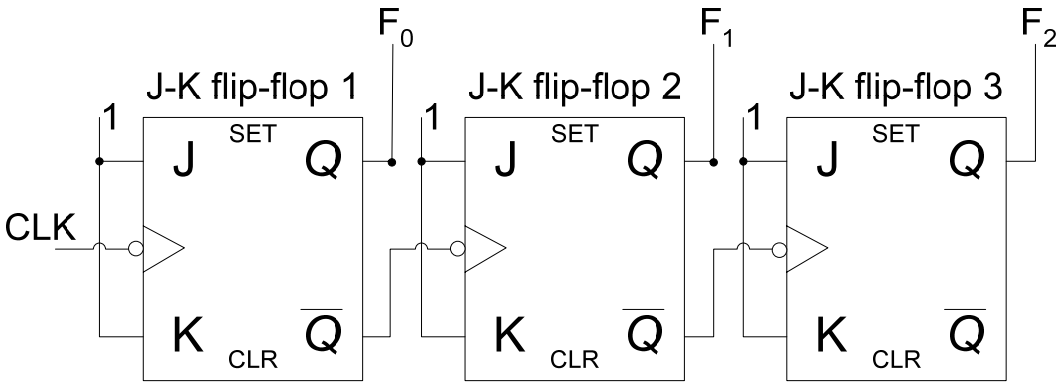


Figure 27.1a 3-bit Asynchronous Down-Counter

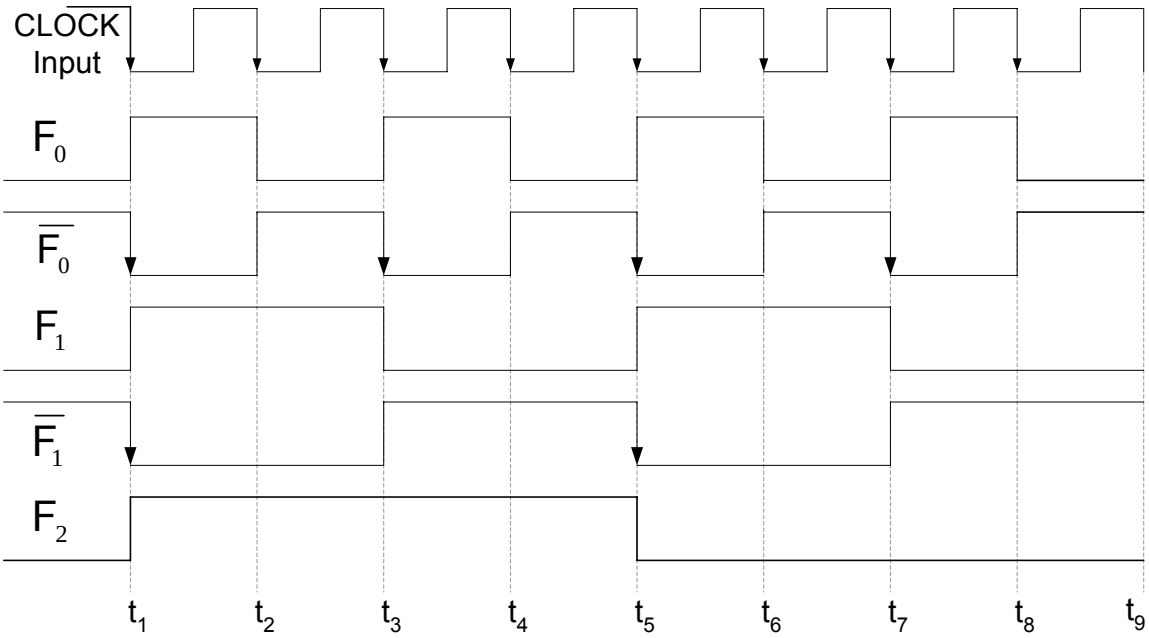


Figure 27.1b Timing diagram of a 3-bit Asynchronous Down-Counter

Down Counter with truncated sequence

A down counter can be configured to count down a truncated sequence, similar to an up-counter which can count up to any truncated sequence. A down counter counts down from the maximum count value to some predefined count value which is the last count value in the truncated sequence. On reaching the last count value the down-counter is preset to the maximum count value instead of clearing the counter to zero count value as done in the case of an up-counter. The circuit shows a 3-bit down-counter configured to count down a truncated sequence from 111 to 011. On reaching the count value 011, the counter is preset to 111 when it is decremented to 010 on the negative clock transition. Figure 27.2

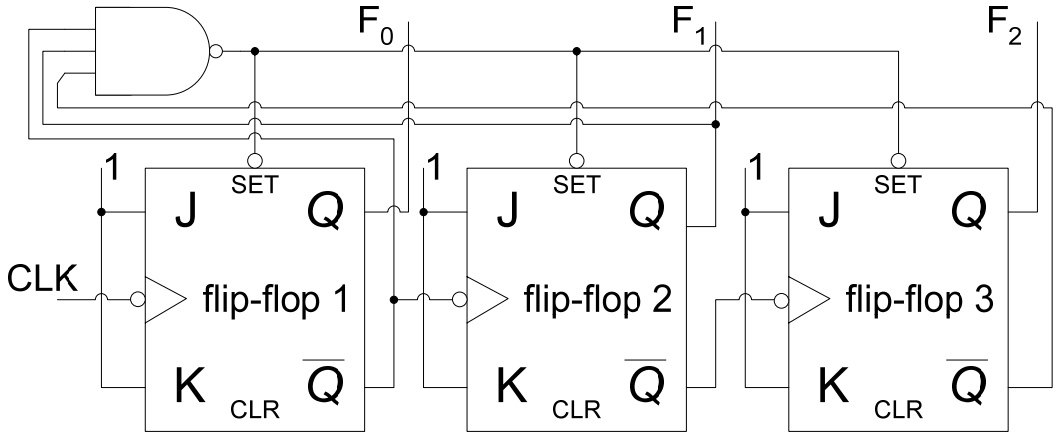


Figure 27.2a Down-counter configured to count a truncated sequence

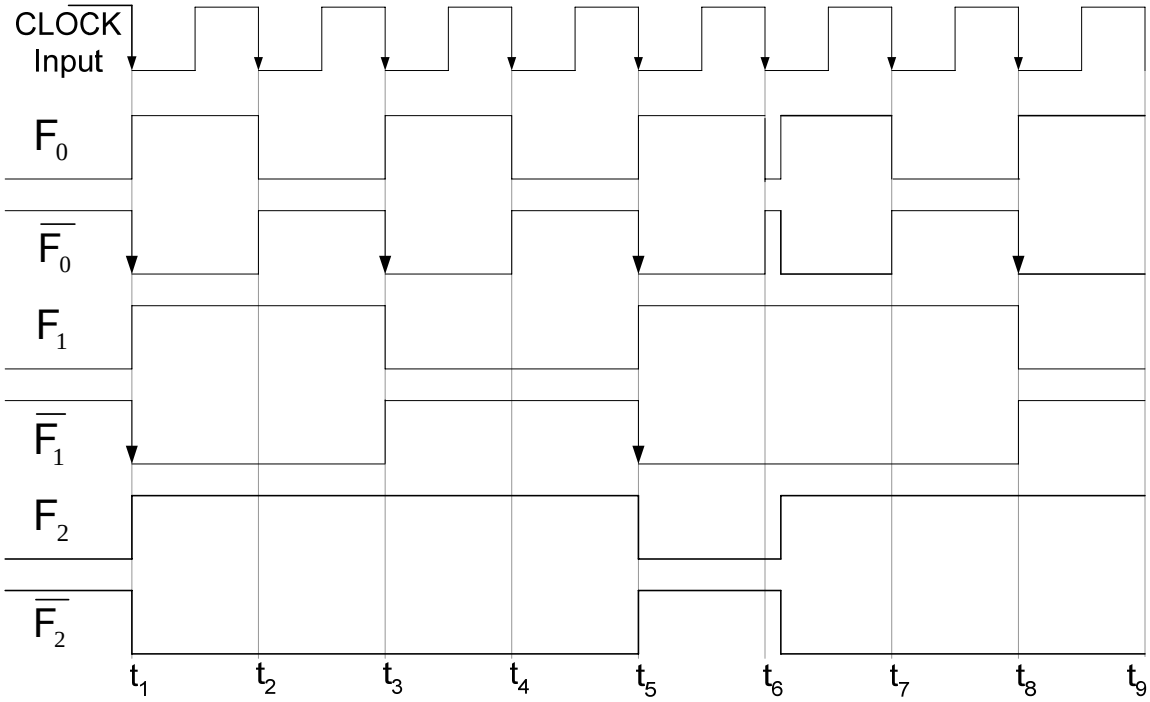


Figure 27.2b Timing diagram of a counter configured to count a truncated sequence

The counter counts down from 111 to 011 from interval t_1 to interval t_5. At interval t_6 the counter counts down to 010, the $\overline{F_0}$, F_1 and $\overline{F_2}$ are set to logic 1, the output of the NAND gate is set to logic 0 which presets all the three flip-flops to state 111. The counter continues with its counting sequence and at the clock transition at interval t_7 and t_8 the counter count down to 110 and 101 respectively.

Synchronous Counters

Asynchronous counters due to the delayed outputs caused by the rippling clock signal do not allow their operation with high frequency clock signals. Asynchronous counters having multiple bits also cause timing problems due to the excessive propagation delays.

Applications requiring 8, 16 and 32 counters and operating at high clock frequencies are implemented using Synchronous Counters. Synchronous counters use a common clock signal connected to the clock inputs of all the counter flip-flops. Therefore, on a clock transition all the flip-flops simultaneously change their output state. Figure 27.3.

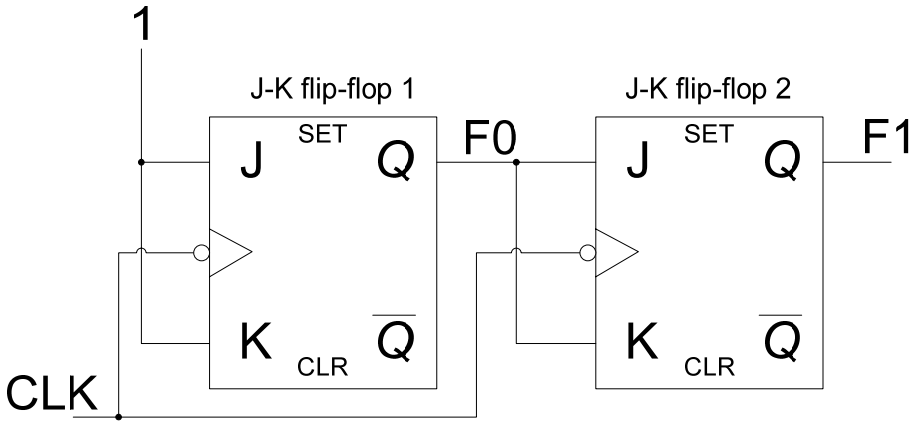


Figure 27.3a 2-bit Synchronous Counter

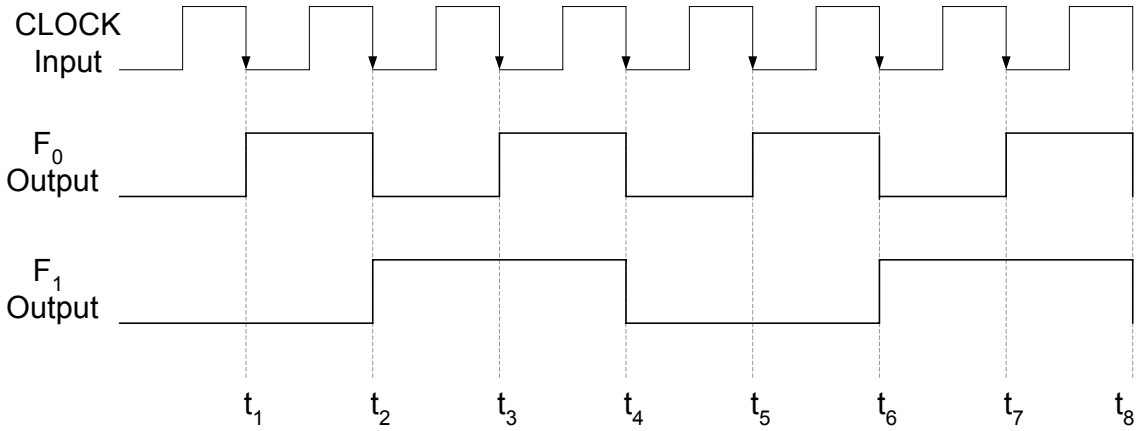


Figure 27.3b Timing diagram of a 2-bit Synchronous Counter

The 2-bit Synchronous counter has both its clock inputs connected to the clock signal. Both the flip-flops are reset to logic low states respectively. On a high to low clock transition at interval t . the F output of the first flip-flop toggles to logic high. Since the clock transition on

the clock input of the second flip-flop also occurs at interval t_1, the J-K inputs of the second flip-flop are at interval t_1 are at logic 0. The change at the inputs J-K to logic 1 of the second flip-flop occurs after a propagation delay t_{PLH} of the first flip-flop. Thus the output of the second flip-flop remains unchanged due to the input condition at the J-K inputs (J=0, K=0). At interval t_2 the output F_0 is at logic high (1) along with the J-K inputs as the three are connected together. On a clock transition at interval t_2 the output F_0 toggles to logic 0. At the very same instant the output F_1 also toggles to logic 1. The inputs J-K of the second flip-flop is set to logic 0 after a propagation delay of t_{PHL} of the first flip-flop. At interval t_3, at the clock transition the output F_0 toggles to logic 1. The inputs J-K of the second flip-flop at time interval t_3 is logic 0 therefore at the clock transition the output F_1 remains unchanged. The inputs J-K of the second flip-flop change after a propagation delay of t_{PLH}. Finally, at time interval t_4, the output F_0 of the first flip-flop toggles to logic 0. The J-K inputs of the second flip-flop are at logic 1, therefore the output F_1 of the second flip-flop is also set to logic 0.

3-bit & 4-bit Synchronous Counters

Multi-bit Synchronous Counters can easily be implemented by connecting together appropriate number of flip-flops together. The clock inputs of all the flip-flops are directly connected to a common clock signal. Implementing of Synchronous Counters larger than 2-bits requires the use of an AND gate. Figure 27.4

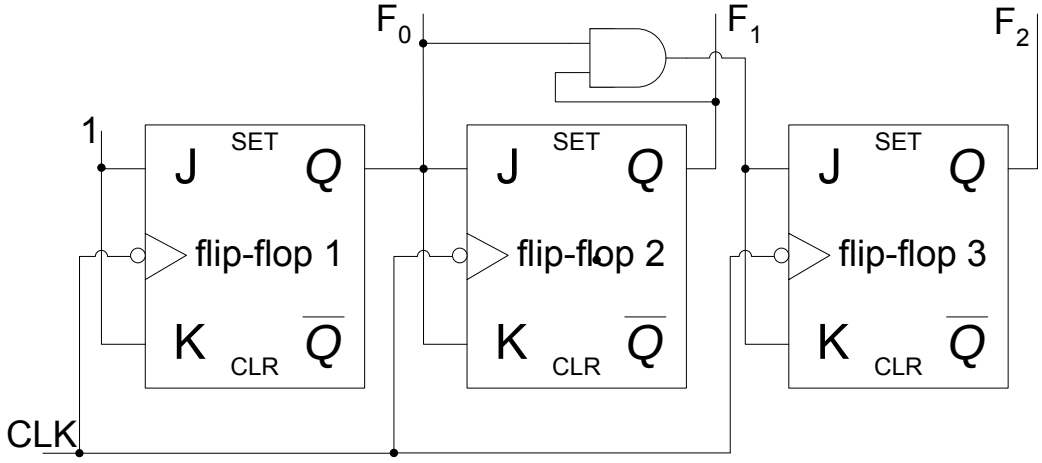


Figure 27.4a A 3-bit Synchronous Counter

The operation of the 3-bit Synchronous Counter and the need for the AND gate can be understood by studying the timing diagram of the 3-bit counter. The timing of the first two flip-flops is identical to the timings of the 2-bit counter discussed earlier. The timing diagram shows that at interval t_4, the 3-bit counter should count from state 011 to 100. Similarly, at interval t_8 the counter should count from state 111 to 000. At both the intervals the F_2 output of the third flip-flop toggles to logic 1 and logic 0 respectively when the outputs F_0 and F_1 are both at logic 1. This is implemented by connecting the two outputs F_0 and F_1 to the inputs of a 2-input AND gate. The output of the AND gate is logic 1 when both its inputs (F_0 and F_1) are at logic 1. The output of the AND gate is connect to the J-K inputs of the third flip-flop. If the AND gate is not used and the J-K inputs of the third flip-flop are directly connected to the output F_1 of the second flip-flop, the third flip-flop will change its state and set its output F_2 to logic 1 at the time interval t_3. The count sequence is thus disturbed.

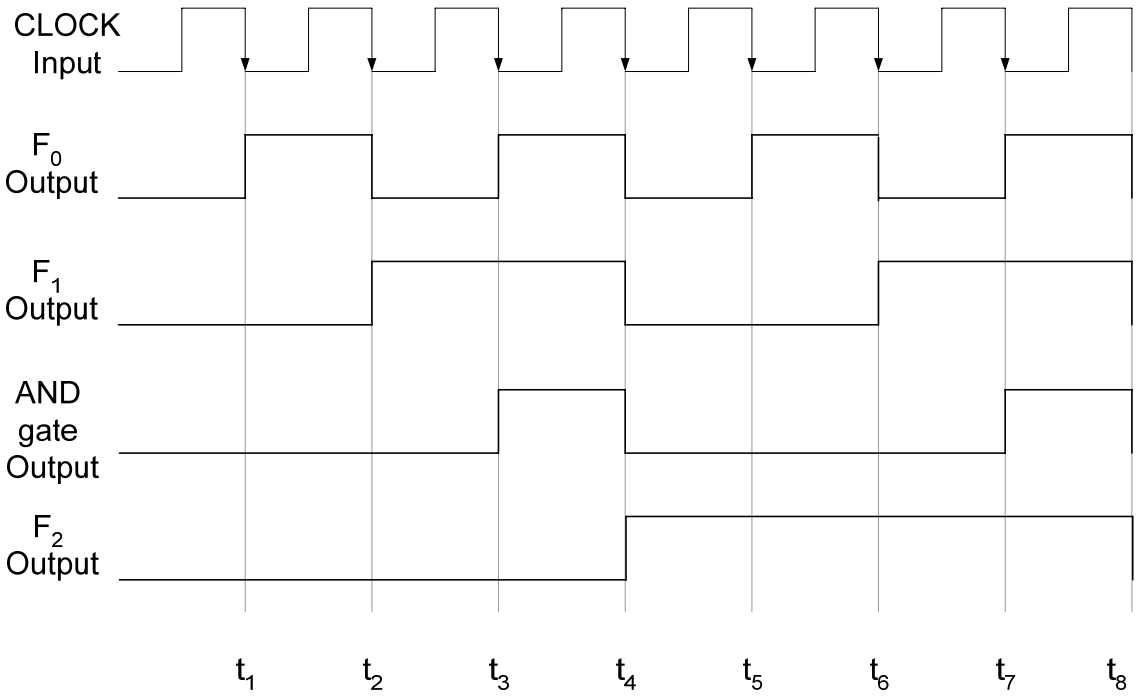


Figure 27.4b Timing diagram of a 3-bit Synchronous Counter

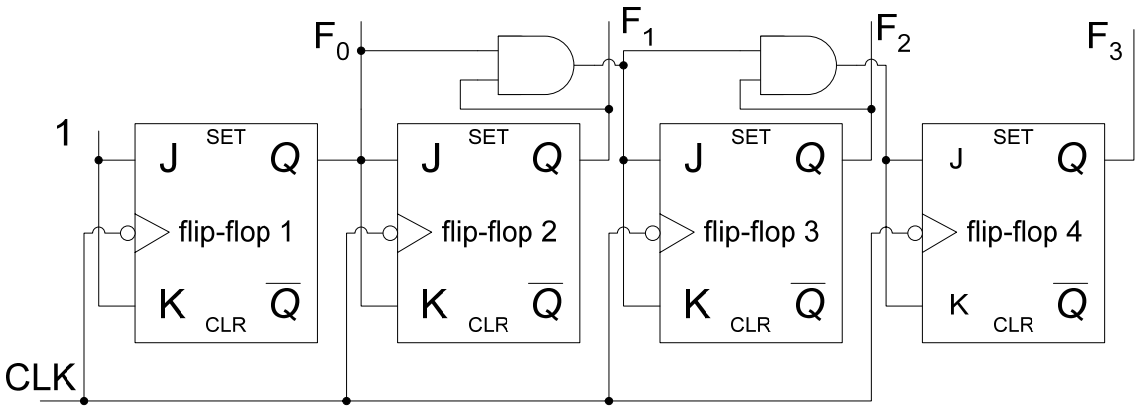


Figure 27.5 4-bit Synchronous Binary Counter

Larger counters can be implemented using similar AND gates. For example, a 4-bit counter uses four flip-flops. The counter circuit for the first three flip-flops is identical to the 3-bit counter circuit. The input of the fourth flip-flop is connected through a 3-input AND gate with inputs F_0, F_1 and F_2. The fourth flip-flop changes its state when the outputs of the first three flip-flops are at logic 1. That is, the when the 4-bit counter is counting from 0111 to 1000 and 1111 to 0000. Figure 27.5

4-bit Synchronous Decade Counter

Earlier, an Asynchronous Decade counter has been discussed, which counts from state 0000 to 1001. The Asynchronous counter is cleared to state 0000 when the counter counts from 1001 to 1010. Synchronous counter can be implemented which counts from 0000 to 1001. In the synchronous counter, all the four flip-flops are connected to a common clock

and are triggered simultaneously. However, instead of using the clear asynchronous inputs to clear the counter to the initial state, logic gates are used to reset the decade counter to state 0000 after it reaches state 1001. The implementation of the Synchronous Decade counter can be understood with the help of a function table that represents the operation of the Decade Counter. Table 27.1.

Input	Output			
Clock Pulses	F_3	F_2	F_1	F_0
1	0	0	0	0
2	0	0	0	1
3	0	0	1	0
4	0	0	1	1
5	0	1	0	0
6	0	1	0	1
7	0	1	1	0
8	0	1	1	1
9	1	0	0	0
10	1	0	0	1

Table 27.1 Output of a Synchronous Decade Counter

The output state of the first flip-flop F_0 is shown to toggle between 1 and 0 on each clock transition. Therefore, the inputs J-K of the first flip-flop are connected to logic high. The output state of the second flip-flop F_1 changes from logic 0 to logic 1 and vice- verse when F_0 output is logic 1 and F_3 output is logic 0. Therefore, the inputs J-K of the second flip-flop are connected to a function determined by the Boolean expression $F_0\overline{F_3}$. The output state of the third flip-flop F_2 changes from logic 0 to logic 1 and vice-versa when F_0 and F_1 outputs are both at logic 1. Therefore, the inputs J-K of the third flip-flop are connected to a function determined by the Boolean expression F_0F_1. The output of the fourth flip-flop F_3 changes its output state when outputs F_0, F_1 and F_2 are at logic 1 or when outputs F_0 and F_3 are at logic 1. Therefore, the J-K inputs of the fourth flip-flop are connected to a function determined by a Boolean expression $F_0F_1F_2 + F_0F_3$. The decade counter is shown in figure 27.6

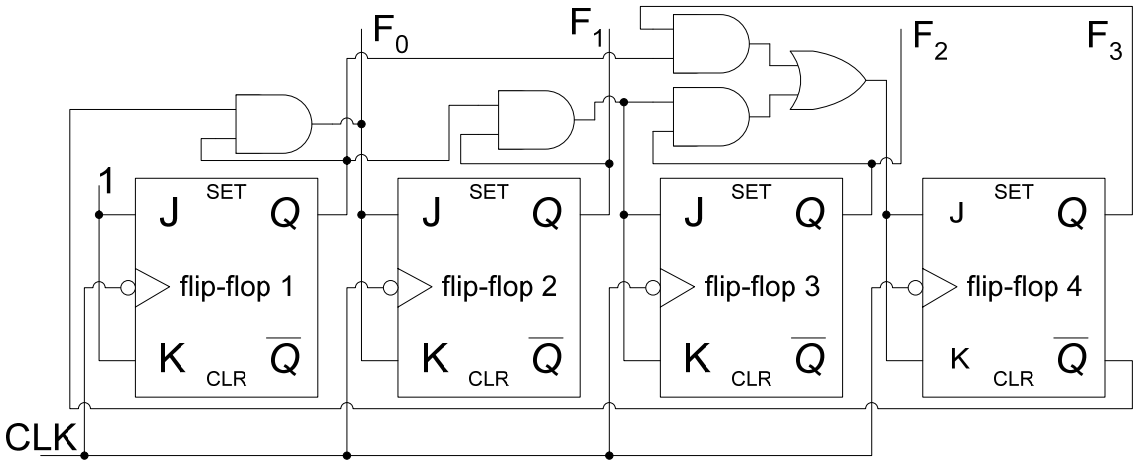


Figure 27.6 Synchronous Decade Counter

Lesson No. 28

TIMING DIAGRAM OF A SYNCHRONOUS DECADE COUNTER

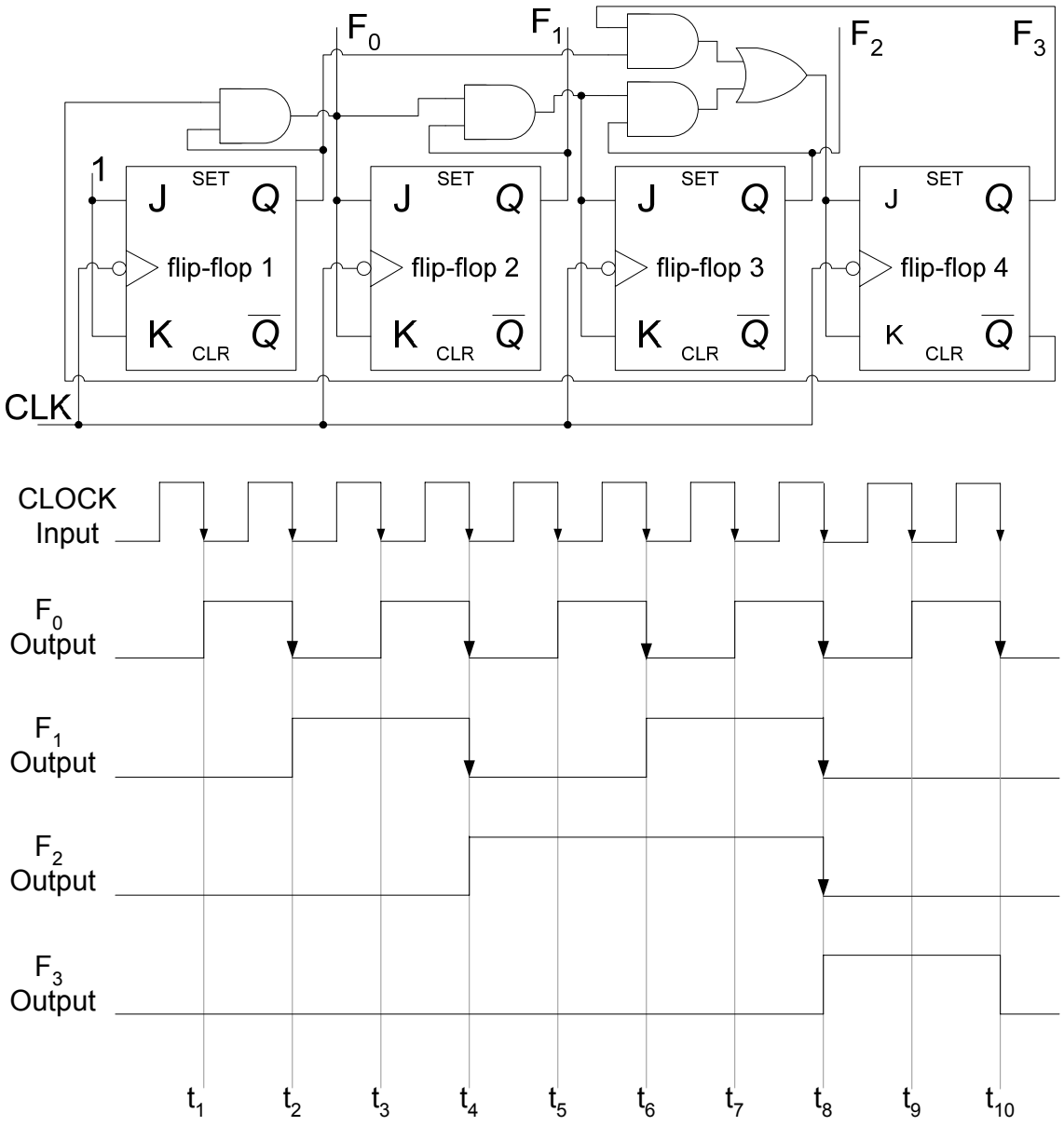


Figure 28.1 Timing diagram of a Synchronous Decade Counter

Mod-n Synchronous Counter

A Mod-n Synchronous can be implemented using appropriate number of J-K flip-flops connected together with their clocks triggered simultaneously. A synchronous counter which counts a truncated sequence of n unique states can be similarly implemented. The Modulus number represents the unique number of states which the counter counts in a sequence. The Modulus number determines the number of flip-flops required based on the relation $n = 2^m$ where m is the number of flip-flops.

Integrated Circuit Synchronous Counters

Instead of connecting a large number of flip-flops together to form large Synchronous counters, counter circuits available in Integrated Circuit form can be quickly connected to form large counters. The 74HC163 is a 4-bit Synchronous Counter. Figure 28.2. The counter has the following pins.

1. Parallel data inputs D_0, D_1, D_2 and D_3
2. Data outputs Q_0, Q_1, Q_2 and Q_3
3. Positive edge-triggered CLOCK signal
4. Active-low CLR input which resets the Counter output to 0000
5. Active-low LOAD input which loads the 4-bit data applied at the counter inputs
6. Active-high ENT and ENP enable inputs. For the counter to operate both the enable inputs have to be high
7. The Ripple Clock Output RCO goes high when the Counter reaches the terminal count 1111. The RCO output along with ENT and ENP enable input pins are used to cascade multiple counter ICs for implementing larger counters

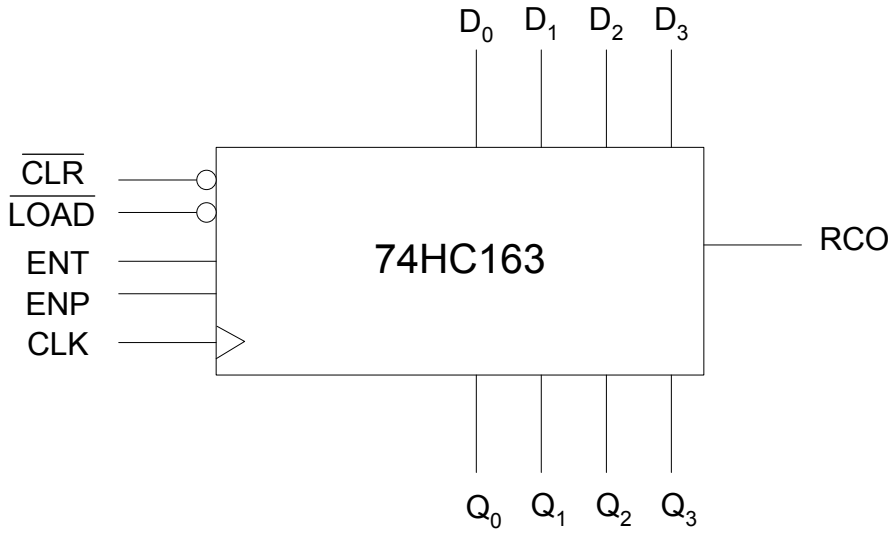


Figure 28.2a 74HC163 4-bit Synchronous Counter

Referring to the timing diagram, the $\overline{CLR}$ signal is activated between interval t_0 and t_1. The counter output is reset synchronously at interval t_1 as the $\overline{CLR}$ signal is active at interval t_1. If the $\overline{CLR}$ signal is deactivated before interval t_1 then the counter output is not reset. The $\overline{LOAD}$ signal is activated between interval t_1 and t_2. At the clock transition at t_2, the counter is loaded with the 4-bit data applied at the inputs D_0, D_1, D_2 and D_3. The ENP and ENT enable signals are activated before interval t_3 and the counter increments to the higher count at clock transition at intervals t_3 and t_4. When the counter reaches the count 15 at interval t_4, the RCO (Ripple Clock Output) is set to high indicating that terminal count has been reached. At intervals t_5, t_6, t_7 and t_8 the counter successively counts to 0, 1, 2 and 3. The counter enable signal ENP is deactivated after interval t_8, which inhibits the counter from counting any further.

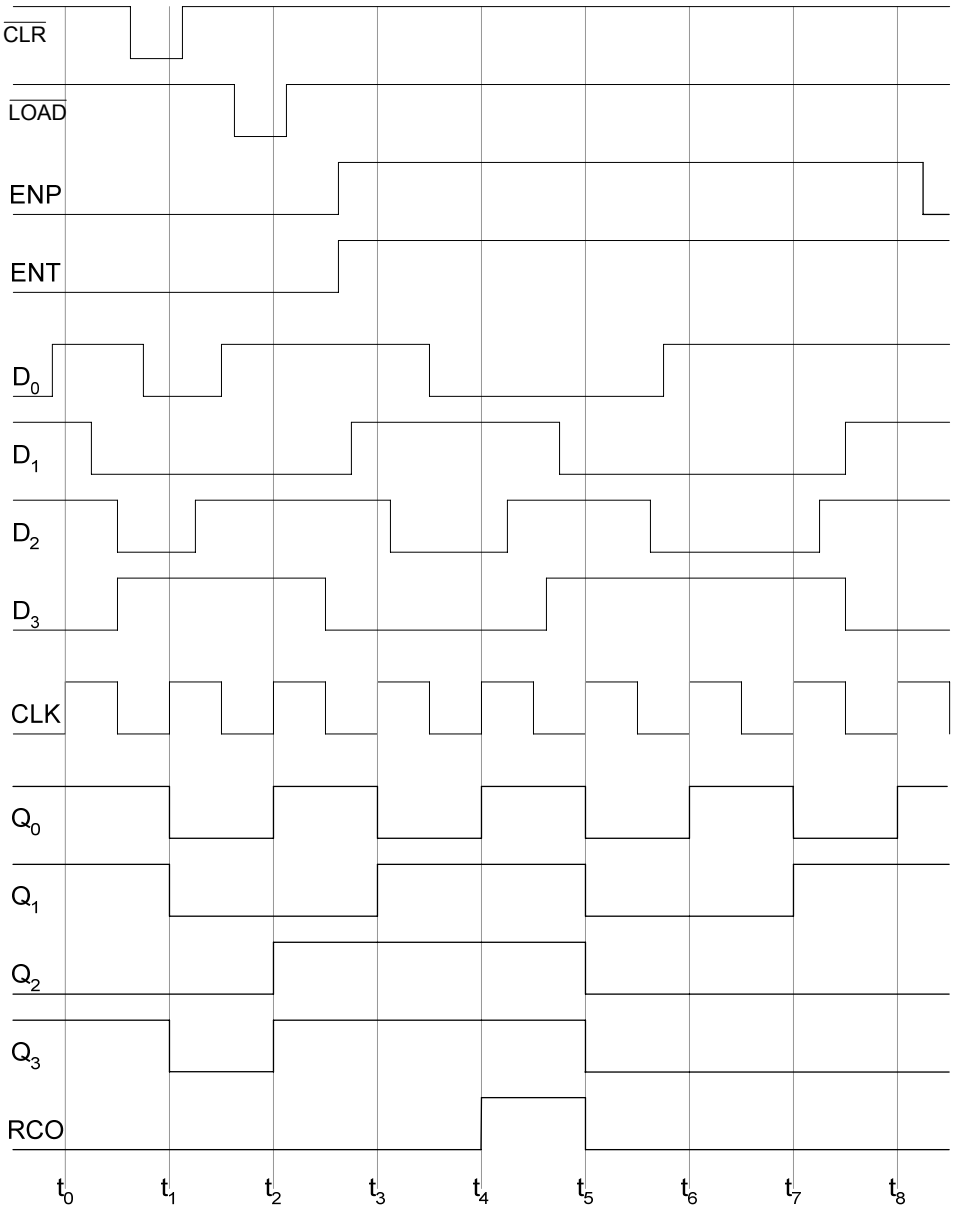


Figure 28.2b Timing diagram of the 74HC163 Synchronous counter

The 74HC160 is a 4-bit Synchronous Decade counter with the same input and output pins as the 74HC163. The RCO output of the decade counter is activated when the counter reaches its terminal count 1001.

Cascading Counters

It is very convenient to cascade Integrated Circuit counters together to form larger counters instead of connecting together flip-flops to implement a large counter. The enable inputs and Ripple Clock Outputs of the Integrated Circuit counters allow cascading of multiple counters together. Two, 74HC160 decade counters are shown connected together to divide the input frequency by 10 and 100. Figure 28.3. The 74HC163 can also be similarly cascaded together.

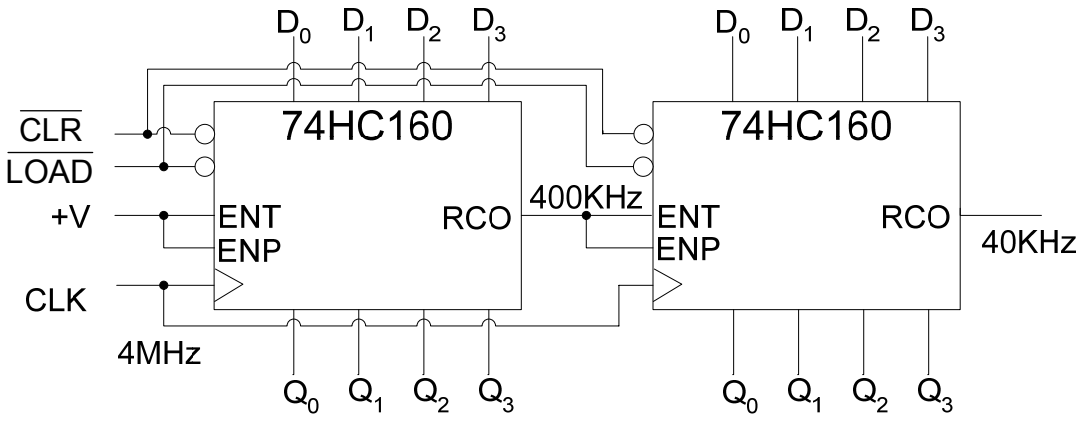


Figure 28.3a Cascaded Decade Counters

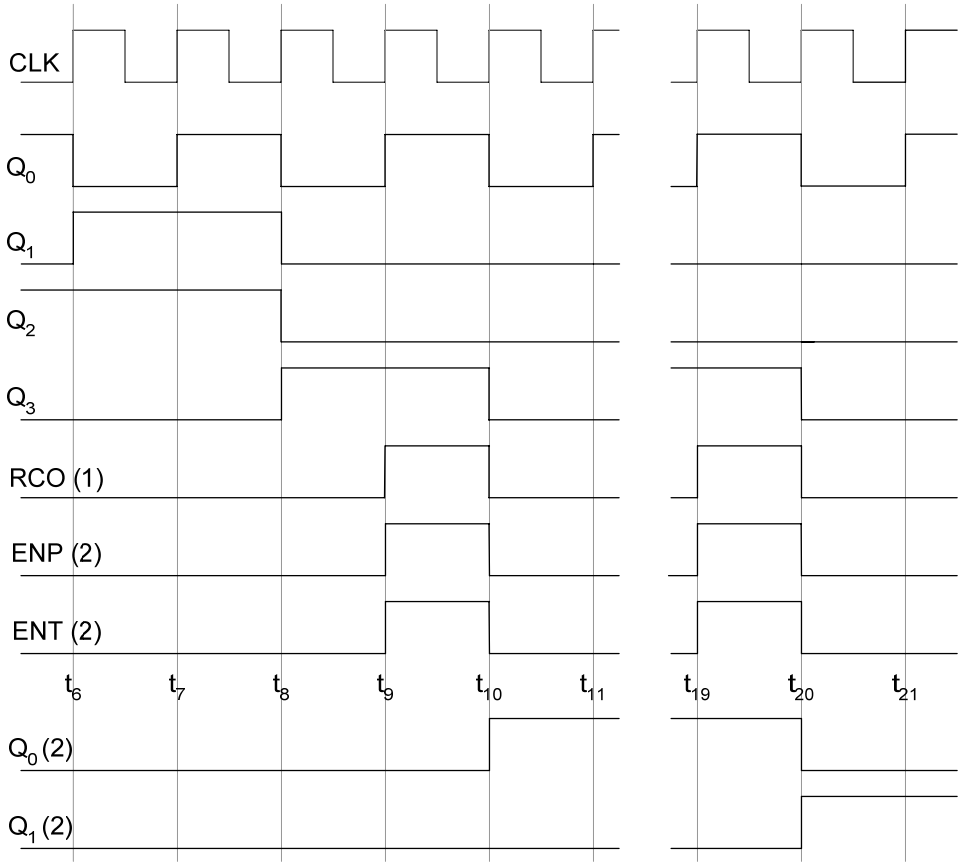


Figure 28.3b Timing diagram of a Cascaded Decade Counter

In the timing diagram, at interval t_9 the first decade counter reaches the terminal count 1001. The RCO output of the counter is set to logic 1. The RCO of the first counter is connected to the ENP and ENT enable pins of the second counter, therefore the counter is enabled. At interval t_{10} on a positive clock transition the first counter increments to count 0000. Since the second counter is also enabled, it is incremented to 0001. As soon as the first counter is incremented to 0000, the ECO signal is deactivated which in-turn also inhibits the second

first counter again reaches its terminal count 1001, the RCO output of counter once again becomes active thereby activating the second counter. At interval t_{20} on a positive clock transition the first and second counters increment to count 0000 and 0010 respectively. The RCO signal is again deactivated inhibiting the second counter from counting. This sequence continues after the first counter reaches its terminal count.

Integrated Circuit Counters with Truncated Sequences

Earlier, a decade counter was implemented by truncating the counting sequence of a MOD-16 counter. The Integrated Circuit Counters can also be configured as MOD-n counters where n represents the truncated sequence and is less than 16. Figure 28.4 shows the circuit diagram of the 74HC163 counter configured as Mod-7 counter. The counter is preset with the count value 1001 by setting the LOAD/NORMAL input to logic 1 at the NOR gate input. At the positive clock transition t_1, the count value is loaded. The counter increments and at interval t_7 it reaches the terminal count. The RCO output is set to logic 1 which sets the $\overline{\text{LOAD}}$ input to logic 0. At the positive clock transition at interval t_8 the preset value 1001 is reloaded and the counter continues its counting sequence.

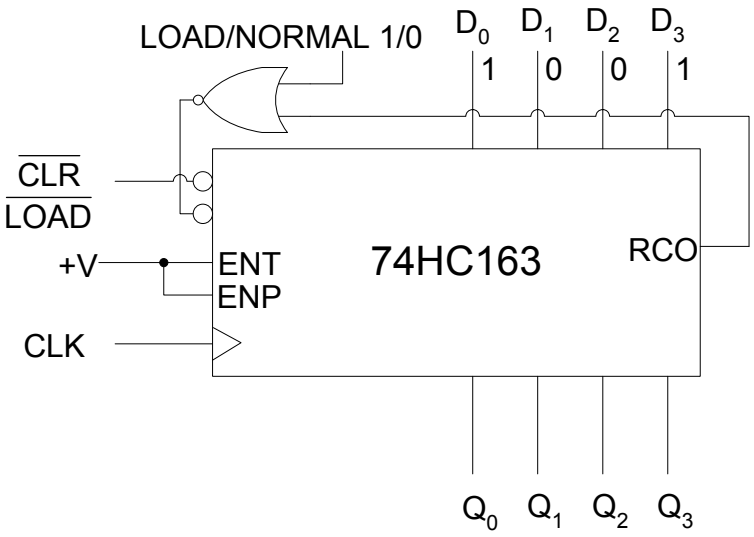


Figure 28.4a 74HC163 configured as Mod-7 counter

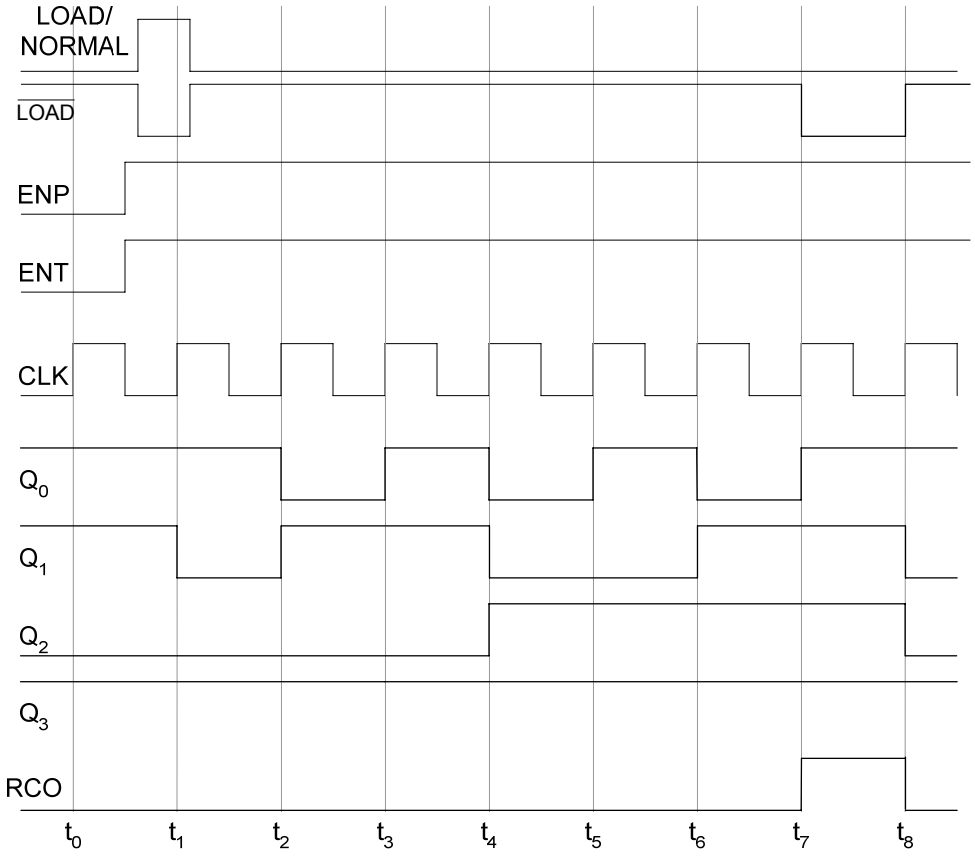


Figure 28.4b The timing diagram of a truncated Mod-7 Counter

Another method to configure an Integrated Circuit counter is to reset the counter when it reaches the maximum count value of its truncated sequence. This requires extra logic in the form of logic gates that determine the terminating state and reset the counter. The circuit diagram of the counter is shown. Figure 28.5

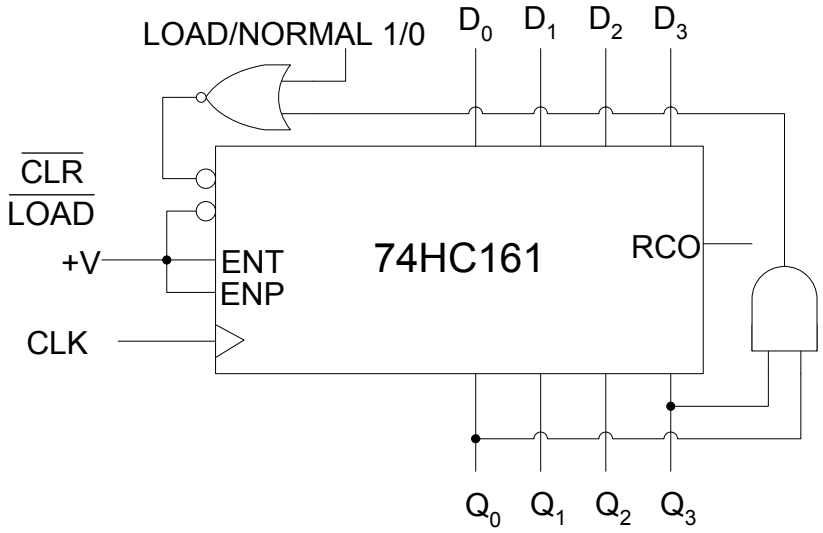


Figure 28.5a 74HC161 configured as Mod-9 counter

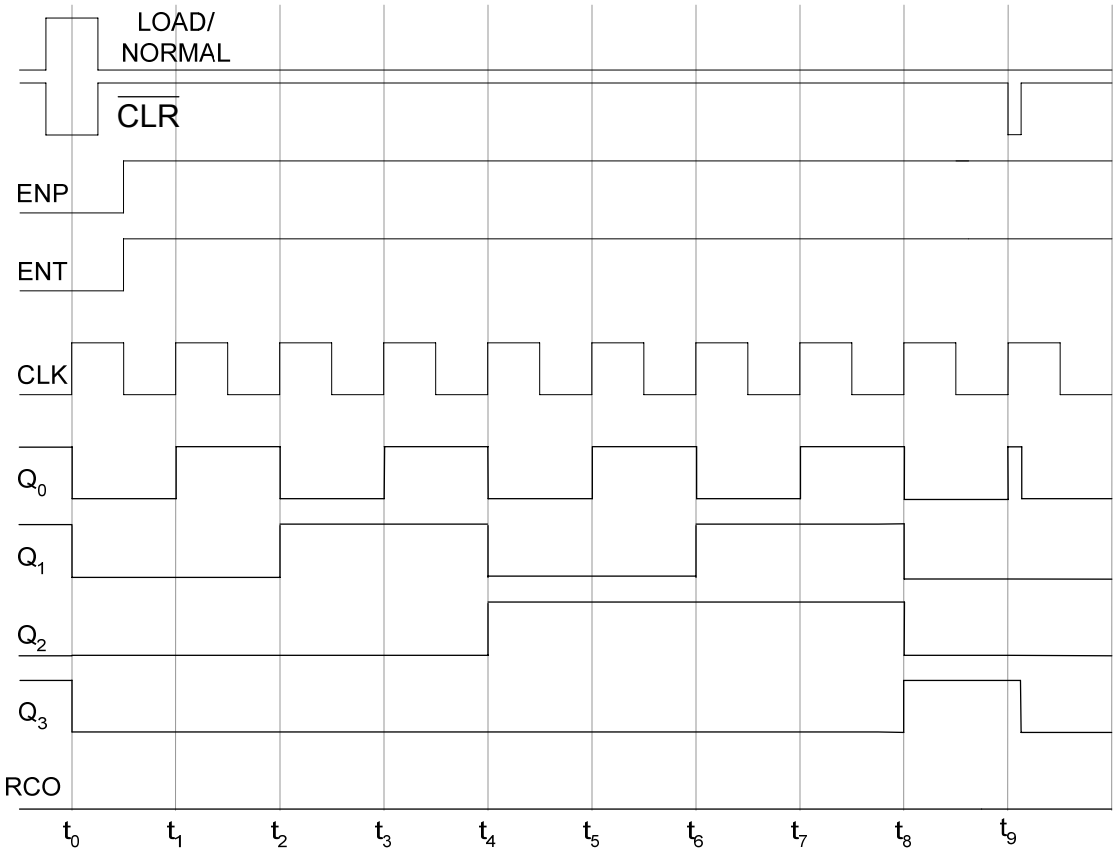


Figure 28.5b Timing diagram of a 74HC161 configured as Mod-9 counter

The counter used is 74HC161 instead of 74HC163. The 74HC161 has an Asynchronous Clear input, where as the 74HC163 counter has a synchronous Clear input. At time interval t_9 the counter increments to 1001 which sets the output of the AND gate to logic 1. The NOR gate output is set to logic low which activates the clear input and resets the counter to 0000. The 74HC163 counter which has a synchronous clear input, will reset counter resets at interval t_{10} when there is a transition at the clock input. It is clear from the timing diagram that to implement a Mod-9 counter the 74HC161 instead of 74HC163 counter has to be used.

Cascaded Counters with Truncated Sequences

Cascaded counters can also be configured to count in a truncated sequence. The circuit diagram of three cascaded 74HC163 is shown. Figure 28.6. The 12-bit cascaded counter is loaded with initial count value 1000 0000 0000. When the counter counts to 1111 1111 1111, the RCO output set to logic 1 by the third counter reloads the initial count values 0000, 0000 and 1000 in all the three counters respectively. The 12-bit counter can be configured for maximum count sequence as Modulus 4096 counter. The counter has been configured to count from 1000 0000 0000 to 1111 1111 1111 that is 2048 states or Modulus 2048 counter.

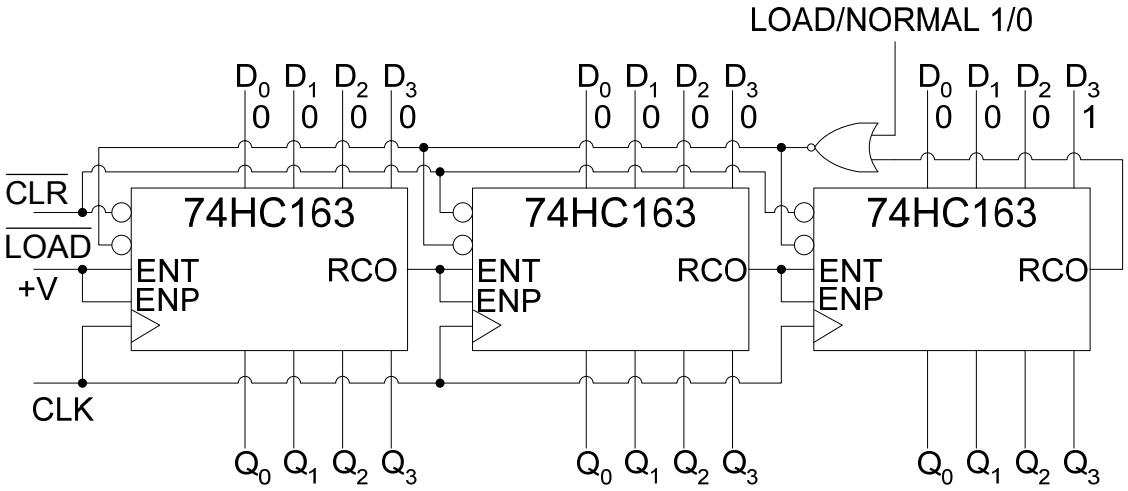


Figure 28.6 74HC163 counters connected for cascaded truncated count sequence

Up-Down Counter

An up-down counter can increment its output count value at each clock transition or decrement its count value at each clock transition, depending upon the count mode it is configured in. The counter can be reconfigured to count in the opposite direction during its count sequence. The circuit of an up-down 3-bit counter can be developed by studying the up-down count sequence of the counter. Table 28.1.

Clock Pulse	Q_2	Q_1	Q_0
0	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

Table 28.1a Up-counting sequence of a 3-bit Synchronous Counter

Clock Pulse	Q_2	Q_1	Q_0
0	1	1	1
1	1	1	0
2	1	0	1
3	1	0	0
4	0	1	1
5	0	1	0
6	0	0	1
7	0	0	0

Table 28.1b Down-counting sequence of a 3-bit Synchronous Counter

A 3-bit Synchronous up-counter has been discussed earlier. Consider the implementation of down-counter, the up and down counter can be combined to form a single configurable up-down counter. For the down-counting sequence the output Q of the first flip-flop toggles

between 0 and 1, therefore the J-K inputs are connected to logic 1. The output Q_1 of the second flip-flop toggles between logic 0 and 1 when the Q_0 output is logic 0 or $\overline{Q_0}$ is logic 1. The output Q_2 of the third flip-flop toggles when Q_0 and Q_1 outputs are both logic 0 or $\overline{Q_0}$ and $\overline{Q_1}$ are both logic 1.

Lesson No. 29

UP/DOWN COUNTER

The down-counter is implemented by connecting the $\overline{Q_0}$ and $\overline{Q_1}$ outputs. Figure 29.1

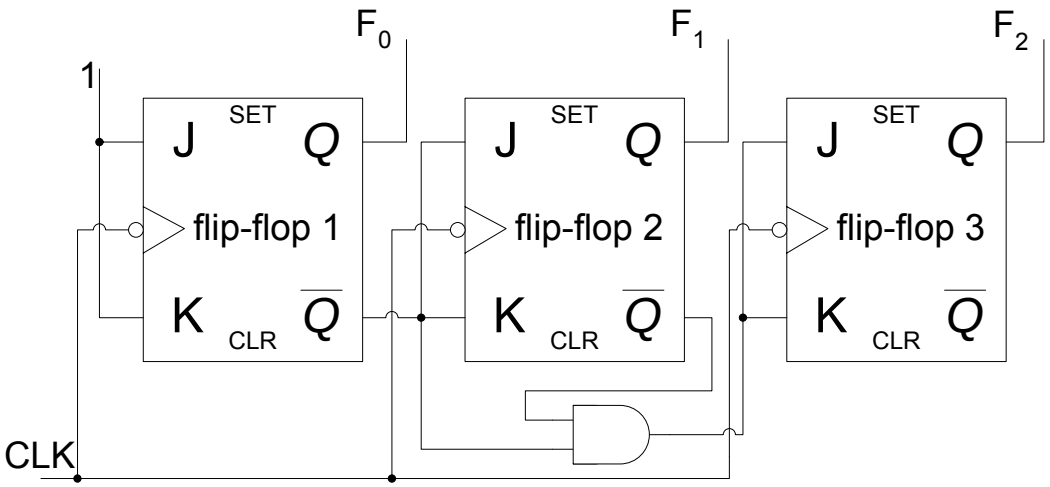


Figure 29.1 3-bit Synchronous Down-counter

The down-counter counter circuit is very similar to the up-counter circuit discussed earlier. The only change is the connection of the AND gate to the complementary outputs of the first and second flip-flops.

The up-counter and down-counter can be implemented as a single counter circuit by adding some extra logic. In the circuit diagram, the Up-down counter is configured to count up or down by setting the $UP/\overline{DOWN}$ input to logic 1 or 0 respectively. When the $UP/\overline{DOWN}$ input is set to logic 1, upper AND gates are enabled, allowing flip-flip 2 to toggle its state when F_0 output of flip-flop 1 is logic 1. Similarly when both F_0 and F_1 outputs are logic 1, flip-flop 3 toggles its state. When the $UP/\overline{DOWN}$ input is set to logic 0, the lower AND gates are enabled. When F_0 is logic 0, $\overline{Q_0}$ is logic 1 and the flip-flop 2 toggles its output state. Similarly, when both outputs F_0 and F_1 are at logic 0, that is, $\overline{Q_0}$ and $\overline{Q_1}$ are at logic 1 the flip-flop 3 toggles its state. During the counting sequence, the $UP/\overline{DOWN}$ input can be set to logic 1 or 0 at any time to reverse the counting sequence. Figure 29.2

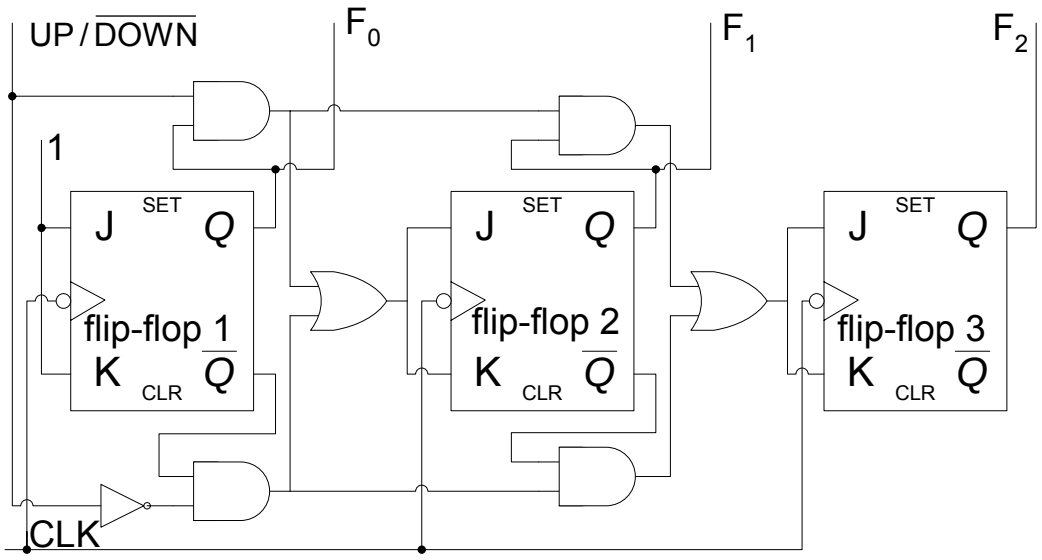


Figure 29.2a Up-Down Synchronous Counter

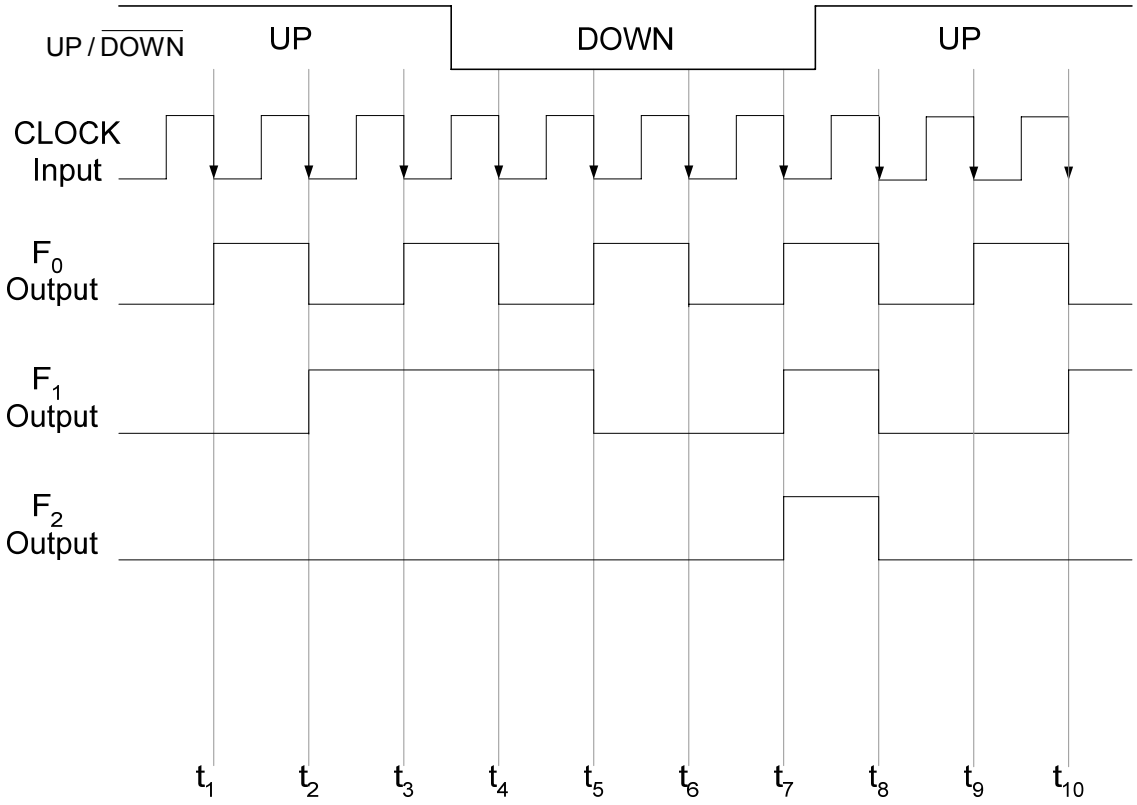


Figure 29.2b Timing diagram of an Up-Down Synchronous Counter

Integrated Circuit Up/Down Decade Counter

Implementing a 4-bit Up/Down counter by connecting flip-flops and logic gates increases the circuit size and requires many connections. The 74HC190 is a 4-bit Up/Down Synchronous Counter available in an Integrated Circuit form. Figure 29.3. The counter has the following pins.

1. Parallel data inputs D_0, D_1, D_2 and D_3
2. Data outputs Q_0, Q_1, Q_2 and Q_3
3. Positive edge-triggered CLOCK signal
4. Active-low LOAD input which loads the 4-bit data applied at the counter inputs
5. Active-low CTEN counter enable input
6. $D/\overline{U}$ the count down/up input. When the input is set to logic 1, the counter counts down and when the input is set to logic 0, the counter counts up
7. The MAX/MIN output that is set to high when the terminal count 1001 is reached when counting up or when the terminal count 0000 is reached when counting down. The MAX/MIN output is logic high for one complete cycle when a terminal count is reached.
8. The Ripple Clock Output RCO goes low when the Counter reaches the terminal count 1001 or 0000 when counting up or down respectively. The RCO output remains low during the negative half of the clock cycle. The RCO, the MAX/MIN output along with CTEN input is used to cascade multiple counter ICs for implementing larger counters.

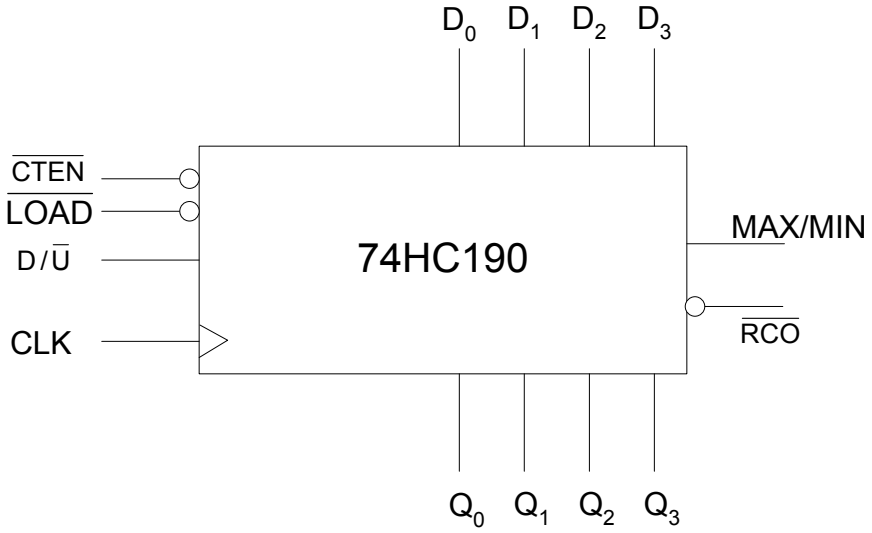


Figure 29.3 74HC190 4-bit Synchronous Up/Down Counter

Counter Decoding

In digital circuits the counter outputs are decoded using decoders or logic gates to determine when the counter is in a certain state in its counting sequence. For example, a 4-bit Modulus-16 counter counts from state 0 to state 15. A digital circuit is enabled when the count reaches count value 4, a second circuit is enabled when the count value reaches 8 and a third circuit is enabled when the count value reaches 12. A decoder using AND or NAND gates logic gates can be implemented. Figure 29.4

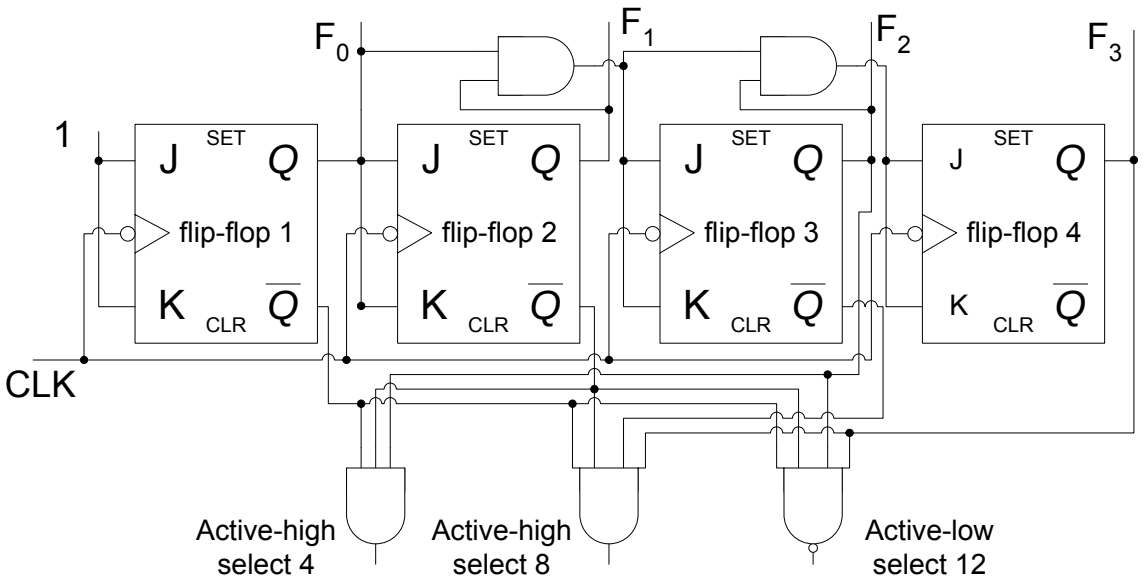


Figure 29.4a Decoder circuit decoding counter outputs 4, 8 and 12

The output of the first AND gate is set to logic high when the counter output is set to 0100 (4). The output of the second AND gate is set to logic high when the counter output is set to 1000 (8). The NAND gate is set to logic low when the counter output is set to 1100 (12). The propagation delay due to ripple effect in Asynchronous counters, discussed earlier causes these Asynchronous counters to work erratically. The propagation problem also exists in Synchronous counters to some degree due to the propagation delays from the clock transition to the Q output of the flip-flop which varies slightly for each flip-flop. The timing diagram for the decoder circuit shows that the decoder outputs are activated for different time intervals at different intervals which are not in a proper sequence. Figure 29.4b. The counter output for count 2 is detected by the AND gate decoder during interval t_{2A} to t_3 and again for a very short interval at t_4. Similarly, the counter output 8 is selected for a very short duration between intervals t_{AB} and t_9. The decoder outputs for very short durations at interval t_2, t_4, t_6 and t_8 are known as 'gliches'.

Glitches can be eliminated by enabling the decoder outputs after the glitches have settled down. Glitches are removed by using the clock signal to enable the decoder circuit. Figure 29.5. The clock signal is connected to the inputs of each of the three decoder gates. During the second, positive half of the clock signal the three gates are enabled, all the glitches occur during the first negative half of the clock cycle during which the decoder gates are disabled. This method is known as Strobing method where the decoder outputs are activated after some delay allowing the glitches to settle down.

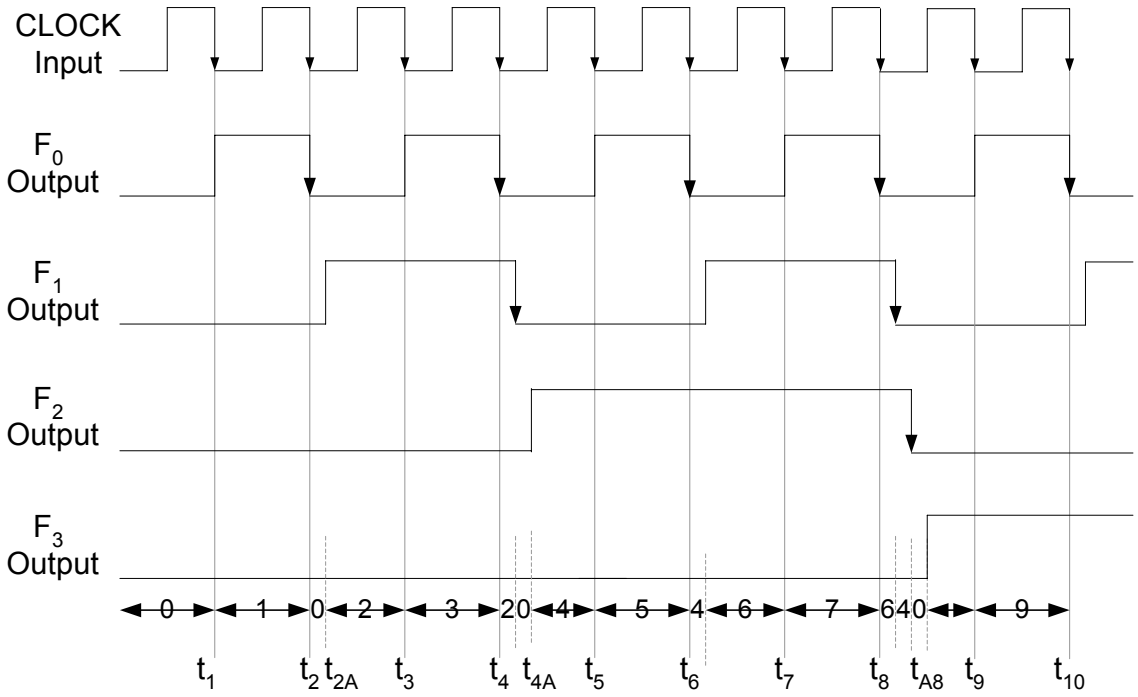


Figure 29.4b Decoded Outputs of Synchronous Counter

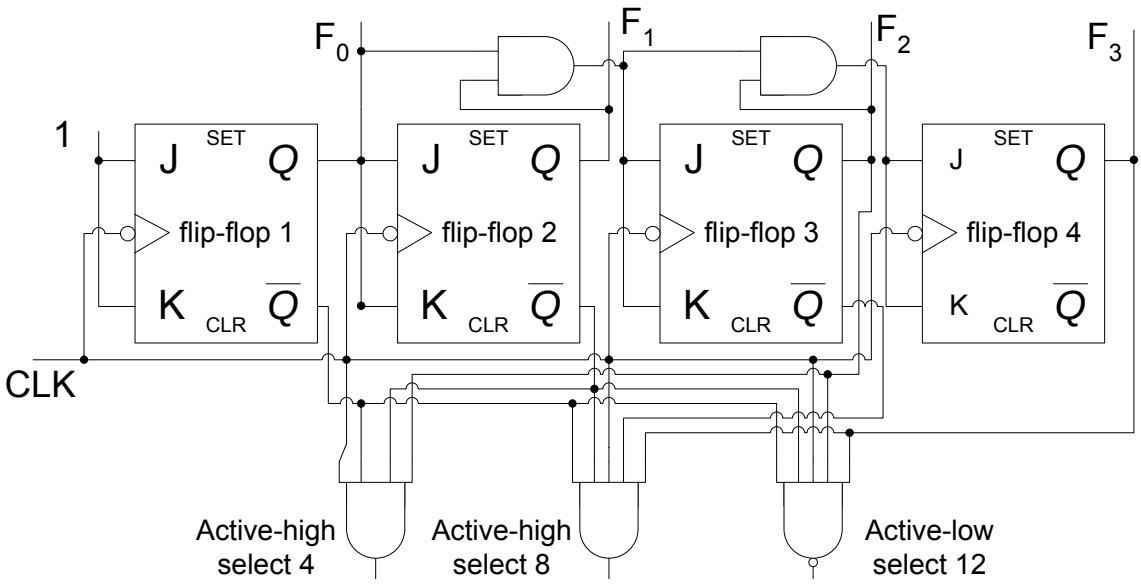


Figure 29.5 The Decoder circuit connected to remove glitches

Glitches occur even with Integrated circuits due to different propagation delays between the clock transition and the variable path lengths between different inputs and outputs within the integrated circuit. The Glitches that occur at the output of a 74x 138 3-to-8 decoder connected to a 74HC163 counter can be removed by enabling the decoder during the second half of the clock signal. Figure 29.6

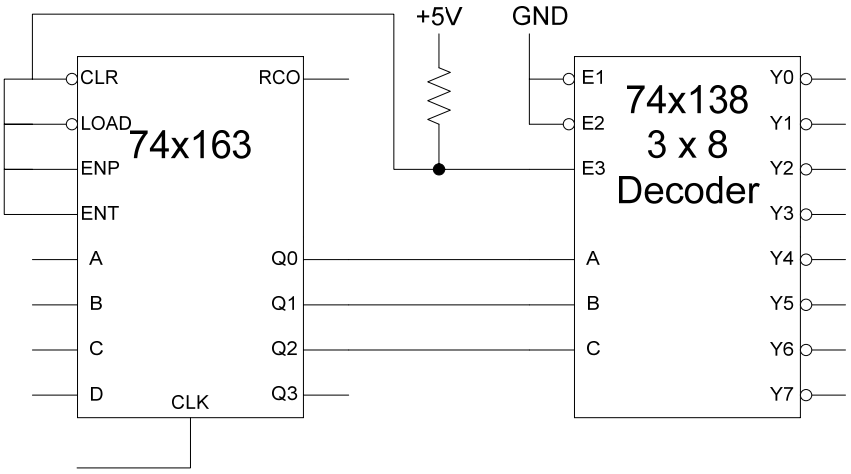


Figure 29.6 74x163 Counter output Decoded using a 3 x 8, 74x138 Decoder

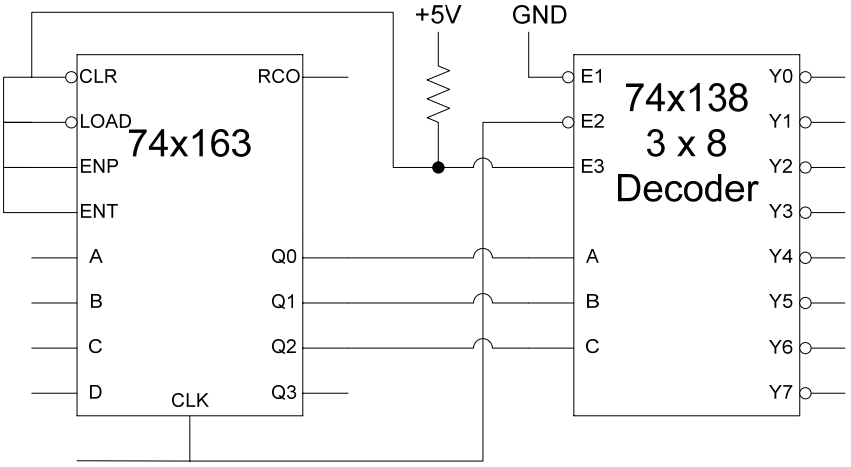


Figure 29.7 74x138 Decoder enabled by a clock signal

Counter Applications

1. Digital Clock

The primary use of counter is in counting applications and sequencing through a set of operations. A digital clock can be implemented using the AC 50 Hz frequency as the clock signal. Figure 29.8

In the digital clock circuit the 50 Hz, 220 volt ac mains sinusoidal signal is shaped into a 50 Hz, 5 volt square-wave signal. A divide-by-50 counter divides the input 50 Hz signal to a 1 Hz signal. The Seconds, divide-by-60 counter counts up to sixty seconds (0 to 59). The Minutes, divide-by-60 counter also counts up to sixty minutes (0-59). The Hours, decade counter counts from 0 to 9. The flip-flop connected to the output of the decade counter is set to 0 or 1 to represent hours from 0 to 9 and 10 to 12 respectively.

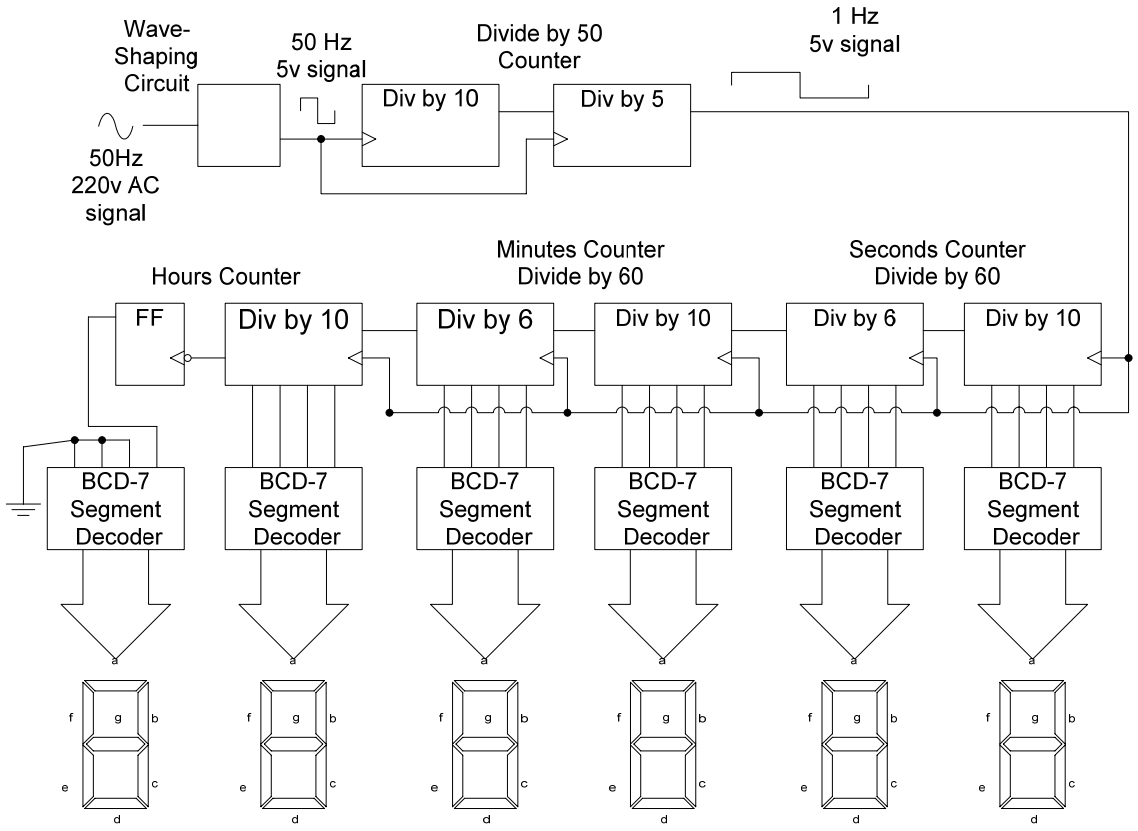


Figure 29.8 Digital Clock Circuit

The circuit diagram of the Divide by 60 Seconds and Minutes counter is shown in figure 29.9. The 74HC160A decade counter is sued which has Asynchronous clear. The divide by 60 counter is implemented by cascading two 74HC160A counters. The least significant counter which is the units counter is configured as a decade counter and counts from 0000 to 1001. On reaching the terminal count value, the RCO output of the Units counter is set to high which enables the tens counter. The tens counter is configured as a Mod-6 counter, thus it counts from 000 to 101. The NAND gate output is set to low when the counter counts up to 110, the NAND gate output is connected to the asynchronous clear input which resets the counter to 000. When the tens counter reaches its terminal count 101, and the units counter reaches its terminal count 1001, the AND gate output is set to logic high to indicate the terminal count 59 of the divide by 60 counter. The output of the AND gate is connected to the counter enable pins ENT and ENP of the next stage, thus on reaching the terminal count the next stage is enabled and the count is incremented by 1 on a clock transition.

The hours counter is implemented using a single decade counter and a flip-flop. Two counters are not required as the hours counter counts 12 unique output states. Implementation of a Mod-12 requires 5 flip-flops. Figure 29.10

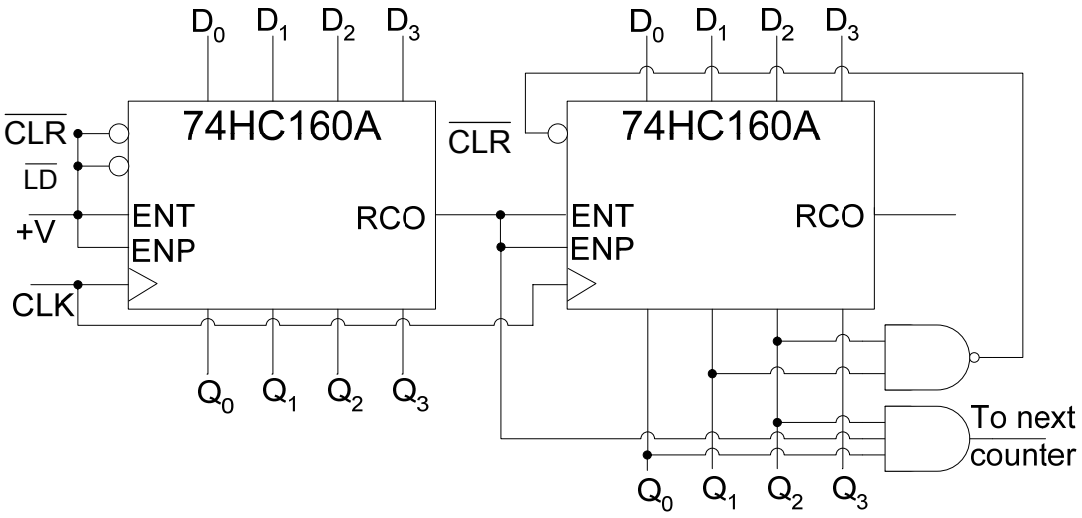


Figure 29.9 Divide by 60 Minutes and Seconds counter

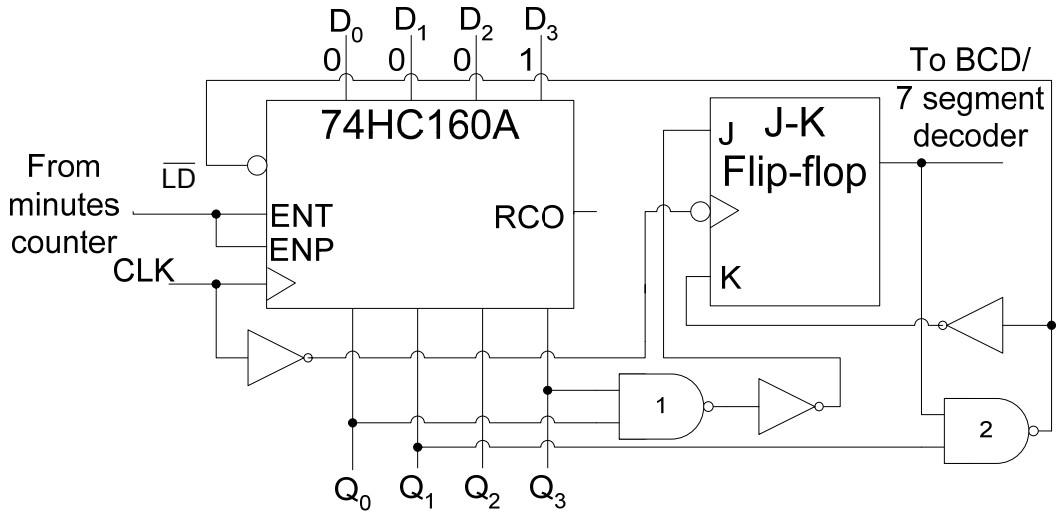


Figure 29.10a Hours Counter Circuit

The hours unit counter circuit is configured as a decade counter, counting from 0000 to 1001 when it is enabled by the Minutes counter circuit. The terminal count 1001 is detected by the NAND gate (1) which sets the J input of the flip-flop to logic 1. The K input of the flip-flop is at Logic 0, therefore on a clock transition the J–K flip-flop output is set to logic 1 when the units counter recycles to 0000. The NOT gate connected to the clock input of the J-K flip-flop allows the J-K flip-flop to trigger when the units counter is triggered to count from 1001 to 0000. The unit counter counts to 0001 and 0010 to represent hours 11 and 12 respectively along with the output of the J-K flip-flop which is set to logic 1. On the next clock transition when the units counter counts to 0011 the NAND gate (2) output set to logic 0 reloads the units counter with the count value 0001 and the J-K flip-flop toggles to 0 output as its K input which is set to logic 1.

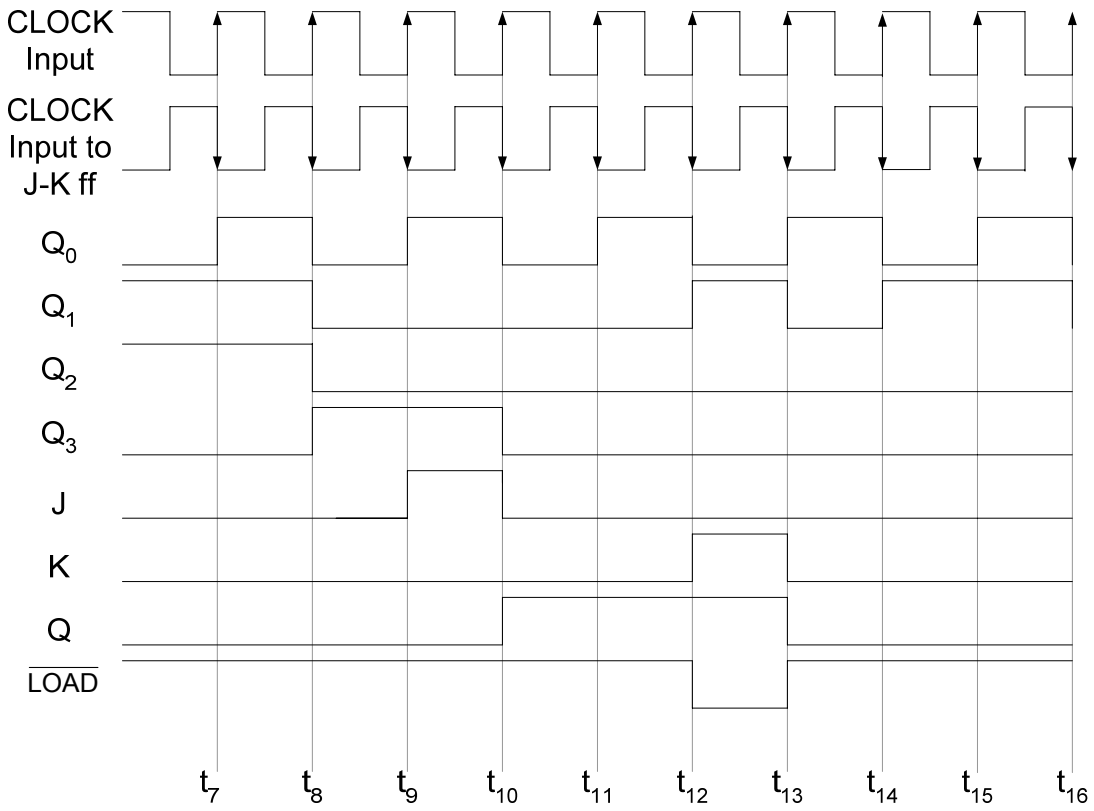


Figure 29.10b Hours Counter timing diagram

2. Frequency Counter

A frequency counter is used to measure the frequency of an input signal. The basis for the operation of a frequency counter is counting of the clock pulses in predetermined time interval. The frequency of periodic signal is the number of cycles in a time period of one second. The frequency of the unknown signal can be calculated by counting the number of clock pulses of the unknown signal and dividing the count number by the time interval in which the clock pulses are counted, Figure 29.11

In the circuit shown, the input signal with unknown frequency is applied at the AND gate input. The second input of the AND gate is connected to a signal which determines the sampling interval. The signal is set to logic high at interval t_1 to enable the AND gate allowing the input signal to be connected to the clock input of the counter circuit. The sampling interval signal is set to logic low at the end of the sampling interval t_2 to disable the AND gate and inhibit the counter from counting. Before the counter counts the clock pulses of the input signal it is reset by activating the Asynchronous input to clear the counter at interval t_0. At the end of the sampling interval the counter output is displayed on 7-segment displays.

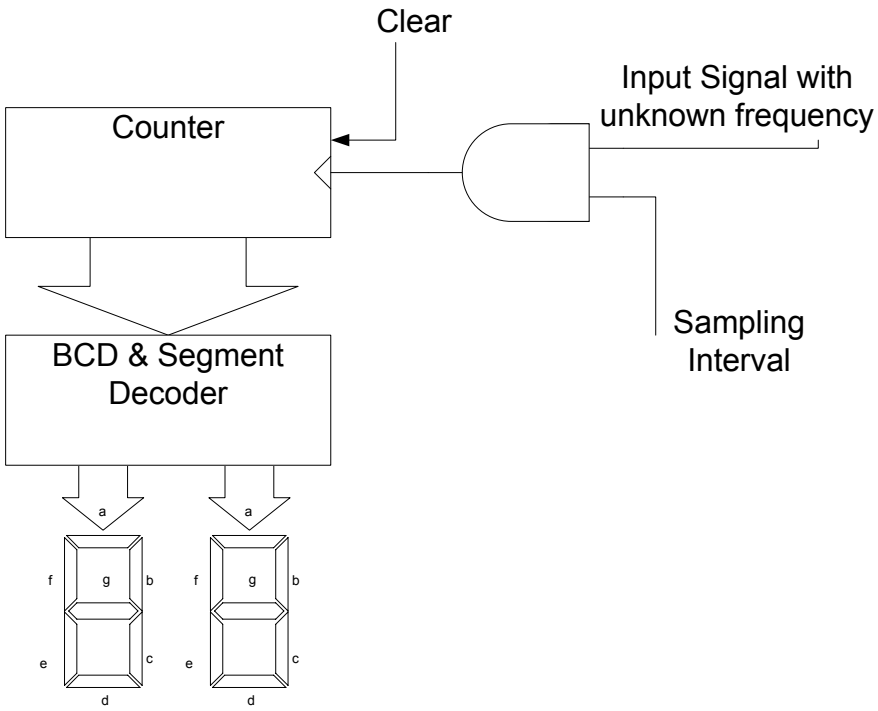


Figure 29.11a Frequency Counter Circuit

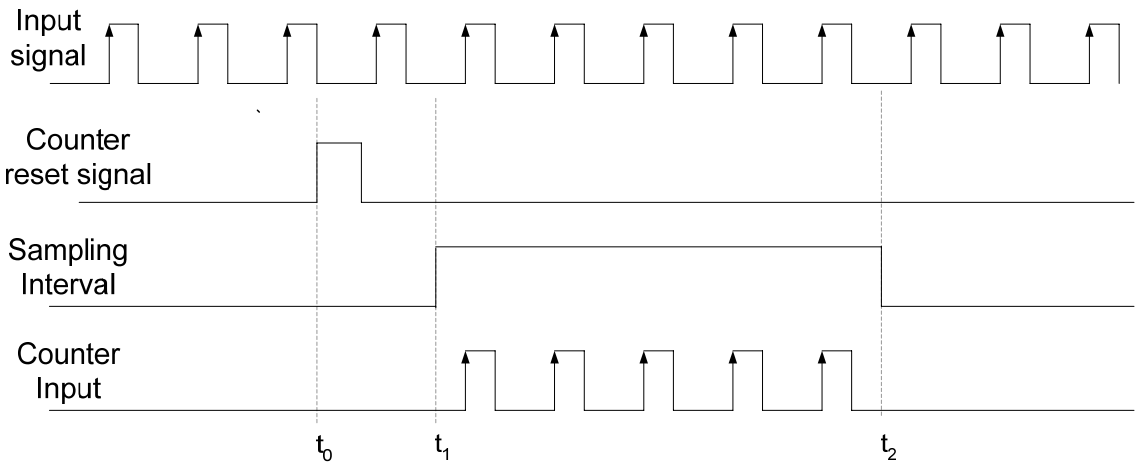


Figure 29.11b Timing diagram of the Frequency Counter Circuit

The accuracy of the frequency counter depends on the duration of the timing sampling interval, which must be very accurate. Consider that during a sampling interval of 1 second 4573 clock pulses of the input signal are measured. Thus, the frequency of the unknown signal is 4573 Hz. If the same input signal is sampled using a 0.1 second sampling interval then 457.3 pulses are counted, which means that either 457 or 458 will be counted depending on the start of the sampling interval at t_1. Similarly, if the sampling interval is reduced to 0.01 seconds, the numbers of clock pulses measured are 45.73, which means that either 45 or 46 will be read.

Very accurate sampling intervals are implemented using cascaded counter which is connected to a very accurate timing signal generated by a crystal controlled oscillator (Astable multi-vibrator). The output timing signal of each cascade section is available at a switch which is used to select the appropriate timing signal for controlling the sampling interval. The output of the switch is connected to the clock input of a negative triggered J-K flip-flop, which divides the input signal by 2. Thus, when the 1 Hz sampling interval is selected, the signal at the output of the J-K flip-flop has a time period of 2 seconds. Figure 29.12

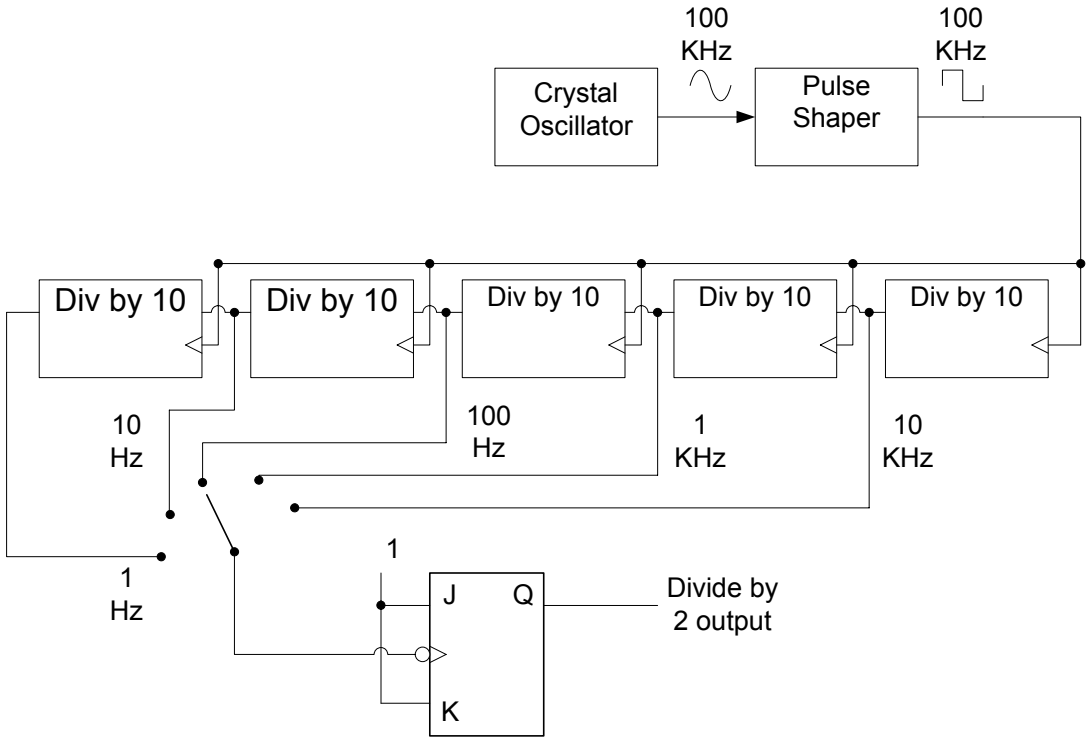


Figure 29.12 Cascaded Counter circuit for obtaining accurate sampling intervals

The detailed circuit diagram and the timing diagram of the frequency diagram are shown in figure 29.13. In the timing diagram the Sampling Interval pulse is obtained from the output of the J-K flip-flop shown in figure 29.8. The duration of the Sampling interval pulse can be selected through the switch. The sampling interval signal is connected to the input of the 3-input AND gate and the clock input of the second J-K flip-flop which toggles its output at each negative transition of the clock. When the output of the second flip-flop changes to logic 1 (interval t_1) it triggers the One-Shot which generates a short output pulse which clears the Counter circuit. At interval t_2 during the positive half of the sampling interval when the output of the second J-K flip-flop is high the 3-input AND gate is enabled and the input signal with unknown frequency is applied at the input of the counter, which count the input signal pulses. At interval t_3 there is negative transition of the sampling signal, which triggers the second flip-flop changing its output to logic 0. Logic 0 output of the flip-flop disables the 3-input AND gate inhibiting the counter from counting. The pulses counted by the counter during interval t_2 to t_3 are directly displayed.

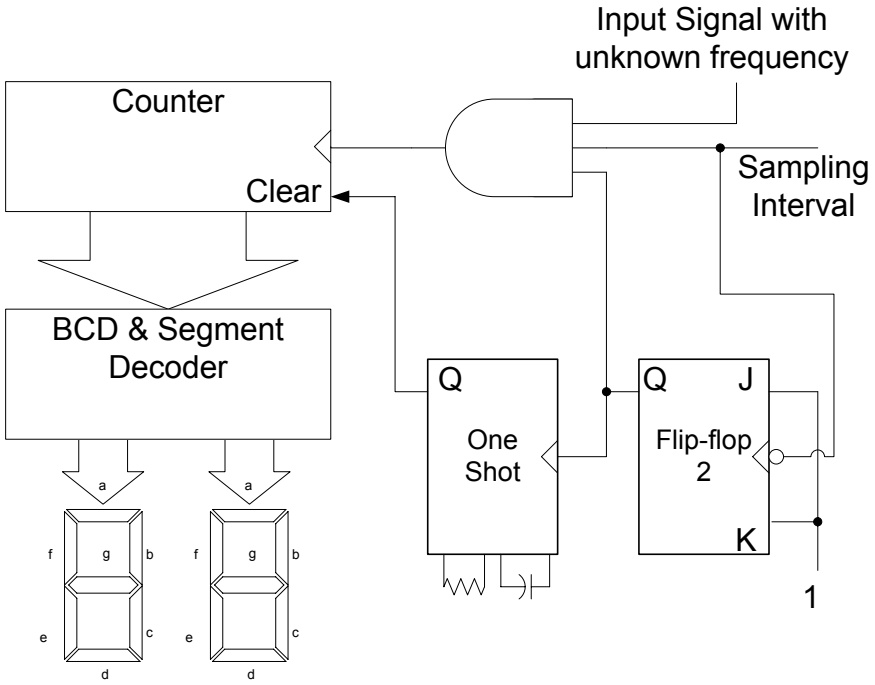


Figure 29.13a Detailed circuit diagram of a frequency counter

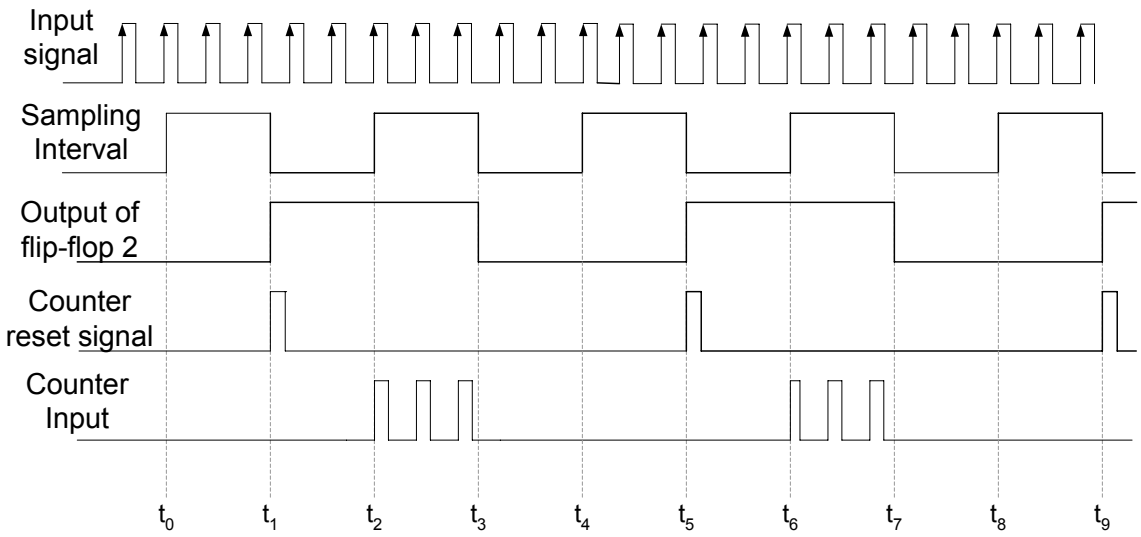


Figure 29.13b Timing diagram of the frequency counter circuit

Design of Synchronous Counters

The counters that have been discussed are binary counters that count in a sequence either upwards or downwards. The count start and end sequence of a counter can also be set arbitrarily and the counter can then count up or down with in the terminal count limits.

through a set of predefined arbitrary values. Counters can also be implemented using D flip-flops instead of J-K flip-flops. Counters are sequential circuits which are designed using standard set of steps.

Sequential Circuit (State Machine)

A general Sequential circuit consists of a combinational circuit and a memory circuit (flip-flop). In a clocked Sequential circuit the memory element has a clock input. At any given instant the memory element is in its present state. On a clock transition the output of the memory element changes to the next state. The next state is determined by the inputs applied at the memory input at the time of clock transition. The inputs to the memory which allow the memory to change its state on a clock transition are known as excitation inputs or excitation variables. The present state of the memory is represented by state variables. The state variables and the inputs to the sequential circuit determine the sequential circuit output. Figure 29.14

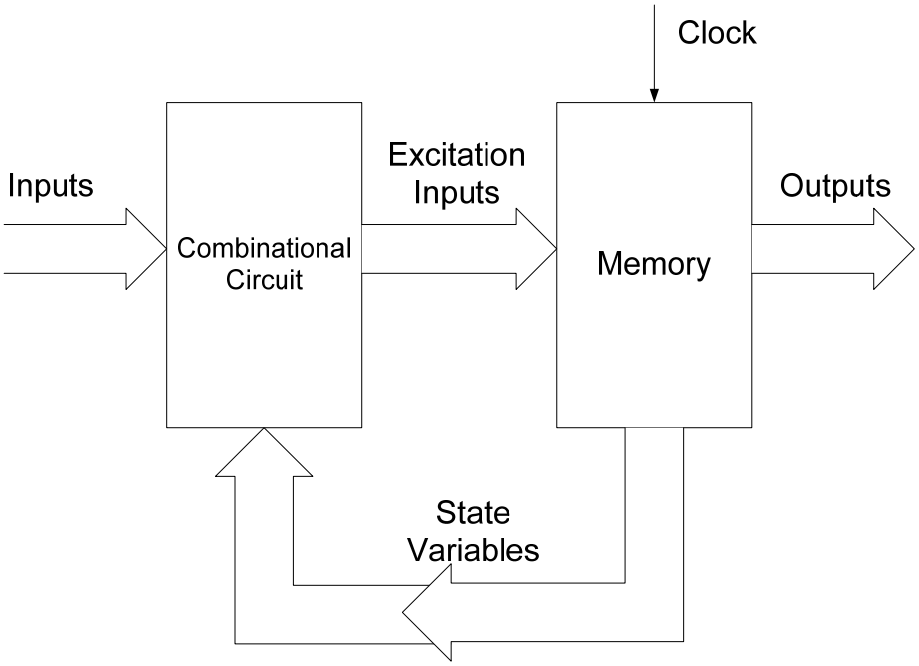


Figure 29.14 Clocked Sequential Circuit Block diagram

Design Procedure

The design procedure is based on a number of steps starting from defining the state diagram and ending at the implementation of State machine.

1. State Diagram

A sequential circuit (state machine) is described by a state diagram, which shows the sequence of state through which the sequential circuit progresses when it is clocked. The state diagram of a 3-bit Synchronous Up-Counter (sequential circuit) is shown in the **figure. 28.3**

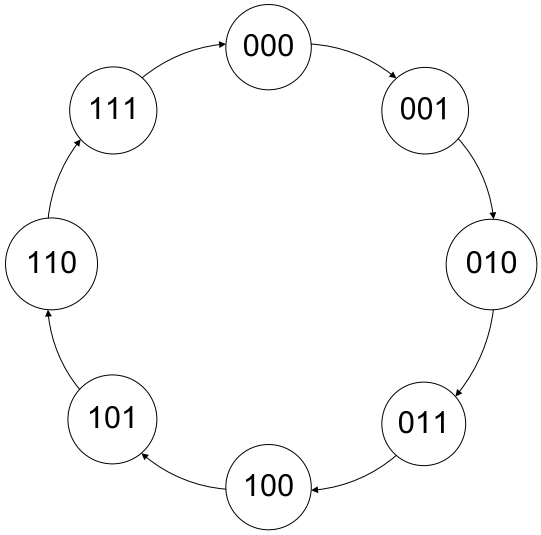


Figure 28.3 State diagram of a 3-bit Up-Counter

2. Next-State Table

Once the state diagram of the sequential circuit is defined, a Next-State Table is derived which lists each present state and the corresponding next state. The next state is the state to which the sequential circuit switches when a clock transition occurs. **Table 28.1**

Present State			Next State		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
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

Table 28.1 Next-State Table for a 3-bit Up-Counter

3. Flip-flop Transition Table

The Memory element of the Sequential circuit is implemented using flip-flops. The number of flip-flops used is determined by the total number of states. When there is a clock transition at the clock input of the flip-flops they change from their present state to the next state. The Flip-flop transition table lists all the possible flip-flop input combinations which allows the present state to change to the next state on a clock transition. The flip-flop transition table is based on the flip-flop used (D, S-R or J-K). **Table 28.2**

Flip-flop Inputs		Output Transitions	
J	K	Q_t	Q_{t+1}
0	x	0	0
1	x	0	1
x	1	1	0
x	0	1	1

Table 28.2 J-K flip-flop Transition table

4. Karnaugh Maps

For each state variable shown in the Next-State table, the change from present state to the next state on a clock transition depends upon the J-K inputs. **Table 28.3.** Considering the state variable Q_2, J_2 and K_2 inputs set to 0 and x (don't care) allow Q_2 to change from present state 0 to next state 0 shown in the first row. Similarly, the state variable Q_0 changes from 1 to 0 when J_0 and K_0 inputs are set at x (don't care) and 1 respectively. The table is completed using the information in the Next-State table and the J-K flip-flop transition table. The J-K inputs can be directly mapped to Karnaugh maps. **Table 28.4**

Present State			Next State			J-K flip-flop inputs					
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0
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

Table 28.3 J-K flip-flop input table

Q_2Q_1/Q_0	0	1
00	0	0
01	0	1
11	x	x
10	x	x

Q_2Q_1/Q_0	0	1
00	x	x
01	x	x
11	0	1
10	0	0

Table 28.4a Karnaugh Map for J_2 and K_2 inputs

Q_2Q_1/Q_0	0	1
00	0	1
01	x	X
11	x	X
10	0	1

Q_2Q_1/Q_0	0	1
00	x	x
01	0	1
11	0	1
10	x	x

Table 28.4b Karnaugh Map for J_1 and K_1 inputs

Q_2Q_1/Q_0	0	1
00	1	x
01	1	x
11	1	x
10	1	x

Q_2Q_1/Q_0	0	1
00	x	1
01	x	1
11	x	1
10	x	1

5. Logic expressions for Flip-flop Inputs

Simplified expressions for J_2-K_2, J_1-K_1 and J_0-K_0 are directly obtained from the Karnaugh maps.

$$J_2 = Q_1Q_0$$
$$K_2 = Q_1Q_0$$

$$J_1 = Q_0$$
$$K_1 = Q_0$$

$$J_0 = 1$$
$$K_0 = 1$$

6. Sequential Circuit Implementation

The Boolean expressions obtained in the previous step are implemented using logic gates. The sequential circuit implemented is shown in **figure 28.4.**

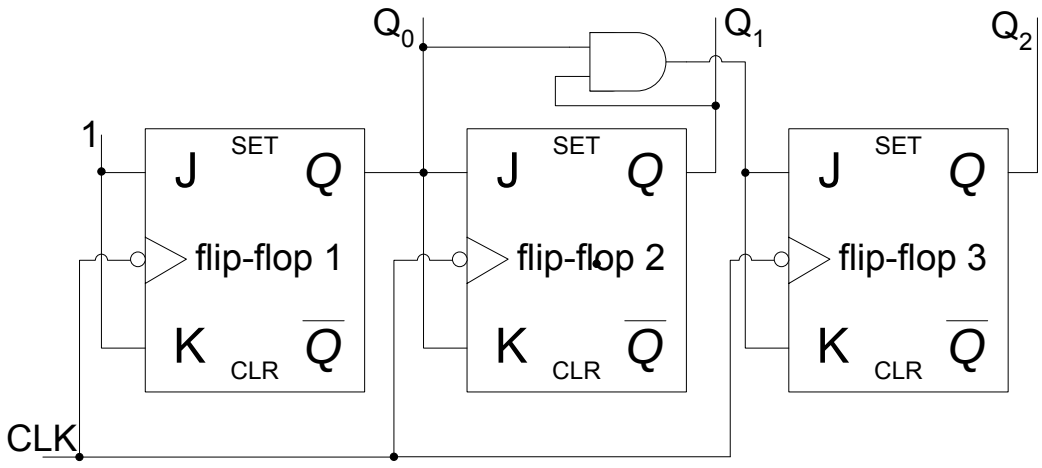


Figure 28.4 Implementation of the Sequential Circuit

Implementing a 3-bit Up/Down Counter

1. State Diagram

The state diagram of a 3-bit Up/Down Synchronous Counter is shown in the **figure. 28.5**. X=0 and X =1 indicates that the counter counts up when input X = 0 and it counts down when X =1.

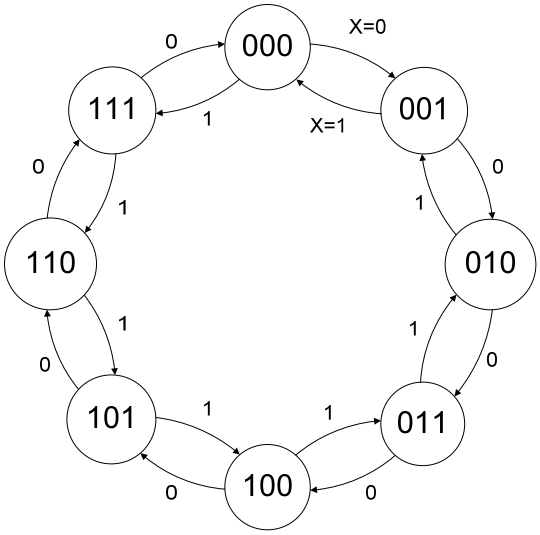


Figure 28.5 State diagram of a 3-bit Up-Counter

2. Next-State Table

The next state is the state to which the sequential circuit switches when a clock transition occurs. **Table 28.5.** The next state outputs for X=0 and X=1 are shown separately.

Present State			Next State X=0			Next State X=1		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	0	1	1	1	1
0	0	1	0	1	0	0	0	0
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	0	1	0
1	0	0	1	0	1	0	1	1
1	0	1	1	1	0	1	0	0
1	1	0	1	1	1	1	0	1
1	1	1	0	0	0	1	1	0

Table 28.5 Next-State Table for a 3-bit Up-Counter

3. Flip-flop Transition Table

The flip-flop transition table is based on the J-K flip-flop. **Table 28.6**

Flip-flop Inputs		Output Transitions	
J	K	Q_t	Q_{t+1}
0	x	0	0
1	x	0	1
x	1	1	0
x	0	1	1

Table 28.6 J-K flip-flop Transition table

4. Karnaugh Maps

The J-K flip-flop inputs for change in state variables when X=0 and X=1 are shown in the **table 28.7**. The J-K inputs can be directly mapped to 4-Variable Karnaugh maps. **Table 28.8.**

Present State			Next State X=0			J-K flip-flop inputs					
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0
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

Table 28.7a J-K flip-flop input table for X=0

Present State			Next State X=1			J-K flip-flop inputs					
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	1	1	1	1	x	1	x	1	x
0	0	1	0	0	0	0	x	0	x	x	1
0	1	0	0	0	1	0	x	x	1	1	x
0	1	1	0	1	0	0	x	x	0	x	1
1	0	0	0	1	1	x	1	1	x	1	x
1	0	1	1	0	0	x	0	0	x	x	1
1	1	0	1	0	1	x	0	x	1	1	x
1	1	1	1	1	0	x	0	x	0	x	1

Table 28.7b J-K flip-flop input table for X=1

Q_2Q_1/Q_0X	00	01	11	10
00	0	1	0	0
01	0	0	0	1
11	x	x	x	x
10	x	x	x	x

Q_2Q_1/Q_0X	00	01	11	10
00	x	x	x	x
01	x	x	x	x
11	0	0	0	1
10	0	1	0	0

Table 28.8a Karnaugh Map for J_2 and K_2 inputs

Q_2Q_1/Q_0X	00	01	11	10
00	0	1	0	1
01	x	x	x	x
11	x	x	x	x
10	0	1	0	1

Q_2Q_1/Q_0X	00	01	11	10
00	x	x	x	x
01	0	1	0	1
11	0	1	0	1
10	x	x	x	x

Q_2Q_1/Q_0X	00	01	11	10
00	1	1	x	x
01	1	1	x	x
11	1	1	x	x
10	1	1	x	x

Q_2Q_1/Q_0X	00	01	11	10
00	x	x	1	1
01	x	x	1	1
11	x	x	1	1
10	x	x	1	1

Table 28.8c Karnaugh Map for J_0 and K_0 inputs

5. Logic expressions for Flip-flop Inputs

Simplified expressions for J_2-K_2, J_1-K_1 and J_0-K_0 are directly obtained from the Karnaugh maps.

$$J_2 = Q_1 Q_0 \overline{X} + \overline{Q_1}\,\overline{Q_0} X$$
$$K_2 = Q_1 Q_0 \overline{X} + \overline{Q_1}\,\overline{Q_0} X$$

$$J_1 = Q_0 \overline{X} + \overline{Q_0} X$$
$$K_1 = Q_0 \overline{X} + \overline{Q_0} X$$

$$J_0 = 1$$
$$K_0 = 1$$

6. Sequential Circuit Implementation

The Boolean expressions obtained in the previous step are implemented using logic gates. The sequential circuit implemented is shown in **figure 28.6.**

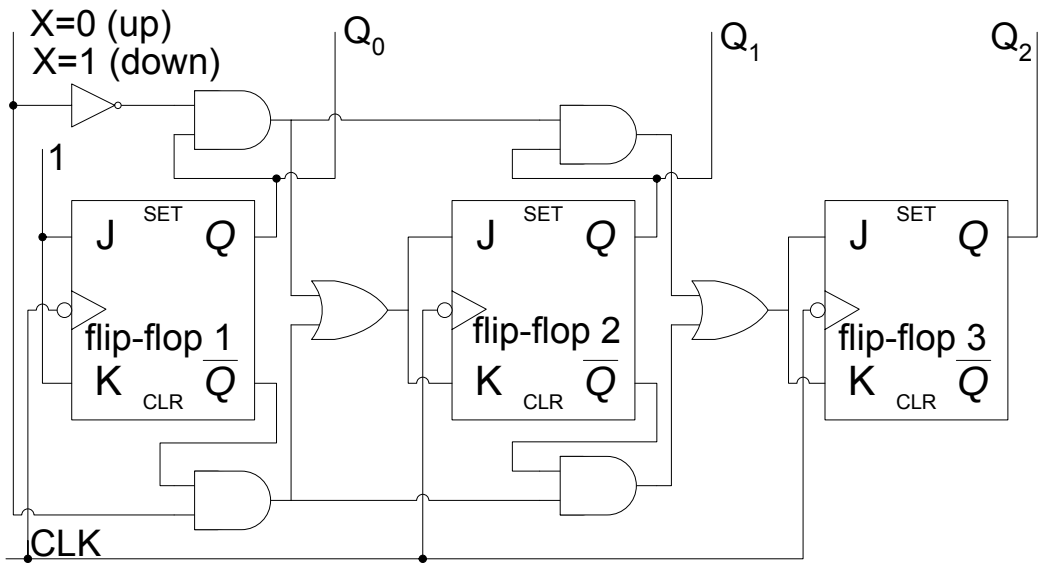


Figure 28.6 Implementation of the Sequential Circuit

Lesson No. 30

DIGITAL CLOCK

The timing diagram figure 30.1a shows the time interval t_6 to t_{11} and t_{19} to t_{21}. At time interval t_9 the units counter counts to 1001 (9) which is the terminal count of the 74x160 decade counter. The RCO signal is set to logic 1 to indicate the terminal count. The RCO signal is connected to the ENP and ENT enable signals of the tens counter. At interval t_{10} the tens counter is incremented by 1, the units counter recycles to 0000 (0) and the RCO signal is deactivated inhibiting the tens counter from incrementing. At interval t_{19} the units counter once again reaches its terminal count activating the RCO signal and enabling the tens counter to increment its initial count 0001 to 0010. The counting sequence continues until the tens counter increments to 0101 (5) and the units counter recycles to 0000 and continues with the counting sequence on each positive clock transition.

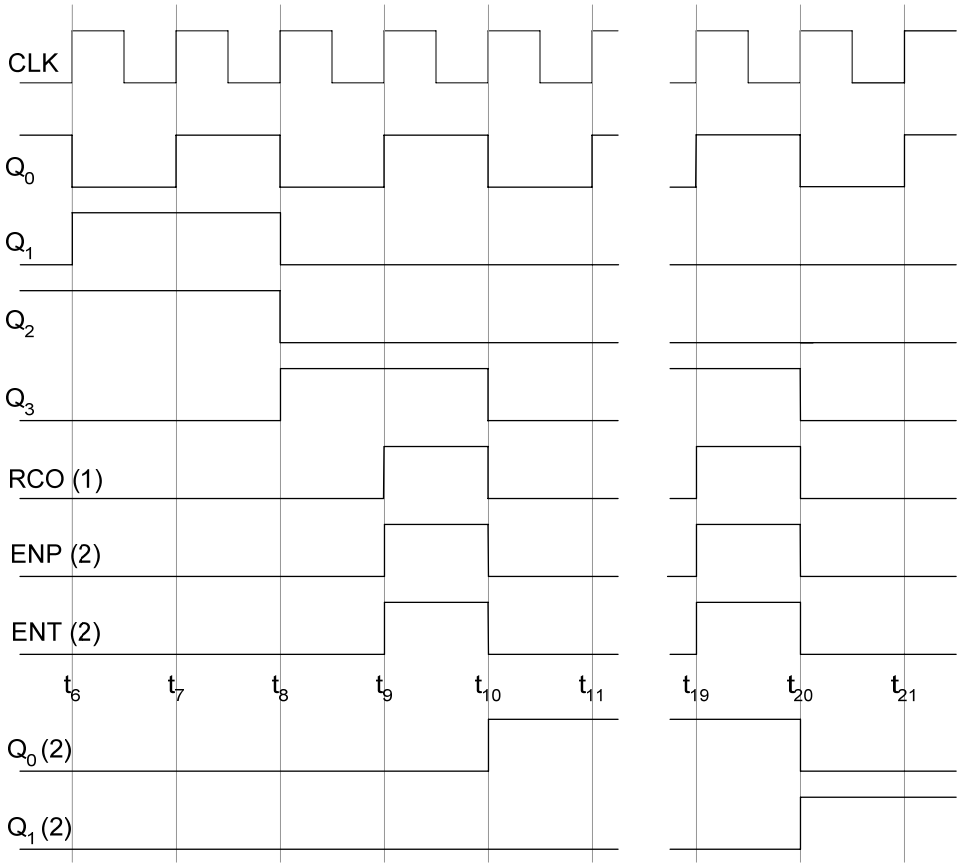


Figure 30.1a Timing diagram of the divide by 60 minutes/seconds counter

The timing diagram fig 30.1b shows the timing sequence from interval t_{56} to t_{64}. The unit counter reaches its terminal count at interval t_{59}. The output of the 3-input AND gate is set to logic high. The output of the AND gate is connected to the ENP and ENT enable inputs of the next counter, thus enabling the next counter. At interval t_{60}, on a positive clock transition the units counter recycles to 0000, the tens counter increments to 0110 (6) setting the output of the NAND gate to logic 0 and the next counter increments its count. The NAND gate output is connected tot the asynchronous active low clear input of the tens counter which is asynchronously cleared to 0000.

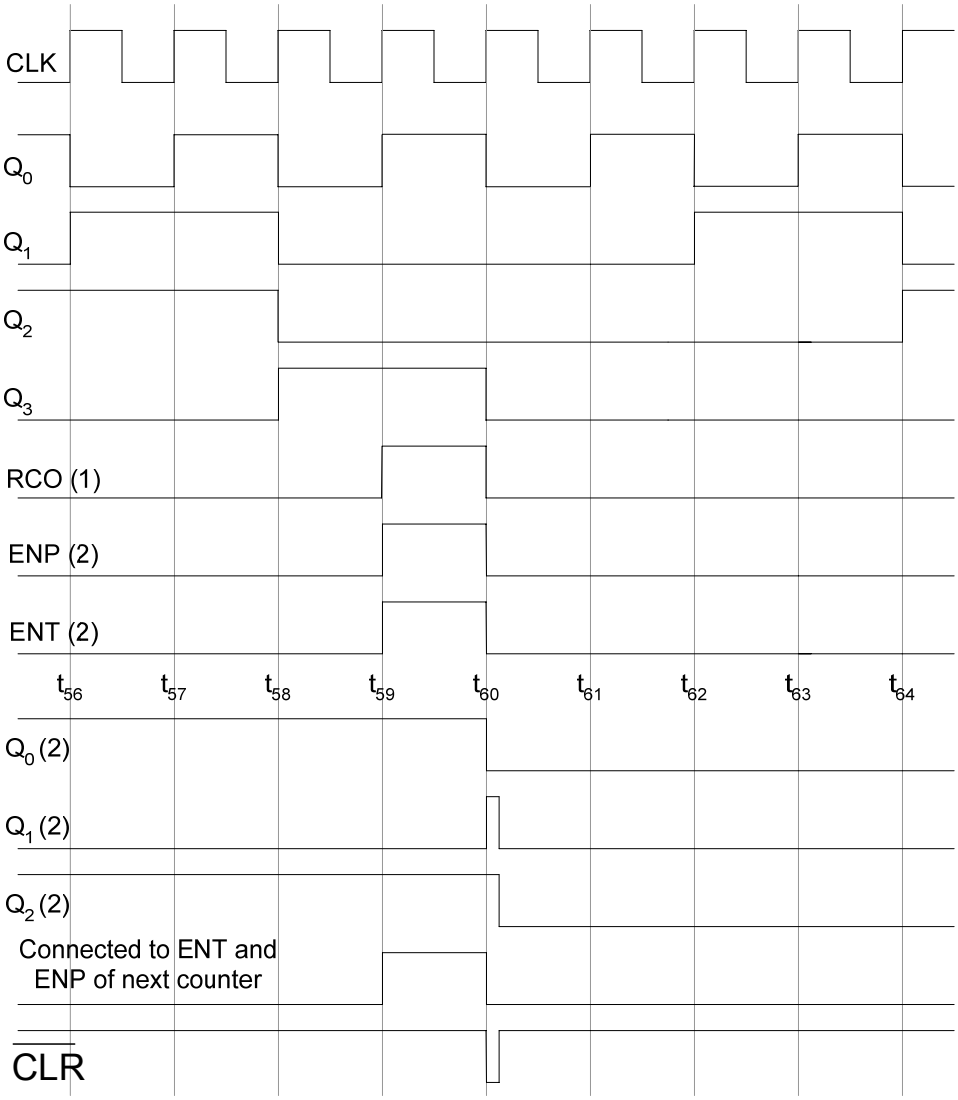


Figure 30.1b Timing diagram of the divide by 60 counter at time interval t_{56} to t_{64}

The hours unit counter circuit is configured as a decade counter, counting from 0000 to 1001 when it is enabled by the Minutes counter circuit. The NOT gate connected to the clock input of the J-K flip-flop allows the negative-edge triggered J-K flip-flop to trigger when the units counter is triggered to count from 0000 to 1001. The terminal count 1001 is detected by the NAND gate (1) at interval t_9 which sets the J input of the flip-flop to logic 1. The K input of the flip-flop is at Logic 0, therefore on a clock transition at interval t_{10} the J–K flip-flop output Q is set to logic 1, the units counter recycles to 0000 resetting J input to logic 0. The unit counter counts to 0001 and 0010 to represent hours 11 and 12 in interval t_{11} and t_{12} respectively. At interval t_{12} as the unit counters count changes from 1011 (11) to 1100 (12), Q_1 output is set to logic 1, which sets the output of the NAND gate to logic 0 as the other input of the NAND is already at logic 1 (Q). The NAND gate sets the K input to logic 1 and setting the active-low LOAD signal to logic 0. At interval t_{13}, at the positive clock transition the unit counter is reloaded with the count 0001, the J-K flip-flop output toggles to logic 0 from logic 1. As the units counter is reloaded with count 0001, the K input is set to logic 0. At intervals t_{14}, t_{15} and t_{16} the hours unit counter increments the hours count by 1.

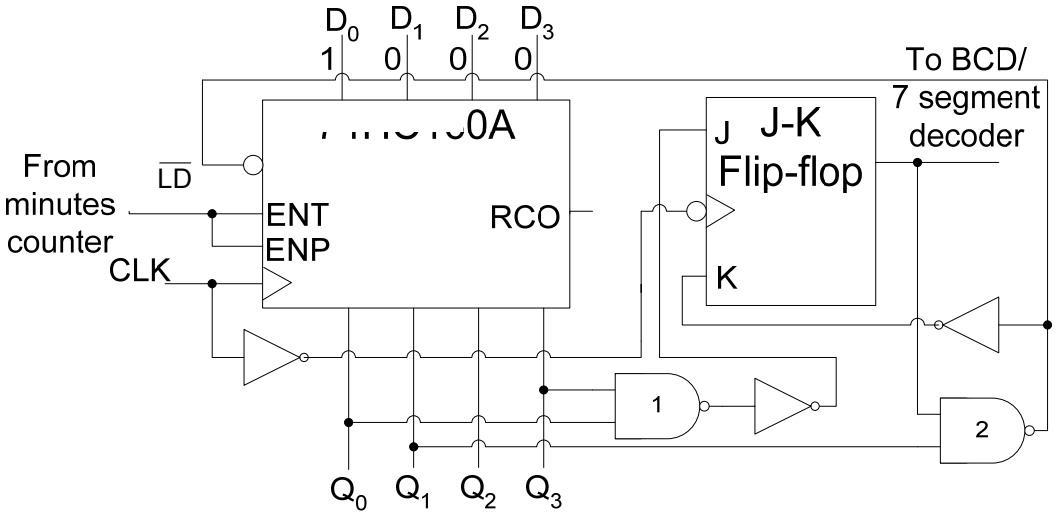


Figure 30.2a Hours Counter Circuit

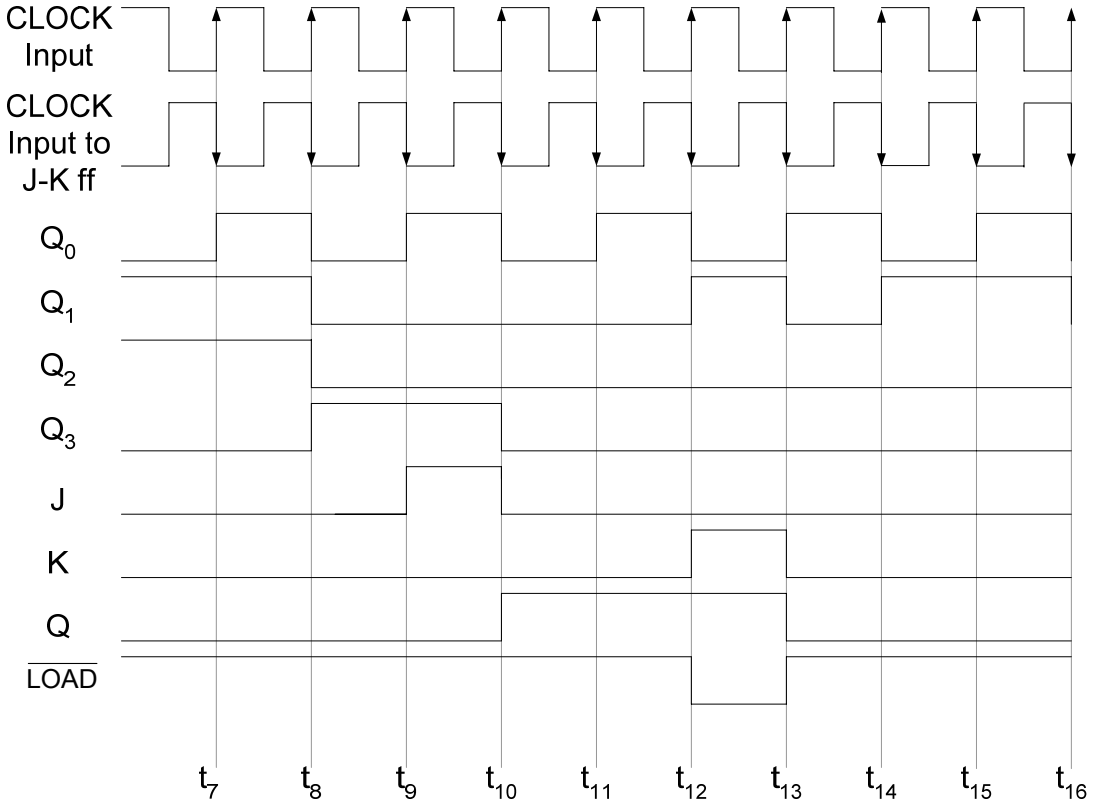


Figure 30.2b Hours Counter timing diagram

3. Frequency Counter

A frequency counter is used to measure the frequency of an input signal. The basis for the operation of a frequency counter is counting of the clock pulses in a predetermined time interval. The frequency of periodic signal is the number of cycles in a time period of one second. The frequency of the unknown signal can be calculated by counting the number of

clock pulses of the unknown signal and dividing the count number by the time interval in which the clock pulses are counted, Figure 30.3

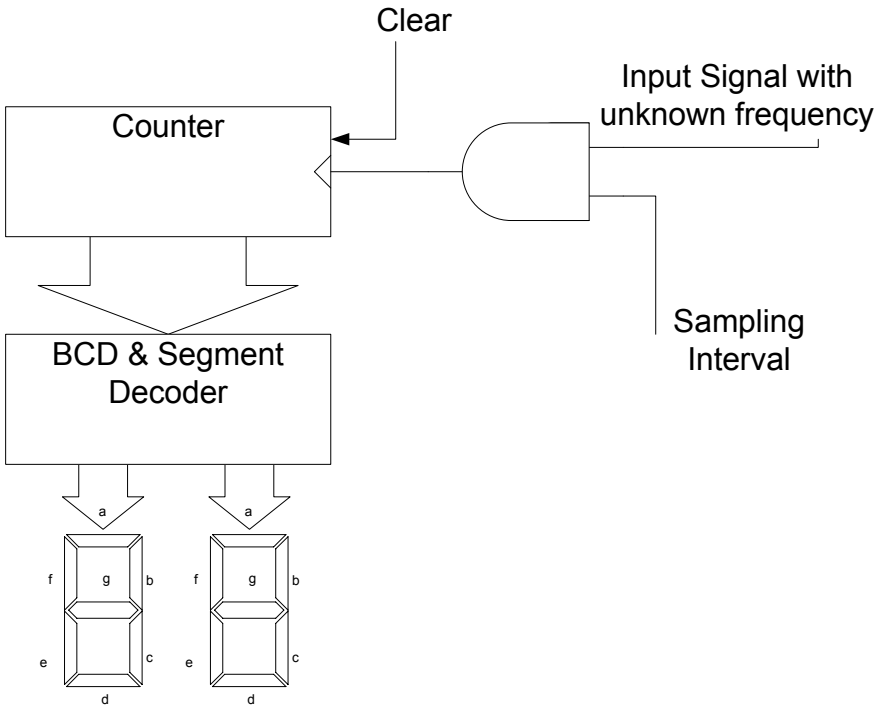


Figure 30.3a Frequency Counter Circuit

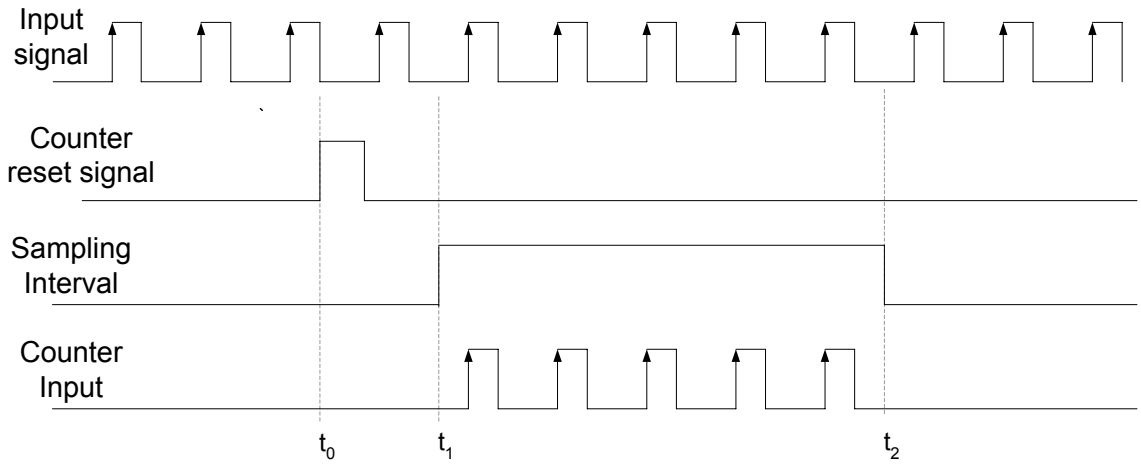


Figure 30.3b Timing diagram of the Frequency Counter Circuit

In the circuit shown, the input signal with unknown frequency is applied at the AND gate input. The second input of the AND gate is connected to a signal which determines the sampling interval. The signal is set to logic high at interval t_1 to enable the AND gate allowing the input signal to be connected to the clock input of the counter circuit. The sampling interval signal is set to logic low at the end of the sampling interval t_2 to disable the AND gate and inhibit the counter from counting. Before the counter counts the clock pulses of the input signal

it is reset by activating the Asynchronous input to clear the counter. At the end of the sampling interval the counter output is displayed on 7-segment displays.

The accuracy of the frequency counter depends on the duration of the timing sampling interval, which must be very accurate. Consider that during a sampling interval of 1 second 4573 clock pulses of the input signal are measured. Thus, the frequency of the unknown signal is 4573 Hz. If the same input signal is sampled using a 0.1 second sampling interval then 457.3 pulses are counted, which means that either 457 or 458 will be counted depending on the start of the sampling interval at t_1. Thus the frequency is determined to be either 4570 or 4580. Similarly, if the sampling interval is reduced to 0.01 seconds, the numbers of clock pulses measured are 45.73, which means that either 45 or 46 will be read indicating a frequency of 4500 or 4600.

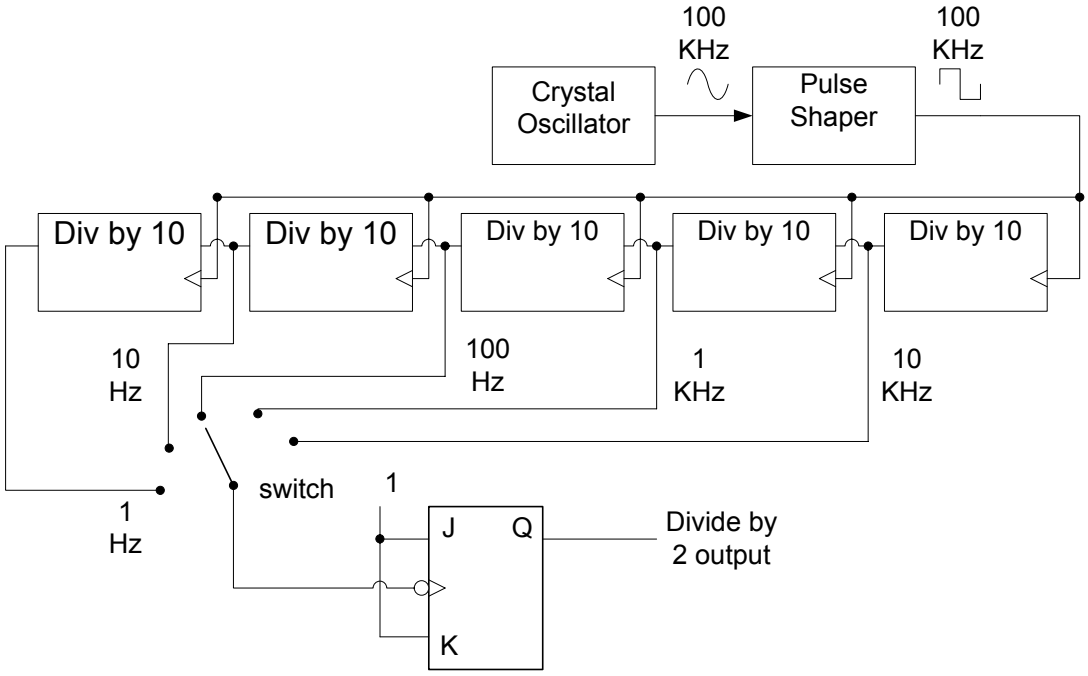


Figure 30.4 Cascaded Counter circuit for obtaining accurate sampling intervals

Very accurate sampling intervals are implemented using cascaded counter which is connected to a very accurate timing signal generated by a crystal controlled oscillator (Astable multi-vibrator). The output timing signal of each cascade section is available at a switch which is used to select the appropriate timing signal for controlling the sampling interval. The output of the switch is connected to the clock input of a negative triggered J-K flip-flop, which divides the input signal by 2. Thus, when the 1 Hz sampling interval is selected, the signal at the output of the J-K flip-flop has a time period of 2 seconds. Figure 30.4.

The detailed circuit diagram and the timing diagram of the frequency diagram are shown in figure 30.5. In the timing diagram the Sampling Interval pulse is obtained from the output of the J-K flip-flop shown in figure 30.4. The duration of the Sampling interval pulse can be selected through the switch. The sampling interval signal is connected to the input of the 3-input AND gate and the clock input of the second J-K flip-flop which toggles its output at each negative transition of the clock. When the output of the second flip-flop changes to logic 1 (interval t_1) it triggers the One-Shot which generates a short output pulse which clears the

Counter circuit. At interval t_2 during the positive half of the sampling interval when the output of the second J-K flip-flop is high the 3-input AND gate is enabled and the input signal with unknown frequency is applied at the input of the counter, which count the input signal pulses. At interval t_3 there is negative transition of the sampling signal, which triggers the second flip-flop changing its output to logic 0. Logic 0 output of the flip-flop disables the 3-input AND gate inhibiting the counter from counting. The pulses counted by the counter during interval t_2 to t_3 are directly displayed.

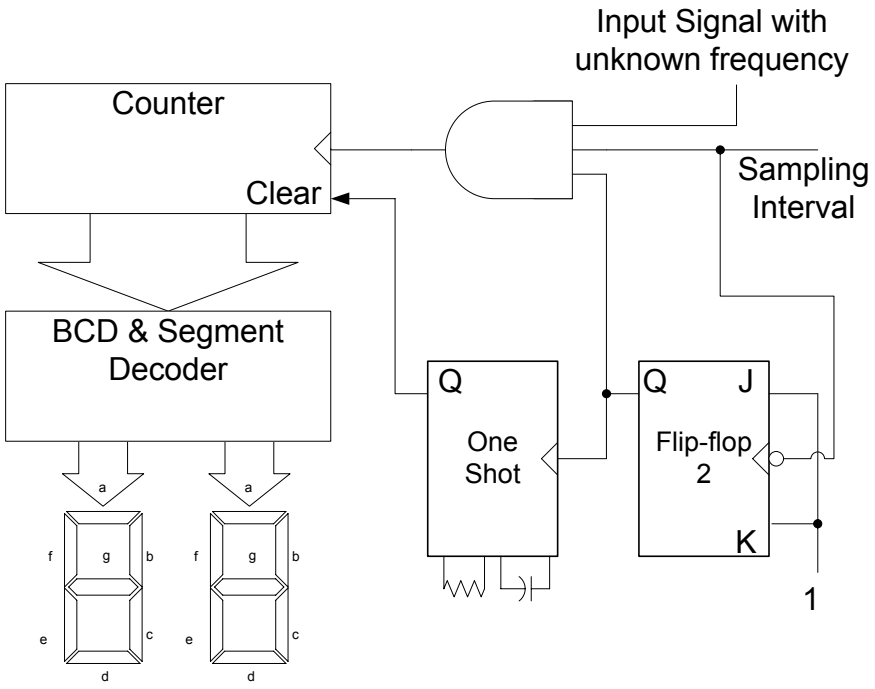


Figure 30.5a Detailed circuit diagram of a frequency counter

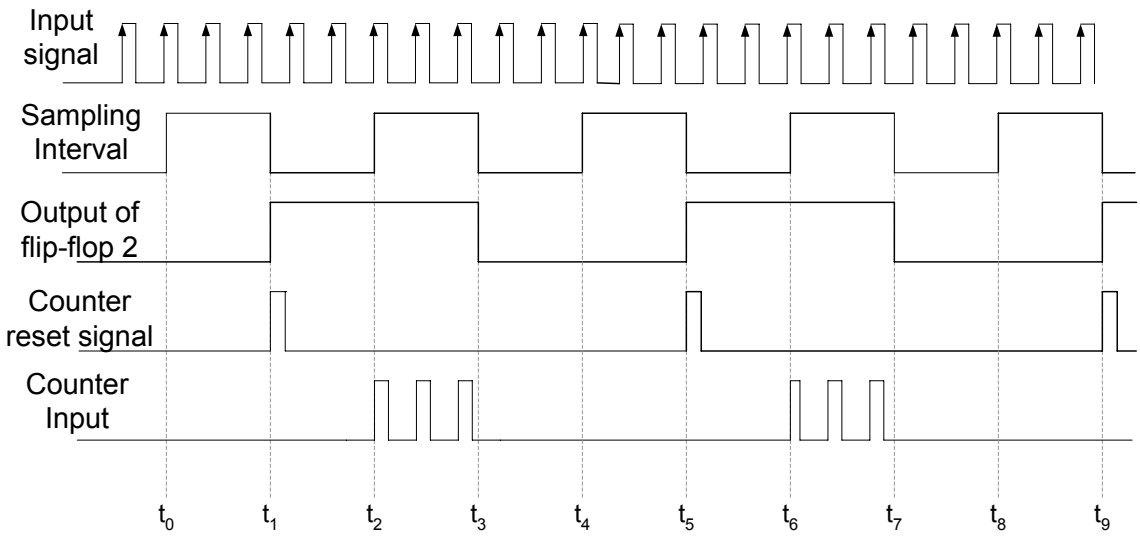


Figure 30.5b Timing diagram of the frequency counter circuit

Design of Synchronous Counters

The counters that have been discussed are binary counters that count in a sequence either upwards or downwards. The count start and end sequence of a counter can also be set arbitrarily and the counter can then count up or down with in the terminal count limits. Counters can also be designed that do not count in a sequence, instead they sequence through a set of predefined arbitrary values. Counters can also be implemented using D flip-flops instead of J-K flip-flops. No formal method of designing Counters has been discussed; however during the study of synchronous counters a general procedure was discussed which helps in the implementation of the counters. The procedure requires listing of the binary counting sequence and then determining the input condition for each flip-flop which promotes a change in their output state. The input conditions are dependent on the previous start outputs of the flip-flops and are implemented by using logic gates. The method does help in implementing counters but it is not a comprehensive method for the design and implementation of different types of counters.

Clocked Synchronous State Machines

The Synchronous Counters are the simplest forms of Clocked Synchronous State Machines. State Machine is a generic name given to Sequential circuits. The Sequential circuits use a clock signal to change from one state to the other and all the flip-flops are connected to a single clock signal, therefore it is a Clocked Synchronous State Machine.

A general Sequential circuit consists of a combinational circuit and a memory element. The memory element is made of a set of n flip-flops all connected to a a common clock. The n flip-flops store 2^n states. The flip-flops change their current state to the next state on each clock transition. The next state is determined by the current state and the external input. The output of the State Machine is determined by the current state and external input. The inputs to the memory which allow the memory to change its state on a clock transition are known as excitation inputs or excitation variables. The present state of the memory is represented by state variables. The state variables and the inputs to the sequential circuit determine the sequential circuit output. The Sequential circuit whose output depends on the current state and the input is known as Mealy Machine. Figure 30.6a. Sequential circuits whose output is determined by the current state only is known as Moore Machine. Figure 30.6b.

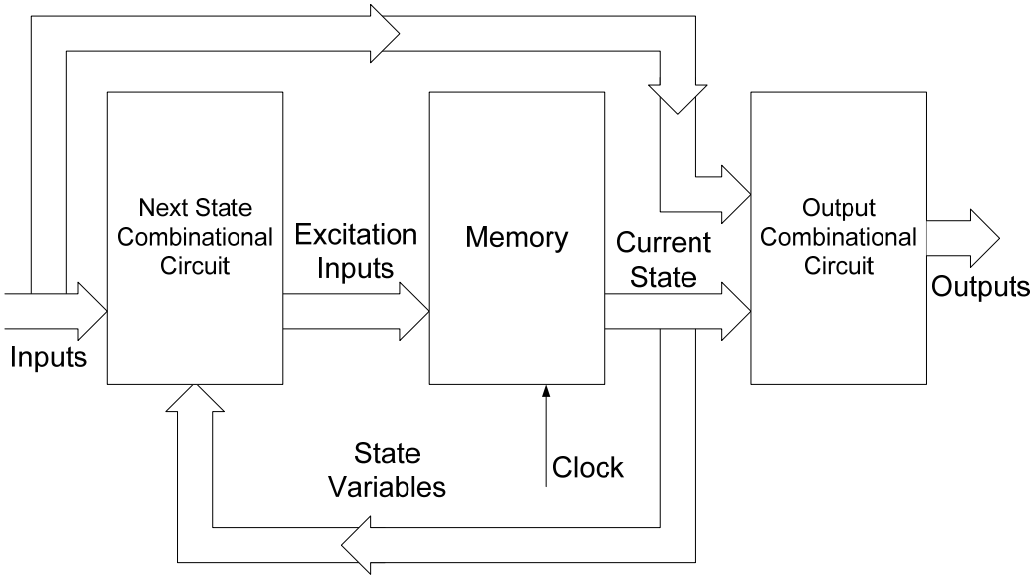


Figure 30.6a Clocked Sequential State Machine (Mealy Machine)

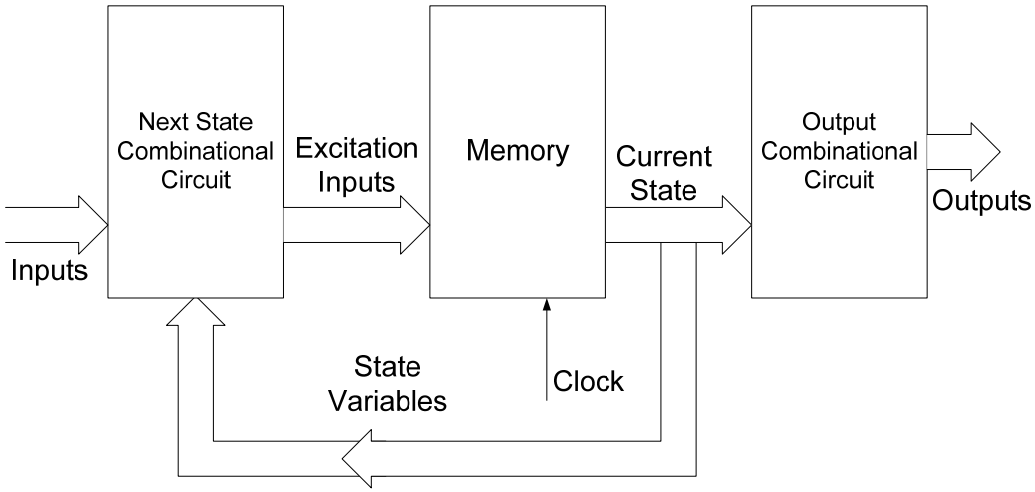


Figure 30.6b Clocked Sequential State Machine (Moore Machine)

Design Procedure

The design and implementation of Synchronous Counters follows an established set of steps and rules which start from defining the state diagram and end at the implementation of State machine.

7. State Diagram

A sequential circuit (state machine) is described by a state diagram, which shows the sequence of state through which the sequential circuit progresses when it is clocked. The state diagram of a 3-bit Synchronous Up-Counter (sequential circuit) is shown in the figure. 30.7

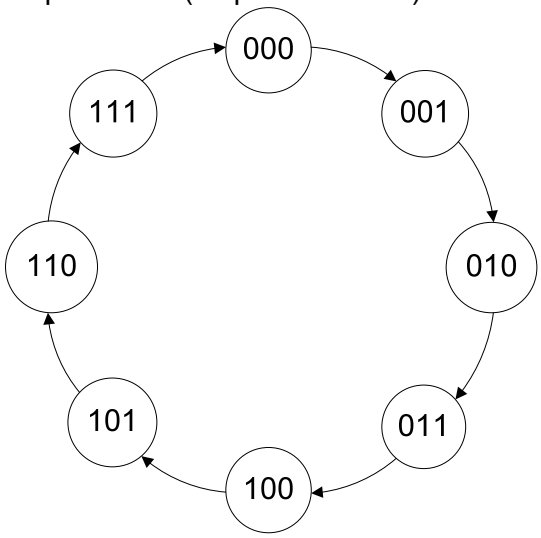


Figure 30.7 State diagram of a 3-bit Up-Counter

Lesson No. 31

2. NEXT-STATE TABLE

Once the state diagram of the sequential circuit is defined, a Next-State Table is derived which lists each present state and the corresponding next state. The next state is the state to which the sequential circuit switches when a clock transition occurs. Table 31.1

Present State			Next State		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
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

Table 31.1 Next-State Table for a 3-bit Up-Counter

3. Flip-flop Transition Table

The Memory element of the Sequential circuit is implemented using flip-flops. The number of flip-flops used is determined by the total number of states. When there is a clock transition at the clock input of the flip-flops they change from their present state to the next state. The Flip-flop transition table lists all the possible flip-flop input combinations which allow the present state to change to the next state on a clock transition. The flip-flop transition table is based on the flip-flop used (D, S-R or J-K). Table 31.2. In the transition table the present state logic 0 changes to next state logic 0, when J-K inputs are 0 and 0 respectively or J-K inputs are 0 and 1 respectively. Thus if input J=0 the next state output is 0. Similarly when J-K inputs are 1 and 1 or 1 and 0 the next state output is set to logic 1. Thus if input J=1 the next state output is 1. Similarly for the other two transition cases K=1 and K=0 sets the next state output to logic 0 and 1 respectively.

Flip-flop Inputs		Output Transitions	
J	K	Q_t	Q_{t+1}
0	x	0	0
1	x	0	1
x	1	1	0
x	0	1	1

Table 31.2 J-K flip-flop Transition table

4. Karnaugh Maps

For each state variable shown in the Next-State table, the change from present state to the next state on a clock transition depends upon the J-K inputs. Table 31.3. Considering the state variable Q_2, J_2 and K_2 inputs set to 0 and x (don't care) allow Q_2 to change from present state 0 to next state 0. Similarly, the state variable Q_0 changes from 1 to 0 when J_0 and K_0

in the Next-State table and the J-K flip-flop transition table. The J-K inputs can be directly mapped to Karnaugh maps. Table 31.4

Present State			Next State			J-K flip-flop inputs					
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0
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

Table 31.3 J-K flip-flop input table

Q_2Q_1/Q_0	0	1
00	0	0
01	0	1
11	x	x
10	x	x

$J_2 = Q_1Q_0$

Q_2Q_1/Q_0	0	1
00	x	x
01	x	x
11	0	1
10	0	0

$K_2 = Q_1Q_0$

Table 31.4a Karnaugh Map for J_2 and K_2 inputs

Q_2Q_1/Q_0	0	1
00	0	1
01	x	X
11	x	X
10	0	1

$J_1 = Q_0$

Q_2Q_1/Q_0	0	1
00	x	x
01	0	1
11	0	1
10	x	x

$K_1 = Q_0$

Table 31.4b Karnaugh Map for J_1 and K_1 inputs

Q_2Q_1/Q_0	0	1
00	1	x
01	1	x
11	1	x
10	1	x

$J_0 = 1$

Q_2Q_1/Q_0	0	1
00	x	1
01	x	1
11	x	1
10	x	1

$K_0 = 1$

5. Logic expressions for Flip-flop Inputs

Simplified expressions for J_2-K_2, J_1-K_1 and J_0-K_0 are directly obtained from the Karnaugh maps. The expressions are shown along with the Karnaugh maps.

6. Sequential Circuit Implementation

The Boolean expressions obtained in the previous step are implemented using logic gates. The sequential circuit implemented is shown in figure 30.8.

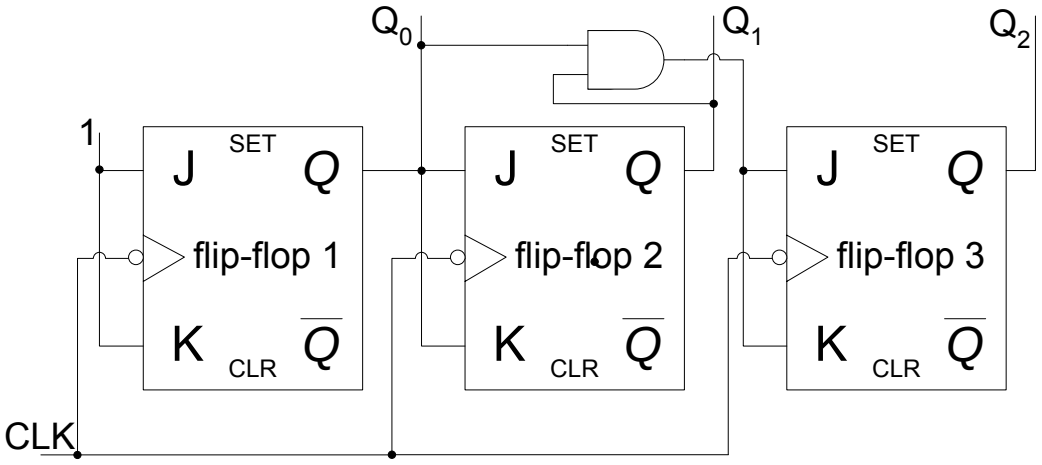


Figure 31.1 Implementation of the Sequential Circuit

The 3-bit up counter can be implemented using S-R flip-flops and D flip-flops. Implementation of the counter using S-R flip-flop requires the use of S-R flip-flop transition table in step 3. The remaining steps follow step 3.

S-R flip-flop based Implementation

Flip-Flop Transition Table

To implement the counter using S-R flip-flops instead of J-K flip-flops, the S-R transition table is used. The S-R flip-flop does not allow S and R inputs to be set to logic 1 and 1 respectively and is considered to be an invalid state. Based on the three set of valid inputs the S-R transition table is shown. Table 31.5

Flip-flop Inputs		Output Transitions	
S	R	Q_t	Q_{t+1}
0	x	0	0
1	0	0	1
0	1	1	0
x	0	1	1

Table 31.5 S-R flip-flop Transition table

Karnaugh Maps

The S-R input table is shown in table 31.6. The Karnaugh maps for the input

Present State			Next State			S-R flip-flop inputs					
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	S_2	R_2	S_1	R_1	S_0	R_0
0	0	0	0	0	1	0	x	0	x	1	0
0	0	1	0	1	0	0	x	1	0	0	1
0	1	0	0	1	1	0	x	x	0	1	0
0	1	1	1	0	0	1	0	0	1	0	1
1	0	0	1	0	1	x	0	0	x	1	0
1	0	1	1	1	0	x	0	1	0	0	1
1	1	0	1	1	1	x	0	x	0	1	0
1	1	1	0	0	0	0	1	0	1	0	1

Table 31.6 S-R flip-flop input table

Q_2Q_1/Q_0	0	1
00	0	0
01	0	1
11	x	0
10	x	x

$S_2 = \overline{Q_2}Q_1Q_0$

Q_2Q_1/Q_0	0	1
00	x	x
01	x	0
11	0	1
10	0	0

$R_2 = Q_2Q_1Q_0$

Table 31.7a Karnaugh Map for S_2 and R_2 inputs

Q_2Q_1/Q_0	0	1
00	0	1
01	x	0
11	x	0
10	0	1

$S_1 = \overline{Q_1}Q_0$

Q_2Q_1/Q_0	0	1
00	x	0
01	0	1
11	0	1
10	x	0

$R_1 = Q_1Q_0$

Table 31.7b Karnaugh Map for S_1 and R_1 inputs

Q_2Q_1/Q_0	0	1
00	1	0
01	1	0
11	1	0
10	1	0

$S_0 = \overline{Q_0}$

Q_2Q_1/Q_0	0	1
00	0	1
01	0	1
11	0	1
10	0	1

$R_0 = Q_0$

Table 31.7c Karnaugh Map for S_0 and R_0 inputs

Logic expressions for Flip-flop Inputs

Simplified expressions for S_2-R_2, S_1-R_1 and S_0-R_0 are directly obtained from the Karnaugh maps. The expressions are shown along with the Karnaugh maps.

Sequential Circuit Implementation

The implementation of the 3-bit synchronous counter based on S-R flip-flops is shown. Figure 31.2

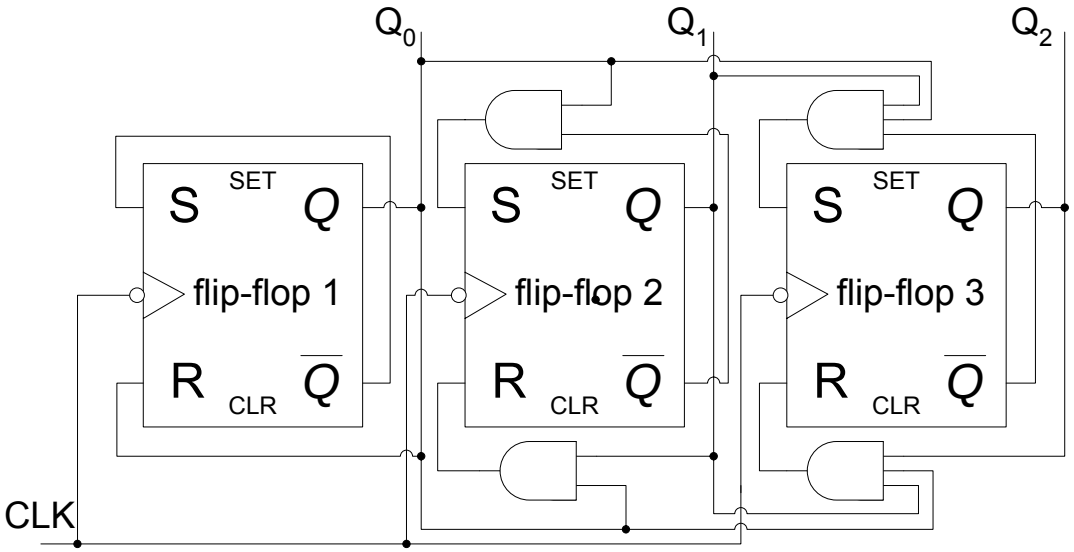


Figure 31.2a S-R flip-flop based implementation of 3-bit Synchronous Counter

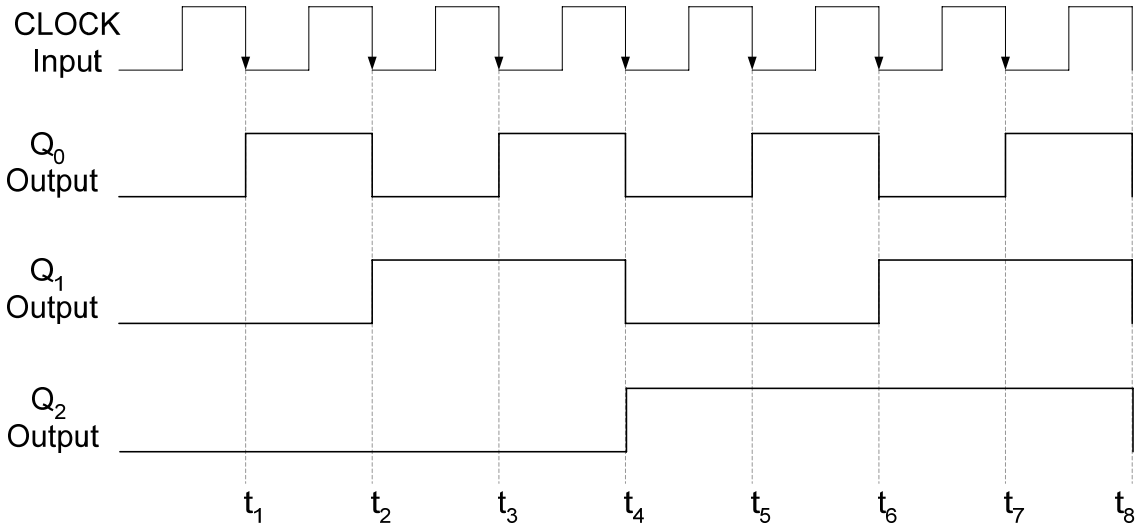


Figure 31.2b Timing diagram of the S-R flip-flop based 3-bit Synchronous Counter

The S-R inputs of the first flip-flop are cross connected to its Q and $\overline{Q}$ outputs. At interval t_1 the Q_0 output is at logic 0, the R input is at logic 0 and S input is at logic 1, thus the flip-flop is set to logic 1. When the Q_0 output is at logic 1, the S and R inputs are at logic 0 and 1 respectively, thus at t_2 the clock transition the flip-flop is reset to 0. At t_1 the S and R inputs of the second-flip-flop are at logic 0 as Q is at logic 0, thus at the clock transition the output of

the second flip-flop remains unchanged. At interval t_2, the S and R inputs of the second flip-flop are set to 1 and 0 respectively, thus it is set to logic 1 on the clock transition. Similarly, at interval t_4 the S-R inputs of the third flip-flop are set to logic 1 and 0 respectively, the flip-flop is set to logic 1 on the clock transition.

© Virtual University of Pakistan

Lesson No. 32

D FLIP-FLOP BASED IMPLEMENTATION

Flip-Flop Transition Table

To implement the counter using D flip-flops instead of J-K flip-flops, the D transition table is used. The D flip-flop only has a single input and the output of the D flip-flop follows the input. The D flip-flop transition table is shown. Table 32.1

Flip-flop Inputs	Output Transitions
D	Q_{t+1}
1	1
0	0

Table 32.1 D flip-flop Transition table

Karnaugh Maps

The D input table is shown in table 32.2. The Karnaugh maps for the input expressions are also derived from the input table.

Present State			Next State			D flip-flop inputs		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	D_2	D_1	D_0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	0
0	1	0	0	1	1	0	1	1
0	1	1	1	0	0	1	0	0
1	0	0	1	0	1	1	0	1
1	0	1	1	1	0	1	1	0
1	1	0	1	1	1	1	1	1
1	1	1	0	0	0	0	0	0

Table 32.2 D flip-flop input table

Q_2Q_1/Q_0	0	1
00	0	0
01	0	1
11	1	0
10	1	1

$D_2 = Q_2 \oplus Q_1Q_0$

Q_2Q_1/Q_0	0	1
00	0	1
01	1	0
11	1	0
10	0	1

$D_1 = Q_0 \oplus Q_1$

Q_2Q_1/Q_0	0	1
00	1	0
01	1	0
11	1	0
10	1	0

$D_0 = \overline{Q_0}$

Table 32.3 Karnaugh Map for D_2, D_1 and D_0 inputs

Logic expressions for Flip-flop Inputs

Simplified expressions for D_2, D_1 and D_0 are obtained from the Karnaugh maps. The expressions are shown along with the Karnaugh maps.

Sequential Circuit Implementation

The first D flip-flop is connected to toggle at each clock transition. The second flip-flop sets its output depending on the D input. The input to the second flip-flop is determined by the expression $D_1 = Q_0 \oplus Q_1$, thus at intervals t_1, t_4, t_5 and t_8 the input D_1 is at logic 1 therefore on the clock transition the output Q_1 is also set to logic 0. At intervals t_2, t_3, t_6 and t_7 the output Q_1 is set to logic 1 as the input D_1 is at logic 1. The input to the second flip-flop is determined by the expression $D_2 = Q_2 \oplus Q_1 Q_0$, thus at intervals t_1, t_2, t_3 and t_8 the output Q_2 is set to 0 as D_2 input is at logic 0. At intervals t_4, t_5, t_6 and t_7 the output Q_2 is set to logic 1 as D_2 input is at logic 1. Figure 32.1

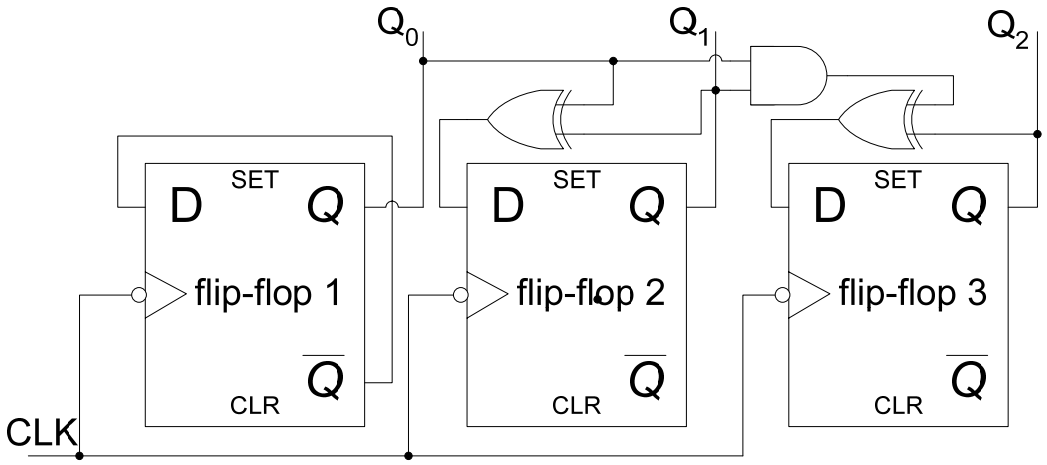


Figure 32.1a D flip-flop based implementation of 3-bit Synchronous Counter

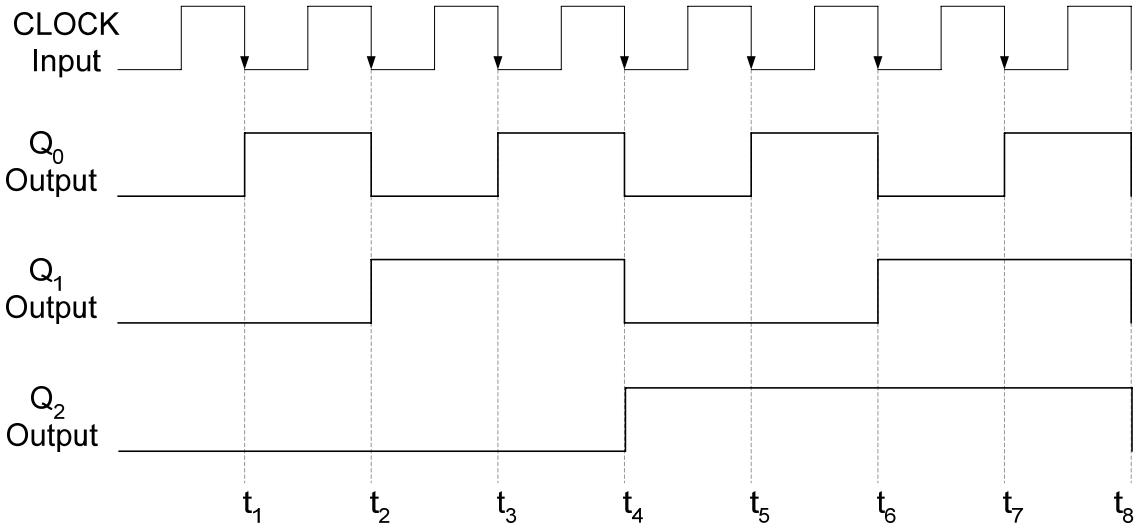


Figure 32.1b Timing diagram of the D flip-flop based 3-bit Synchronous Counter

7. State Diagram

The state diagram of a 3-bit Up/Down Synchronous Counter is shown in the figure. 32.2. X=0 and X =1 indicates that the counter counts up when input X = 0 and it counts down when X =1. X is used as input variable to configure the counter as up or down counter.

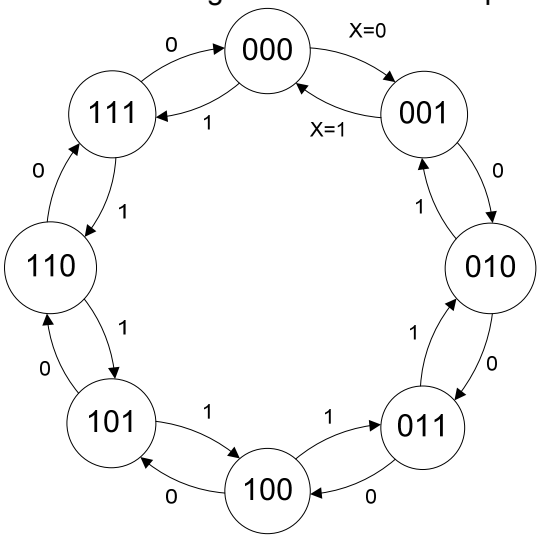


Figure 32.2 State diagram of a 3-bit Up-Counter

8. Next-State Table

The next state is the state to which the sequential circuit switches when a clock transition occurs. Table 32.4. The next state outputs for X=0 and X=1 are shown separately.

Present State			Next State X=0			Next State X=1		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	0	1	1	1	1
0	0	1	0	1	0	0	0	0
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	0	1	0
1	0	0	1	0	1	0	1	1
1	0	1	1	1	0	1	0	0
1	1	0	1	1	1	1	0	1
1	1	1	0	0	0	1	1	0

Table 32.4 Next-State Table for a 3-bit Up-Counter

9. Flip-flop Transition Table

The flip-flop transition table is based on the J-K flip-flop. Table 32.5

Flip-flop Inputs		Output Transitions	
J	K	Q_t	Q_{t+1}
0	x	0	0

x	1	1	0
x	0	1	1

Table 32.5 J-K flip-flop Transition table

10. Karnaugh Maps

The J-K flip-flop inputs when state variables change when X=0 and X=1 are shown in the table 32.6. The J-K inputs can be directly mapped to 4-Variable Karnaugh maps. Table 32.7

Present State			Next State X=0			J-K flip-flop inputs					
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0
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

Table 32.6a J-K flip-flop input table for X=0

Present State			Next State X=1			J-K flip-flop inputs					
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	1	1	1	1	x	1	x	1	x
0	0	1	0	0	0	0	x	0	x	x	1
0	1	0	0	0	1	0	x	x	1	1	x
0	1	1	0	1	0	0	x	x	0	x	1
1	0	0	0	1	1	x	1	1	x	1	x
1	0	1	1	0	0	x	0	0	x	x	1
1	1	0	1	0	1	x	0	x	1	1	x
1	1	1	1	1	0	x	0	x	0	x	1

Table 32.6b J-K flip-flop input table for X=1

Q_2Q_1/Q_0X	00	01	11	10
00	0	1	0	0
01	0	0	0	1
11	x	x	x	x
10	x	x	x	x

$J_0 = 1$

Q_2Q_1/Q_0X	00	01	11	10
00	x	x	x	x
01	x	x	x	x
11	0	0	0	1
10	0	1	0	0

$K_0 = 1$

Table 32.7a Karnaugh Map for J_2 and K_2 inputs

00	0	1	0	1
01	x	x	x	x
11	x	x	x	x
10	0	1	0	1

Q_2Q_1/Q_0X	00	01	11	10
00	x	x	x	x
01	0	1	0	1
11	0	1	0	1
10	x	x	x	x

$$J_1 = Q_0\overline{X} + \overline{Q_0}X \qquad K_1 = Q_0\overline{X} + \overline{Q_0}X$$

Table 32.7b Karnaugh Map for J_1 and K_1 inputs

Q_2Q_1/Q_0X	00	01	11	10
00	1	1	x	x
01	1	1	x	x
11	1	1	x	x
10	1	1	x	x

Q_2Q_1/Q_0X	00	01	11	10
00	x	x	1	1
01	x	x	1	1
11	x	x	1	1
10	x	x	1	1

$$J_2 = Q_1Q_0\overline{X} + \overline{Q_1}\,\overline{Q_0}X \qquad K_2 = Q_1Q_0\overline{X} + \overline{Q_1}\,\overline{Q_0}X$$

Table 32.7c Karnaugh Map for J_0 and K_0 inputs

11. Logic expressions for Flip-flop Inputs

Simplified expressions for J_2-K_2, J_1-K_1 and J_0-K_0 are directly obtained from the Karnaugh maps. The expressions are shown along with the Karnuagh maps.

12. Sequential Circuit Implementation

The Boolean expressions obtained in the previous step are implemented using logic gates. The sequential circuit implemented is shown in figure 32.3

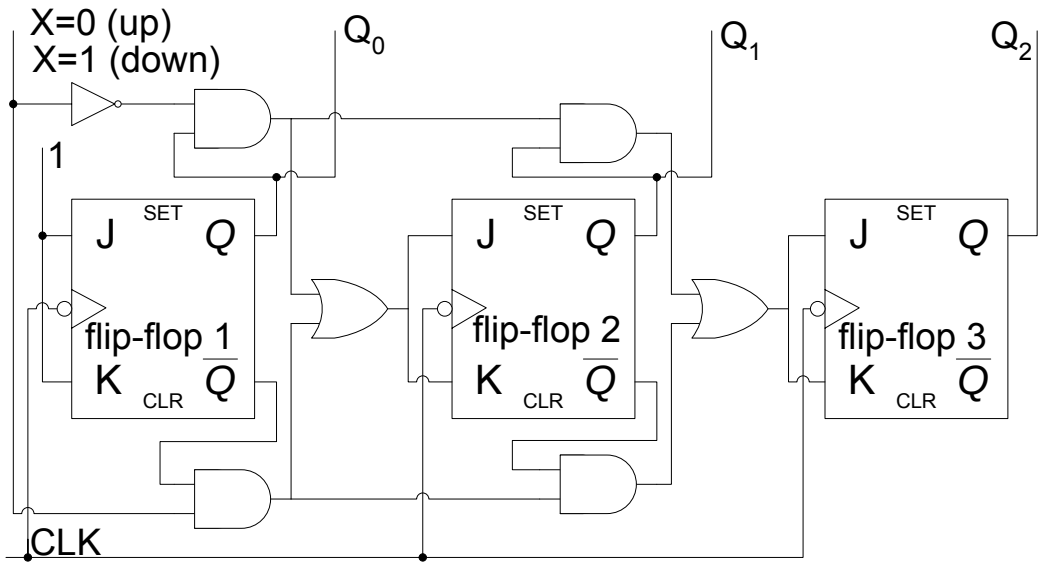


Figure 32.3 Implementation of the Sequential Circuit

State Reduction

A state diagram show the sequence of current and next states through which the state machine sequences. Figure 32.4. The transition from a current state to the next state is determined by current state and the inputs. The outputs of the state machine may also change during the transition from the current state to the next state. The outputs may depend only on the current state (Moore Machine) or a combination of current state and the inputs (Mealy Machine). It is possible that two or more states are equivalent. Two states are considered equivalent if for the same set of inputs the states change to the same next state or equivalent next states and give identical outputs. If equivalent states exist then one of the equivalent state is removed. Reduction in the number of state results in fewer flip-flops and a simpler circuit.

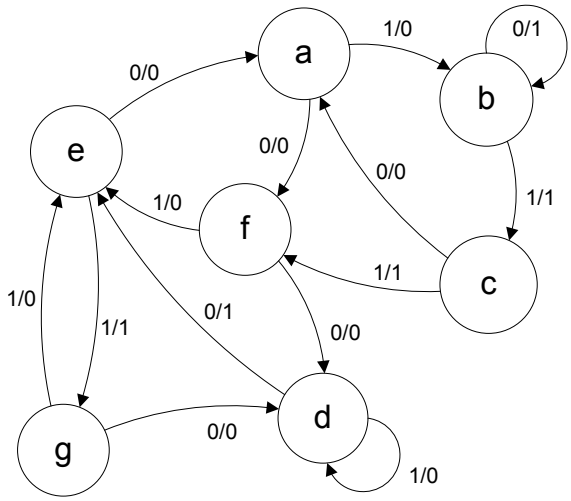


Figure 32.4 State diagram

Reduction in the number of states is possible if one is interested only in the input and output relationship, that is, input and outputs remain unchanged. When external outputs are taken directly from flip-flops, the output must be independent of the number of states before state reduction algorithms are applied. Consider the sequence a, b, c, f, d, d, e, g, e, g, d, e, a, f, d, e, a starting from the initial state a. The inputs and the corresponding outputs are shown in the table. Table 32.8

state	a	b	c	f	d	d	e	g	d	e	a	f	d	e	a
Input	1	1	1	0	1	0	1	0	0	0	0	0	0	0	
Output	0	1	1	0	0	1	1	0	1	0	0	0	1	0	

Table 32.8 The input and output sequence

In the next state table the state ‘f’ is equivalent to state ‘g’ as for each set of inputs states ‘f’ and ‘g’ change to states ‘d’ and ‘e’ respectively. Table 32.9a. Similarly, the outputs also remain identical. Therefore state ‘g’ can be eliminated and in the state table all instances of state ‘g’ are replaced by state ‘f’.

Present State	Next State		Output	
	X=0	X=1	X=0	X=1
a	f	b	0	0
b	b	c	1	1
c	a	f	0	1
d	e	d	1	0
e	a	g	0	1
f	d	e	0	0
g	d	e	0	0

Table 32.9a Next-State table

Present State	Next State		Output	
	X=0	X=1	X=0	X=1
a	f	b	0	0
b	b	c	1	1
c	a	f	0	1
d	e	d	1	0
e	a	f	0	1
f	d	e	0	0

Table 32.9b Next State table, with state ‘g’ eliminated and instances of state ‘g’ replaced by state ‘f’

In the next state table state ‘c’ is equivalent to state ‘e’ as for each input, the current state changes to the same next states. Table 32.9b. The outputs are also identical when changing from the present state to the next state. The state table is simplified by eliminating state e and replacing all instances of state ‘e’ with state ‘c’. table 32.9c. The State diagram represented by the simplified state table is shown. Figure 31.7.

Present State	Next State		Output	
	X=0	X=1	X=0	X=1
a	f	b	0	0
b	b	c	1	1
c	a	f	0	1
d	c	d	1	0
f	d	c	0	0

Table 32.9c Next State table, with state ‘e’ eliminated and instances of state ‘e’ replaced by state ‘c’

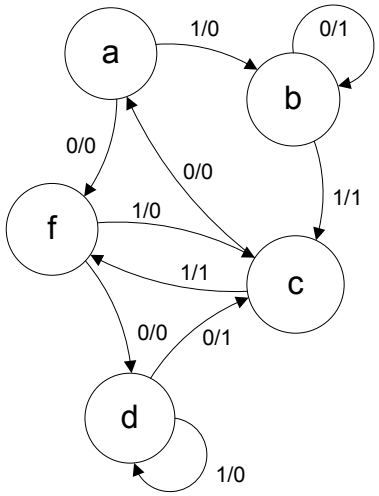


Figure 32.5 Simplified State diagram

Reconsider the initial sequence a, b, c, f, d, d, e, g, e, g, d, e, a, f, d, e, a starting from the initial state a. The inputs and outputs for the state sequence derived from the simplified State diagram are shown in table 32.10.

state	a	b	c	f	d	d	c	f	d	c	a	f	d	c	a
Input	1	1	1	0	1	0	1	0	0	0	0	0	0	0	
Output	0	1	1	0	0	1	1	0	1	0	0	0	1	0	

Table 32.10 The input and output sequence obtained from the simplified state diagram

Elimination of equivalent states results in the reduction in the number of flip-flops. In the example described, the elimination of two states reduces the total number of unique states from seven to five, however the number of flip-flops remain the same which is three. If the number of states had been reduced to four then only two flip-flops would be required.

Lesson No. 33

STATE ASSIGNMENT

Each state in a sequential circuit is identified by a unique combination of binary bits. Unless the output of the sequential is directly taken form the flip-flop outputs such as counters, the states can be selected to allow minimum bit changes when changing from one state to the other. Keeping the bits changes to minimum when changing from one state to the next, results in simpler combinational circuits that determine the next state. Consider the example discussed earlier having states a, b, c, d and f. Since we are interested in only the input and output sequence, therefore it is immaterial how states a, b, c, d and f are uniquely identified. Three possible state assignments are shown. Table 33.1. The Next-State, flip-flop input tables for the three state assignments are shown. Table 33.2, 33.3 and33.4.

State	State Assignment 1	State Assignment 2	State Assignment 3
a	000	001	000
b	001	010	001
c	010	011	011
d	011	100	010
f	100	110	110

Table 33.1 Three possible state assignments for states a, b, c, d and f

Present State	Next State		D flip-flop Inputs						Output	
	X=0	X=1	X=0			X=1			X=0	X=1
000	100	001	1	0	0	0	0	1	0	0
001	001	010	0	0	1	0	1	0	1	1
010	000	100	0	0	0	1	0	0	0	1
011	010	011	0	1	0	0	1	1	1	0
100	011	010	0	1	1	0	1	0	0	0

Table 33.2a Next State flip-flop input table for first State Assignment

Q_2Q_1/Q_0X	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	x	x	x	x
10	0	0	x	x

$D_2 = \overline{Q}_2\overline{Q}_1\overline{Q}_0x + Q_1\overline{Q}_0X$

Q_2Q_1/Q_0X	00	01	11	10
00	0	0	1	0
01	0	0	1	1
11	x	x	x	x
10	1	1	x	x

$D_1 = Q_2 + Q_0X + Q_1Q_0$

Q_2Q_1/Q_0X	00	01	11	10
00	0	1	0	1
01	0	0	1	0
11	x	x	x	x
10	1	0	x	x

$D_0 = Q_2\overline{Q}_0\overline{X} + Q_1Q_0X + \overline{Q}_1Q_0\overline{X} + \overline{Q}_2\overline{Q}_1\overline{Q}_0X$

Table 33.2b Karnaugh Maps and D flip-flop input Boolean expressions for the first State

Present State	Next State		D flip-flop Inputs						Output	
	X=0	X=1	X=0			X=1			X=0	X=1
001	110	010	1	1	0	0	1	0	0	0
010	010	011	0	1	0	0	1	1	1	1
011	001	110	0	0	1	1	1	0	0	1
100	011	100	0	1	1	1	0	0	1	0
110	100	011	1	0	0	0	1	1	0	0

Table 33.3a Next State flip-flop input table for second State Assignment

Q_2Q_1/Q_0X	00	01	11	10
00	x	x	0	1
01	0	0	1	0
11	1	0	x	x
10	0	1	x	x

Q_2Q_1/Q_0X	00	01	11	10
00	x	x	1	1
01	1	1	1	0
11	0	1	x	x
10	1	0	x	x

$$D_2 = \overline{Q_2}\,\overline{Q_1}\,\overline{x} + Q_1Q_0X + Q_2Q_1\overline{X} + Q_2\overline{Q_1}X$$

$$D_1 = \overline{Q_2}\,\overline{Q_0} + \overline{Q_1}Q_0 + Q_1X + \overline{Q_1}\,\overline{X}$$

Q_2Q_1/Q_0X	00	01	11	10
00	x	x	0	0
01	0	1	0	1
11	0	1	x	x
10	1	0	x	x

$$D_0 = Q_2\overline{Q_1}\,\overline{X} + Q_1\overline{Q_0}X + Q_1Q_0\overline{X}$$

Table 33.3b Karnaugh Maps and D flip-flop input Boolean expressions for the second State Assignment

Present State	Next State		D flip-flop Inputs						Output	
	X=0	X=1	X=0			X=1			X=0	X=1
000	110	001	1	1	0	0	0	1	0	0
001	001	011	0	0	1	0	1	1	1	1
011	000	110	0	0	0	1	1	0	0	1
010	011	010	0	1	1	0	1	0	1	0
110	010	011	0	1	0	0	1	1	0	0

Table 33.4a Next State flip-flop input table for third State Assignment

Q_2Q_1/Q_0X	00	01	11	10
00	1	0	0	0
01	0	0	1	0
11	0	0	x	x
10	x	x	x	x

$$D_2 = \overline{Q}_1\overline{Q}_0\overline{x} + Q_1Q_0X$$

Q_2Q_1/Q_0X	00	01	11	10
00	1	0	1	0
01	1	1	1	0
11	1	1	x	x
10	x	x	x	x

$$D_1 = \overline{Q}_0\overline{x} + Q_0X + Q_1\overline{Q}_0$$

Q_2Q_1/Q_0X	00	01	11	10
00	0	1	1	1
01	1	0	0	0
11	0	1	x	x
10	x	x	x	x

$$D_0 = \overline{Q}_2Q_1\overline{Q}_0\overline{x} + Q_2X + \overline{Q}_1Q_0 + \overline{Q}_1X$$

Table 33.4b Karnaugh Maps and D flip-flop input Boolean expressions for the third State Assignment

The third State Assignment is shown to have simpler input Boolean expressions leading to a simpler combinational circuit. Generally, the selection of State Assignment is based on the following guidelines.

- Choose an initial coded state into which the state machine (sequential circuit) can easily be forced to reset (000 or 111)
- Minimize the State Variables that change on each transition
- Maximize the number of state variables that don't change in a group of related states
- If there are unused states, then choose the best state variable combinations to achieve the first three goals.

Moore Machine

State Diagram

The state diagram of a Moore Machine is shown. Figure 33.1. The Clocked Synchronous Sequential Circuit has six states. On each clock transition the machine sequences through the states 011, 111, 001, 010, 100 and 110. The outputs of the flip-flops represent the sequential circuit output.

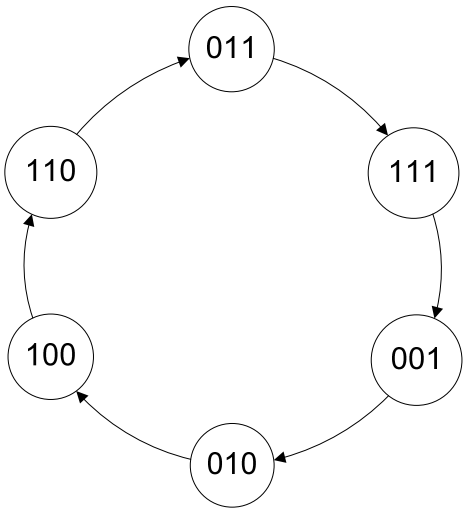


Figure 33.1 State diagram of a Moore Machine

Next-State Table

The Next-State table is derived from the State diagram. The present and the corresponding next states to which the sequential circuit changes at each clock transition are shown. Table 33.5

Karnaugh Maps

The flip-flop input table based on J-K flip-flops is shown. Table 33.6. J-K flip-flop transition table is used to determine the J-K flip-flop inputs. The Karnaugh maps for each of the three J and K inputs of the three J-K flip-flops are shown along with the Boolean expressions. Table 33.7

Present State			Next State		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	1	1	1	1	1
1	1	1	0	0	1
0	0	1	0	1	0
0	1	0	1	0	0
1	0	0	1	1	0
1	1	0	0	1	1

Table 33.5 Next-State table of the Moore Machine

Present State			Next State			J-K flip-flop inputs					
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0
0	1	1	1	1	1	1	x	x	0	x	0
1	1	1	0	0	1	x	1	x	1	x	0
0	0	1	0	1	0	0	x	1	x	x	1
0	1	0	1	0	0	1	x	x	1	0	x
1	0	0	1	1	0	x	0	1	x	0	x
1	1	0	0	1	1	x	1	x	0	1	x

Q_2Q_1/Q_0	0	1
00	x	0
01	1	1
11	x	x
10	x	x

$J_2 = Q_1$

Q_2Q_1/Q_0	0	1
00	x	x
01	x	x
11	1	1
10	0	x

$K_2 = Q_1$

Table 33.7a Karnaugh Map for J_2 and K_2 inputs

Q_2Q_1/Q_0	0	1
00	x	1
01	x	x
11	x	x
10	1	x

$J_1 = 1$

Q_2Q_1/Q_0	0	1
00	x	x
01	1	0
11	0	1
10	x	x

$K_1 = \overline{Q}_2\overline{Q}_0 + Q_2Q_0$

Table 33.7b Karnaugh Map for J_1 and K_1 inputs

Q_2Q_1/Q_0	0	1
00	x	x
01	0	x
11	1	x
10	0	x

$J_0 = Q_2Q_1$

Q_2Q_1/Q_0	0	1
00	x	1
01	x	0
11	x	0
10	x	x

$K_0 = \overline{Q}_1$

Table 33.7c Karnaugh Map for J_0 and K_0 inputs

Implementation

The circuit and timing diagram of the State Machine is shown. Figure 33.2. The sequential circuit is assumed to be reset to state 011. At interval t_1, J-K input of the first flip-flop is set at 0 and 0 respectively; therefore at the clock transition the output of the first flip-flop remains unchanged. The J input of the second flip-flop is permanently connected to logic 1, the K input is set at logic 0, therefore the output of the second flip-flop is set to logic 1 at the clock transition t_1. The J-K input of the third flip-flop is set to logic 1, at clock transition t_1 the output of the flip-flop changes to logic 1. The operation of the sequential circuit can be similarly verified for intervals t_2 to t_7.

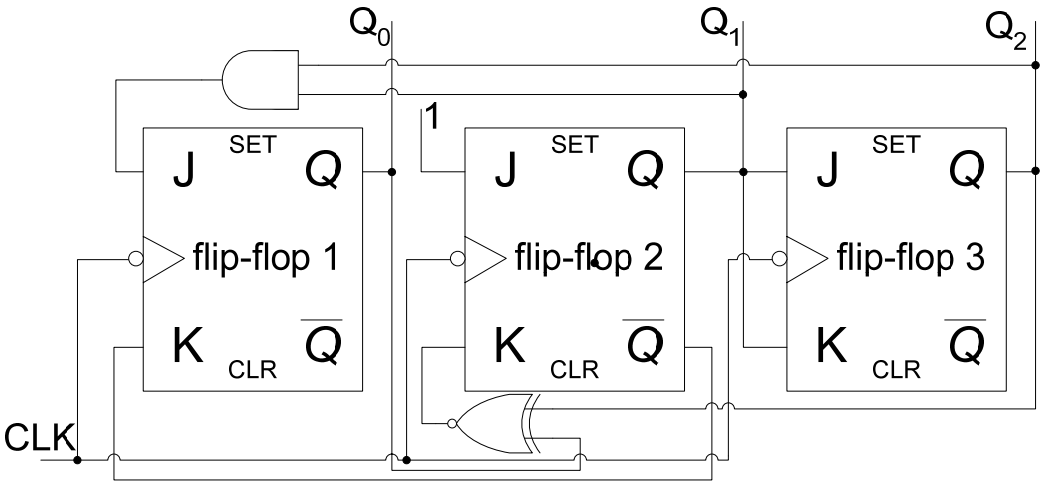


Figure 33.2a Implementation of the Moore Machine

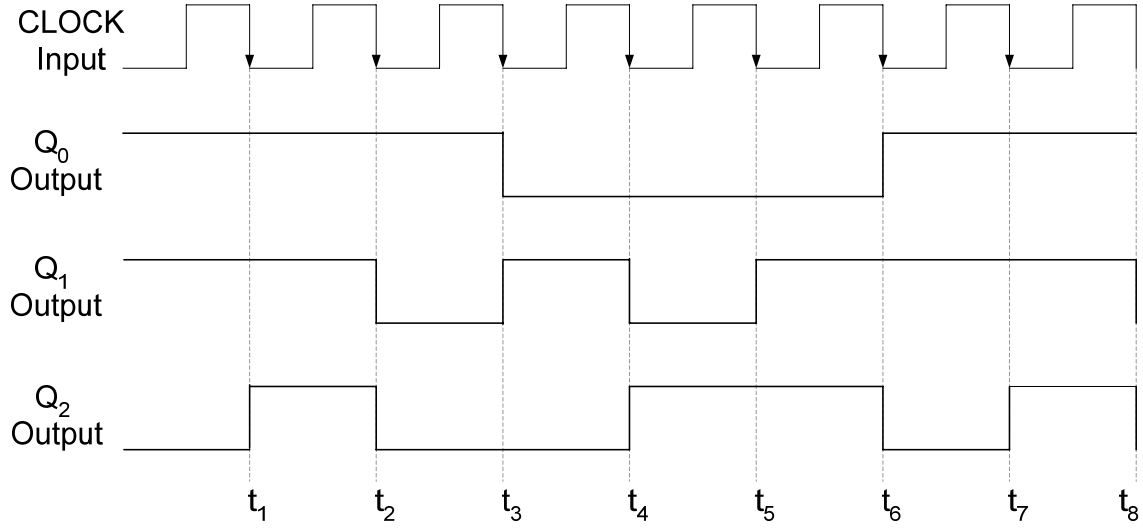


Figure 33.2b Timing diagram of the Moore Machine

Mealy Machine
State Diagram

The sequential circuit represented earlier as a Moore Machine is described as a Mealy Machine. Figure 33.3. The output of a Mealy machine depends upon the present state at the inputs. The state diagram shows the six states. When the input is 1, the machine switches from its present state to the next. If the input is 0, the machine remains in its present state. The outputs of the machine when it switches to the next state or it remains the present state are shown with the directed arrows. For, example at state ‘a’, when the input is 1 the machine changes to the next state and the output is set to 111. When the input is set at 0, the machine remains in its current state with outputs 011.

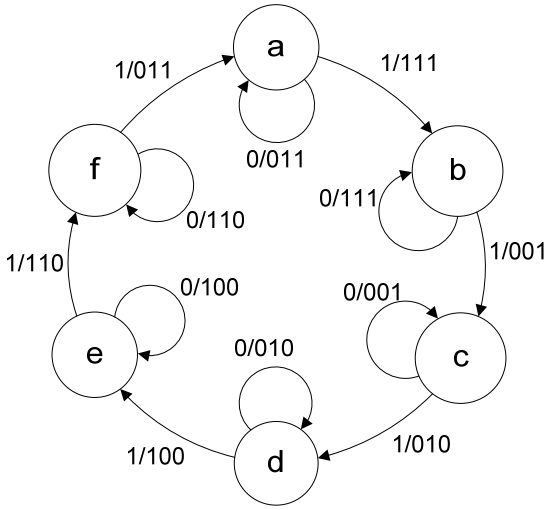


Figure 33.3 State diagram of a Mealy Machine

Next-State Table

The Next-State table for the Mealy Machine is shown. Table 33.8. The Next-State table is directly derived from the State diagram. The present state, and the corresponding next state when the input X=0 and X=1 are shown in separate columns respectively. Similarly, the Moore Machine outputs are also shown for each present state when the inputs are X=0 and X=1 respectively.

Present State	Next State		Output	
	X=0	X=1	X=0	X=1
a	a	b	011	111
b	b	c	111	001
c	c	d	001	010
d	d	e	010	100
e	e	f	100	110
f	f	a	110	011

Table 33.8 Next-State table of a Mealy Machine

State Assignments

Based on the guidelines for State Assignment, States are assigned keeping the bit changes to minimum. The corresponding next states for input X=0 and X=1 are also shown. Table 33.9.

Present State	Next State	
	X=0	X=1
000	000	001
001	001	011
011	011	010
010	010	110
110	110	100
100	100	000

Karnaugh Maps

The J-K flip-flop input tables for the Mealy Machine are shown. Table 33.10. The J-K inputs for the three flip-flops are based on the J-K flip-flop transition tables.

Present State			Next State X=0			J-K flip-flop inputs X=0						Output X=0		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0	O_2	O_1	O_0
0	0	0	0	0	0	0	x	0	x	0	x	0	1	1
0	0	1	0	0	1	0	x	0	x	x	0	1	1	1
0	1	1	0	1	1	0	x	x	0	x	0	0	0	1
0	1	0	0	1	0	0	x	x	0	0	x	0	1	0
1	1	0	1	1	0	x	0	x	0	0	x	1	0	0
1	0	0	1	0	0	x	0	0	x	0	x	1	1	0

Table 33.10a J-K flip-flop input table for the Moore Machine (X=0)

Present State			Next State X=1			J-K flip-flop inputs X=1						Output X=1		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0	O_2	O_1	O_0
0	0	0	0	0	1	0	x	0	x	1	x	1	1	1
0	0	1	0	1	1	0	x	1	x	x	0	0	0	1
0	1	1	0	1	0	0	x	x	0	x	1	0	1	0
0	1	0	1	1	0	1	x	x	0	0	x	1	0	0
1	1	0	1	0	0	x	0	x	1	0	x	1	1	0
1	0	0	0	0	0	x	1	0	x	0	x	0	1	1

Table 33.10b J-K flip-flop input table for the Moore Machine (X=1)

The Karnaugh maps for the three sets of J-K inputs and the three outputs are shown. The Boolean expressions are written along with the Karnuagh maps. Table 33.11

Q_2Q_1/Q_0X	00	01	11	10
00	0	0	0	0
01	0	1	0	0
11	x	x	x	x
10	x	x	x	x

$J_2 = Q_1\overline{Q}_0X$

Q_2Q_1/Q_0X	00	01	11	10
00	x	x	x	x
01	x	x	x	x
11	0	0	x	x
10	0	1	x	x

$K_2 = \overline{Q}_1X$

Table 33.11a Karnaugh Map for J_2 and K_2 inputs

Q_2Q_1/Q_0X	00	01	11	10
00	0	0	1	0
01	x	x	x	x
11	x	x	x	x
10	0	0	x	x

$J_1 = Q_0X$

Q_2Q_1/Q_0X	00	01	11	10
00	x	x	x	x
01	0	0	0	0
11	0	1	x	x
10	x	x	x	x

$K_1 = Q_2X$

Q_2Q_1/Q_0X	00	01	11	10
00	0	1	x	x
01	0	0	x	x
11	0	0	x	x
10	0	0	x	x

Q_2Q_1/Q_0X	00	01	11	10
00	x	x	0	0
01	x	x	1	0
11	x	x	x	x
10	x	x	x	x

$$J_0 = \overline{Q}_2\overline{Q}_1X \qquad K_0 = Q_1X$$

Table 33.11c Karnaugh Map for J_0 and K_0 inputs

Q_2Q_1/Q_0X	00	01	11	10
00	0	1	0	1
01	0	1	0	0
11	1	1	x	x
10	1	0	x	x

Q_2Q_1/Q_0X	00	01	11	10
00	1	1	0	1
01	1	0	1	0
11	0	1	x	x
10	1	1	x	x

$$O_2 = Q_2\overline{X} + Q_2Q_1 + \overline{Q}_2\overline{Q}_0X + \overline{Q}_1Q_0\overline{X}$$
$$O_1 = Q_1Q_0X + \overline{Q}_2\overline{Q}_0\overline{X} + Q_2X + \overline{Q}_1\overline{Q}_0 + \overline{Q}_1\overline{X}$$

Q_2Q_1/Q_0X	00	01	11	10
00	1	1	1	1
01	0	0	0	1
11	0	0	x	x
10	0	1	x	x

$$O_0 = \overline{Q}_1X + \overline{Q}_2\overline{Q}_1 + Q_0\overline{X}$$

Table 33.11d Karnaugh Map for O_2, O_1 and O_0 outputs

Implementation

The implementation of the Mealy Machine is shown. Figure 33.4. The circuit shows only the part of the circuit that allows the Mealy Machine to switch from its current state to the next state. The operation of the machine can be verified with the help of the timing diagram.

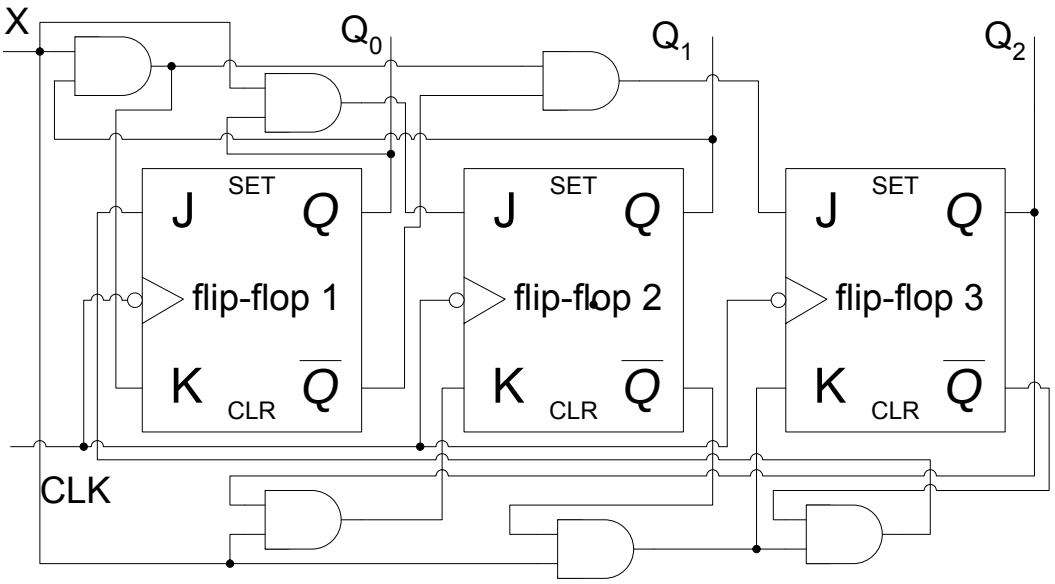


Figure 33.4a Implementation of the Mealy Machine

The machine is assumed to be reset to state 'a' 000. At interval t_1, J-K inputs of the first flip-flop are set at logic 1 and 0 respectively, therefore at the clock transition the output of the first flip-flop switches from 0 to 1. The J-K inputs of the second flip-flop are set at logic at logic 0 and 0 respectively, thus the output state of the second flip-flop remains unchanged at the clock transition. The J-K inputs of the third flip-flop are set to logic 0 and 1 respectively, thus at clock transition t_1 the flip-flop is reset to logic 0. Transition at intervals t_2 to t_7 can similarly be verified.

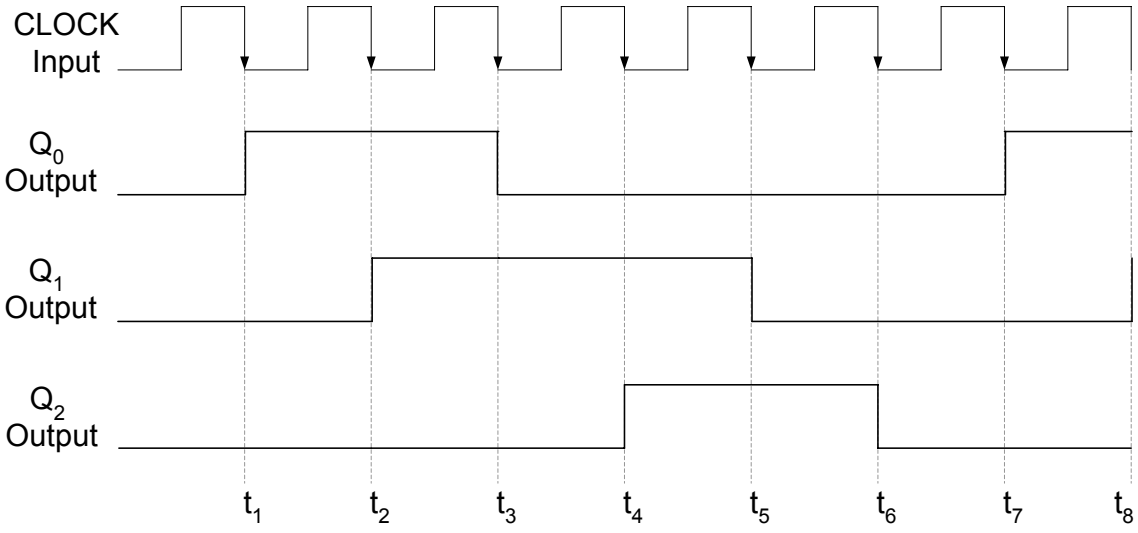


Figure 33.4b Timing diagram of the Mealy Machine

The circuit which implements the Mealy Machine outputs has not been shown. The machine outputs are implemented through the three Boolean expressions for outputs O_2, O_1 and O_0 respectively. At interval t_1, before the clock transition, Q_0, Q_1 and Q_2 outputs are set at

O_2, O_1 and O_0 is 0, 1 and 1 respectively. At the clock transition t_1 when the X input is set to 1, the output O_2, O_1 and O_0 is set to 1, 1 and 1. The outputs for all six states ‘a’, ‘b’, ‘c’, ‘d’, ‘e’ and ‘f’ can similarly be verified.

Lesson No. 34

SHIFT REGISTERS

The need to storage binary data was discussed earlier. In digital circuits multi-bit data has to be stored temporarily until it is processed. A flip-flop is able to store a single binary bit of information. Multiple bits of data are stored by using multiple flip-flops which have their clock inputs connected together. Thus, by activating the clock signal multiple-bits of data are stored.

Technically, a register performs two basic functions. It stores data and it moves or shifts data. The shifting of data involves shifting of bits from one flip-flop to the other within the register or moving data in and out of the register. The shift operation of the binary data is carried out by applying clock signals. Several different kinds of shift operations can be identified. The different shift operations are described using a 4-bit shift register.

1. Serial In/Shift Right/Serial Out Operation

Data is shifted in the right-hand direction one bit at a time with each transition of the clock signal. Figure 34.1. The data enters the shift register serially from the left hand side and after four clock transitions the 4-bit register has 4-bits of data. The data is shifted out serially one bit at a time from the right hand side of the register if clock signals are continuously applied. Thus after 8 clock signals the 4-bit data is completely shifted out of the shift register.

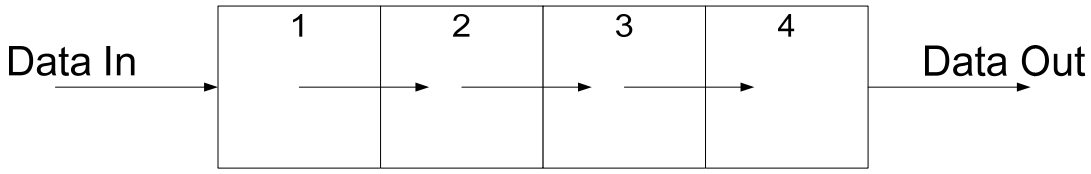


Figure 34.1 Serial In/Serial Right/Serial Out Operation

2. Serial In/Shift Left/Serial Out Operation

Data is shifted in the left-hand direction one bit at a time with each transition of the clock signal. Figure 34.2. The data enters the shift register serially from the right hand side and after four clock transitions the 4-bit register has 4-bits of data. The data is shifted out serially one bit at a time from the left hand side of the register if clock signals are continuously applied. Thus after 8 clock signals the 4-bit data is completely shifted out of the shift register.

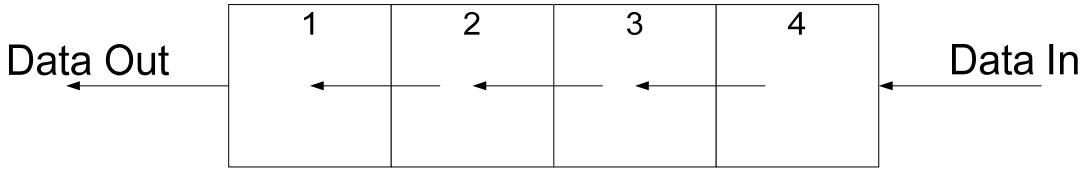


Figure 34.2 Serial In/Serial Left/Serial Out Operation

The Serial Shift register has been discussed earlier, implemented using J-K flip-flops. Serial shift registers can be implemented using any type of flip-flops. A serial shift register implemented using D flip-flops with the serial data applied at the D input of the first flip-flop and serial data out obtained at the Q output of the last flip-flop is shown. Figure 34.3. At each clock transition 1–bit of serial data is shifted in and at the same instant 1-bit of serial data is shifted out. For a 4-bit shift register, 8 clock transitions are required to shift in 4-bit data and

completely shift out the 4-bit data. As the data is shifted out 1-bit at a time, a logic 0 value is usually shifted in to fill up the vacant bits in the shift register.

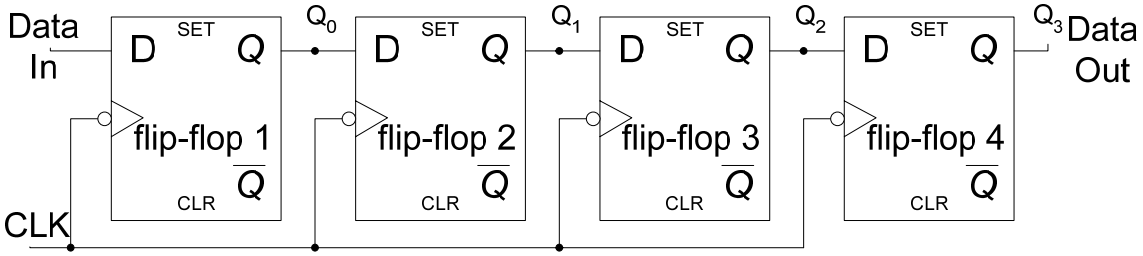


Figure 34.3a Serial In/Shift Right/Serial Out Register

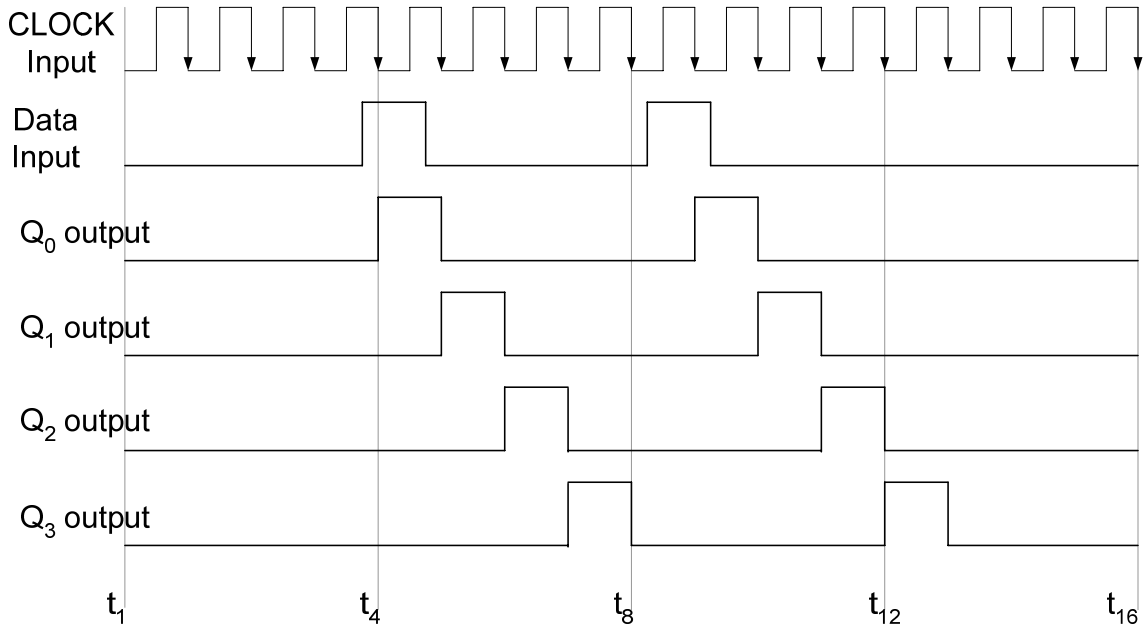


Figure 34.3b Timing diagram of a Serial In/Shift Right/Serial Out Register

The shift left and shift right shift registers are identical in their working. They are connected differently for shift left and shift right operations. Bidirectional Shift Registers are available which allow data to be shifted left or right. Figure 34.4. The 4-bit register is configured to shift left or right by setting the $RIGHT/\overline{LEFT}$ signal to logic high or low respectively. When the register is configured to shift right, the AND gates marked 1 are enabled. The input of the first flip-flop is connected to the serial Input, the inputs of the next three flip-flops are connected to the Q outputs of the previous flip-flops. Thus on a clock transition data is shifted 1-bit towards the right. The serial data is shifted out of the register through output Q_3. When the register is configured to shift left the AND gates marked 2 are enabled, connecting the Q outputs of the flip-flop on the right hand side to the D input of the flip-flop on the left hand side. Thus on each clock transition data is shifted 1-bit towards left. Serial date out is available through the Q_0 output. Serial data is input through the Serial Data in line which is connected to the fourth AND gate marked 2 on the extreme right hand side.

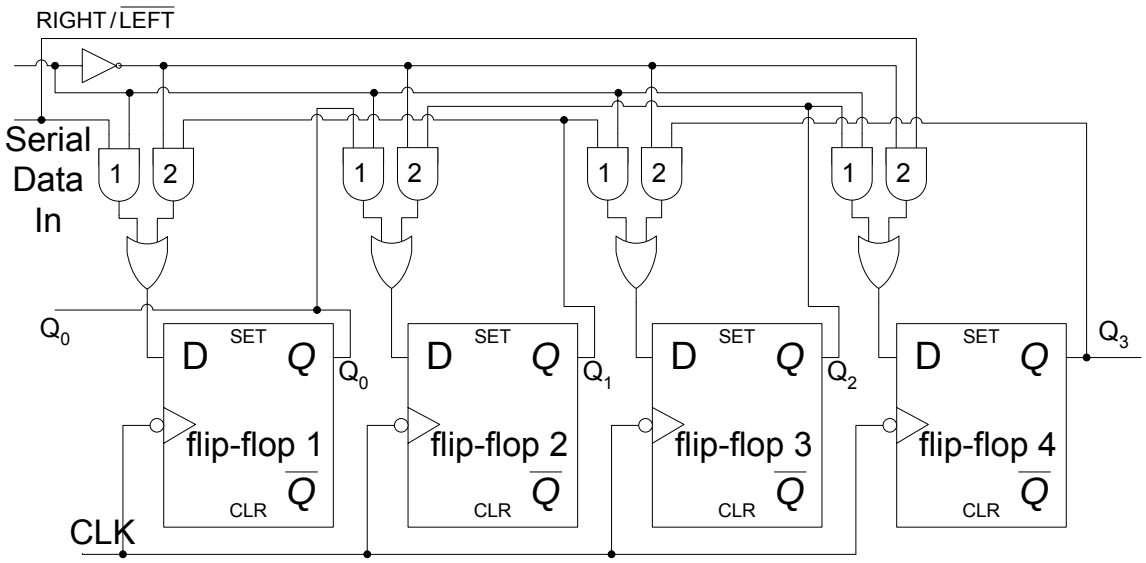


Figure 34.4a Bi-directional, 4-bit Shift register

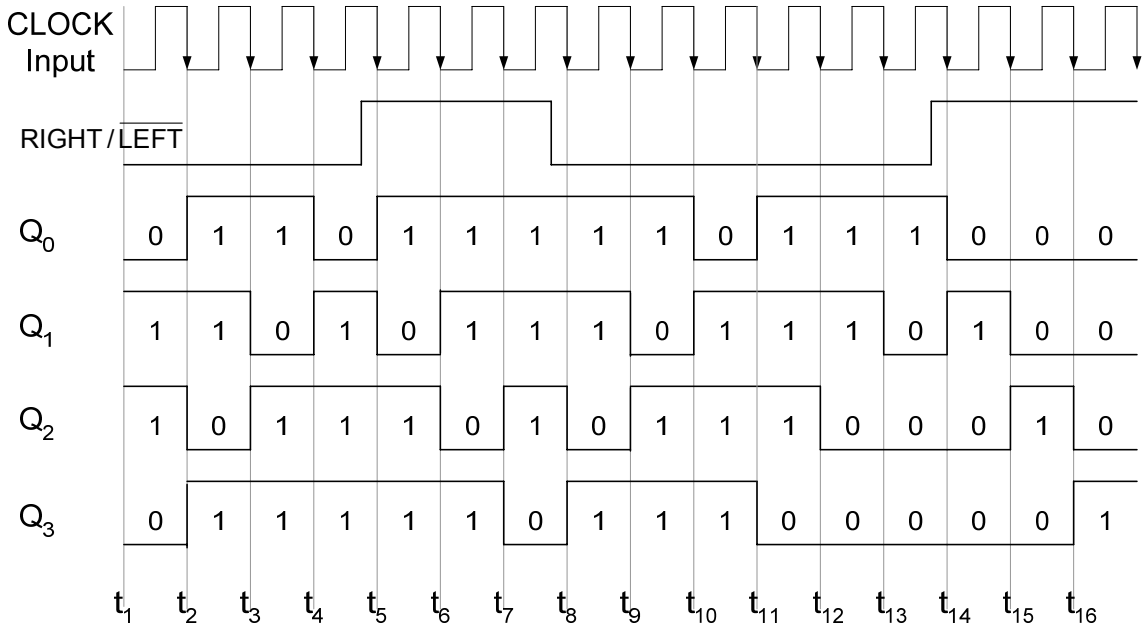


Figure 34.4b Timing diagram of a Bi-directional, 4-bit Shift register

The timing diagram shows the operation the Bi-directional shift register which initially shifts data towards the left. At interval t_5, the registered is configured to shift right and at t_8 towards left and again towards the right at interval t_{14}. A logic 1 is applied at the Serial data input from intervals t_1 to t_{10}. At interval t_{11} and onwards a logic 0 is applied at the Serial data input.

3. Serial In/Parallel Out Operation

Data is shifted in the left-hand direction one bit at a time with each transition of the clock signal. The data enters the shift register serially from the right hand side and after four clock transitions the 4-bit register has 4-bits of data. The data is shifted out in parallel by the

application of a single clock signal. The shift register has 4 parallel outputs. The circuit diagram of the Serial In/Parallel Out register is shown. Figure 34.5.

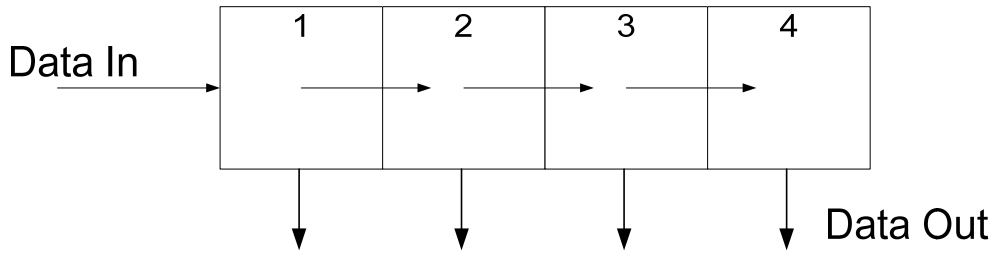


Figure 34.5 Serial In/Parallel Out Operation

The 74HC164 is an MSI 8-bit Serial In/Parallel Out Shift Register. The Shift register has 8 parallel Outputs, an Asynchronous Active-low CLR input which clears the shift register. The shift register is triggered on the positive clock transition. The Serial data is applied through inputs A and B. Input pins A and B are internally connected through an internal NAND gate. The two pins act as a data input and shift register enable inputs. Serial data is applied at either input A or B. The other input when set to logic high enables the shift operation. The Figure 34.6

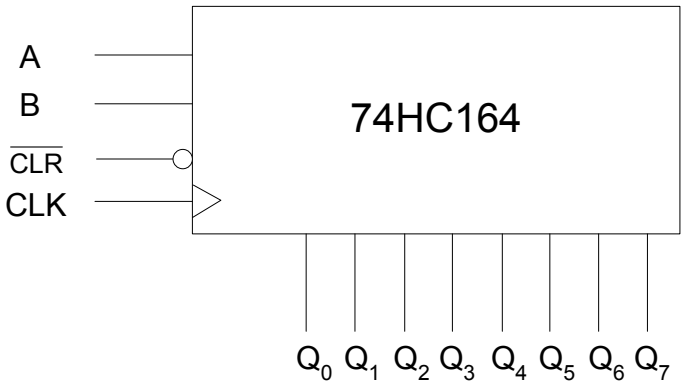


Figure 34.6a 74HC164, 8-bit Serial In/Parallel Out Shift Register

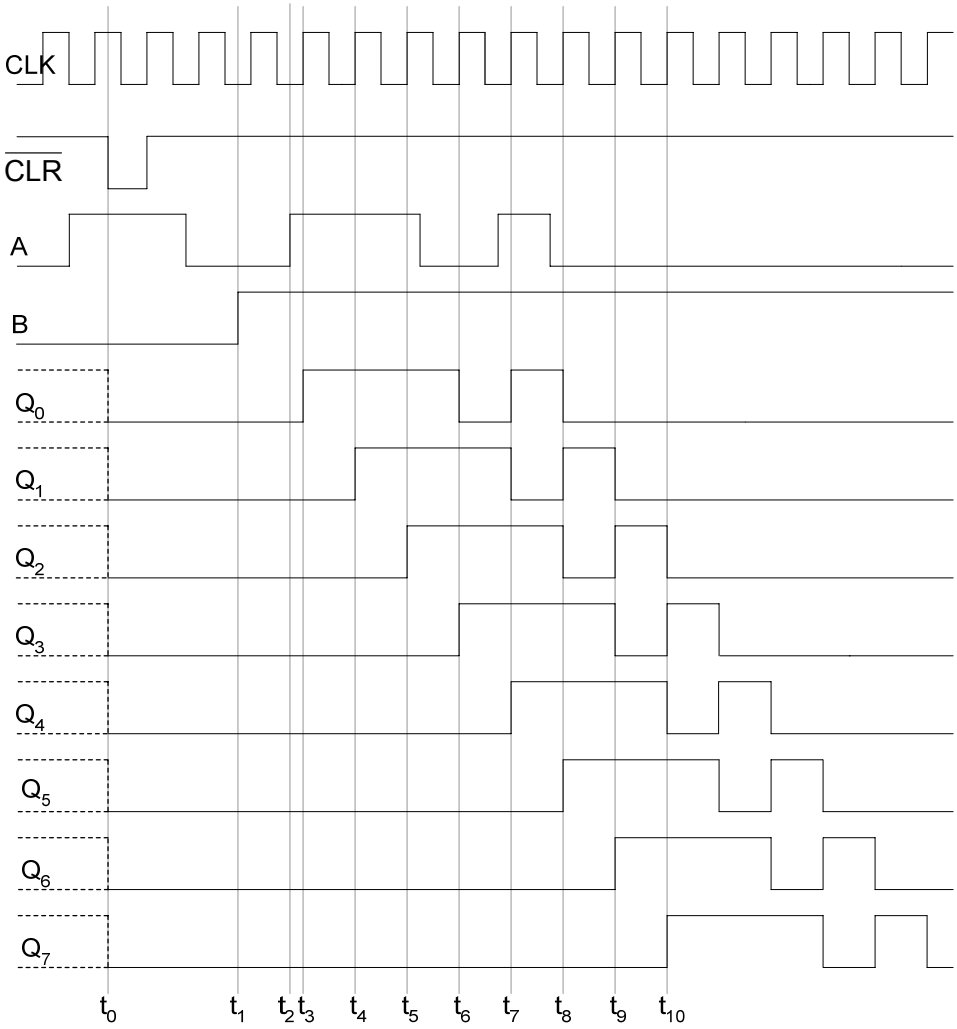


Figure 34.6b Timing diagram of a 74HC164, 8-bit Serial In/Parallel Out Shift Register

In the timing diagram, the register is cleared asynchronously by activating the active-low CLR input at interval t_0. The serial data is applied at input A of the register before interval t_0. However, the register is enabled to perform shift operation at interval t_1, when input B is set to logic high. At interval t_2, there is a low to high transition in the serial data input which is latch by the first flip-flop at the positive clock transition at interval t_3. AT each positive clock transition from interval t_4 to t_{10} the data is shifted right by 1-bit.

4. Parallel In/Serial Out Operation

The register has parallel inputs, data bits are loaded into the register in parallel by activating a load signal. The data is shifted out serially by application of clock signals. Thus in a 4-bit shift register, after 4 clock signals the 4-bit data is completely shifted out of the shift register. Figure 34.7

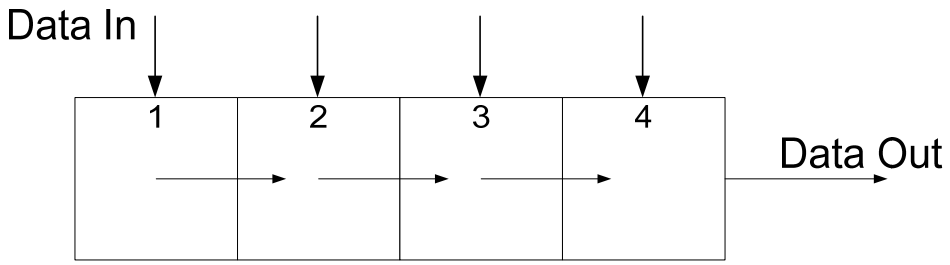


Figure 34.7 Parallel In/Serial Out Operation

The internal circuit of a 4-bit Parallel In/Serial Out Shift register is shown. Figure 34.8. The 4-bit data is initially loaded in Parallel into the shift register by setting the $SHIFT/\overline{LOAD}$ input to logic low. The AND gates marked 2 are enabled allowing data to be applied at the inputs of the respective D flip-flops. On a positive clock transition the data is latched by the respective flip-flops. To shift the data, the $SHIFT/\overline{LOAD}$ is set to logic high which enables AND gates marked 1 connecting the Q outputs of the each flip-flop connected to the D input of the next flip-flop.

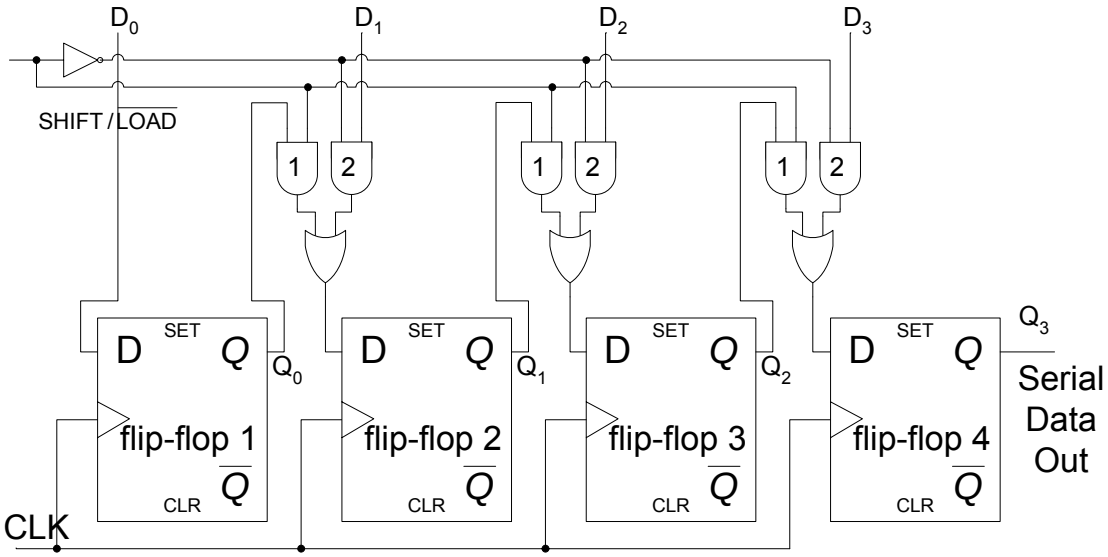


Figure 34.8 4-bit Parallel In/Serial Out Shift register

The 74HC165 is an 8-bit Parallel In/Serial Out register which can also work as an 8-bit Serial In/Serial Out register. The Parallel Data is loaded asynchronously by using the Asynchronous Set/Clear Inputs. After loading the parallel 8-bti data, the serial shift operation is carried out by enabling the clock signal. The CLK and CLK INH signal are internally connected through an OR gate to the clock inputs of the eight flip-flops. The clock signal is enabled by setting the CLK INH signal to logic low. Figure 34.9.

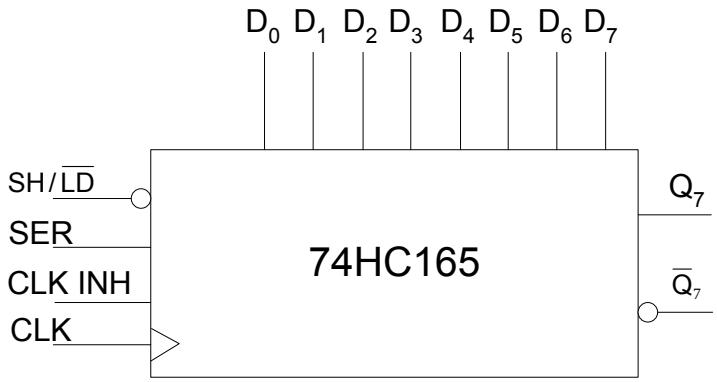


Figure 34.9 74HC165, 8-bit Parallel In/Serial Out Shift Register

5. Parallel In/Parallel Out Operation

The register has parallel inputs and parallel outputs. Data is entered in parallel by applying a single clock pulse. Data is latched by the flip-flops on the clock transition and is available in parallel form at the flip-flop outputs. Figure 34.10. The internal circuit of 4-bit Parallel In/Parallel Out Register is shown. Figure 34.11. The Parallel In/Parallel Out register stores Parallel data and usually does not allows any shift operations.

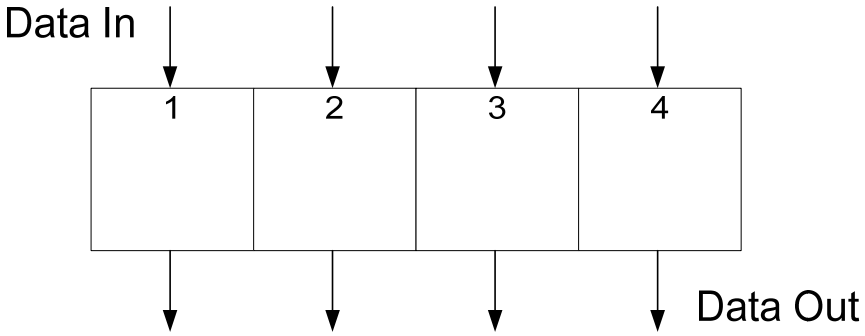


Figure 34.10 Parallel In/Parallel Out Operation

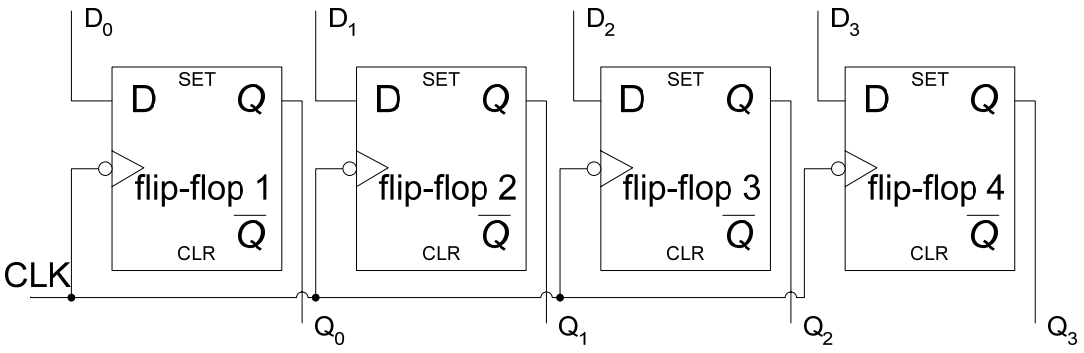


Figure 34.11 A D-flip-flop based 4-bit Parallel In/Parallel Out Register

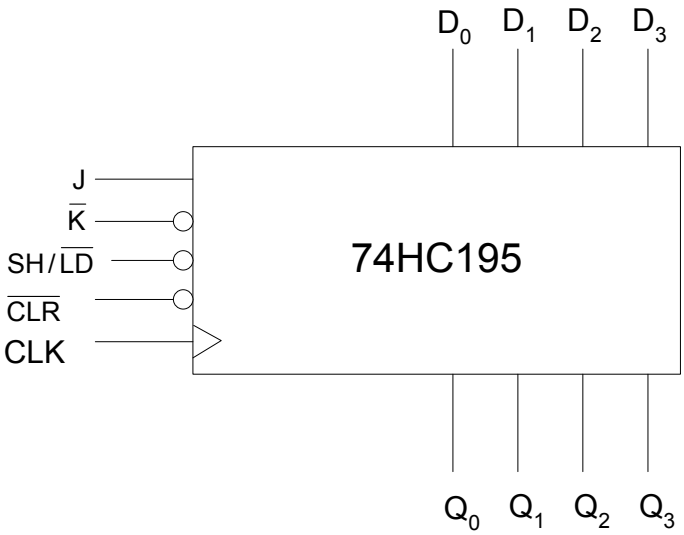


Figure 34.12 74HC195, 4-bit Parallel In/Parallel Out Shift Register

The 74HC195 is a 4-bit Parallel In/Parallel Out Register. It also has a Serial In input, therefore the register can also be used as a Serial In/Parallel Out or as a Serial In/ Serial Out register. The output Q_3 is used as a Serial Out pin. 74HC195. The J and $\overline{K}$ inputs are used to input data serially. These inputs are connected to the first flip-flop. The $SH/\overline{LD}$ is used to load the Parallel Data and to allow shift operations on the clock transition. The $\overline{CLR}$ input is used to clear the register asynchronously. Figure 34.12.

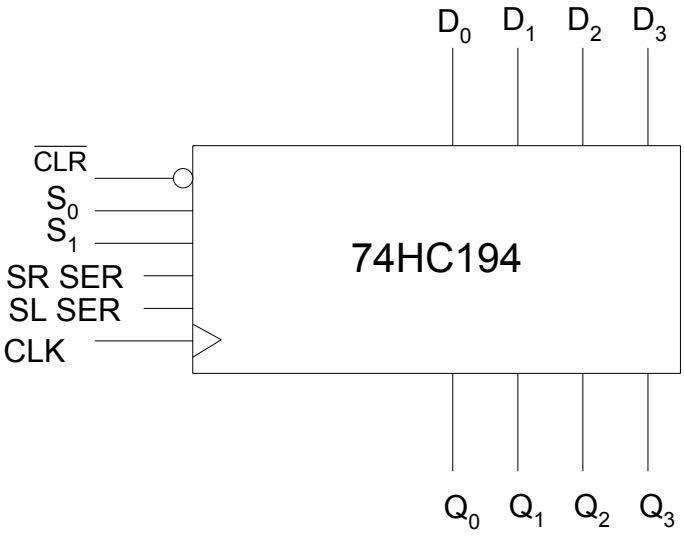


Figure 34.13 Bi-directional 4-bit Universal Shift Register

The 74HC194 is a 4-bit Bidirectional Shift register that shifts data in the left and right hand directions and has both Parallel and Serial input and output capability. Figure 34.13. The register has 4-bit Parallel Inputs D_0 to D_3 and Parallel Outputs Q_0 to Q_3. An active low Asynchronous CLR input clears the register. The register shifts data on a positive clock transition. S_0 and S_1 inputs control the operation of the register. When S_0 and S_1 both are at

When S_0 is high, shift right operation is carried out, serial data is entered through the SR SER input. When S_1 is high, shift left operation is carried out, serial data is entered through the SL SER input. When both S_0 and S_1 are logic low the register is inhibited.

6. Rotate Right Operation

The serial output of the register is connected to the serial input of the register. By applying clock pulses data is shifted right. The data shifted out of the serial out pin at the right hand side is re-circulated back into the shift register input at the left hand side. Thus the data is rotated right within the register. Figure 34.14

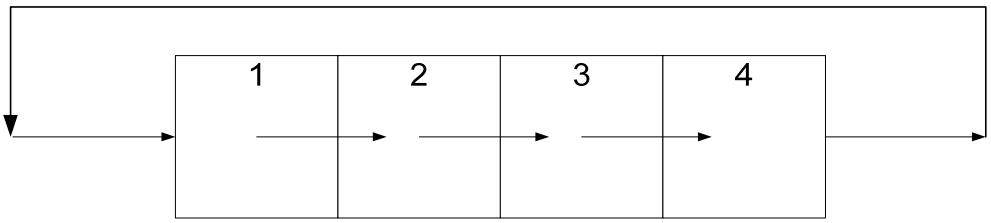


Figure 34.14 Rotate Right Operation

7. Rotate Left Operation

The serial output of the register is connected to the serial input of the register. By applying clock pulses data is shifted left. The data shifted out of the serial out pin at the left hand side is re-circulated back into the shift register input at the right hand side. Thus the data is rotated left within the register. Figure 34.15

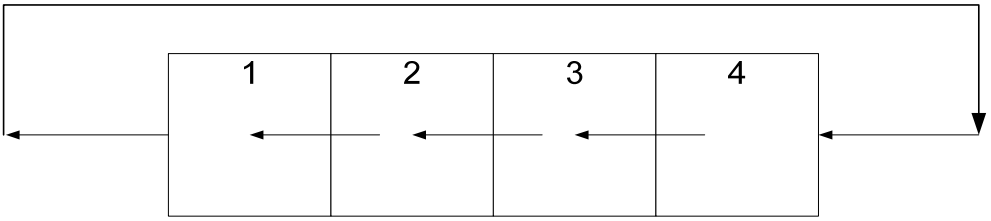


Figure 34.15 Rotate Left Operation

Shift Register Counters

Shift register counters are basically, shift registers connected to perform rotate left and rotate right operations. When data is rotated through a register counter a specific sequence of states is repeated. Two commonly used register counters in digital logic are the Johnson Counter and the Ring Counter.

1. Johnson Counter

In a Johnson counter, the $\overline{Q}$ output of the last flip-flop of the shift register is connected to the data input of the first flip-flop. The circuit of a 4-bit, D flip-flop based Johnson Counter is shown in figure 34.16. The sequence of states that are implemented by a n-bit Johnson counter are 2n. Thus a 4-bit Johnson counter sequences through 8 states and a 5-bit Johnson

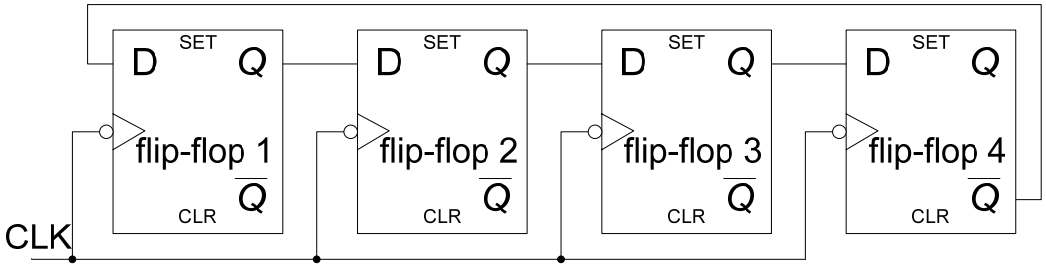


Figure 34.16 4-bit Johnson Counter

Clock Pulse	Q_0	Q_1	Q_2	Q_3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

Table 34.1 Sequence of states of a 4-bit Johnson Counter

2. Ring Counter

The Ring Counter is similar to the Johnson counter, except that the Q output of the last flip-flop of the shift register is connected to the data input of the first flip-flop of the shift register. All the flip-flops of the counter are cleared to logic low except for the first flip-flop which is preset to logic high. Figure 34.17.

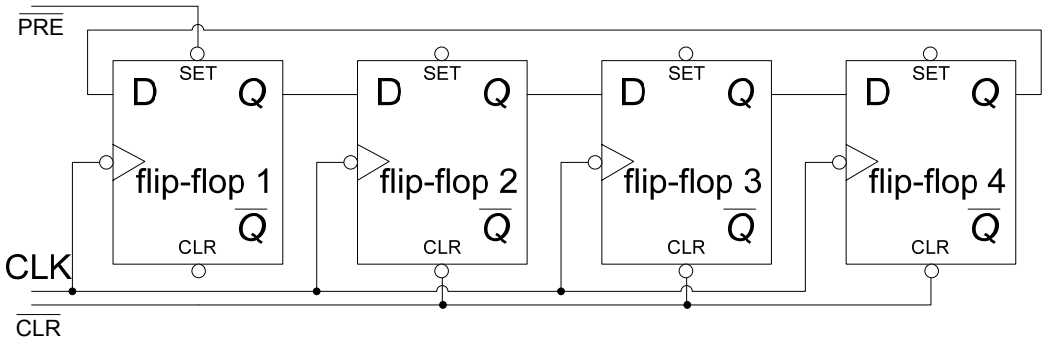


Figure 34.17 4-bit Ring Counter

After the initialization of the counter, the logic high set at the output of the first flip-flop is shifted right at each clock transition. Table 34.2. With a Ring Counter circuit no decoding gates are required. Each state of the ring counter has a unique output.

Clock Pulse	Q_0	Q_1	Q_2	Q_3
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1

Lesson No. 35

APPLICATIONS OF SHIFT REGISTERS

The major application of a shift register is to convert between parallel and serial data. Shift registers are also used as keyboard encoders. The two applications of the shift registers are discussed.

1. Serial-to-Parallel Converter

Earlier, Multiplexer and Demultiplexer based Parallel to Serial and Serial to Parallel converters were discussed. The Multiplexer and Demultiplexer require registers to store the parallel data that is converted into serial data and parallel data which is obtained after converting the incoming serial data. A Parallel In/Serial Out shift register offers a better solution instead of using a Multiplexer-Register combination to convert parallel data into serial data. Similarly, a Serial In/Parallel Out shift register replaces a Demultiplexer-Register combination.

In Asynchronous Serial data transmission mode, a character which is constituted of 8-bits (which can include a parity bit) is transmitted. To separate one character from another and to indicate when data is being transmitted and when the serial transmission line is idle (no data is being transmitted) a set of start bit and stop bits are appended at both ends of the 8-bit character. A character is preceded by a logic low start bit. When the line is idle it is set to logic high, when a character is about to be transmitted the start bit sets the line to logic low. The logic low start bit is an indication that 8 character bits are to follow and the transmission line is no longer in an idle state. After 8-character bits have been transmitted, the end of the character is indicated by two stop bits that are at logic high. The two logic bits indicate the end of the character and also set the transmission line to the idle state. Therefore a total of 11 bits are transmitted to send one character from one end to the other. The logic low start bit is also a signal for the receiver circuit to start receiving the 8 character bits that are following the start bit. The 11-bit serial character format is shown. Figure 35.1.

<table>
<tr><td>0</td><td>0/1</td><td>0/1</td><td>0/1</td><td>0/1</td><td>0/1</td><td>0/1</td><td>0/1</td><td>0/1</td><td>1</td><td>1</td></tr>
<tr><td>Stop bit</td><td colspan="8">Data bits</td><td colspan="2">Stop bits</td></tr>
</table>

Figure 35.1 11-bit Serial Data format

A Serial to Parallel converter circuit based on shift registers is shown. Figure 35.2. The serial data is preceded by a logic low start bit which triggers the J-K flip-flop. The output of the flip-flop is set to logic high which enables the clock generator. The clock pulses generated are connected to the clock input of a Serial In/Parallel Out shift register and also to the clock input of an 8-bit counter. On each clock transition, the Serial In/Parallel Out shift register shifts in one bit data. When the 8-bit counter reaches its terminal count 111, the terminal count output signal along with the clock signal trigger the One-Shot and also allow the Parallel In/Parallel Out register to latch in the Parallel data at the output of the Serial In/Parallel Out shift register. The One-shot resets the J-K flip-flop output Q to logic 0 disabling the clock generator and also clears the 8-bit counter to count 000.

© Virtual University of Pakistan

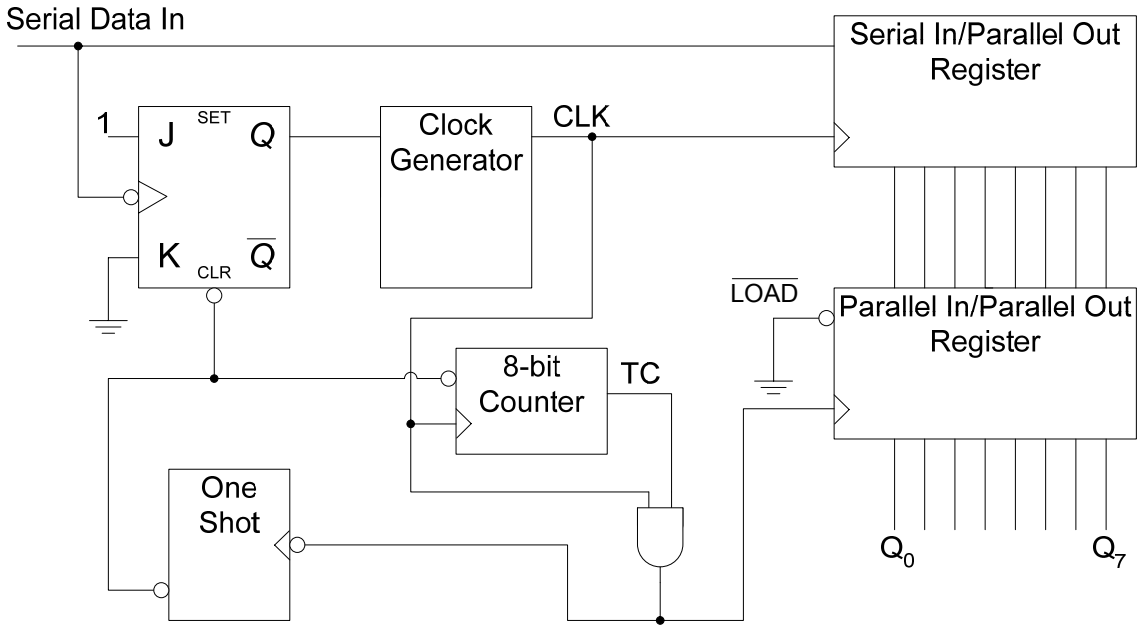


Figure 35.2 Series-to-Parallel Converter

2. Keyboard Encoder

Earlier a simple keypad encoder circuit was discussed where, the 0 to 9 digit keypad was connected through a decade to BCD encoder. Pressing any keypad key enables the corresponding input of the encoder circuit which encodes the input as a 4-bit BCD output. Computer keyboards which have more keys employ a keyboard encoder circuit that regularly scans the keyboard to check for any key press. Figure 35.3. The scanning is done by organizing the keys in the form of rows and columns. With the help of a shift register based ring counter one row is selected at a time. The two counters are connected as an 8-bit Ring counter which sequences through a bit pattern having all 1's and a single 0. The 8 state sequence selects one row at a time by setting it to logic 0. If a key is pressed, the corresponding column also becomes logic 0 as it connected to the selected row. The row and column which are selected are encoded by the row and column encoders. When a key is pressed, the selected column which is set to logic 0 sets the output of the NAND gate to logic 1 which triggers two One Shots. The first One Shot inhibits the clock signal to the ring counters for a short interval until the Key Code is stored. The One Shot also triggers the second One-Shot that sends a pulse to the clock input of the Key Code register. The Key Code Register stores the key ID represented as 3-bit column and 3-bit row code.

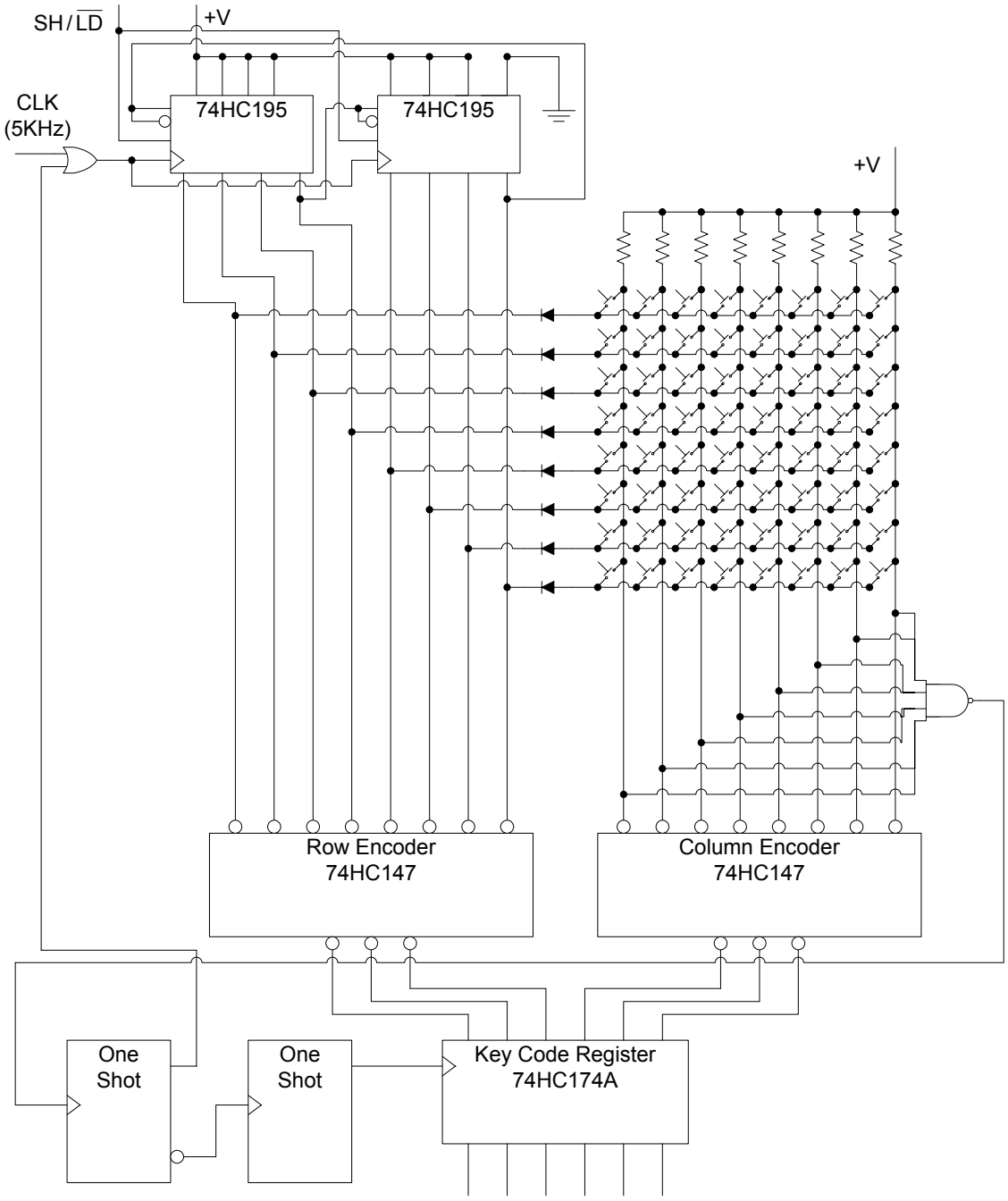


Figure 35.3 Keyboard Encoder circuit

Programmable Sequential Logic

Earlier PLD devices were discussed and their Combinational Modes were discussed. PLD devices can be programmed to implement Sequential Circuits. The AND-OR gate array of a PLD device is used to implement the excitation inputs for the memory element. The Memory element is implemented in the form of a flip-flop in each OLMC module of the PLD device. The

the input combinational circuit that generates the excitation inputs for the memory element. The output of the sequential circuit is obtained from the tri-state buffer which connects the output of the OLMC module to the output pin of the PLD device. The output of the tri-state buffer is determined by the current state of the flip-flop and the combinational external input which is connected to the control input of the tri-state buffer which enables or disables the tri-state buffer output.

The Registered Mode

In the discussion on Combinational Logic with PLDs, the two active-low and active-high Combinational Modes of the PLD device were discussed. In Sequential Logic with PLD devices the Registered Active-low and Active high Modes are used. Figure 35.4

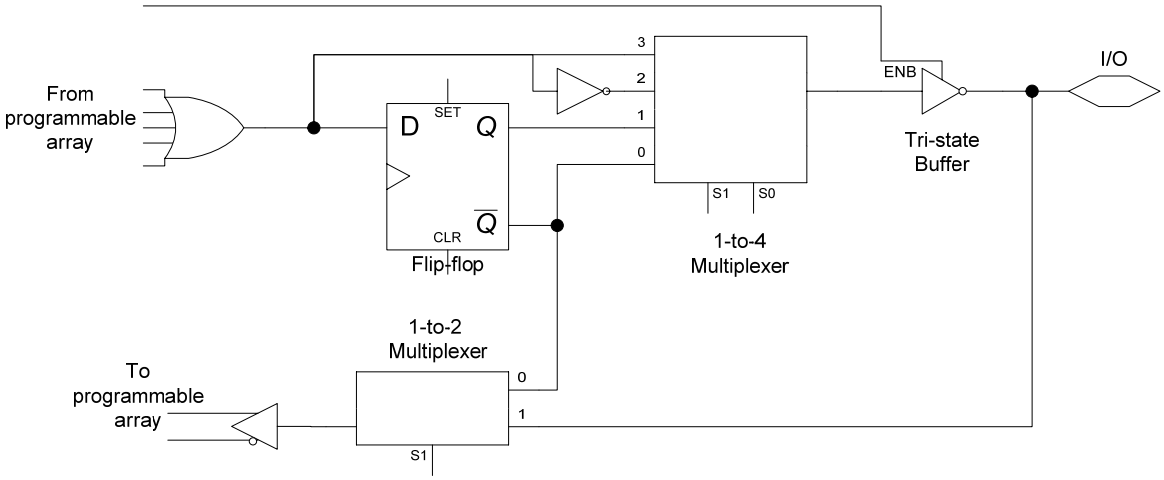


Figure 35.4 OLMC of the GAL22V10 device

The PLD is selected for Sequential operation by configuring the OLMC in the Registered Mode by setting the 1-to-4 MUX select inputs S1 and S0 to 01 or 00. By setting the MUX select inputs S1 and S0 to 01 respectively, the Q output of the D flip-flop is made available at the out of the Multiplexer which is connected to the output tri-state buffer. The S1 select input of the 1-to-2 MUX is also set to 0 in the Registered Mode operation, which allows the $\overline{Q}$ output of the D flip-flop to be feed back to the AND gate array. In the Registered Mode the feedback from the tri-state buffer output can not be used as a feedback to the AND gate array. By setting the S1 and S0 select inputs of the 1-to-4 MUX to 00 respectively, the output of the MUX is connected to the $\overline{Q}$ output of the D flip-flop instead of the Q output. The feedback to the AND gate array however remains the same, which is connected to the $\overline{Q}$ output of the D flip-flop.

Software Mode Specification

The Combinational or Registered Modes of the OLMC are selected by programming statements in the declaration part of the input file and the way logic descriptions are written.
The ISTYPE statement is used in the declaration part with the statements assigning PIN numbers to output variables.

```
X     PIN 22     ISTYPE ‘reg’;
Y     PIN 23     ISTYPE ‘com’;
```

The declaration statements describe variable X and Y available at output pins 22 and 23 respectively. The X variable is a 'Registered' output available from the D-flip-flop. The Y variable is a 'Combinational' output available directly from the AND-OR gate array output. The active-low or active-high output of the Registered Mode can also be specified in the declaration statement

```
X       PIN 22          ISTYPE 'reg.buffer;
Z       PIN 20          ISTYPE 'reg.invert';
```

Figure 35.5b ISTYPE statement to specifying active-high or active-low Registered Mode output

The first declaration statement describes X output variable as an active-high Registered Mode output. The second statement describes Z output variable as an active-low Registered Mode output.

The assignment operators ':=' and ':>' are used in logic descriptions to indicate a Registered output.

```
X := D;
Y = D;
```

Figure 35.5c Assignment Operators for Registered Mode

The first logical declaration statement indicates that X will be assigned the value of D on the clock transition and will hold the value until the next clock transition. The second logical declaration indicates that output Y is equal to input D.

The dot extension '.CLK' is used to indicate that the register device is a clocked flip-flop. A statement using the dot extension must accompany a logical declaration statement.

```
X := D;
X.CLK = Clock;
```

Figure 35.5d Dot assignment to indicate clocked flip-flop

Example1: Parallel Input/Parallel Output 8-bit Register with inverted outputs

A PLD device such as GAL22V10 can be programmed to work as an 8-bit D flip-flop based register with inverted outputs. The ABEL statements for configuring the PLD are shown. Figure 35.6

The pin declarations are

```
Clock, D0, D1, D2, D3, D4, D5, D6 D7   PIN    1, 2, 3, 4, 5, 6, 7, 8, 9;
Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7         PIN    22, 21, 20, 19, 18, 17, 16, 15
                                              ISTYPE 'reg.invert';
```

The logical declarations are

```
[Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7] := [D0, D1, D2, D3, D4, D5, D6, D7];
[Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7].CLK = Clock;
```

The logical declarations can also be written as

```
Q0 := D0;
Q0.CLK = Clock;
Q1 := D1;
Q1.CLK = Clock;
Q2 := D2;
Q2.CLK = Clock;
Q3 := D3;
Q3.CLK = Clock;
Q4 := D4;
Q4.CLK = Clock;
Q5 := D5;
Q5.CLK = Clock;
Q6 := D6;
Q6.CLK = Clock;
Q7 := D7;
Q7.CLK = Clock;
```

Figure 35.6a ABEL Statements for implementing an 8-bit register with inverted outputs

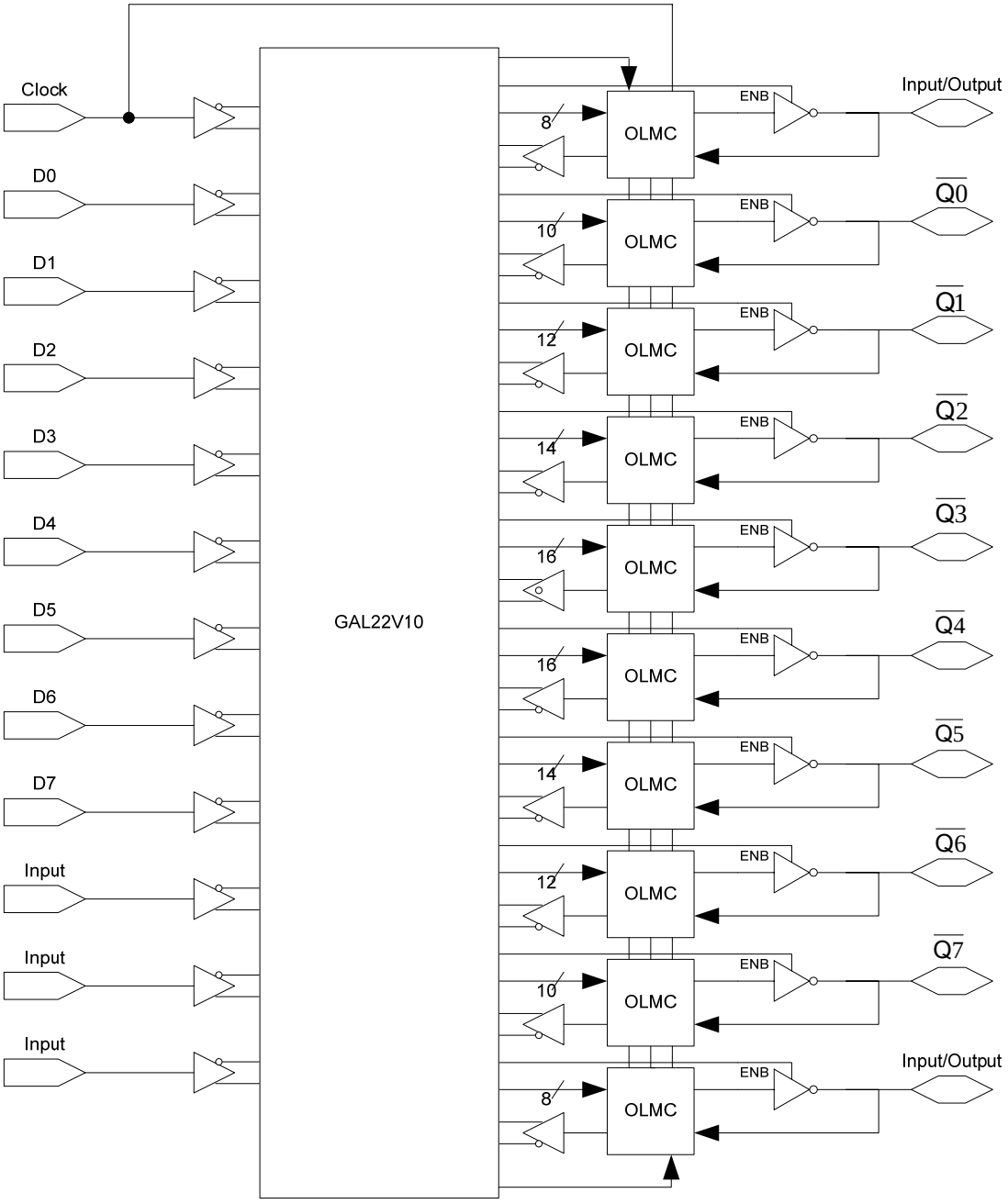


Figure 35.6b GAL22V10 configured as an 8-bit inverted output register

Example2: 8-bit Serial In/Parallel Out Shift Register

An 8-bit Serial In/Parallel out shift register based on an identical D type flip-flop is shown. Figure 35.7. The Clear signal has to be set to logic 0 to asynchronously clear all the flip-flops. The Enable input has to be set to logic 1 to allow serial data to be shifted in. An 8-bit Serial In/Parallel Out Shift Register is implemented using the GAL22V10 PLD. The D flip-flop implemented in the OLMC is triggered on the positive clock edge. It also has active-high,

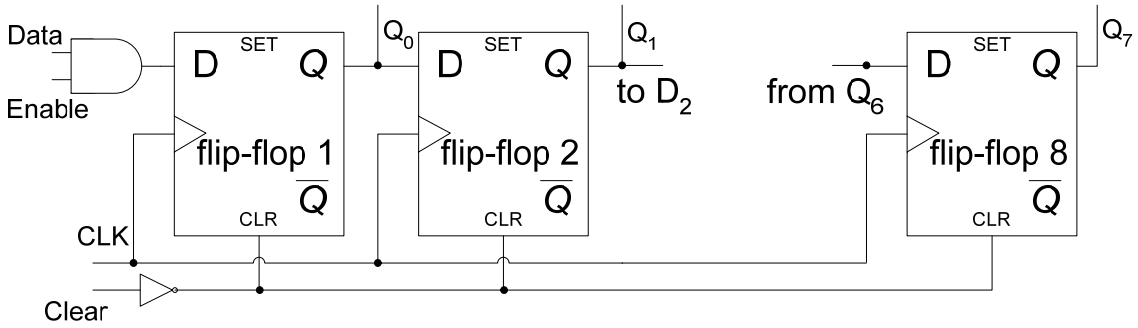


Figure 35.7 8-bit Serial In/Parallel Out Shift Register

Module Right_bit_shift_register

Title '8-bit shift register in a GAL22V10'

"Device Declaration
Register Device 'P22V10'

"Pin Declaration

Clock, Clear	Pin 1, 2;
Data, Enable	Pin 3, 4;
Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7	Pin 16, 17, 18, 19, 20, 21, 22, 23 ISTYPE 'reg.buffer';

Equations
Q0 := Data & Enable;
[Q1, Q2, Q3, Q4, Q5, Q6, Q7] := [Q0, Q1, Q2, Q3, Q4, Q5, Q6];
[Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7].CLK = clock;
[Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7].AR = !clear;

Test_Vectors

([Clock, Clear, Data, Enable] -> [Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7])
[.x. , 0 , .x. , .x.] -> [0 , 0 , 0 , 0 , 0 , 0 , 0 , 0];
[.c. , 1 , 1 , 0] -> [0 , 0 , 0 , 0 , 0 , 0 , 0 , 0];
[.c. , 1 , 0 , 1] -> [0 , 0 , 0 , 0 , 0 , 0 , 0 , 0];
[.c. , 1 , 1 , 1] -> [1 , 0 , 0 , 0 , 0 , 0 , 0 , 0];
[.c. , 1 , 0 , 1] -> [0 , 1 , 0 , 0 , 0 , 0 , 0 , 0];
[.c. , 1 , 1 , 1] -> [1 , 0 , 1 , 0 , 0 , 0 , 0 , 0];
[.c. , 1 , 0 , 1] -> [0 , 1 , 0 , 1 , 0 , 0 , 0 , 0];
[.c. , 1 , 1 , 1] -> [1 , 0 , 1 , 0 , 1 , 0 , 0 , 0];
[.c. , 1 , 0 , 1] -> [0 , 1 , 0 , 1 , 0 , 1 , 0 , 0];
[.c. , 1 , 1 , 1] -> [1 , 0 , 1 , 0 , 1 , 0 , 1 , 0];
[.c. , 1 , 0 , 1] -> [0 , 1 , 0 , 1 , 0 , 1 , 0 , 1];
[.c. , 1 , 1 , 1] -> [1 , 0 , 1 , 0 , 1 , 0 , 1 , 0];
[.c. , 0 , 1 , 1] -> [0 , 0 , 0 , 0 , 0 , 0 , 0 , 0];
END

The ABEL Input file format for the shift register is shown in figure 35.4. The Equations and the Test_Vectors declarations are,

- Q0 := Data & Enable;

  The Q0 output is active high and depends upon the product of Data input and the Enable input and Q0 will be assigned the product value at the positive transition of the clock.

- [Q1, Q2, Q3, Q4, Q5, Q6, Q7] := [Q0, Q1, Q2, Q3, Q4, Q5, Q6];
  [Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7].CLK = Clock;

  The Q0, Q1, Q2, Q3, Q4, Q5, Q6 outputs are assigned to Q1, Q2, Q3, Q4, Q5, Q6, Q7 respectively on a clock transition.

- [Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7].AR = !Clear;

  The Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7 outputs are reset on a $\overline{\text{Clear}}$ signal applied at the Asynchronous Reset (AR) Input.

- ([Clock, Clear, Data, Enable] -> [Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7])
  [.x. , 0 , .x. , .x.] -> [0 , 0 , 0 , 0 , 0 , 0 , 0 , 0];
  [.c. , 1 , 1 , 0] -> [0 , 0 , 0 , 0 , 0 , 0 , 0 , 0];
  [.c. , 1 , 0 , 1] -> [0 , 0 , 0 , 0 , 0 , 0 , 0 , 0];
  [.c. , 1 , 1 , 1] -> [1 , 0 , 0 , 0 , 0 , 0 , 0 , 0];

  The Test_Vector specifies ‘x’ as don’t care, ‘c’ as clock signal, thus the first vector specifies logic 0 outputs when Clear input is logic 0. Clock, Data and Enable inputs are don’t care. The second vector specifies a clock transition with, Clear, Data and Enable inputs set to logic 1, 1 and 0 respectively. The Enable input is set to logic 0 therefore the shift operation is inhibited. The third vector enables the shift operation with logic 0 shifted in. The fourth vector shifts in logic 1.

Example3: 4-bit Parallel In/Serial Out Shift Register

A 4-bit Parallel In/Serial Out shift register is shown. Figure 35.9. It is very similar to the register discussed earlier, except that the shift register shown has an asynchronous reset input which clears the shift register. The ABEL Input file for the 4-bit Parallel In/Serial Out shift register is shown in figure 35.10.

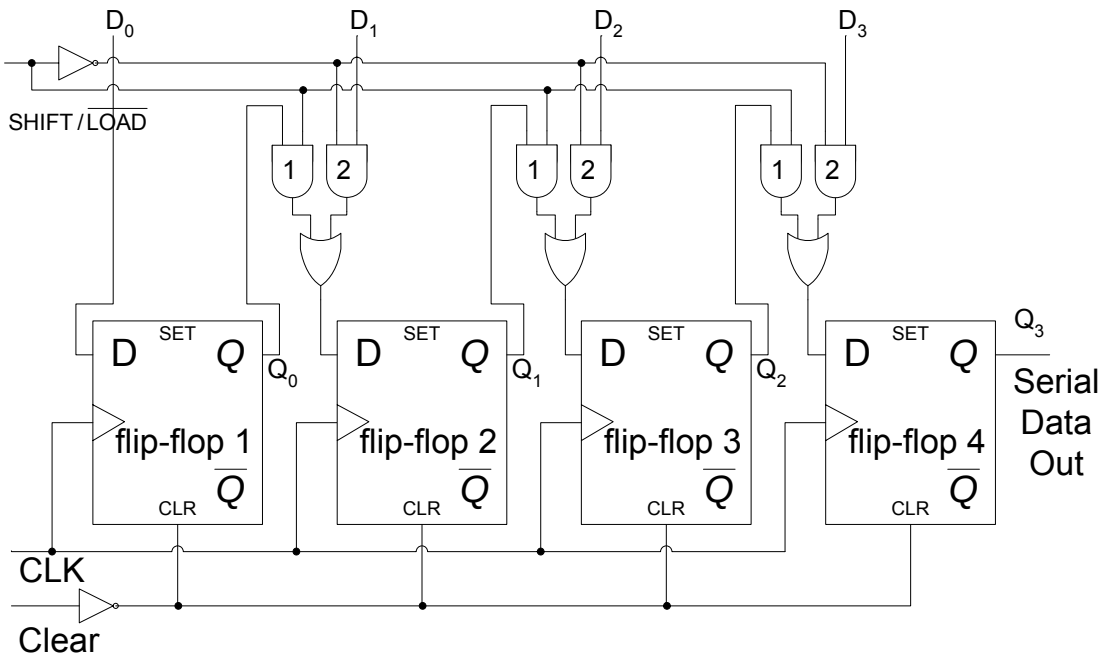


Figure 35.9 4-bit Parallel In/Serial Out Shift Register

Module Four_bit_shift_register

Title '4-bit shift register in a GAL22V10'

"Device Declaration

Register Device 'P22V10'

"Pin Declaration

Clock, Clear	Pin 1, 2;
SHLD	Pin 3;
D0, D1, D2, D3	Pin 4, 5, 6, 7 ISTYPE 'reg.buffer';
Q0, Q1, Q2, Q3	Pin 14, 15, 16, 17 ISTYPE 'reg.buffer';

Equations

```
Q0 := D0;
Q1 := Q0 & SHLD # D1 & !SHLD;
Q2 := Q1 & SHLD # D2 & !SHLD;
Q3 := Q2 & SHLD # D3 & !SHLD;
[Q0, Q1, Q2, Q3].CLK = clock;
[Q0, Q1, Q2, Q3].AR = !clear;
```

Test_Vectors

```
([Clock, Clear, SHLD, D0, D1, D2, D3] -> [Q3])
```

```
[  .c.  ,   1   ,   0   , 0 , 1 ,  0 ,  1 ] -> [ 1 ];
[  .c.  ,   1   ,   0   , 1 , 0 ,  1 ,  0 ] -> [ 0 ];
[  .c.  ,   1   ,   1   , 1 , 0 ,  1 ,  0 ] -> [ 1 ];
[  .c.  ,   1   ,   1   , 1 , 0 ,  1 ,  0 ] -> [ 0 ];
[  .c.  ,   1   ,   1   , 1 , 0 ,  1 ,  0 ] -> [ 1 ];
[  .c.  ,   0   ,   0   , 1 , 0 ,  1 ,  0 ] -> [ 0 ];
```

END

Figure 35.10 ABEL Input file for a 4-bit Parallel In/Serial Out Shift register

The Equations and the Test_Vectors declarations are,

- Q0 := D0;

  The Q0 output is active high and is assigned the value Do at the positive transition of the clock.

- Q1 := Q0 & SHLD # D1 & !SHLD;
  Q2 := Q1 & SHLD # D2 & !SHLD;
  Q3 := Q2 & SHLD # D3 & !SHLD;
  [Q0, Q1, Q2, Q3].CLK = clock;

  The Q1, Q2 and Q3 output is assigned the value based on the Boolean expression $Q0.SHLD + D1.\overline{SHLD}$, $Q1.SHLD + D2.\overline{SHLD}$ and $Q2.SHLD + D3.\overline{SHLD}$ on a positive clock transition.

- [Q0, Q1, Q2, Q3].AR = !clear;

  The Q0, Q1, Q2, Q3 outputs are reset on a $\overline{Clear}$ signal applied at the Asynchronous Reset (AR) Input.

- ([Clock, Clear, SHLD, D0, D1, D2, D3] -> [Q3])

```
[  .x.  ,   0   ,   .x.  , .x. , .x. , .x. , .x. ] -> [ 0 ];
[  .c.  ,   1   ,   0   , 0 , 1 ,  0 ,  1 ] -> [ 1 ];
```

  The Test_Vector specifies ‘x’ as don’t care, ‘c’ as clock signal, thus the first vector specifies logic 0 output at Q3 when Clear input is logic 0. Clock, SHLD, D0, D1, D2 and D3 inputs are don’t care. The second vector specifies a clock transition with, Clear, SHLD, D0, D1, D2 and D3 inputs set to logic 1, 0, 0, 1, 0 and 1 respectively. The data 0101 is loaded into the register with the Q3 output set to 1.

Lesson No. 36

EXAMPLE4: 3-BIT UP/DOWN COUNTER

The 3-bit Up/Down Counter was earlier implemented using J-K flip-flops. A D flip-flop based 3-bit Up/Down Counter is implemented by mapping the present state and next state information in D Input table. Table 36.1. The Karnaugh maps and the simplified Boolean expressions derived from the D Input table, table 36.2 are used to implement the 3-bit Up/Down counter circuit. Figure 36.1

Present State			Next State X=0			D flip-flop inputs		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	D_2	D_1	D_0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	0
0	1	0	0	1	1	0	1	1
0	1	1	1	0	0	1	0	0
1	0	0	1	0	1	1	0	1
1	0	1	1	1	0	1	1	0
1	1	0	1	1	1	1	1	1
1	1	1	0	0	0	0	0	0

Table 36.1a D flip-flop input table for X=0

Present State			Next State X=1			D flip-flop inputs		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	D_2	D_1	D_0
0	0	0	1	1	1	1	1	1
0	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	1
0	1	1	0	1	0	0	1	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	0	1	0	0
1	1	0	1	0	1	1	0	1
1	1	1	1	1	0	1	1	0

Table 36.1b D flip-flop input table for X=1

Q_2Q_1/Q_0X	00	01	11	10
00	0	1	0	0
01	0	0	0	1
11	1	1	1	0
10	1	0	1	1

$$D_2 = \overline{Q}_2\overline{Q}_1\overline{Q}_0X + \overline{Q}_2Q_1Q_0\overline{X} + Q_2\overline{Q}_0\overline{X} + Q_2Q_1X + Q_2\overline{Q}_1Q_0$$

Table 36.2a Boolean expression for D_2 inputs

Q_2Q_1/Q_0X	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	1	0	1	0
10	0	1	0	1

$$D_1 = \overline{Q}_1\overline{Q}_0X + Q_1\overline{Q}_0\overline{X} + Q_1Q_0X + \overline{Q}_1Q_0\overline{X}$$
$$D_1 = (\overline{Q}_1\overline{Q}_0 + Q_1Q_0)X + (Q_1\overline{Q}_0 + \overline{Q}_1Q_0)\overline{X}$$
$$D_1 = Q_0 \oplus Q_1 \oplus X$$

Table 36.2b Boolean expression for D_1 inputs

Q_2Q_1/Q_0X	00	01	11	10
00	1	1	0	0
01	1	1	0	0
11	1	1	0	0
10	1	1	0	0

$$D_0 = \overline{Q}_0$$

Table 36.2c Boolean expression for D_0 inputs

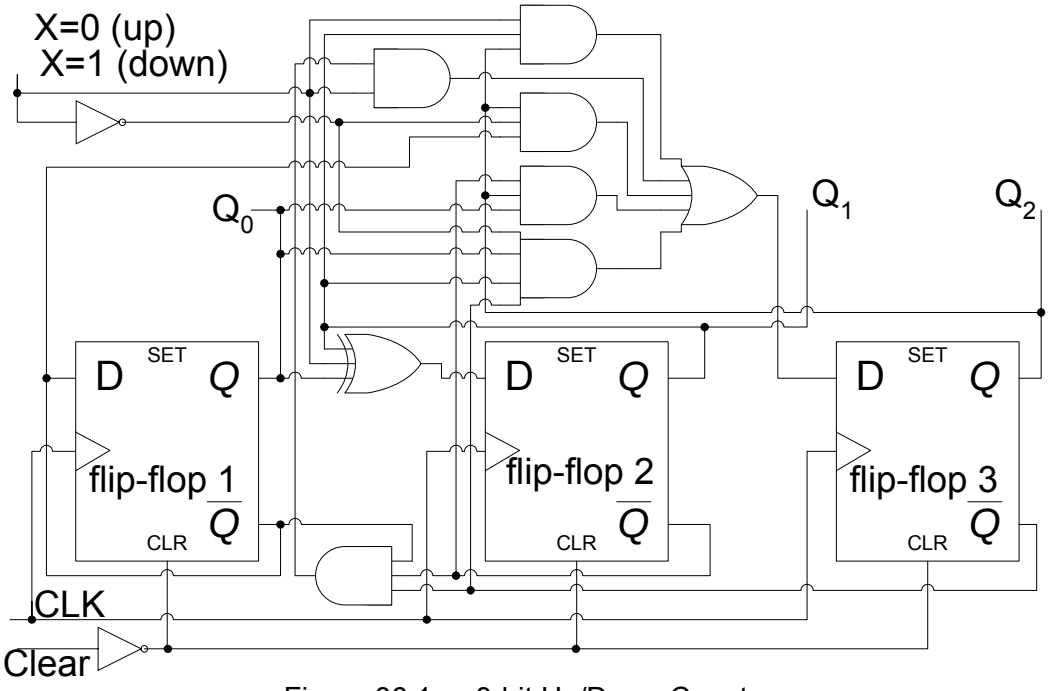


Figure 36.1 3-bit Up/Down Counter

The main definitions and declarations of the ABEL input file for the Up/Down Counter is shown. Table 36.3.

Pin Definition	
CLOCK, CLEAR, X	pin 1, 2, 3;
Q0, Q1, Q2	pin 21, 22, 23 ISTYPE 'reg,buffer';

Table 36.3a Input/Output Pin Definition of 3-bit Up/Down Counter

The Clock, Clear and X input variables applied at pins 1, 2 and 3 are used to provide the clock signal, the asynchronous clear pulse and the external input to control the direction of the count sequence. The Q0, Q1 and Q2 outputs are available from the D flip-flops of the three OLMCs at pins 21, 22 and 23. Since these outputs are sequential outputs available from the flip-flops therefore they are defined of type register 'reg.buffer'. The three outputs are the State variables that define the different states. Table 36.3a

```
Equations
Q0 := !Q0;
Q1 := Q0 $ Q1 $ X;
Q2 := !Q2 & !Q1 & !Q0 & X # !Q2 & Q1 & Q0 & !X # Q2 & !Q0 & !X
                                    # Q2 & Q1 & X # Q2 & !Q1 & Q0;
[Q0, Q1, Q2].CLK = clock;
[Q0, Q1, Q2].AR = !clear;
```

Table 36.3b Equation Definition of 3-bit Up/Down Counter

The ABEL Equations definition defines the Next state outputs for the three state variables. The ABEL assignment equations represent the Boolean expressions derived for the three D flip-flop inputs, table 36.2. Thus the next state output for variable Q0 depends upon the D0 input which is defined by the Boolean expression as $\overline{Q}_0$. Similarly, for the next state output for variable Q1 depends upon the D1 input which is defined by the Boolean expression as $Q_0 \oplus Q_1 \oplus X$. The ABEL next state expression for state variable Q3 is similarly based on the Boolean expression for Q3.

The ABEL statements [Q0, Q1, Q2].CLK = clock and [Q0, Q1, Q2].AR = !clear declare the change from the present state to the next state on a clock transition and the Asynchronous reset (AR) of all the three D flip-flops in the OLMC by the $\overline{\text{Clear}}$ input signal. Table 36.3b

The Test Vector definition defines the test vectors for all the three counter inputs and the three counter outputs. Since the Asynchronous input overrides the Synchronous X input, therefore in the first test vector when the Clear Asynchronous input is 0 the output is cleared to 000 irrespective of the X Synchronous input. When the Clear input is set to 1, the counter functions normally, the X input set to 0 sets the counter to increment and the X input set to 1 sets the counter to decrement. Table 36.3c

```
Test Vector
([Clock, Clear, X] -> [Q2, Q1, Q0])
[   .c.   ,    0    ,.x.] -> [0  ,  0  ,   0 ];
[   .c.   ,    1    , 0 ] -> [0  ,  0  ,   1 ];
[   .c.   ,    1    , 0 ] -> [0  ,  1  ,   0 ];
[   .c.   ,    1    , 0 ] -> [0  ,  1  ,   1 ];
[   .c.   ,    1    , 0 ] -> [1  ,  0  ,   0 ];
[   .c.   ,    1    , 0 ] -> [1  ,  0  ,   1 ];
[   .c.   ,    1    , 0 ] -> [1  ,  1  ,   0 ];
[   .c.   ,    1    , 0 ] -> [1  ,  1  ,   1 ];
[   .c.   ,    1    , 0 ] -> [0  ,  0  ,   0 ];
[   .c.   ,    1    , 1 ] -> [1  ,  1  ,   1 ];
[   .c.   ,    1    , 1 ] -> [1  ,  1  ,   0 ];
[   .c.   ,    1    , 1 ] -> [1  ,  0  ,   1 ];
[   .c.   ,    1    , 1 ] -> [1  ,  0  ,   0 ];
```

```
[  .c.  ,   1   , 1 ] -> [0  ,  1  ,   1 ];
[  .c.  ,   1   , 1 ] -> [0  ,  1  ,   0 ];
[  .c.  ,   1   , 1 ] -> [0  ,  0  ,   1 ];
[  .c.  ,   1   , 1 ] -> [0  ,  0  ,   0 ];
```

Table 36.3c Test Vector Definition of 3-bit Up/Down Counter

Using a Truth-Table to specify Sequential Circuit

The ABEL Input file can use a truth table instead of the equation to specify the Boolean expressions. The Equation definition of the ABEL input file reduces to the two statements defining the change in the output state based on the clock transition and the Asynchronous reset of the D flip-flops through the Asynchronous Clear signal. Table 36.4a.

Equations
[Q0, Q1, Q2].CLK = clock; [Q0, Q1, Q2].AR = !clear;

Table 36.4a Equation Definition for Truth Table based Sequential Circuit definition

The 3-bit Up/Down Sequential circuit's complete operation can be described by a truth table which has external inputs Clear, X and Present State variables Q0, Q1 and Q2. The output of the counter circuit are the Next State variables Q0, Q1, Q2. The first statement of the truth table definition indicates that when Clear is set to 0, the counter output is reset to 000 irrespective of the X input and the present state inputs. The next 16 statements define the 8 input combinations and the corresponding counter outputs when the counter is counting up and the 8 input combinations and its corresponding outputs when the counter is counting down. Table 36.4b.

Truth Table

```
Truth_Table   ([Clear, X, Q2, Q1, Q0] :> [Q2, Q1, Q0])
               [  0   ,.x., .x. , .x. , .x. ] :> [ 0 ,  0 ,  0 ];
               [  1   , 0 , 0  , 0  , 0  ] :> [ 0 ,  0 ,  1 ];
               [  1   , 0 , 0  , 0  , 1  ] :> [ 0 ,  1 ,  0 ];
               [  1   , 0 , 0  , 1  , 0  ] :> [ 0 ,  1 ,  1 ];
               [  1   , 0 , 0  , 1  , 1  ] :> [ 1 ,  0 ,  0 ];
               [  1   , 0 , 1  , 0  , 0  ] :> [ 1 ,  0 ,  1 ];
               [  1   , 0 , 1  , 0  , 1  ] :> [ 1 ,  1 ,  0 ];
               [  1   , 0 , 1  , 1  , 0  ] :> [ 1 ,  1 ,  1 ];
               [  1   , 0 , 1  , 1  , 1  ] :> [ 0 ,  0 ,  0 ];
               [  1   , 1 , 0  , 0  , 0  ] :> [ 1 ,  1 ,  1 ];
               [  1   , 1 , 1  , 1  , 1  ] :> [ 1 ,  1 ,  0 ];
               [  1   , 1 , 1  , 1  , 0  ] :> [ 1 ,  0 ,  1 ];
               [  1   , 1 , 1  , 0  , 1  ] :> [ 1 ,  0 ,  0 ];
               [  1   , 1 , 1  , 0  , 0  ] :> [ 0 ,  1 ,  1 ];
               [  1   , 1 , 0  , 1  , 1  ] :> [ 0 ,  1 ,  0 ];
               [  1   , 1 , 0  , 1  , 0  ] :> [ 0 ,  0 ,  1 ];
               [  1   , 1 , 0  , 0  , 1  ] :> [ 0 ,  0 ,  0 ];
```

Using a State Diagram to specify Sequential Circuit

The ABEL Input file can also use a State diagram to specify the states of the Sequential circuit. Before specifying the State diagram, the states have to be defined. After defining the states the state diagram is defined by indicating how the present state changes to the next state. ABEL, IF-THEN-ELSE statements or GOTO statements are used to specify the how the present state changes to the next state.

State Definition	
QSTATE	= [Q2, Q1, Q0];
A	= [0 , 0 , 0];
B	= [0 , 0 , 1];
C	= [0 , 1 , 0];
D	= [0 , 1 , 1];
E	= [1 , 0 , 0];
F	= [1 , 0 , 1];
G	= [1 , 1 , 0];
H	= [1 , 1 , 1];

Table 36.5a State definition of the 3-bit Up/Down Counter

The QSTATE variable defines the eight states of the counter circuit. Each state defined by the three state variables is identified by state names A, B to H. Table 36.5a.

State Diagram	
State A:	if X then H else B;
State B:	if X then A else C;
State C:	if X then B else D;
State D:	if X then C else E;
State E:	if X then D else F;
State F:	if X then E else G;
State G:	if X then F else H;
State H:	if X then G else A;

Table 36.5b Defining the next states using IF-THEN-ELSE

The ABEL, IF-THEN-ELSE statements are used to define the input conditions for which the present state changes to the next state. For example, in the State Diagram, the State A changes to state H if the input variable X=1 otherwise the next state is state B. Similar, if then else statements are used to define the next states for each of the present states. If the present state switches to the next state without checking any conditions then an ABEL, GOTO statement is used. For example, for a 3-bit Up counter GOTO statements are used to specify the next state without checking any condition. Table 36.5c

State Diagram	
State A:	GOTO B;
State B:	GOTO C;
State C:	GOTO D;
State D:	GOTO E;

State F:	GOTO G;
State G:	GOTO H;
State H:	GOTO A;

Table 36.5c Defining the next states using GOTO

The ABEL Input file which uses the State diagram instead of equations or the truth table replaces the Equation part by State Definitions and State Diagram. The Equation definition only defines the change in state on a clock transition and the Asynchronous input as shown in table 36.4a

Design Example: Elevator Control System

An elevator is installed in a building that moves from one floor to the other. A person going to the second floor from the first floor presses a request button on the first floor. When the elevator arrives at the first floor, the doors open and the person walks in to the elevator. The person presses the request button for the second floor. When the elevator reaches the second floor, it stops and the doors open. The doors are opened for a specified time. A person inside the elevator can keep the doors open for a longer duration of time if an 'Open' button is pressed. Inside the elevator and outside near each entrance to the elevator there is a 7-segment display which displays the floor on which the elevator currently is. The direction, Up or Down in which the elevator is moving is also displayed.

Input and Output Signals

Different inputs and outputs are required to control the operation of the elevator. The operation of the elevator is based on a sequential state machine. A State diagram describes all the operations of the elevator. The inputs that are received from the person in the form of requests are

- Request buttons to call the elevator, REQ1 and REQ2
- Floor request buttons inside the elevator, FLOOR1 and FLOOR2
- Open door button, OPEN

The duration for which the elevator doors are opened, and remain open, and the time it takes for the elevator to move form one floor to the next is determined by a clock signal. When the elevator arrives at a floor a floor sensor generates an ARRIVE signal. Thus the State machine uses to additional input signals.

- Clock signal, CLOCK
- Arrive signal, ARRIVE

The elevator generates three output signals to indicate the doors OPEN/CLOSED, direction of movement of elevator UP/DOWN and the motion of the elevator WAITING/MOVING.

- Door Open/Close, DOOR=0 and DOOR=1
- Direction Up/Down, DIR=0 and DIR=1
- Motion Waiting /Moving, MOTION=0 and MOTION=1

In addition to the three output signals the elevator generates signals to display the floor number and the direction in which the elevator is moving.

Elevator State Diagram

The State diagram of the elevator operation has six states. Figure 36.2. Three state variables are required to define the six states. The state outputs directly determine the status of the door, the direction of the motion and control the motion.

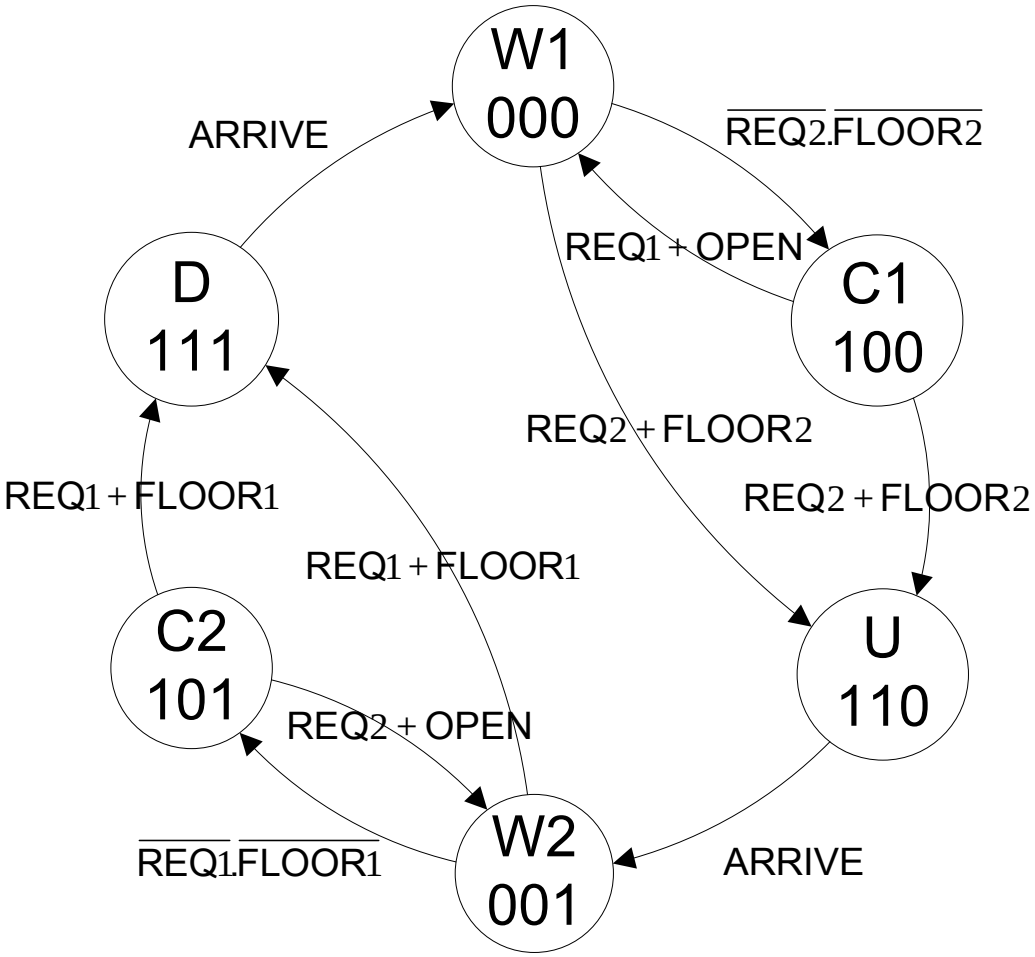


Figure 36.2 State Diagram of Elevator

Wait 1 State (W1): The initial state of the elevator is the Wait 1 State (W1), where the elevator is waiting on the first floor with the door open. The W1 state 000 identifies the state outputs.

- Door=0 (open)
- Motion=0 (waiting)
- Dir=0 (up)

Close 1 State (C1): If for a fixed interval of time, a request from second floor is not received or the button for floor 2 is not pressed, the system goes to state Close1 (C1). In this state the elevator remains on the first floor with its doors closed. The C1 state 100 identifies the output signals.

- Door=1 (close)

- Dir=0 (up)

If at any time a person inside the elevator presses the Open door switch, OPEN or a person waiting on first floor requests for the elevator REQ1, the system switches back to Wait 1 State, W1.

Up State (U): The system switches to Up state (U) when either a request is received from the second floor REQ2 or a person presses the Floor2 switch F2. The U State represents the state when the elevator is moving up. The outputs of this state are 110.

- Door=1 (close)
- Motion=1 (moving)
- Dir=0 (up)

Similarly when the system is in state W1, the system switches to Up state (U), when either a request is received from the second floor REQ2 or a person presses the Floor2 switch F2.

Wait 2 State (W2): The system switches from Up state (U), to Wait 2 state (W2) when the Arrive signal is received from the floor sensor. In the W2 state the elevator is waiting on floor 2 with its door open. The output of the W2 state are 001

- Door=0 (open)
- Motion=0 (waiting)
- Dir=1 (down)

Close 2 State (C2): The elevator waits with its door open on the second floor for a specified period of time. If a request from first floor is not received or the button for floor 1 is not pressed, the system switches to Close2 state (C2). In this state the elevator remains on the second floor with its doors close. The output signals of C2 state are 101

- Door=1 (close)
- Motion=0 (waiting)
- Dir=1 (down)

If at any time a person inside the elevator presses the Open door switch, OPEN or a person waiting on second floor requests for the elevator REQ2, the system switches back to Wait 2 State, W2.

Down State (D): The system switches to Down state (D), when either a request is received from the first floor REQ1 or a person presses the Floor1 switch F1. The D, Down represents the state when the elevator is moving down. The outputs of this state are 111.

- Door=1 (close)
- Motion=1 (moving)
- Dir=1 (down)

Similarly when the system is in state W2, the system switches to state D, Down when either a request is received from the first floor REQ1 or a person presses the Floor1 switch F1.The system switches to the Wait 1 state (W1) when the Arrive signal is received from the floor sensor on the first floor.

State Table

Present State	Next State	Next State	Next State	Next State	Next State	Next State
	REQ1=0	REQ1=1	FLOOR1=0	FLOOR1=1	OPEN=0	OPEN=1
W1(000)	x	x	x	x	x	x
C1(100)	C1	W1	x	x	C1	W1
UP(110)	x	x	x	x	x	x
W2(001)	C2	DO	C2	DO	x	x
C2(101)	C2	DO	C2	DO	x	x
DO(111)	x	x	x	x	x	x

Table 36.6a State table for Elevator Control for REQ1, FLOOR1 and OPEN inputs

Present State	Next State	Next State	Next State	Next State	Next State	Next State
	REQ2=0	REQ2=1	FLOOR2=0	FLOOR2=1	OPEN=0	OPEN=1
W1(000)	C1	UP	C1	UP	x	x
C1(100)	C1	UP	C1	UP	x	x
UP(110)	x	x	x	x	x	x
W2(001)	x	x	x	x	x	x
C2(101)	C2	W2	x	x	C2	W2
DO(111)	x	x	x	x	x	x

Table 36.6b State table for Elevator Control for REQ2, FLOOR2 and OPEN inputs

The Next State tables for the Elevator Control are obtained directly from the State Diagram. The State table show the present and the next states for each of the five external inputs that are activated by people using the elevator. The ARRIVE external input is activated by a separate sensor circuit and is not activated by press of a button. The next states when the ARRIVE signal is active are not shown in the State Tables.

Input Latches

The request buttons to call the elevator, REQ1 and REQ2, the Floor request buttons inside the elevator, F1 and F2 and the Open door button, OPEN can be pressed at any time. A sequential circuit switches from one state to the next on the basis of its present state and external input. Supposedly, a person presses and releases the REQ1 button to request for the elevator in a time period between two consecutive clock transitions. At the clock transition, the status of the REQ1 switch is inactive therefore the REQ1 is not entertained. It is therefore important that all input buttons, REQ1, REQ2, F1, F2 and OPEN are connected through latches which are not controlled by a clock signal. Figure 36.3. A total of 5 latches are required to store the inputs received from the buttons REQ1, REQ2, F1, F2 and OPEN.

© Virtual University of Pakistan

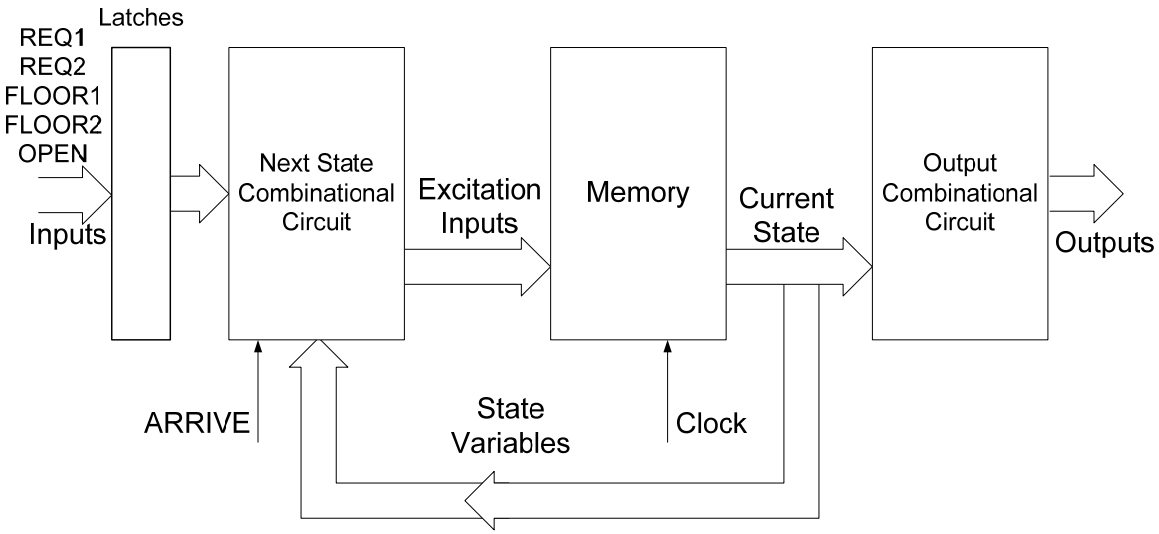


Figure 36.3 Block diagram of the Elevator State Machine

Lesson No. 37

REDUCED NUMBER OF INPUT LATCHES

The number of latches used to store the external inputs can be reduced to two if the REQ1, FLOOR1 and OPEN button outputs (pressed when elevator is on the first floor) are stored on one latch and the REQ2, FLOOR2 and OPEN button outputs pressed when elevator is on the second floor) are stored on the second latch. This also simplifies the Boolean expressions required to generate the excitation inputs for the next states. The next state table for REQ1, FLOOR1 and OPEN inputs indicates that the REQ1 can be pressed at any time either on the first floor or the second floor. The FLOOR1 request can also be pressed at any time, however if the elevator is already on the first floor then requests for FLOOR1 can be discarded. Similarly, if OPEN button is pressed when the elevator is on the first floor is considered as a valid request. The Boolean expressions that set the latch SR1 is $REQ1 + FLOOR1.DIR + OPEN.\overline{DIR}$. The circuit diagram of the SR1 latch is shown. Figure 37.1.

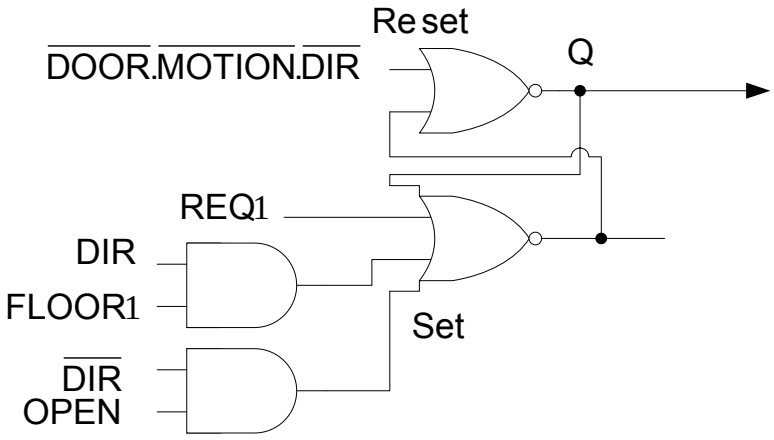


Figure 37.1 SR1 latch which stores the status of the REQ1, FLOOR1 and OPEN buttons

The DIR variable indicates the current floor. IF DIR=0, the elevator is on the first floor and if DIR=1, the elevator is on the second floor. Similarly, the OPEN input sets the SR1 latch when it is pressed when the elevator is on the first floor. The simplified next state table for inputs REQ1, FLOOR1 and OPEN in terms of SR1 latch is shown. Table 37.1.

Present State	Next State	Next State
	SR1=0	SR1=1
W1(000)	x	x
C1(100)	C1	W1
UP(110)	x	x
W2(001)	C2	DO
C2(101)	C2	DO
DO(111)	x	x

Table 37.1 Simplified State table for Elevator Control for REQ1, FLOOR1 and OPEN inputs

The state of inputs REQ2, FLOOR2 and OPEN can similarly be represented by setting/resetting the second latch SR2. The circuit diagram and the Boolean expression can similarly be represented. Figure 37.2 and Table 37.2.

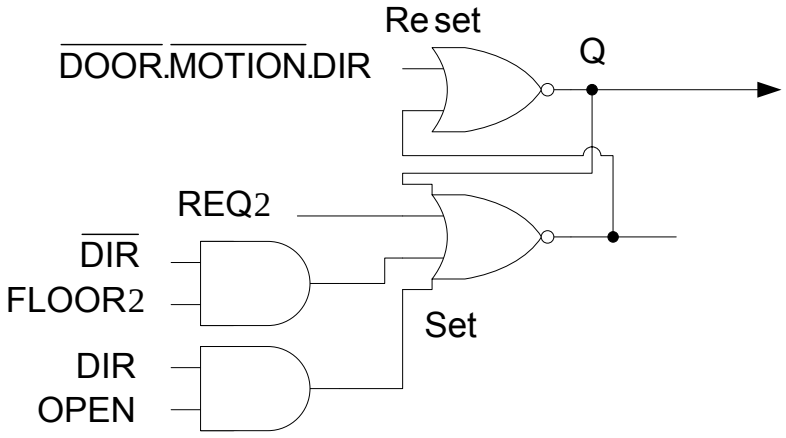


Figure 37.2 SR2 latch which stores the status of the REQ2, FLOOR2 and OPEN buttons

Present State	Next State	Next State
	SR2=0	SR2=1
W1(000)	C1	UP
C1(100)	C1	UP
UP(110)	x	x
W2(001)	x	x
C2(101)	C2	W2
DO(111)	x	x

Table 37.2 Simplified State table for Elevator Control for REQ2, FLOOR2 and OPEN inputs

The modified Block diagram of the Elevator State Machine which uses the two SR1, SR2 latches instead of the previously discussed five latches is shown. Figure 37.3. The Next State Combinational Circuit block is replaced by latches SR1 and SR2 which handle the REQ1, FLOOR1, REQ2, FLOOR2 and OPEN external inputs. The external input ARRIVE is connected to the Next State Combinational circuit along with the Present State inputs which determine the excitation inputs for the memory element.

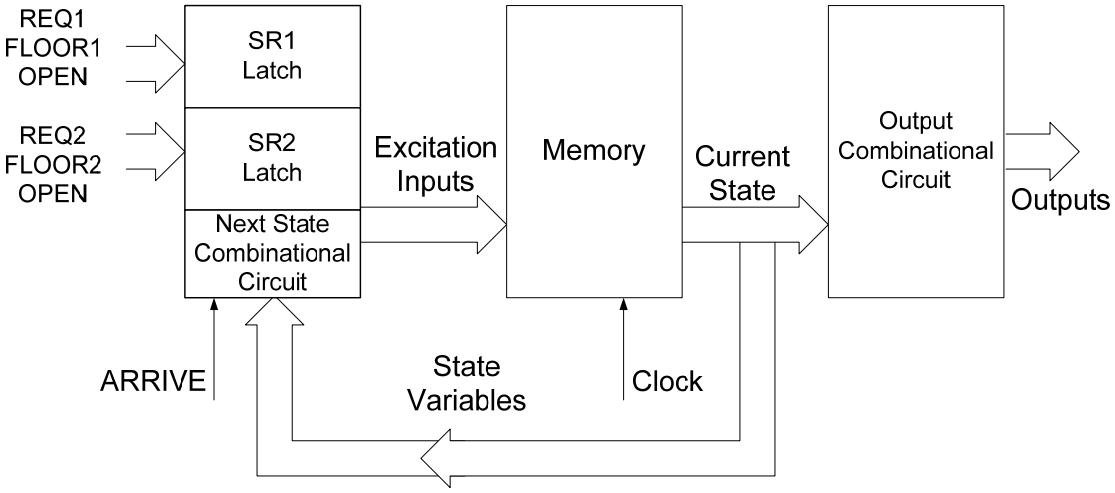


Figure 37.3 Modified Block diagram of the Elevator State Machine

The Next State Table for the Elevator State Machine based on the inputs SR1, SR2 and ARRIVAL is shown. Table 37.3. The Next State table is implemented using the two simplified state tables, table 37.1 and table 37.2 and the ARRIVE input.

Present State	Next State	Next State	Next State	Next State	Next State	Next State
	SR1=0	SR1=1	SR2=0	SR2=1	ARRIVAL=0	ARRIVAL=1
W1(000)	x	x	C1(100)	UP(110)	x	x
C1(100)	C1(100)	W1(000)	C1(100)	UP(110)	x	x
UP(110)	x	x	x	x	UP(110)	W2(001)
W2(001)	C2(101)	DO(111)	x	x	x	x
C2(101)	C2(101)	DO(111)	C2(101)	W2(001)	x	x
DO(111)	x	x	x	x	DO(111)	W1(000)

Table 37.3 The Next State Table based on SR1, SR2 and ARRIVAL inputs

The ABEL Input file for Elevator State Machine

The main declaration and definition sections of the ABEL input file for the Elevator State Machine are described. Table 36.4.

The SR1, SR1_, SR2, SR2_ variables are the S-R latch Q and $\overline{Q}$ outputs for latches SR1 and SR2. These latches are implemented using the AND-OR gates of the PLD device, there outputs are available at the output pins 16, 17, 18 and 19 of the GAL16V8 device. These outputs are generated by combinational circuits therefore these outputs are defined as ISTYPE ‘com.buffer’. These outputs are feed back to the AND gate array for connection to the D flip-flops. The outputs from the three D flip-flops, DOOR, MOTION and DIR are declared as ISTYPE ‘reg.buffer’ as these three outputs are the outputs of the sequential circuit D flip-flops in the OLMC modules. Table 36.4a.

Pin Declaration		
CLK, !OLE	Pin	1,11;
REQ1, REQ2	Pin	2,3;
FLOOR1, FLOOR2, OPEN, ARRIVE	Pin	4, 5, 6, 7;
SR1, SR1_	Pin	16, 17 ISTYPE ‘com.buffer’;
SR2, SR2_	Pin	18, 19 ISTYPE ‘com.buffer’;
DOOR, MOTION, DIR	Pin	12, 13, 14 ISTYPE ‘reg.buffer’;

Table 36.4a Pin Declarations of the Elevator Input and Output signals

The operation of the Sequential state machine in the ABEL file is defined in the form of a State diagram instead of Boolean expressions. Before defining the State Diagram, all the states are defined. The six states can be defined using the statement WAIT1 = [0,0,0], CLOSE1 = [1,0,0] etc. The alternate method for defining the states is by prefixing the binary number with ^B. Table 37.4b.

State Definition
CONSTATE = [DOOR, MOTION, DIR];
WAIT1 = ^B000;
CLOSE1 = ^B100;
UP = ^B110;
WAIT2 = ^B001;
CLOSE2 = ^B101;
DOWN = ^B111;

Table 37.4b State Definition of the Elevator Controller

The statements defining the State diagram for the Elevator State Machine are derived from the State Table. Table 37.3. The State Diagram definition is defined in Table 37.4c.

State Diagram	
State_diagram CONSTATE	
State WAIT1:	if (SR2) then UP else CLOSE1;
State CLOSE1:	if (SR2) then UP else if SR1 then WAIT1 else CLOSE1;
State UP:	if (ARRIVE) then WAIT2 else UP;
State WAIT2:	if (SR1) then DOWN else CLOSE2;
State CLOSE2:	if (SR1) then DOWN else if SR2 then WAIT2 else CLOSE2;
State DOWN:	if (ARRIVE) then WAIT1 else DOWN;

Table 37.4c State diagram for the Elevator Controller

The equations defining the Set and Reset input for the two latches SR1 and SR2 are defined in the Equation Definition part of the ABEL input file. The CONSTATE.CLK = Clock is used to indicate that the CONSTATE state variables change on a clock transition. Table 37.4d. The ‘FB’ indicates that the DOOR, MOTION and DIR output signals are feed back to the AND gate array.

Equation Definition

CONSTATE.CLK = Clock;
SR1 = REQ1 # !DIR.FB & OPEN # DIR.FB & FLOOR1 # !SR1_;
SR1_ = (!DOOR.FB & !MOTION.FB & !DIR.FB) # !SR1;
SR2 = REQ2 # !DIR.FB & OPEN # DIR.FB & FLOOR2 # !SR2_;
SR2_ = (!DOOR.FB & !MOTION.FB & DIR.FB) # !SR2;

Table 37.4d Equations for the latches SR1 and SR2

A separate GAL16V8 is used to implement the 7-Segment floor display and the Up and Down direction arrows. The floor display circuit is a combinational circuit which uses the MOTION and DIR inputs two determine the floor number and the direction of the display arrow.

Design Example: Traffic Signal Control System

A road intersection is shown on the diagram. Figure 37.4. On each section of the road two sensors determine the presence and arrival of vehicles. Sensor 1 is activated if a car is waiting and Sensor 2 is activated when an arriving vehicle passes over the sensor. The sensors installed on the North and South section of the road are connected together and determine the presence of vehicle(s) on the North-South section of the road. The sensors installed on the East and West section of the road are connected together and determine the presence of vehicle(s) on the East-West section of the Road. During the day when traffic flow is heavy at the intersection, the traffic light is cycled every 5 minutes. That is, the traffic signal controlling the North-South section is Green for 5 minutes and then Red for 5 minutes, Similarly, the traffic signal controlling the East-west section is Red for 5 minutes and Green for the next 5 minutes. During the night when traffic is relatively light it stops a car for a maximum time of 1 minute, unless a car approaches the intersection on the cross road in which case the traffic signal turns red and stops the approaching car and allows the waiting car to proceed. For example, a car is waiting at South approach of the intersection. A car approaching the intersection on the cross road from the East direction is stopped and the waiting car on the South section is allowed to proceed. The approaching car is detected by Sensor 2 installed on the East road section. If no other cars are arriving at the intersection the waiting car on the East approach is allowed to proceed after 1 minute.

Traffic Signal Controller Inputs and Outputs

The State Machine which controls the Traffic Signal has several inputs and outputs. The inputs are the

- NSSR: The NSSR is activated when a car is over either of the four sensors on the North-South section of the road
- EWSR: The EWSR is activated when a car is over either of the four sensors on the East-West section of the road

  A Timer is used to count the 5 minute and 1 minute traffic signal cycle during the day and night. Two signals LTIME and STIME provide the timing inputs to the State Machine.
- LTIME: The LTIME signal is activated if 5 minutes have elapsed; the signal remains active unless the timer is reset.
- STIME: The STIME signal is activated if 1 minute has elapsed; the signal remains active unless the timer is reset.

The outputs of the State Machine are

- NSGrn: The Green signal controlling the traffic on the North-South section

- NSRed: The Red signal controlling the traffic on the North-South section
- EWGrn: The Green signal controlling the traffic on the East-West section
- EWYel: The Yellow signal controlling the traffic on the East-West section
- EWRed: The Red signal controlling the traffic on the East-West section
- TMRST: The Reset signal which resets the timer after the LTIME or the STIME signals are activated to indicate a time interval of 5 and 1 minutes respectively.

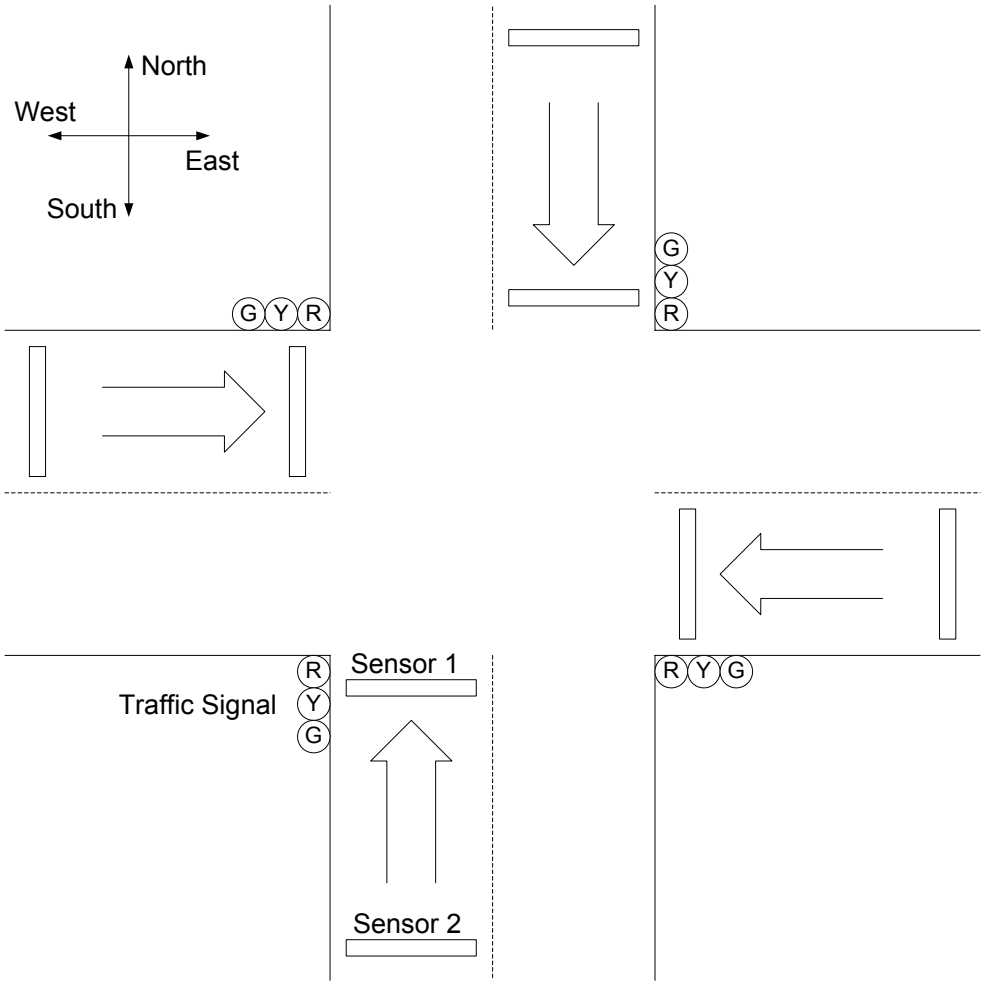


Figure 37.4 The Traffic signals and sensors at a Traffic Intersection

Assuming that the initial state is the NSG (North-South Green) State, during the day time the LTIME timer output is available as an input signal to the State Machine. As long as LTIME is inactive the State Machine remains in its current state NSG, otherwise it switches to the next state NSY (North-South Yellow). During the night time the STIME timer output is available as an input signal to the State machine. If the STIME is inactive the State Machine remains in its current state NSG, otherwise it switches to the NSY state. If a car arrives at the East-West cross road it is made to stop, that is when $EWSR.\overline{NSSR}$, the state NSG remains unchanged. If cars arrive on both the NS and EW sections, both the cars have to be stopped and the state changes to NSY. If a car arrives at the NS section it has to be stopped therefore the state changes to NSY. The information is represented by a flowchart. Figure 37.5

The State Machine can be implemented using a GAL16V8 device. The declaration and definition parts of the ABEL input file for the Traffic Light Controller are described in tables. Table 37.5

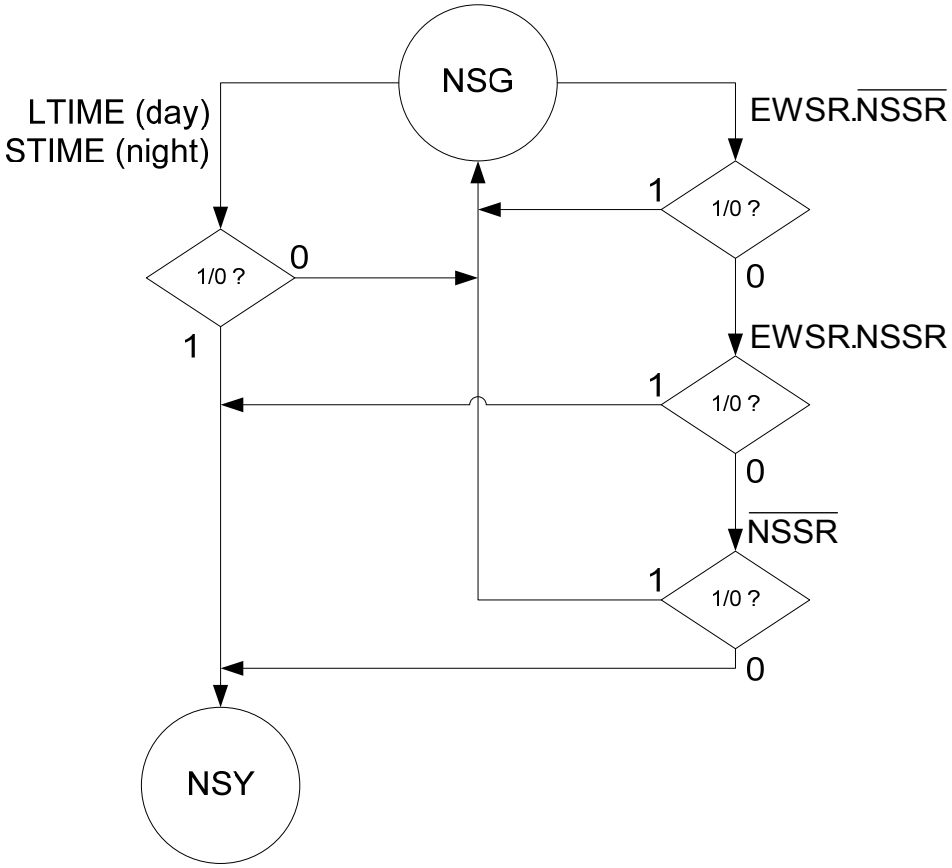


Figure 37.5 Flow chart of conditions which switch the state from NSG to NSY

The pin declaration defines the pins for the CLOCK, NSSR, EWSR, LTIME and STIME inputs to the Traffic Light Controller and the pins for the Q0, Q1, Q2 State variable outputs and the Timer reset TMRST outputs. The state variable outputs are available from the D flip-flops of the OLMC modules and are available in the inverted form, therefore they are defined of type 'reg.invert'. The TMRST signal is an active low signal which resets the counter when the Controller switches to certain states. The TMRST is an active low signal and is based on a combinational circuit therefore its is defined of type 'com.invert'. Table 37.5a

Pin Declaration	
CLOCK, !OLE	pin 1, 11;
NSSR, EWSR, LTIME, STIME	pin 2, 3, 8, 9;
Q0, Q1, Q2	pin 17, 16, 15 ISTYPE 'reg.invert';
TMRST	pin 14 ISTYPE 'com.invert';

Table 37.5a Pin Declarations for the Input and Output pins to the Controller circuit

Definitions
TRSTATE = [Q2, Q1, Q0]; NSG = [0 , 0, 0]; NSY = [0, 0, 1]; NSY2 = [0, 1, 1]; NSR = [0, 1, 0]; EWG = [1, 1, 0]; EWY = [1, 1, 1]; EWY2 = [1, 0, 1]; EWR = [1, 0, 0];

Table 37.5b State definitions for the Traffic Light Controller

State Diagram TRSTATE	
State NSG:	if (!STIME) then NSG else if (LTIME) then NSY else if (EWSR & !NSSR) then NSG else if (EWSR & NSSR) then NSY else if (!NSSR) then NSG else NSY;
State NSY:	goto NSY2;
State NSY2:	goto NSR;
State NSR:	goto EWG;
State EWG:	if (!STIME) then EWG else if (LTIME) then EWY else if (NSSR & !EWSR) then EWG else if (EWSR & NSSR) then EWY else if (!EWSR) then EWG else EWY;
State EWY:	goto EWY2;
State EWY2:	goto EWR;
State EWR:	goto NSG;

Table 37.5c State Diagram for the Traffic Light Controller

The Controller operation is defined by using a State Diagram. Before defining the State Diagram the States have to be defined. The Traffic Light Controller has eight states. Each State is defined using three state variables. The state assignment used restricts the bit changes when switching from one state to the next to a single bit. Table 37.5b.

The ABEL State Diagram statements define each state and the transition to the next state. The transition from the present state NSG to the next state NSG or NSY depending upon the external inputs is defined in State Diagram statements. Table 37.5c.

Lesson No. 38

EQUATION DEFINITION

The Equation definition for the Traffic Controller defines the TRSTATE variable dependent upon the clock transition. The Timer is reset when the state is either NSY2 or EWY2. Table 38.1.

Equations
TRSTATE.CLK = clk; TMRST := (TRSTATE = = NSY2) # (TRSTATE = = EWY2);

Table 38.1 Equation definition for the Traffic Light Controller

The circuit diagram of the Timer connected to the GAL16V8 based Traffic Light Controller is shown. Figure 38.1. The first GAL16V8 is connected to the external inputs NSSR, EWSR and the CLK signal. It is also connected to the two Timer signals LTIME and STIME which determine the Green cycle time of the controller during the day and night respectively. The output of the controller is the TMRST which resets the Timer when the Controller is in state NSY2 or EWY2. The state outputs $\overline{Q0}$, $\overline{Q1}$ and $\overline{Q2}$ are the inverted state outputs, which determine the current state and are also connected to the input of the second GAL16V8 which is programmed for a combinational circuit to turn on/off the North-South and East-West road section traffic signal lights NSRED, NSYEL, NSGRN, EWRED, EWYEL and EWGRN. The chip is also connected to the MANUAL and FLASHCLK inputs. The MANUAL input when activated puts the traffic signal in the Manual Mode where the Yellow signal on the North-South and the East-West road section repeatedly flashes. The flash rate is determined by the FLASHCLK signal.

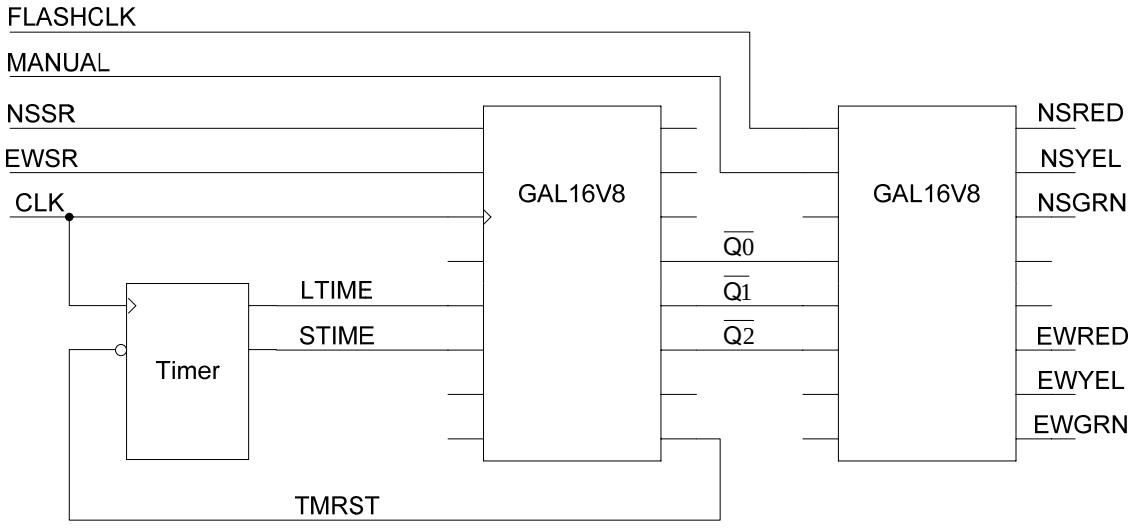


Figure 38.1 The circuit diagram of the Traffic Light Controller

Switching of Traffic Lights

The main definitions and declaration of the ABEL Input file for turning on/off the traffic lights is given. Table 38.2. The Pin Declarations are defined in Table 38.2a. The MANUAL

Yellow lamps. The flash rate is determined by the frequency of the input signal connected at the FLASHCLK input pin. The appropriate lamp is turned on/off on the basis of the Traffic Controller States which are determined by the $\overline{Q0}$, $\overline{Q1}$ and $\overline{Q2}$ inputs. The outputs NSRED, NSYEL, NSGRN, EWRED, EWYEL and EWGRN represent the outputs that are connected to the traffic signal lamps.

Pin Declarations	
FLASHCLK, MANUAL	pin 1, 2;
!Q0, !Q1, !Q2	pin 4, 5, 6;
NSRED, NSYEL, NSGRN	pin 19, 18, 17;
EWRED, EWYEL, EWGRN	pin 14, 13, 12;

Table 38.2a Pin declarations for the turning on/off traffic lamps

The Red, Yellow and Green signals that are turned on/off at different states are shown. Table 38.2b. The equations defining the six outputs that turn on the Red, Yellow and Green signals on the North-South and the East-West road section are defined. Table 38.2c. When the MANUAL signal is set the NSYEL and EWYEL outputs are set to logic high and low depending upon the input signal FLASHCLK.

State	NSGRN	NSYEL	NSRED	EWGRN	EWYEL	EWRED
NSG	on	off	off	off	off	on
NSY	off	on	off	off	off	on
NSY2	off	on	off	off	off	on
NSR	off	off	on	off	off	on
EWG	off	off	on	on	off	off
EWY	off	off	on	off	on	off
EWY2	off	off	on	off	on	off
EWR	off	off	on	off	off	on

Table 38.2b Switching of traffic lamps at different states

Equations

NSRED = !MANUAL & (TRSTATE !=NSG) & (TRSTATE != NSY)
& (TRSTATE != NSY2);
NSYEL = !MANUAL & ((TRSTATE = = NSY) # (TRSTATE = = NSY2))
# MANUAL & FLASHCLK; 	
NSGRN = !MANUAL & (TRSTATE = = NSG);
EWRED = !MANUAL & (TRSTATE !=EWG) & (TRSTATE != EWY)
& (TRSTATE != EWY2);
EWYEL = !MANUAL & ((TRSTATE = = EWY) # (TRSTATE = = EWY2))
# MANUAL & FLASHCLK; 	
EWGRN = !MANUAL & (TRSTATE = = EWG);

Table 38.2c Equation definition for the turning on/off traffic lamps

Analysis of Clocked Synchronous State Machines

Analysis of Clocked Synchronous State Machine is opposite to the Design and Implementation procedure studied. In the analysis procedure an implemented State Machine is described in terms of a state table or a state diagram which specifies all the next states outputs for all possibilities of the current state and input. The analysis of a clocked Synchronous state machine has three basic steps.

- Determine the next-state output functions F and G
  - Next State = F(Current State, Input)
  - Output = G(Current State, Input)
- Use the functions F and G to construct a state/output table
- Draw a State diagram that represents the information in graphical form

The functional behaviour of a flip-flop or a latch is described by a characteristic equation that is a function of its current state and inputs. The characteristic equation does not take into account the exact timing behaviour; it simply describes the functional response. These characteristic equations can be derived from the excitation tables discussed earlier. The excitation table for an S-R latch is shown. Table 38.3. The information in the table is mapped to a three-variable Karnaugh map to derive the Characteristic equation. Figure 38.2.

Flip-flop Inputs		Output Transitions	
S	R	Q_t	Q_{t+1}
0	x	0	0
1	0	0	1
0	1	1	0
x	0	1	1

Table 38.3 S-R flip-flop Transition table

$$Q_{t+1} = S + \overline{R}Q_t$$

SR/Q_t	0	1
00	0	1
01	0	0
11	x	x
10	1	1

Figure 38.2 Characteristic Equation for S-R Latch

The characteristic equations for other flip-flops and latches can be derived similarly. Characteristic equations for some of the flip-flops or latches discussed so far are listed in table 38.4.

Device Type	Characteristic Equation
S-R Latch	$Q_{t+1} = S + \overline{R}Q_t$
D Latch	$Q_{t+1} = D$
Edge-triggered D flip-flop	$Q_{t+1} = D$
J-K flip-flop	$Q_{t+1} = J\overline{Q}_t + \overline{K}Q_t$

Two examples of Synchronous State machines are described.

State Machine Analysis Example1:

A State Machine with two positive-edge triggered D flip-flops is shown. Figure 38.3.

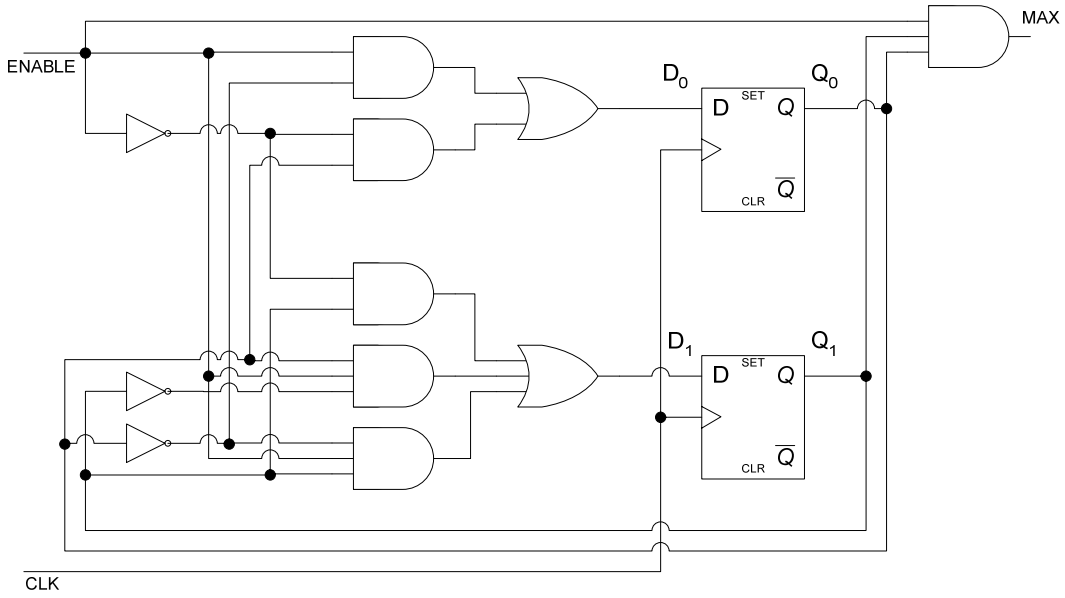


Figure 38.3 Clocked Synchronous State Machine based on D flip-flops

The two flip-flops transfer their D input values to their respective outputs, based on the Characteristic equation for the D-flip-flop. The excitation inputs to the two D flip-flops are determined by the combinational circuit shown. The two excitation equations for D_0 and D_1 inputs are given. Table 38.5. The two Transition equations for the inputs D_0 and D_1 are given. Table 38.6.

D Flip-flop Inputs	Excitation Inputs
D_0	$D_0 = Q_0\overline{EN} + \overline{Q}_0 EN$
D_1	$D_1 = Q_1\overline{EN} + Q_0\overline{Q}_1 EN + \overline{Q}_0 Q_1 EN$

Table 38.5 Excitation Equations for D flip-flop inputs D_0 and D_1

Transition Equations
$Q_{0(t+1)} = Q_0\overline{EN} + \overline{Q}_0 EN$
$Q_{1(t+1)} = Q_1\overline{EN} + Q_0\overline{Q}_1 EN + \overline{Q}_0 Q_1 EN$

Table 38.6 Transition Equations for D flip-flops

In the State Machine, two D flip-flops are used and the outputs Q_0 and Q_1 represent the state variables. Two State variables allow a maximum of four states. From the Transition equations a transition table can be prepared. The Transition Table is shown. Table 38.7.

Present State		Next State ENABLE=0		Next State ENABLE=1	
Q_1	Q_0	Q_1	Q_0	Q_1	Q_0
0	0	0	0	0	1
0	1	0	1	1	0
1	0	1	0	1	1
1	1	1	1	0	0

Table 38.7 Transition Table for D flip-flop based State Machine

The Transition table is very similar to the State table. The state table can be derived from the Transition table by assigning State Names to each State and including the output of the State Machine. The output of the State Machine is determined by the Output Equation

$$MAX = Q_0 Q_1 EN$$

The State Table for a Mealy Machine is given. Table 38.8. The Transition Table represents the function of the Mealy State Machine which is a 2-bit Counter. The Counter doesn't count when the input ENABLE=0 and increments when input ENABLE=1. The output MAX of the State Machine is dependent upon the current state and the Input ENABLE. The State Diagrams for the Mealy State machine derived from the State Table is shown. Figure 38.4.

Present State	Next State ENABLE=0	Next State ENABLE=1	Output MAX ENABLE=0	Output MAX ENABLE=1
A	A	B	0	0
B	B	C	0	0
C	C	D	0	0
D	D	A	0	1

Table 38.8 State table of a Mealy Machine

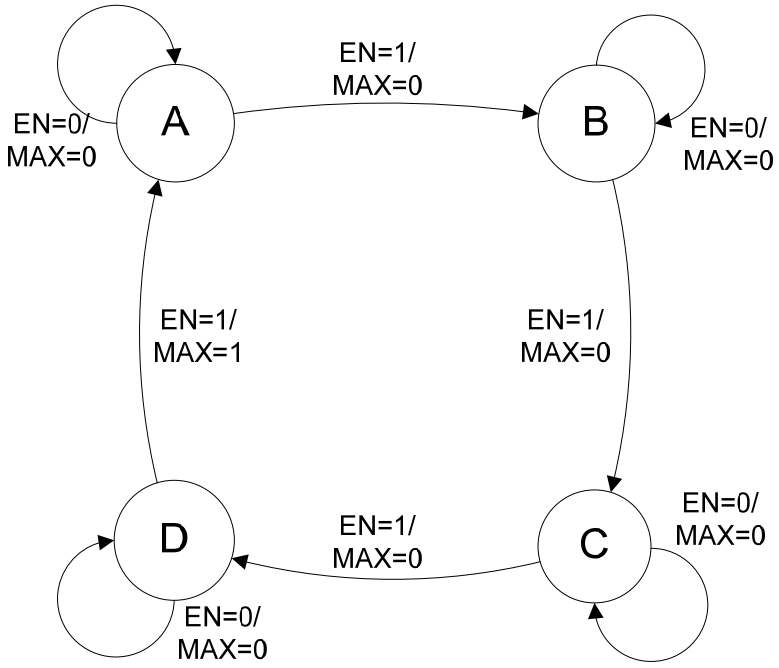


Figure 38.4 State Diagram of a Mealy Machine

In the circuit diagram of the State machine if the output was independent of the external input and only dependent upon the current state of the flip-flops, then the resulting machine is a Moore Machine with a simplified State Table. Table 38.9 and State diagram. Figure 38.5.

Present State	Next State ENABLE=0	Next State ENABLE=1	Output MAX
A	A	B	0
B	B	C	0
C	C	D	0
D	D	A	1

Table 38.9 State table of a Moore Machine

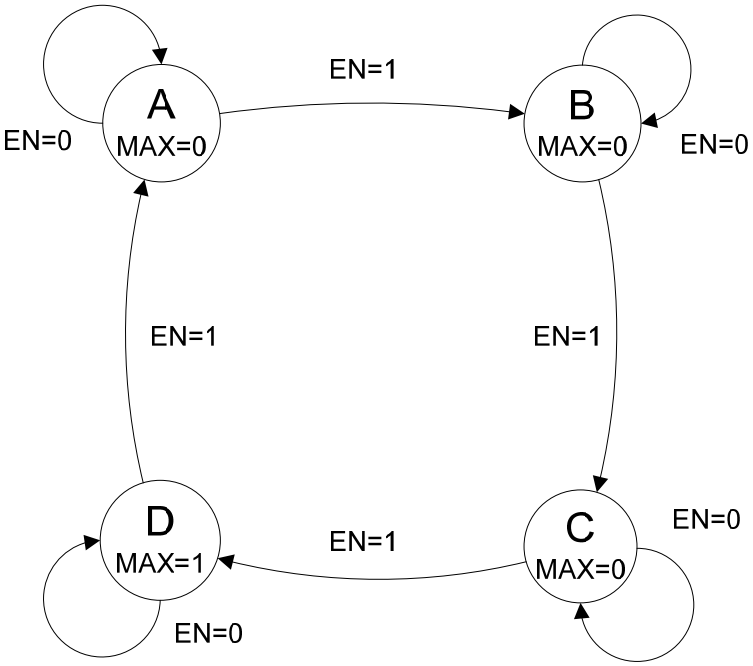


Figure 38.5 State Diagram of a Moore Machine

State Machine Analysis Example2:

A State Machine with two edge triggered J-K flip-flops is shown. Figure 38.6

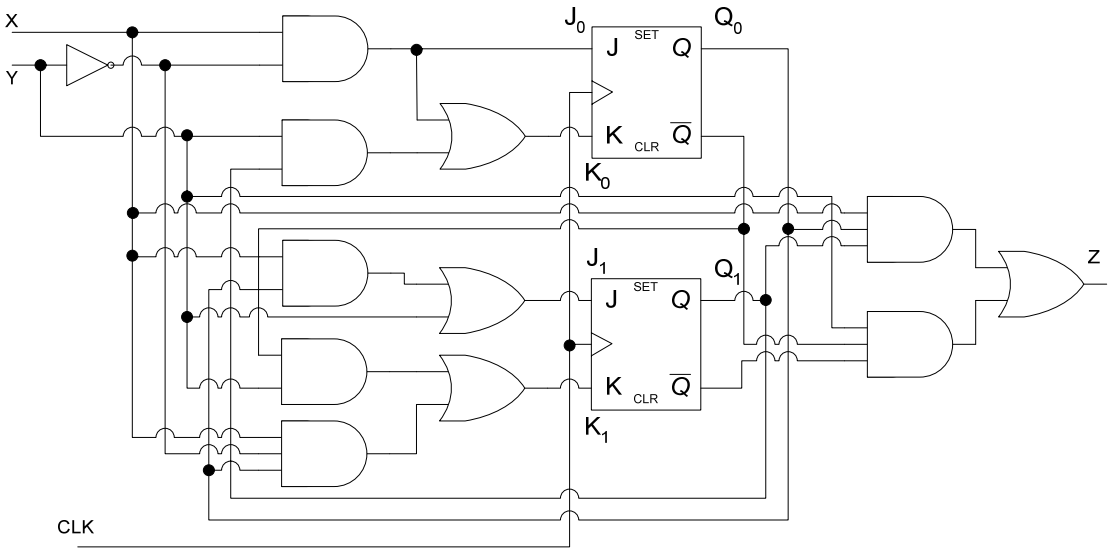


Figure 38.6 Clocked Synchronous State Machine based on J-K flip-flops

The two flip-flops Set/Reset their respective Q outputs based on the J-K input defined by the Characteristic equation for the J-K-flip-flop. The excitation inputs to the two J-K flip-flops are determined by the combinational circuit shown. The two sets of excitation equations for J_0 K_0 and J_1 K_1 inputs are given. Table 38.10. The Transition equations for the inputs J_0 K_0 and J_1 K_1

J-K Flip-flop Inputs	Excitation Inputs
J_0	$J_0 = X\overline{Y}$
K_0	$K_0 = X\overline{Y} + Q_1Y$
J_1	$J_1 = XQ_0 + Y$
K_1	$K_1 = \overline{Q}_0Y + X\overline{Y}Q_0$

Table 38.10 Excitation Equations for J-K flip-flop inputs J_0 K_0 and J_1 K_1

Transition Equations
$Q_{0(t+1)} = J_0\overline{Q}_0 + \overline{K}_0Q_0$
$Q_{0(t+1)} = X\overline{Y}\overline{Q}_0 + (\overline{X\overline{Y} + Q_1Y})Q_0$
$Q_{0(t+1)} = X\overline{Y}\overline{Q}_0 + (\overline{X\overline{Y}})(\overline{Q_1Y})Q_0$
$Q_{0(t+1)} = X\overline{Y}\overline{Q}_0 + (\overline{X} + Y)(\overline{Q}_1 + \overline{Y})Q_0$
$Q_{0(t+1)} = X\overline{Y}\overline{Q}_0 + \overline{X}\overline{Q}_1Q_0 + \overline{X}\overline{Y}Q_0 + Y\overline{Q}_1Q_0$
$Q_{1(t+1)} = J_1\overline{Q}_1 + \overline{K}_1Q_1$
$Q_{1(t+1)} = (XQ_0 + Y)\overline{Q}_1 + (\overline{\overline{Q}_0Y + X\overline{Y}Q_0})Q_1$
$Q_{1(t+1)} = XQ_0\overline{Q}_1 + Y\overline{Q}_1 + (\overline{\overline{Q}_0Y})(\overline{X\overline{Y}Q_0})Q_1$
$Q_{1(t+1)} = XQ_0\overline{Q}_1 + Y\overline{Q}_1 + (Q_0 + \overline{Y})(\overline{X} + Y + \overline{Q}_0)Q_1$
$Q_{1(t+1)} = XQ_0\overline{Q}_1 + Y\overline{Q}_1 + \overline{X}Q_0Q_1 + YQ_0Q_1 + \overline{X}\overline{Y}Q_1 + \overline{Y}\overline{Q}_0Q_1$

Table 38.11 Transition Equations for J-K flip-flops

In the State Machine, two J-K flip-flops are used and the outputs Q_0 and Q_1 represent the state variables. Two State variables allow a maximum of four states. The present state changes to the next state depending upon external inputs X and Y. From the Transition equations a transition table can be prepared. The Transition Table is shown. Table 38.12.

Present State		Next State XY=00		Next State XY=01		Next State XY=10		Next State XY=11	
Q_1	Q_0	Q_1	Q_0	Q_1	Q_0	Q_1	Q_0	Q_1	Q_0
0	0	0	0	1	0	0	1	1	0
0	1	0	1	1	1	1	0	1	1
1	0	1	0	0	0	1	1	0	0
1	1	1	1	1	0	0	0	1	0

Table 38.12 Transition Table for D flip-flop based State Machine

The Transition table is very similar to the State table. The state table can be derived from the Transition table by assigning State Names to each State and including the output of the State Machine. The output of the State Machine is determined by the Output Equation

$$Z = XQ_0Q_1 + Y\overline{Q}_0\overline{Q}_1$$

The State Table for a Mealy Machine is given. Table 38.13. The State Diagram for the Mealy State machine derived from the State Table is shown. Figure 38.7.

Present State	Next State				Output Z			
	XY 00	XY 01	XY 10	XY 11	XY 00	XY 01	XY 10	XY 11
A	A	C	B	C	0	1	0	1
B	B	D	C	D	0	0	0	0
C	C	A	D	A	0	0	0	0
D	D	C	A	C	0	0	1	1

Table 38.13 State Table of a Mealy Machine

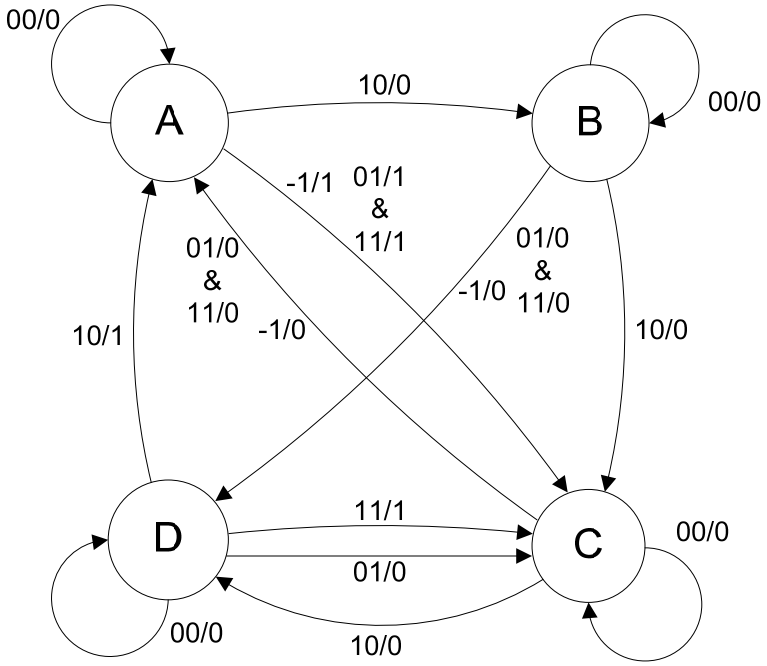


Figure 38.7 State Diagram of a Mealy Machine

Lesson No. 39

MEMORY

Sequential circuits can not operate without a memory element. Memory elements used in Sequential circuits are relatively small and store few binary bits of information. Large memories capable of storing very large amounts of information are used in Computer systems. A computer which executes an application program has the application stored in the form of program instructions in large memories.

Memories store data in units that have one, four, eight or higher number of bits. Smallest unit of binary data is a bit. Data is also handled in a 4-bit unit called a Nibble. In many applications the data is handled as an 8-bit unit called a byte, which is a combination of two 4-bit units that are called Nibbles. A complete unit of information is sometimes called a Word and consists of one or more bytes.

Each storage element of a memory can either store a logic 0 or a logic 1 and is called a cell. Memories are arranged in an array and each cell can be identified by specifying a row and a column number. Figure 39.1. Each square in the diagram represents a memory cell capable of storing a binary 1 or 0. The first eight bits of binary information 11001010 in the first row are stored in eight cells. The addresses of the eight consecutive cells staring from the left most cell are (1,1), (1,2), (1,3), (1,4), (1,5), (1,6), (1,7) and (1,8) representing the first row and columns 1 to 8 respectively. Individual cells at row 5 and column 3 have a binary 1 and a cell at row 6 and column 7 have a binary 0 stored.

Row								
1	1	1	0	0	1	0	1	0
2	0	1	0	0	1	0	1	0
3	0	1	0	0	1	1	0	1
4	1	0	1	0	0	1	0	1
5	1	0	1	1	1	0	1	1
6	0	1	0	1	0	0	0	0
7	1	0	0	1	0	1	1	0
8	0	0	1	1	0	0	0	0
	1	2	3	4	5	6	7	8

Column

Figure 39.1 64-cell Memory Array

Memory Organization

The Memory array can be organized in several ways depending on the unit of data.

has eight locations and each location stores a single byte. The 64-cell array organized as 4 x 16 cell array stores 16 nibbles and if organized as 1 x 64 stores 64 single bit values. The 4 x 16 memory array allows data to be accessed in the form of 4-bit nibbles. The 1 x 64 array allows data to be accessed in units of 1 bit. Figure 39.2.

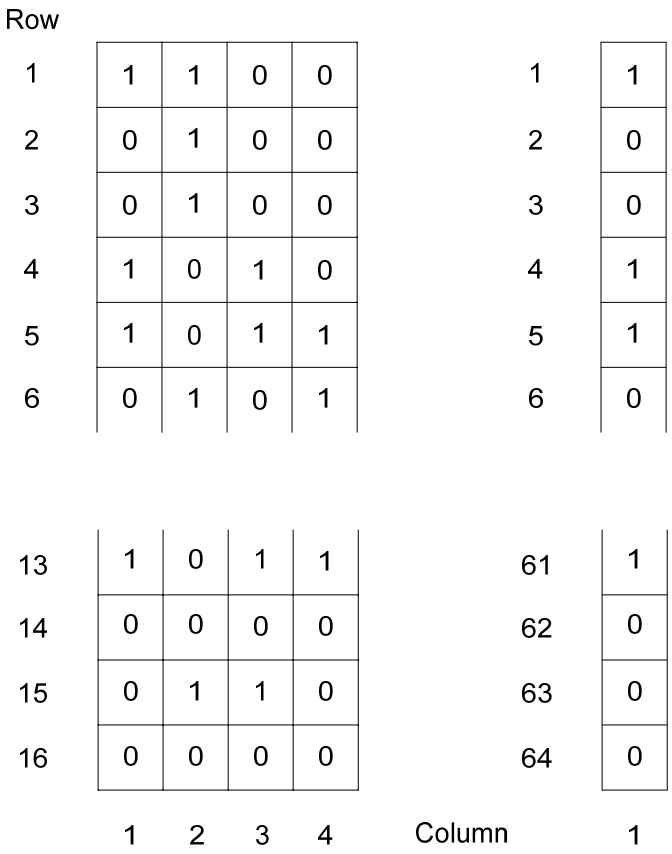


Figure 39.2 Memory Organized as 4 x 16 and 1 x 64 Arrays

A memory is identified by the number of units it can store times the unit size, thus the 8 x 8 memory is identified as an 8 Byte memory, the 16 x 4 memory is used as a 16 Nibble memory and the 64 x 1 is known as a 64 bit memory. Practical memory chips are organized as 16 K x 8 memory, storing 16K bytes or 16 x 1024 = 16384 bytes. A 32 K x 4 memory stores 32K nibbles or 32 x 1024 = 32768 nibbles.

Memory Capacity and Density

Each memory array has a maximum capacity to store information in the form of bits. Thus a 16 K x 8 memory, stores 16K bytes or 16 x 1024 = 16384 bytes or 131072 bits. A 32 K x 4 memory stores 32K nibbles or 32 x 1024 = 32768 nibbles or 131072 bits. The total number of cells in each case is 131072. Memory density on the other hand specifies the number of bits stored per unit area. More the number of bits stored in a unit area more dense the memory, that is, more bits are stored in less space. The capacity and the density of a memory are determined by the total number of cells implemented in a unit area.

Memory Signals and Basic Operations on Memory

Two basic operations are performed on memories, that is, reading of information from the memory and writing of data to the memory. To support the two read and write operations memories provide several signals. Figure 39.3.

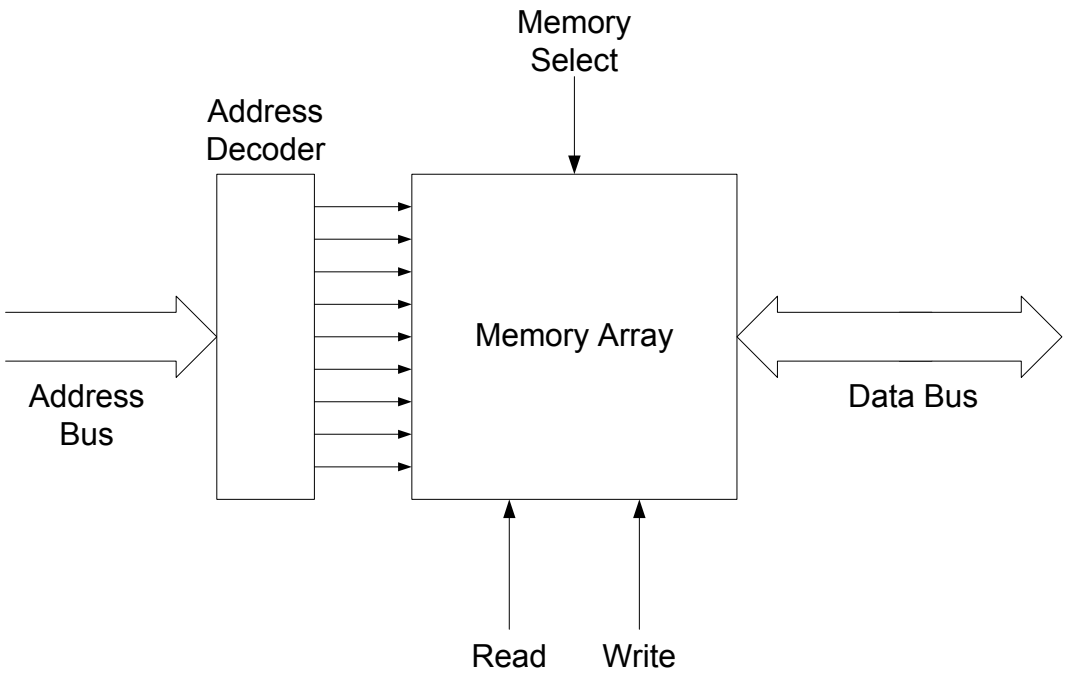


Figure 39.3 Block Diagram of a Read-write Memory

Read and Write Signals

Read/Write signals are required to configure the memory for read and write operation. Memory chips have a single Read/Write signal. When the signal is set to high it allows data to be read from the memory. When the signal is set to low data is written into the memory. Some memory chips have two separate Read and Write signals. The read and write signals are separately asserted to control the Read and Write operation.

Address Signals

Address signals are required to specify the location in the memory from which information is accessed (read or written). A set of parallel address lines known as the address bus carry the address information. The number of bits (lines) comprising the address bus depends upon the size of the memory. For example, a memory having four locations to store data has four unique addresses (00, 01, 10, 11) specified by a 2-bit address bus. The size of the address bus depends upon the total addressable locations specified by the formula 2^n, where n is the number of bits. Thus 2^4=16 (n=4) specifies 4 bits to uniquely identify 16 different locations.

Data Signals

Data lines are required to retrieve the information from the memory array during a read operation and to provide the data that is to be stored in the memory during a write operation. As the memory reads or writes one data unit at a time therefore the data lines should be equal to the number of data bits stored at each addressable location in the memory. A memory

lines or the size of the data bus should be 8-bits or 1 byte. A memory organized to store nibble data values requires a 4-bit wide data bus. Generally, the wider the data bus more data can be accessed at each read or write operation.

Memory Select or Enable Signal

In a computer system there are more than one memory chips to store program information. At any particular instant a read or write operation is carried out on a single addressable location. The unique location can only be accessed in one of the several memory chips, thus a single memory chip has to be selected before a read or write operation can be carried out. All memory chips have a chip enable or chip select signal which has to be activated before the memory can be accessed.

Memory Read operation

Memory Read operation is carried out by first selecting the memory chip by activating the Memory Select signal. The Read signal is asserted to configure the memory circuitry for reading data from the memory. An address (100) is applied on the Address Lines. The internal address decoder of the memory decodes the address and selects one unique row from which data is read. Figure 39.4.

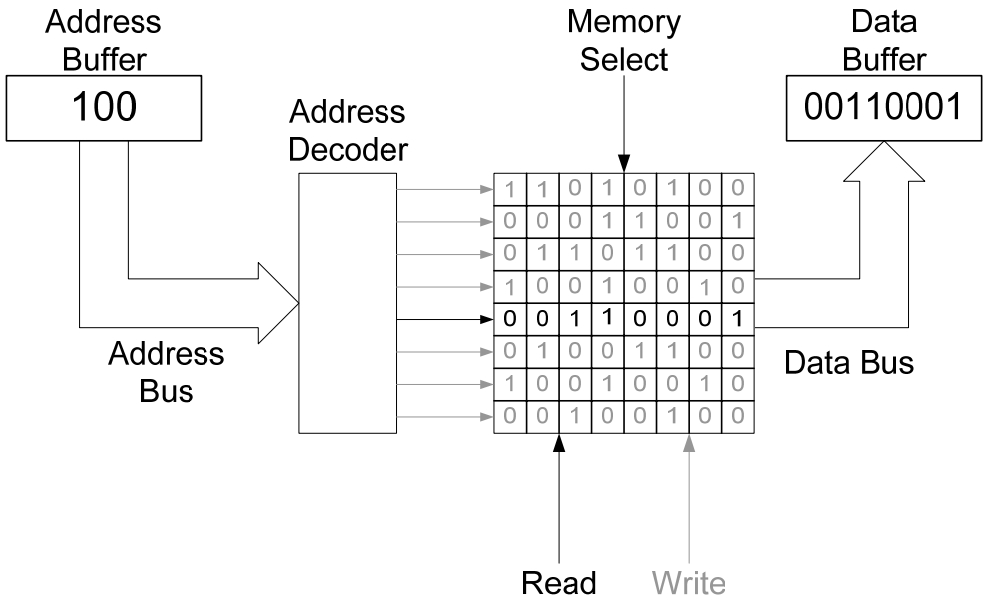


Figure 39.4 Memory Read Operation

The address of the location in the memory from which data is to be read is supplied by the microprocessor. The microprocessor stores the address in its address buffer. The data read from the memory is stored in a data buffer inside the microprocessor. In the diagram shown, a microprocessor places an address 100 on its external address bus connected to the address lines of the memory. The internal address decoder of the memory decodes the address 100 and activates a row select line which selects the row location 4. The data (00110001) at the location is read from the memory and placed on the data bus where it is latched by the microprocessor and stored in its data buffer.

Memory Write operation

Memory Write operation is carried out by first selecting the memory chip by activating

writing data to the memory. An address (011) is placed on the Address Lines by the microprocessor. The internal address decoder of the memory decodes the address and selects one unique row select line which selects the row location 3. The data (10110010) to be written to the selected memory location is placed on the external data bus by the microprocessor which is stored in the selected location. Figure 39. 5

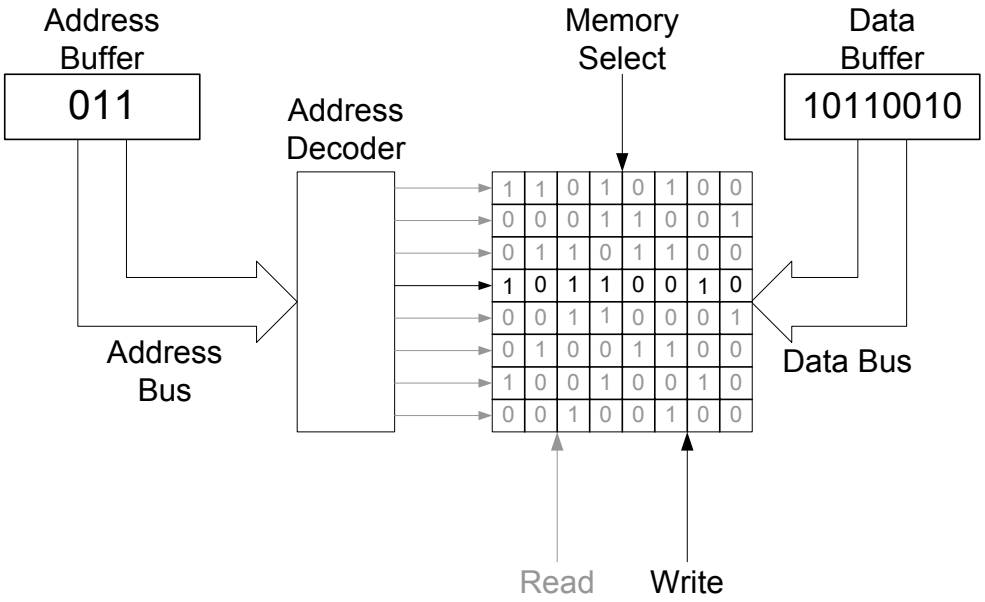


Figure 39.5 Memory Write Operation

Memory Types

Two major categories of memory chips are the Random Access Memory (RAM) and Read-Only Memory (ROM). RAM allows a read or write operation to be carried out at any address. All locations are accessible in equal time. RAM memories do not store permanent data. As soon as the power supply to the memory chip is turned off, the entire data stored in the memory is lost permanently. RAM memories are also known as volatile memories as they lose data when the power is turned off.

ROM chips retain data permanently even if the power to a ROM chip is turned off. ROM chips are also known as non-volatile memory chips due to their ability to retain data permanently. Since ROM chips are read only, therefore user can not write any information to ROM chips. ROM chips are programmed by the manufacturer and contain important information which is required to start (Boot Up) the computer.

Random Access Memory (RAM)

RAM is divided into two types, Static RAM which uses flip-flops as storage elements and Dynamic RAM which uses capacitors to store binary information. In a Static RAM each cell which is capable of storing a binary 0 or 1 is made up of a flip-flop which retains information as long as power continues to be supplied to the flip-flop. Dynamic RAM on the other hand uses a capacitor to store a single bit of data. To store binary 1, the capacitor is charged and to store binary 0, the capacitor is in the uncharged state. Capacitors over a period of time lose their charge and unless the Capacitors are refreshed the information stored by the capacitor is lost. Dynamic memories periodically charge their capacitors by implementing a Refresh cycle. Static Memories are faster than Dynamic memories therefore

memories on the other hand have a high density and can store much more data per unit area and at a lesser cost. Dynamic memories have a high storage density, as capacitors are simpler to implement and occupy a very small semiconductor area as compared to flip-flops.

Static RAM

Each cell of a Static RAM is implemented using a flip-flop which is implemented using several MOSFET transistors. External power is required to operate the transistors. As long as the external power is applied the static memory cell retains the data. The circuit of a single flip-flop based cell which can store a binary 0 or 1 is shown. Figure 39.6.

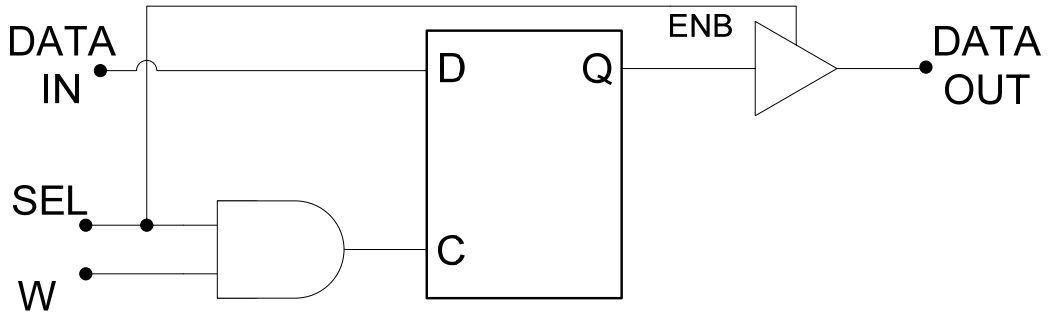


Figure 39.6 Circuit diagram of a Static Memory Cell based on a flip-flop

The flip-flop used to store a binary bit works like a latch. When the SEL signal is activated, the output buffer is enabled allowing data to be read out from the memory cell. When both the SEL and W(rite) signals are activated the latch is configured in the transparent mode and the data applied at the Data In line flows through the latch to the output. The Data In and Data Out lines can be connected together to form a bi-directional line which does not cause any problems with the reading or writing of data. This is possible as the read and write operations takes place at different time intervals.

The flip-flop based cells are combined to form an array. Additional logic is added to select cells at appropriate locations and to read and write data. A 3 x 8 decoder decodes a 3-bit address to select any one of the eight locations comprising of a group of 4-cells. For example, when the address is 000, the first output line of the 3 x 8 decoder is activated which is connected to the SEL input of the four latches in the first row. Similarly, address 111 activates the eighth output line of the 3 x 8 decoder which selects the four latches in the last row (location). The memory array has four Data In lines to store the 4-bit data values at the eight locations. Data In 3 and Data In 0 represents the most and least significant bits of the 4-bit data respectively. The four Data In lines connect the Data In inputs of all the latches in each column respectively. The memory array also has four Data Out lines, each data line connects the output of each latch in a column. The read and write operations are controlled through the three signals W, CS and OE. The Chip Select (CS) signal along with the Output Enable (OE) signal enable each of the four tri-state buffers connected to end of each Data Out line. When data is to be read from a memory array, the memory chip is selected and the output enabled. The Write (W) signal along with the CS signal are used to write data into any 4-bit location. Figure 39.7.

To write data 1001 at the 6^{th} memory location, the address A_2, A_1 and A_0 bits are set to 110 which select the 6^{th} row of the memory array. The data 1001 is placed at the four Data In

row to transparent mode allowing data 1001 applied at the four Data In lines to be available at the Q outputs of the four latches respectively. As soon as the CS and W signals are deactivated, the latches store the data value.

A 16K x 8 memory is shown. Figure 39.8. The memory is capable of storing byte values in 16 x 1024 locations. To address these unique locations, fourteen address lines are required. The memory has eight bi-directional data lines through which data is read/written at selected memory locations. The three CS, WE and OE are shown to be active low.

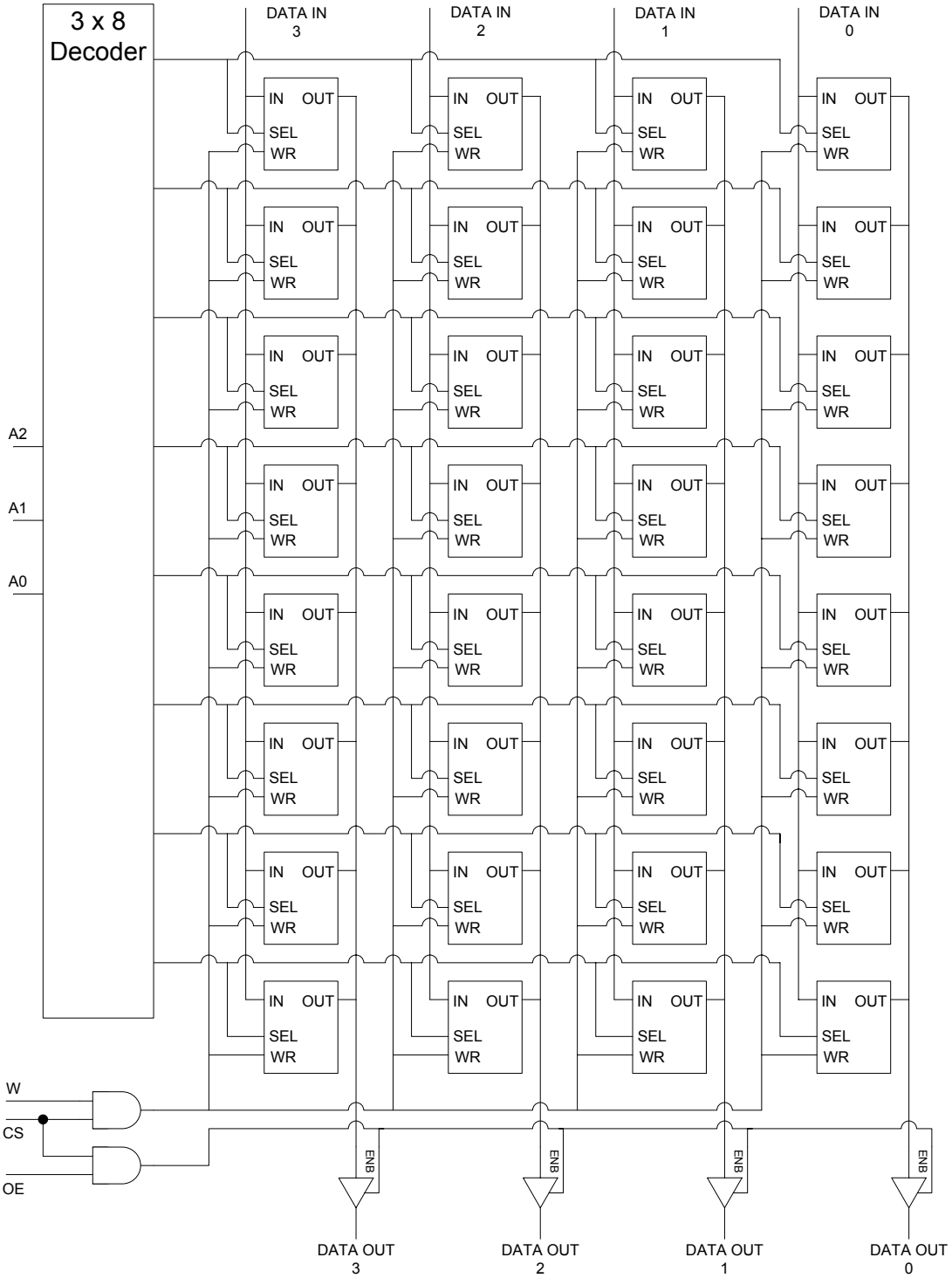


Figure 39.7 Internal Structure of a 8 x 4 Static RAM

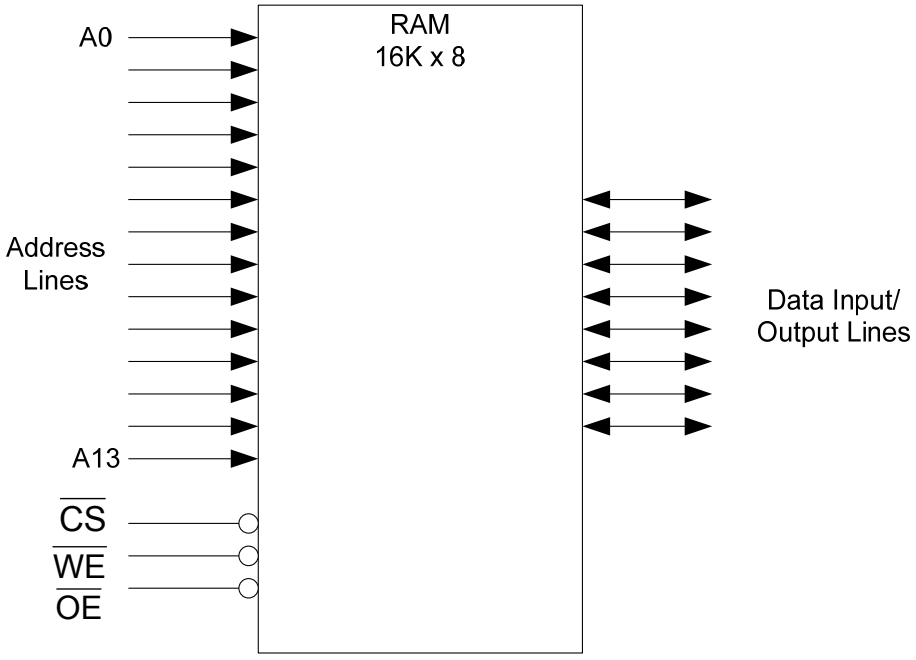


Figure 39. 8 16K x 8 Static RAM

Lesson No. 40

DECODING LARGE MEMORIES

Large memories such as the 16 KB memory have row and column decoders that split the input address into a row address and a column address and activate a row and column select lines respectively. The row and column select lines select a location in the memory array. The memory is arranged in a two-dimensional manner instead of the linear address method discussed earlier. The reason for adopting a row and column decoder to independently but simultaneously select a location by its unique row and column number is to speed up the decoding process. As the memories get larger the decoders that decode and select a unique memory location also become very large with large number of gates. Due to the increased level of gates of the decoding circuitry the delay in decoding the input address increases, thereby slowing the memory access. A large address split into row and column addresses and separately decoded by row and column decoders requires comparatively smaller decoders with fewer number of gates resulting in fast decoding times and thereby faster memory access. The block diagram of a memory using row and column decoders is shown. Figure 40.1.

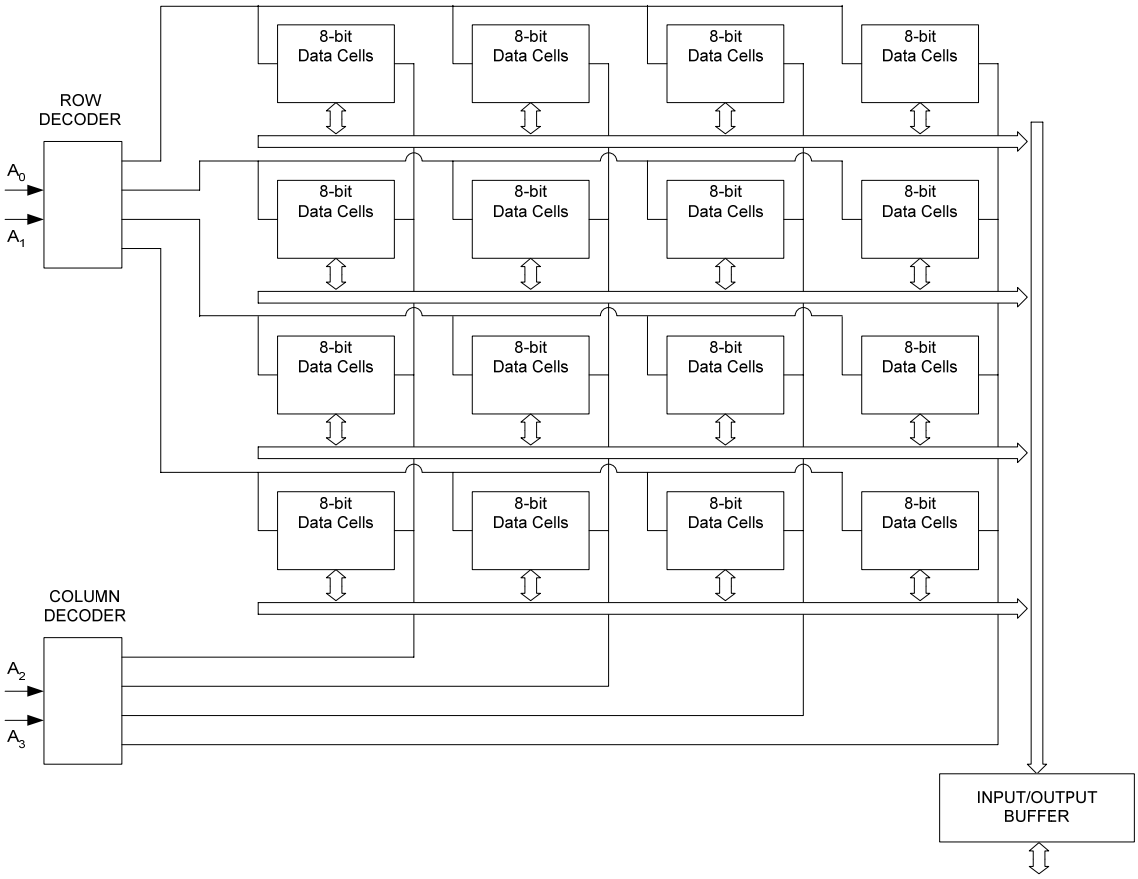


Figure 40.1 Memory array decoded by Row and Columns Decoders

Detail circuitry of the Input/Output Buffer is shown which manages the control of the Data In and Data Out lines. Figure 40.2. When the W, write signal is active and the memory chip is selected CS, the top AND gate is selected and the bottom AND gate is disabled. The data applied at the Data In/Out bi-directional lines is stored in the selected latches. When the

which enables the tri-state buffers connected at the end of the data out lines leading from the latch outputs. This allows data from the selected latches to be available on the Data In/Out lines.

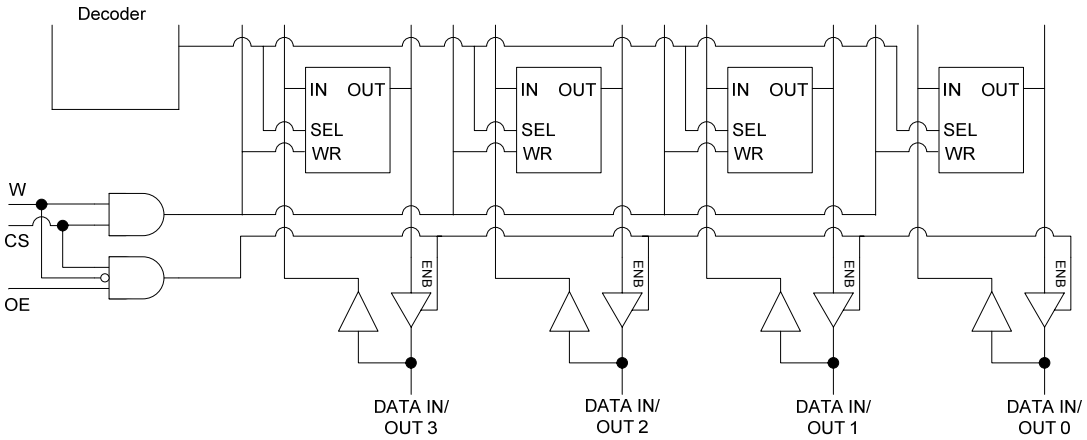


Figure 40.2 Input/Output Data Circuit

The Reading and Writing of data is done by activating the various memory signals in a proper sequence. The Memory Read Cycle controls the memory for reading of data and a Memory Write Cycle controls the memory for writing of data.

Memory Read Cycle

The timing diagram of the read cycle is shown. Figure 40.3. To read data from the memory, the Read Cycle is initiated by applying the address signals. The valid address needs to be maintained stable for a specified duration t_{RC} the read cycle time. Next, the $\overline{CS}$ and the $\overline{OE}$ signals are activated, after a delay of t_{GQ}, the output enable access time measured with respect to the high-to-low transition of the $\overline{OE}$ signal, valid data appears on the data lines. The t_{AQ}, address access time is measured from the beginning of the valid address that appears on the address lines to the appearance of valid data on the data lines. The time t_{EQ} measures the chip enable access time which is the time for the valid data to appear after the high-to-low transition of the chip select signal $\overline{CS}$.

Memory Write Cycle

The timing diagram of the write cycle is shown. Figure 40.4. To write data to the memory, the Write Cycle is initiated by applying the address signals. The valid address needs to be maintained stable for a specified duration t_{WC} the write cycle time. Next, the $\overline{CS}$ and the $\overline{WE}$ signals are activated. The write enable signal $\overline{WE}$ is activated after a minimum time of $t_{s(A)}$ the address setup time which is measured from the beginning of the valid address. The time for which the $\overline{WE}$ signal remains active is known as the write pulse width. After the $\overline{WE}$ signal becomes active the data that is to be written in the memory at the addressed location is applied at the data lines. The $\overline{WE}$ signal must remain valid after data is applied at the data input lines and must remain valid for a minimum time duration t_{WD}. The data must remain valid for a time $t_{h(D)}$, hold time after the $\overline{WE}$ signal is deactivated.

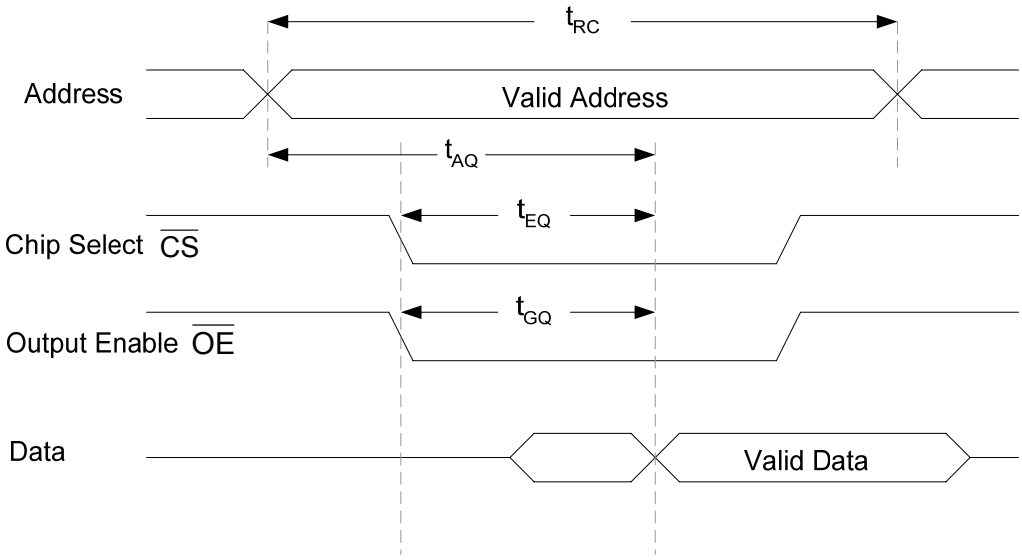


Figure 40.3 Timing diagram of a Read Cycle

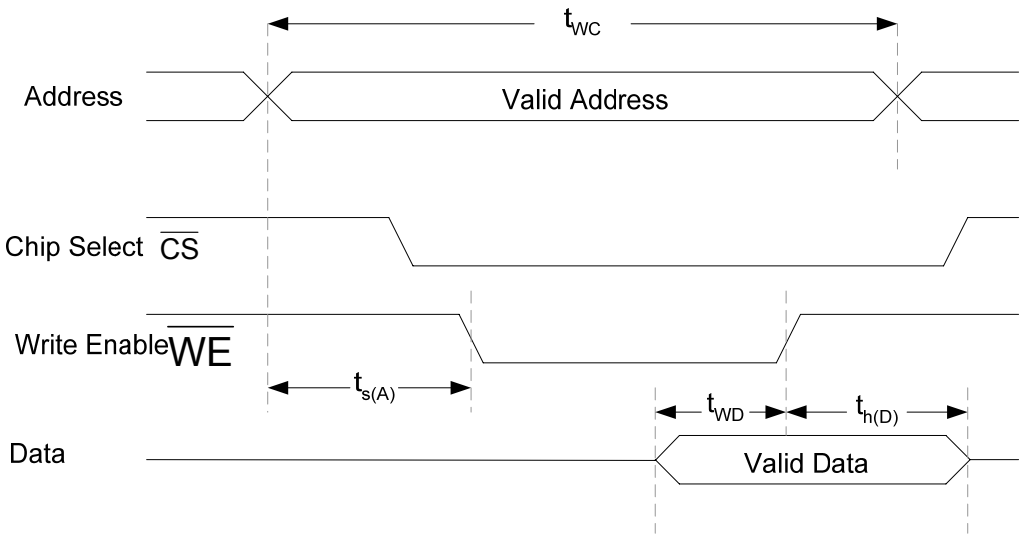


Figure 40.4 Timing diagram of a Write Cycle

Synchronous Burst SRAM

RAM chips are subdivided into Asynchronous RAM (ASRAM) and Synchronous Burst RAM (SB SRAM).The Static memory described is an Asynchronous SRAM, the operation of which does not depend upon the clock signal. The read and write operations are carried out asynchronously. Synchronous SRAM uses a clock signal which is used by the microprocessor to synchronize its activities to synchronize the read and write operations for faster operation. The block diagram of a Synchronous Burst SRAM is shown. Figure 40.5.

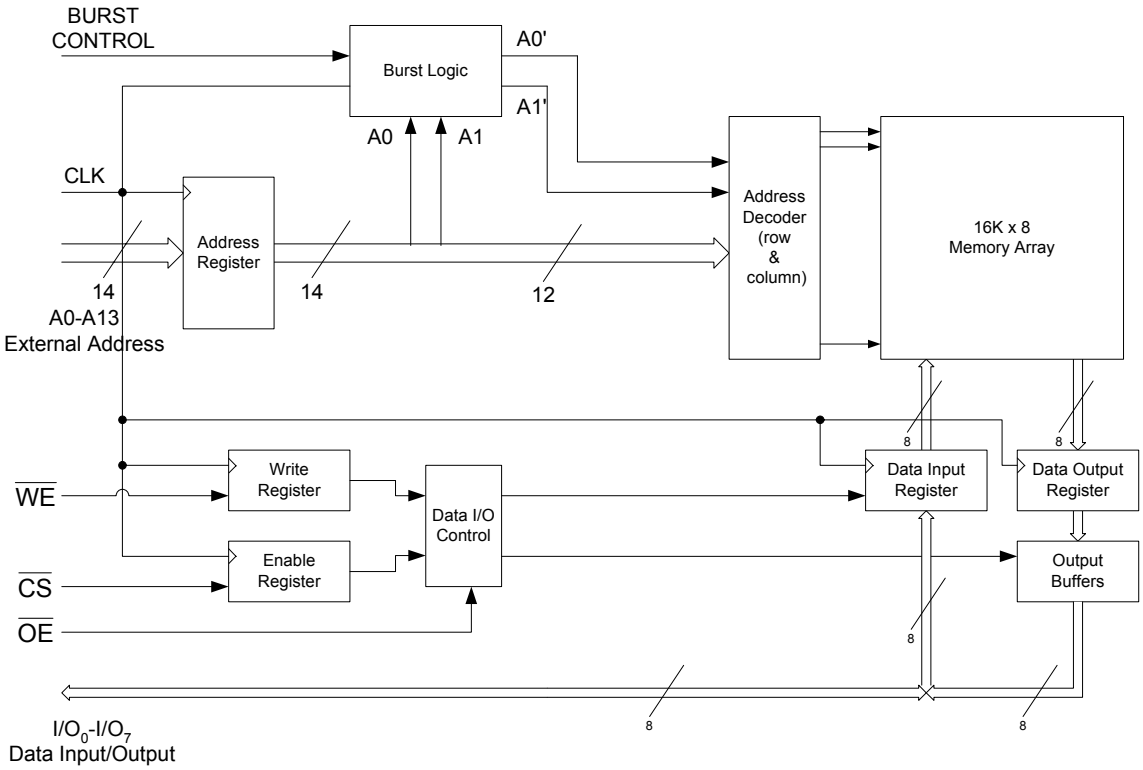


Figure 40.5 Block diagram of a Synchronous Burst RAM

Synchronous RAM is very similar to the Asynchronous RAM, in terms of the memory array, the address decoders, read/write and enable inputs. In the Asynchronous memory the various input signals are asynchronous and are not tied to the clock, whereas in the Synchronous memory all the inputs are synchronized with respect to the clock and are latched into their various registers on an active clock pulse edge. In the diagram, the external address, the $\overline{WE}$ and the $\overline{CS}$ external signals are latched in on a positive clock transition simultaneously. The data that is to be written into the memory is also latched into the Data Input Register at the same positive clock transition. For a read operation the data is latched in the Data Output register on the positive clock transition. There are two variations of the Synchronous SRAM, the Flow-through and the Pipelined SRAM. In the Flow-through SRAM there is no Data Output Register so the data is asynchronously available on the data lines during a read operation. In the Pipelined version there is a Data Output Register which latches in the data read from the memory array.

The Synchronous SRAM also has a Burst feature which allows the Synchronous SRAM to read or write up to four locations using a single address. When an external address is latched in by the Address register, the lower two bits of the address are connected to the Burst logic circuitry which internally increments the addresses at each clock transition producing four different addresses 00, 01, 10 and 11. For example, if an external base address of 37A0 H is stored in the Address Register, the Burst Logic circuitry produces addresses 37A0, 37A1, 37A2 and 37A3. The detailed Burst Logic circuit is shown. Figure 40.6.

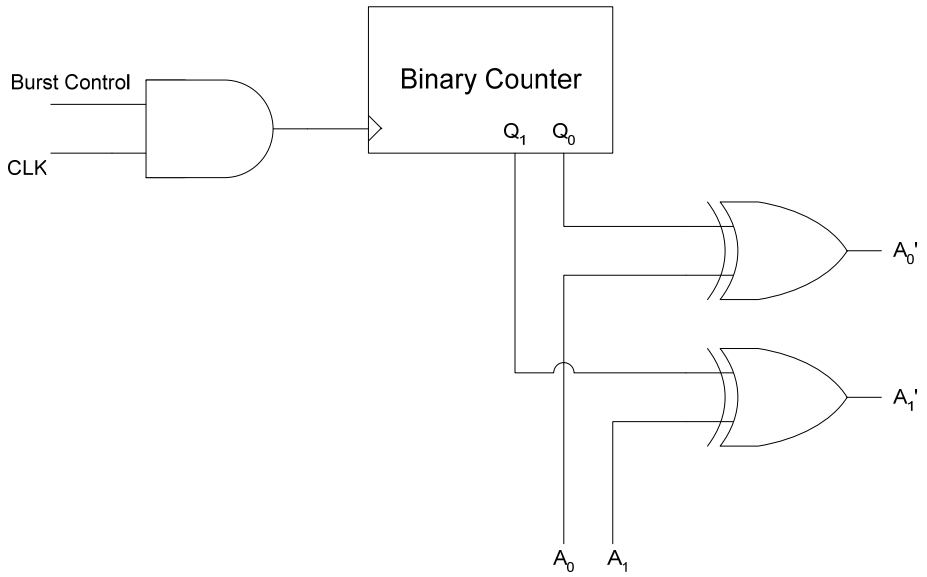


Figure 40.6 Burst Logic Circuit

Dynamic RAM

A static RAM uses a latch to store a single bit of information. Four gates are used to implement a latch. In terms of transistors, 4 to 6 transistors are required to implement a single storage cell. In order to build memories with higher densities, a single transistor is used to store a binary value. A single transistor can not store a binary value however it is used to charge and discharge a capacitor. A single memory cell is thus implemented using a single transistor and a capacitor which occupy lesser space as compared to the six transistors which are used to implement a single Static RAM cell. Thus the density of the capacitor based memory is significantly increased. The capacitor based memory is known as a Dynamic RAM (DRAM). The drawback of DRAM is the discharging of the capacitor over a period of time. Unless the capacitor is periodically recharged all the information stored in terms of binary bits in a capacitor based memory array is lost. The extra circuitry required to refresh the capacitor complicates the operation of the DRAM.

The circuit diagram of a single DRAM capacitor based memory cell is shown. Fig 40.7a. The capacitor is connected through a MOSFET which connects or disconnects the column line at B to the capacitor at D. If the row is set at logic high the MOSFET connects the column line to the capacitor. If the row line is set to logic low the MOSFET disconnects the column line form the capacitor.

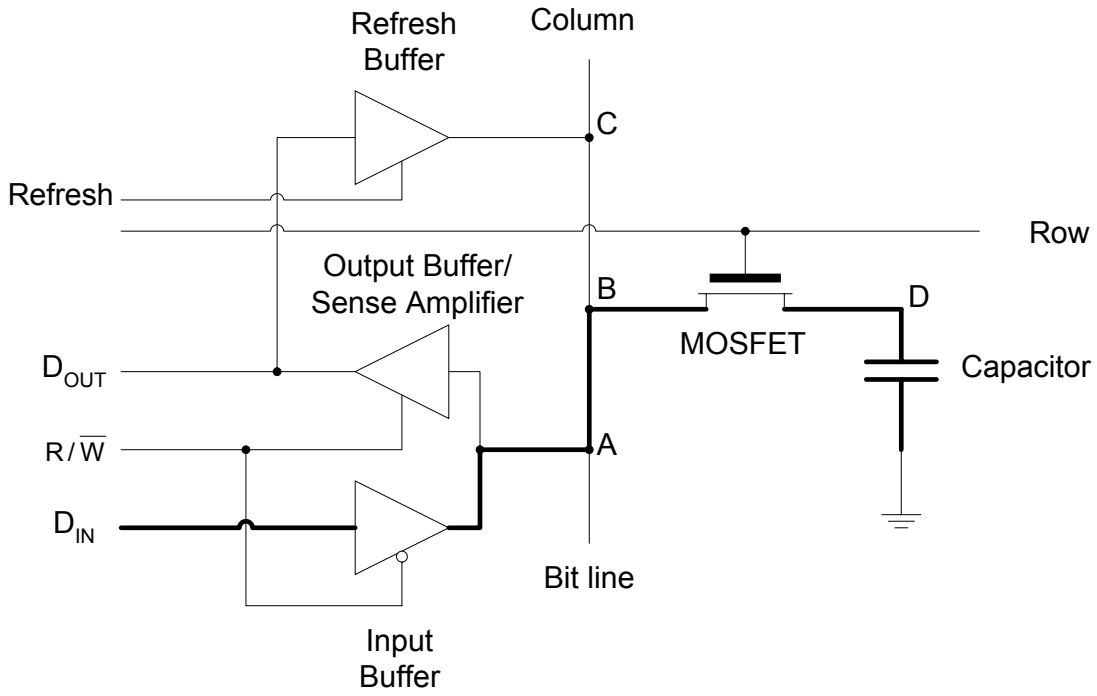


Figure 40.7a Writing a 1 or 0 into the DRAM cell

A write operation allows a logic 1 or 0 to be stored in a DRAM cell (capacitor). The appropriate cell is selected by specifying the address of the memory location which is decoded and the row connecting the desired cell is activated. The $R/\overline{W}$ signal is set to logic low indicating a write operation which enables the tri-state Input Buffer. The logic 1 which is to be stored in the memory cell is applied at the D_{IN} data line which is available at A on the column line. The row line is selected (set to logic high) which allows the MOSFET to connect column B to capacitor D. The capacitor is charged to logic 1 voltage level via ABD. Figure 40.7a. A Write operation to store logic 0 in a DRAM cell is similar. The appropriate row is selected by specifying the storage location address. The $R/\overline{W}$ signal is set to logic low which enables the Input Buffer. The logic 0 to be stored in the DRAM cell is applied at the D_{IN} which is stored on the capacitor via ABD. Figure 40.7a. The thick line in the diagram indicates the data path from D_{IN} to the storage capacitor.

The read operation is accomplished by specifying the address of the location from which data is to be read. The DRAM address decoder activates the appropriate row. The $R/\overline{W}$ signal is set to logic high which enables the output buffer. The logic 1 or 0 stored on the capacitor is available at D_{OUT} through path DBA. Figure 40.7b.

The capacitor can not retain the charge, therefore it has to be periodically charged through a refresh cycle. The Refresh Buffer is enabled by setting the Refresh signal to high. The input of the Refresh Buffer is connected to the output buffer/sense amplifier. The $R/\overline{W}$ signal is set to logic high during the Refresh cycle allowing the information stored on the capacitor to be available at the output of the Output Buffer/Sense amplifier. The information is feed back to the capacitor through the Refresh Buffer via path CBD. Figure 40.7c.

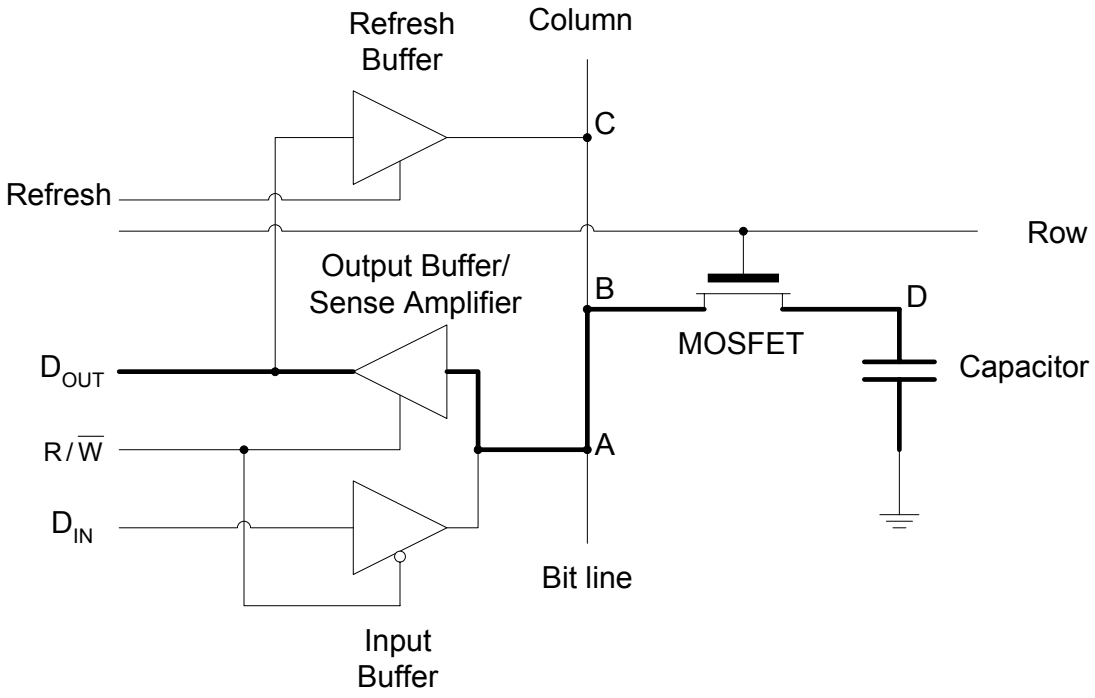


Figure 40.7b Reading a 0 or 1 from the DRAM cell

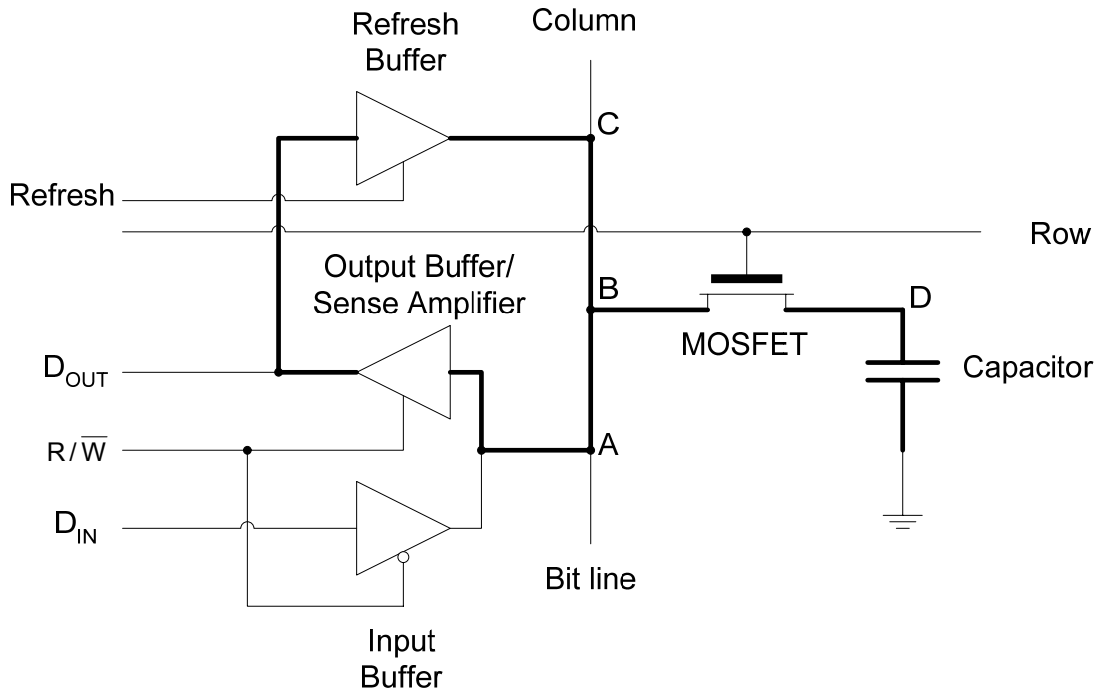


Figure 40.7c Refreshing a DRAM cell

Address Multiplexing

DRAM chips use address multiplexing to reduce the number of address lines by half. The address required to select a memory location is split into row and column addresses. To

access a DRAM location for reading or writing of information the row address is first applied at the address lines. The row address is latched by the Row Address Latch of the DRAM memory chip. The column address is applied next at the same address lines. The column address is latched by the Column Address Latch. Two signals $\overline{RAS}$ and $\overline{CAS}$ are used as strobe signals to control the Row Address and Column Address latches respectively. The external address lines are multiplexed as the same set of address lines are used to apply the row address and the column address at different time instances. The outputs of the Row Address Latch and the Column Address Latch are connected to the Row and Column Decoders which select a single row and column line selecting the storage cell to be accessed. Figure 40.8

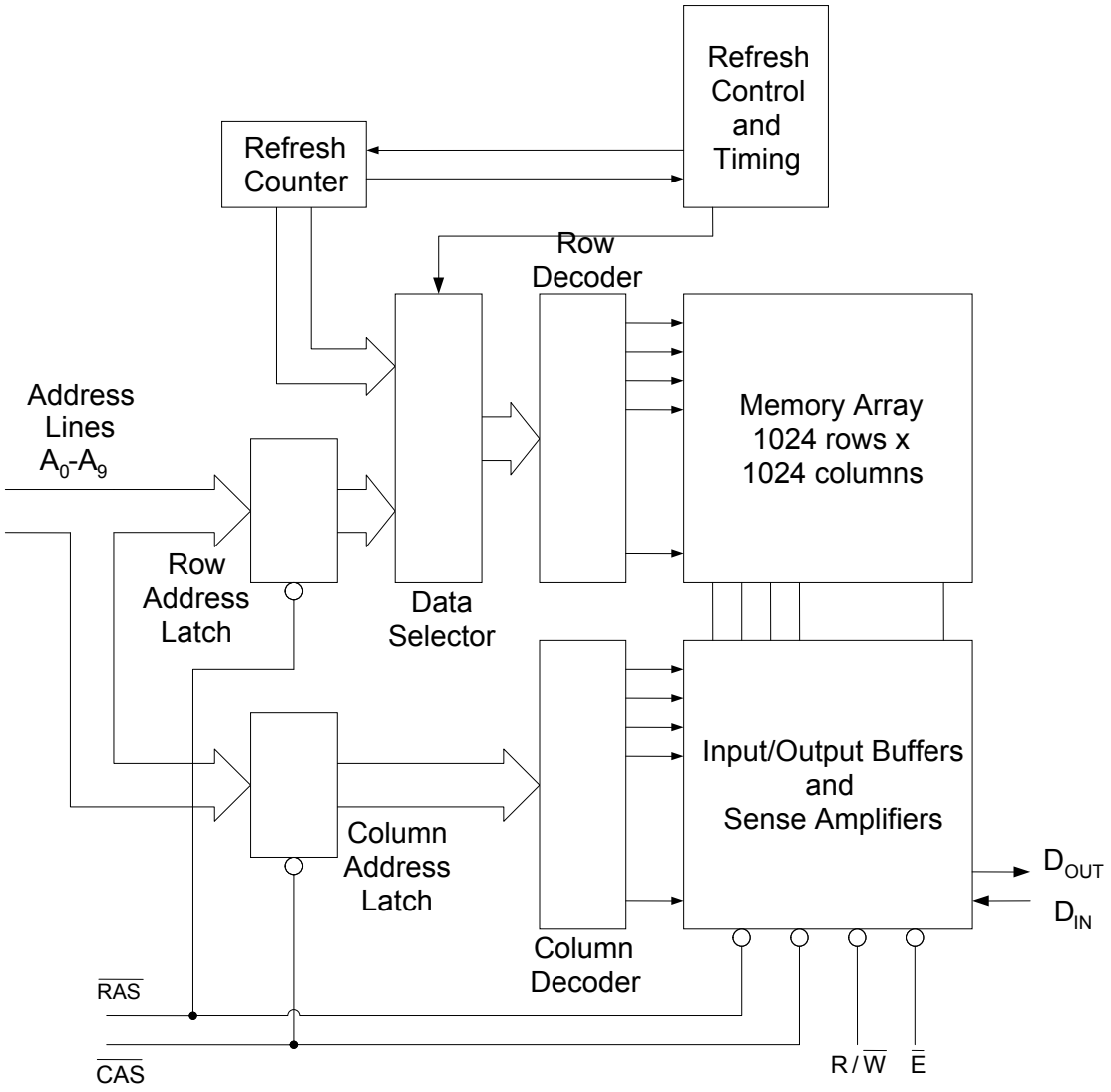


Figure 40.8 Circuit Diagram of a 1M x 1 DRAM

The $R/\overline{W}$ signal controls the Reading and Writing of data through the D_{OUT} and D_{IN} lines. The $\overline{E}$ signal enables the DRAM chip. The refresh cycle is controlled by the Refresh

generated by the refresh counter. During the refresh cycle all memory cells connected to the selected row are refreshed simultaneously. Therefore, a 1M bit DRAM arranged as 1024 rows and 1024 columns is refreshed by selecting all the 1024 rows in a sequence.

© Virtual University of Pakistan

Lesson No. 41

READ AND WRITE CYCLES

The read cycle is shown. Figure 41.1a. The $\overline{RAS}$ and $\overline{CAS}$ signals are activated one after the other to latch the multiplexed row and column addresses respectively applied at the multiplexed address input lines. The $R/\overline{W}$ signal is activated to read data which is made available on the D_{OUT} data line.

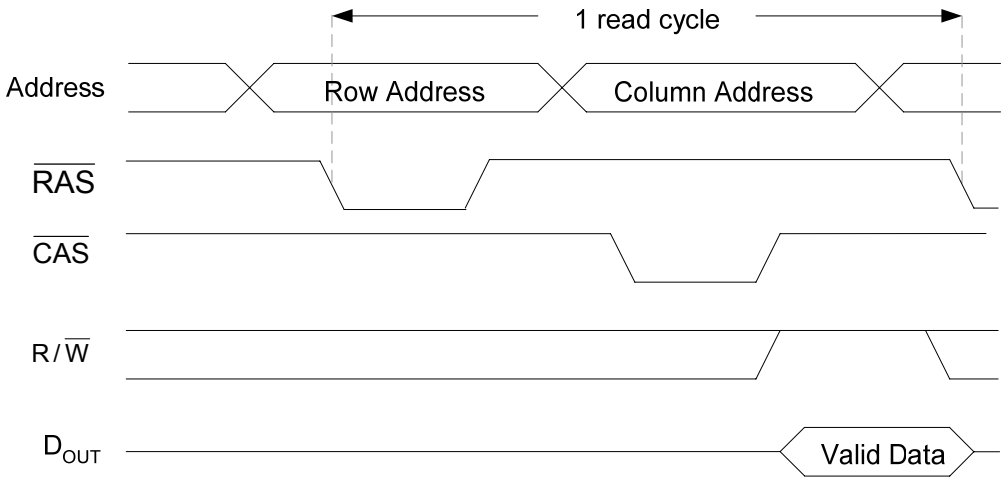


Figure 41.1a DRAM Read Cycle

The write cycle is similar to the read cycle. Figure 41.1b. The row and column addresses are applied at the multiplexed address inputs along with the strobe signals that latch the row and column addresses in the row and column address latches respectively. The write signal is activated allowing data placed at the D_{IN} data line to be stored in the selected memory cell.

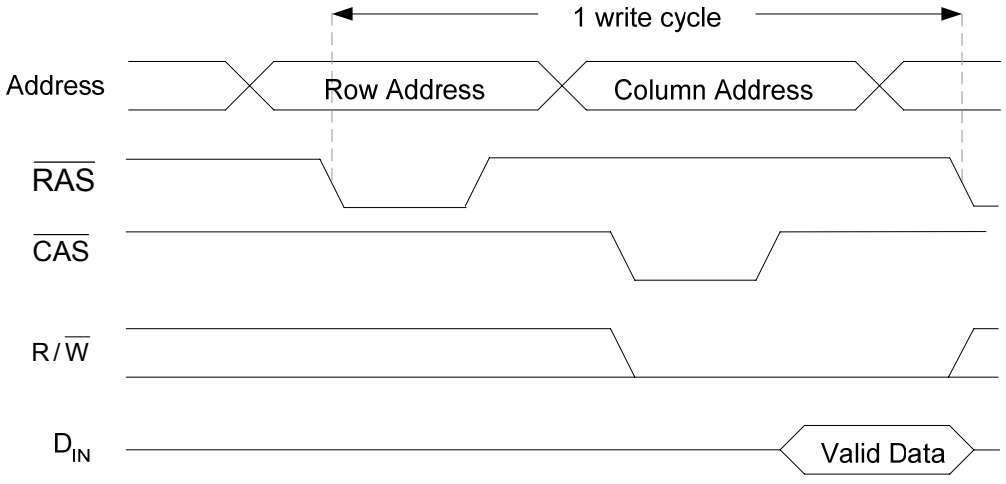


Figure 41.1b DRAM Write Cycle

In a FAST Mode Page Access all the columns in the same row are either read or written. A single row is considered to be a 'Page' of memory storing data values in successive memory locations. The FAST Mode Page Access allows faster memory read and access times when reading successive data values stored in consecutive locations on the same row. The row address is first latched by the $\overline{RAS}$ signal going low. The $\overline{RAS}$ signal remains low throughout the Page Access. The $\overline{CAS}$ signal is toggled between high and low which selects successive columns in the selected row. Data is read or written to successive column locations. During the read cycle when $\overline{CAS}$ goes high the D_{OUT} line is disabled. Therefore, data on the D_{OUT} line has to be latched by the external system before $\overline{CAS}$ goes high. Figure 41.2.

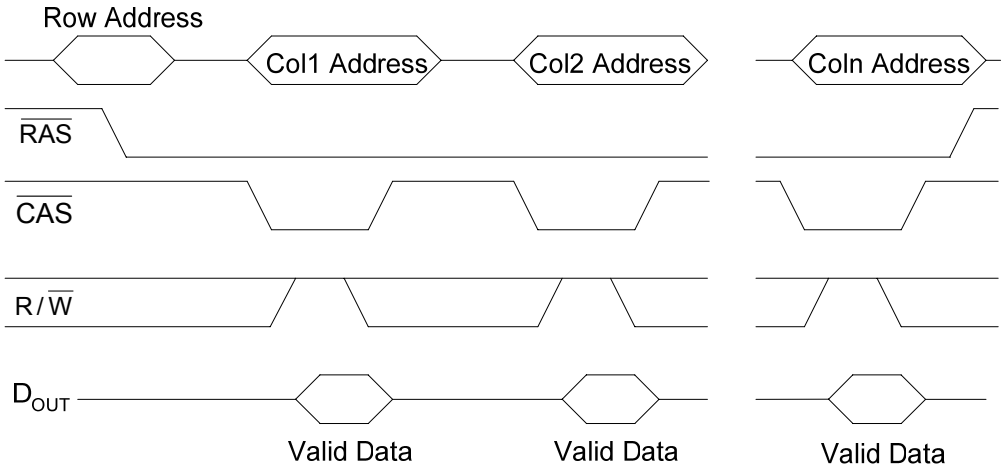


Figure 41.2 FAST Page Mode Read Cycle

Burst Refresh and Distributed Refresh

DRAM chips are refreshed using either the Burst Refresh Mode or the Distributed Refresh Mode. In the Burst Refresh Mode, all rows in the DRAM chip are refreshed consecutively in each refresh cycle. For a DRAM having a refresh cycle of 8 msec, a burst refresh of all rows occurs once every 8 msec. During the refresh cycle the memory can not be accessed for normal read and write operations, therefore the read/write operations are suspended until the refresh cycle is completed.

In Distributed Mode, the refresh cycle is interspersed between normal read and write cycles. For the 1024 x 1024 DRAM memory and a refresh cycle of 8 msec, each of the 1024 rows has to be refreshed in 7.8 microsec when Distributed refresh is used.

RAS only Refresh and CAS before RAS Refresh

The refresh cycles are of two types, $\overline{RAS}$ only refresh and $\overline{CAS}$ before $\overline{RAS}$ refresh. In the $\overline{RAS}$ only refresh the $\overline{RAS}$ signal goes low latching the row address to be refreshed, the $\overline{CAS}$ signal remains inactive high throughout the refresh cycle. An external counter is used to provide the row addresses for the refresh operation. In the $\overline{CAS}$ before $\overline{RAS}$ refresh mode, the $\overline{CAS}$ goes low before $\overline{RAS}$ goes low. This sequence activates an internal refresh

counter that generates the row address to be refreshed. This address is switched by the data selector into the row decoder.

Types of DRAMs

There are several different types of DRAMS available.

- Fast Page Mode DRAM: Compared to random access read/write, FAST Page Mode is faster where successive columns on the same row are read/written in successively by asserting the $\overline{CAS}$ strobe signal. The $\overline{CAS}$ signal when de-asserted, disables the D_{OUT} data line, therefore the next column address can not occur unless the data at the current address is latched by the external system reading data from the DRAM. The access speed of the DRAM during read operation is therefore limited by the external system latching the data available on the D_{OUT} line.
- Extended Data Output (EDO) DRAM: The memory in its operation is similar to the FPM DRAM, however the $\overline{CAS}$ signal doesn't disable the D_{OUT} when it goes to its non-asserted state. Thus the valid data on the D_{OUT} line can be remain until the $\overline{CAS}$ signal is asserted again to access the next column. Thus the next column address can be accessed before the external system accepts the current data.
- Synchronous DRAM: The DRAM operations are tied to a clock signal that also times the microprocessor operations. This allows the DRAM to closely synchronize with the microprocessor.

ROM Read-Only Memory

A ROM contains permanent data that can not be changed. Thus ROM memory does not allow write operation. A ROM stores data that are used repeatedly in system applications, such as tables, conversions, programmed instructions for system initialization and operation. ROMs retain data when the power is turned off. ROMs are of different types.

- Mask ROM: Data is permanently stored during the manufacturing process.
- PROM: Programmable ROM allows storage of data by the user using a PROM programmer. The PROM once programmed stores the data permanently.
- EPROM: Erasable PROM allows erasing of stored data and reprogramming.
- UV EPROM: Is a programmable ROM. Data is erased by exposing the PROM to Ultraviolet light.
- EEPROM: Electrically Erasable PROM is erased electrically. EEPROM allows in-circuit programming and doesn't need to be removed from the circuit for erasure or programming.

Mask ROM

The structure of a storage cell in a Mask ROM is shown. Figure 41.3. The storage cell in a Mask ROM is implemented using a MOS transistor. The Gate of the transistor is connected to the row line and the output of the transistor is connected to the column line. When a row is selected all the MOS transistors with their Gates connected to the row are turned on and connect their high output to their respective column line. Transistors with their GATE connections to the row removed are not turned on and the corresponding column lines have a logic low output. During the manufacturing process the cells that store logic one have the transistors configured with their Gates intact and cells having logic 0 have the transistors configured with their Gate connections removed.



Figure 41.3 ROM cell storing a logic 1 and logic 0

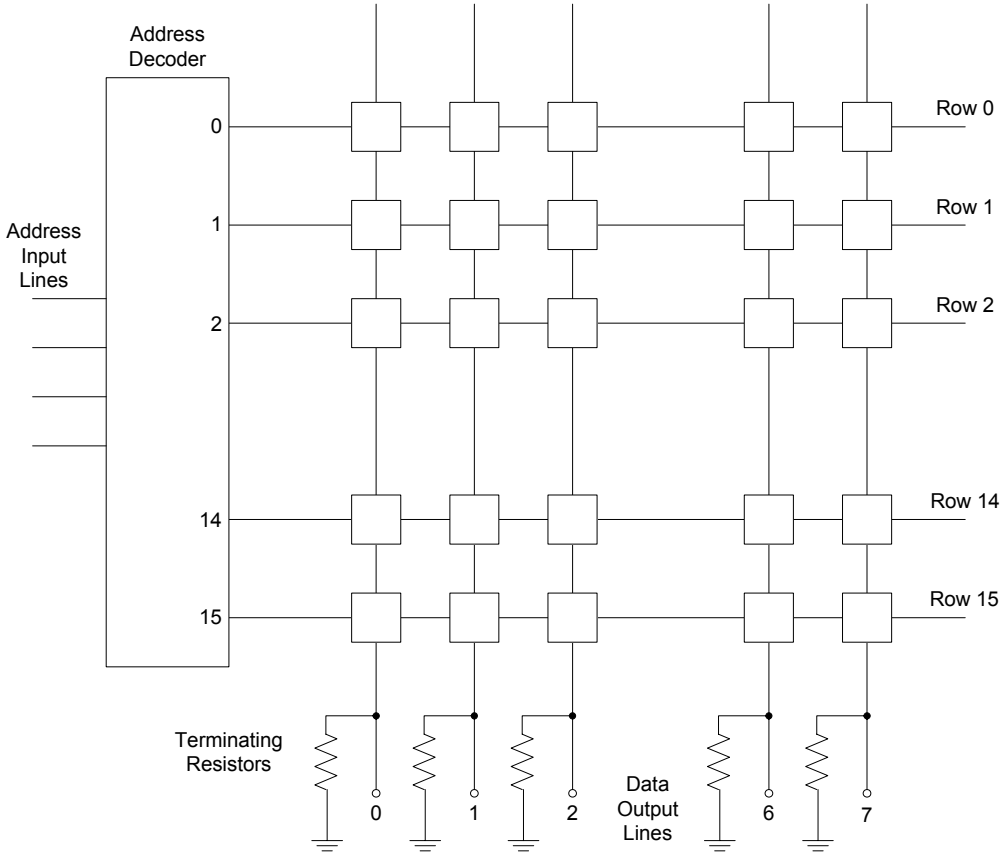


Figure 41.4 General Architecture of a ROM

A 16 x 8 ROM is shown. Figure 41.4. A 4-bit address is decoded by a 4 x 16 decoder which selects the appropriate row line. The MOSFETs connected to the selected row output logic 1 on the respective column lines. The MOSFETs that are not connected output logic 0. The terminating resistor connected to the end of each column line ensures that the output line stays low when a MOSFET outputs logic 0.

The Internal Structure of Mask ROM chip is different from the simplified structure shown in figure 41.4. A 256 x 4 ROM device is organized in the form of a 32 x 32 row-column structure. The 8-bit address is split into a 5-bit row address which selects one out of the 32 rows and the remaining 3 bits are used to select 1 out of 8 column lines by four 8 x 1

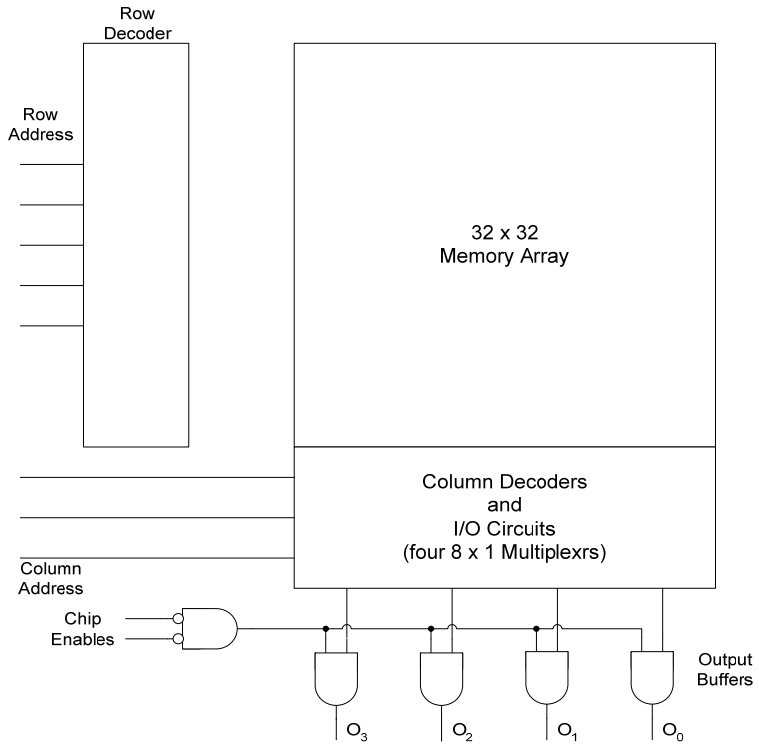


Figure 41.5 Internal structure of a 264 x 4 ROM

ROM Applications

The 264 x 4 ROM can be used as conversion table to convert 4-bit binary values to 4-bit equivalent Gray Code values. The 4-bit code which is to be converted is applied as an address at the 4-bit address input of the ROM. At each of the 256 locations corresponding to the 256 addresses 256 Gray Code values are stored. The 4-bit Gray Code contents stored at the first 16 locations of the ROM are shown. Table 41.1. ROM can also be used as a simple table. Each location in the ROM stores a value which can be accessed by specifying the location address. Look-Up tables used in computers can be implemented using ROMs.

Address	Data
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101
1010	1111
1011	1110
1100	1010
1101	1011
1110	1001
1111	1000

ROM Read Cycle & Access Time

The Access Time of a ROM is the time it takes for the data to appear at the Data Output of the ROM chip after an address is applied at the address input lines. The access time can also be measured with respect to the activation of the chip enable signal and the appearance of the data at the output lines when address is already on the address lines. Figure 41.6.

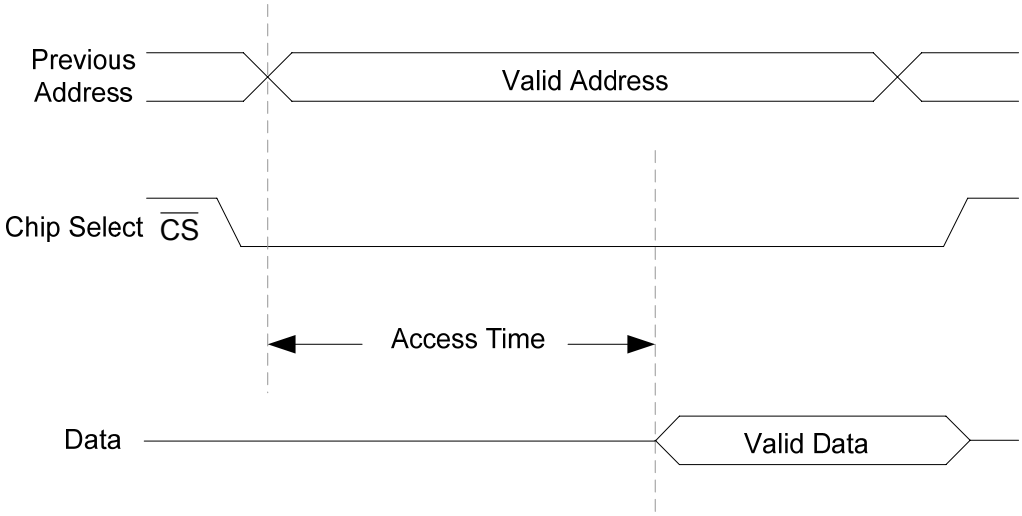


Figure 41.6 Read Access Time of a ROM chip

PROM (Programmable ROMs)

The Mask ROM are programmed at the manufacturing time and can not e programmed by the user. Mask ROM allows read-only operation. A PROM can be programmed once by the user using a PROM Programmer. Once the PROM is programmed its contents can not be erased and programmed again. PROM uses a fusible link to connect the output of the MOS transistors to the column line. The row lines are permanently connected the gates of the transistors. When the fuse is intact, logic high is seen on the column line when the corresponding cell is selected. When the fuse is blown the column line outputs a logic low. Figure 41.7.

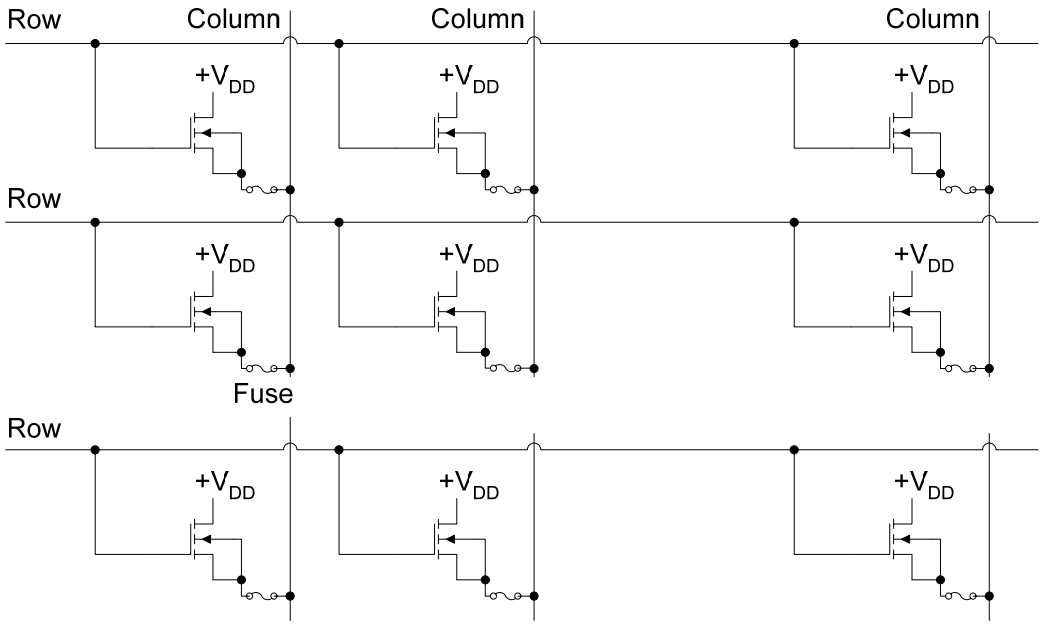


Figure 41.7 PROM array with fusible links

EPROM Erasable PROM

An EPROM is an Erasable PROM. The contents of the memory can be erased and the memory can be reprogrammed. The EPROM uses NMOSFET array with an isolated-gate structure. The isolated gate structure can store a charge for indefinite periods of time. The data bits are represented by the presence or absence of gate charges. Erasure of data bit is the removal of gate charge.

Two basic types of EPROMs are the Ultra Violet EPROM (UV EPROM) and the Electrically Erasable EPROM (EEPROM). In the UV EPROM, the programming process causes the electrons to be removed from the isolated gate. The UV EPROM has a quartz window on the package through which the memory array is exposed to high-intensity UV which causes the positive charge stored on the gate to neutralize after an exposure time of few minutes. A typical UV EPROM memory chip is shown. Figure 41.8

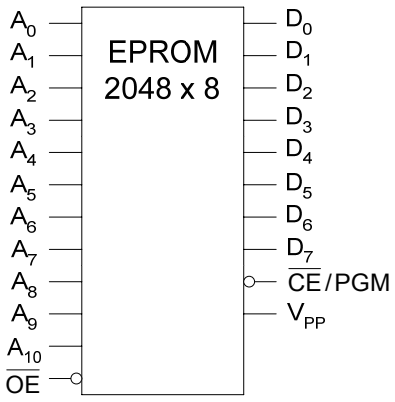


Figure 41.8 A 2 KB EPROM

The 2 KByte EPROM has 2K locations each storing a byte data value. Addressing 2K locations requires 2^{10} or 10 address lines A_0 to A_{10}. Each memory location stores a byte value, therefore 8 data lines are required. To read data from the EPROM, the chip has to be selected by setting the chip enable/program $\overline{CE}/PGM$ signal to active-low and the output enable $\overline{OE}$ is set to logic low.

Programming EPROM

Programming the EPROM chip is done by applying a high dc voltage at the V_{PP} pin and setting the output enable $\overline{OE}$ to logic high. The data to be programmed is applied at the eight data lines and the address at which the data is to be programmed is applied at the address lines. A high level pulse is applied at the enable/program $\overline{CE}/PGM$ signal to program the data at the required address. The entire EPROM is programmed by applying data values at the data lines the corresponding address at the address input lines and high level pulses at the $\overline{CE}/PGM$ pin.

EEPROM Electrically Erasable PROM

An electrically erasable PROM is programmed and erased by applying electric pulses. Since this PROM does not need to be exposed to Ultra Violet light for erasing data therefore the EEPROM can be rapidly programmed and erased in-circuit.

FLASH Memory

An ideal memory should have high density, have read/write capability, should be non-volatile, have fast access time and should be cost effective. The ROM, PROM, EPROM, EEPROM, SRAM and DRAM all exhibit some of these characteristics, however none of these memories have all the mentioned characteristics except for the FLASH Memory.

FLASH memories have high density, that is, they store more information per unit area as more storage cells are implemented per unit area. These memories have read/write capability and are non-volatile and can store data for indefinite time period. The high density FLASH memory cell is implemented using a single floating-gate MOS transistor. A data bit is stored as a charge (logic 0) or the absence of a charge (logic 1) on the floating gate. The amount of charge present o the floating gate determines if the transistor will turn and conduct current from the drain to the source when a control voltage is applied at the Control rate during the read operation. Figure 41.9

© Virtual University of Pakistan

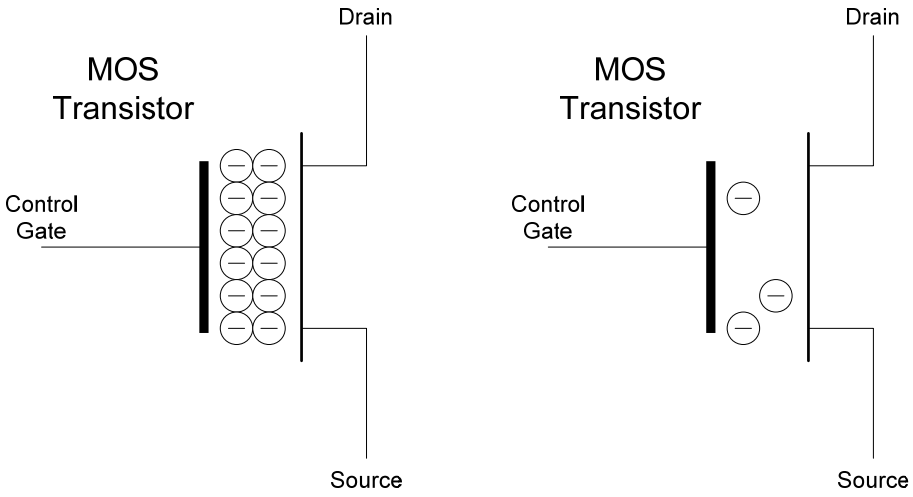


Figure 41.9 MOS transistor with charge (logic 0) and no charge (logic 1)

FLASH Memory Operations

FLASH Memory operations are classified into

- Programming Operation
- Read Operation
- Erase Operation

Programming Operation

Initially, all cells are at the logic 1 state, that is with no charge. The programming Operations adds charge to the floating gate of those cells that are to store a logic 0. No charge is added to those gates that are to store a logic 1. The charges are stored by applying a positive voltage at the Control Gate with respect to the Source which attracts electrons to the floating gate. Once the gate is charged it, retains the charge for years. Figure 41.10

Read Operation

During the read operation a positive voltage is applied to the MOS transistor control gate. If a negative charge is stored on the gate then the positive read voltage is not sufficient to overcome the negative charge therefore the transistor is not turned on. On the other hand if there is no or small amount of negative charge stored, the positive read voltage is sufficient to overcome the negative charge turning on the transistor. When the transistor is turned on there is a current from the drain to the source of the cell transistor. The presence of this current is sensed to indicate a 1. The absence of this current indicates a 0. Figure 41.10.

© Virtual University of Pakistan

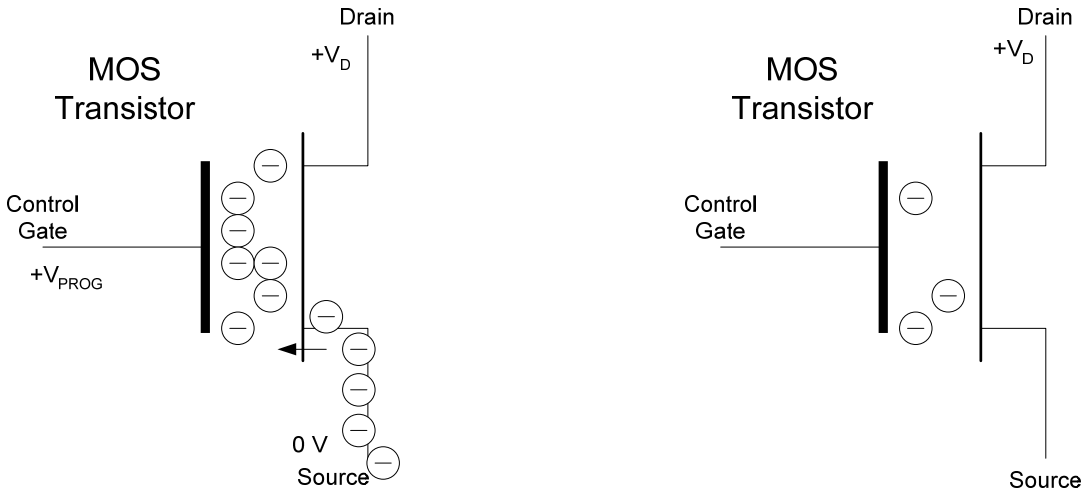


Figure 41.10 FLASH Memory Cell programmed with logic 0 and logic 1

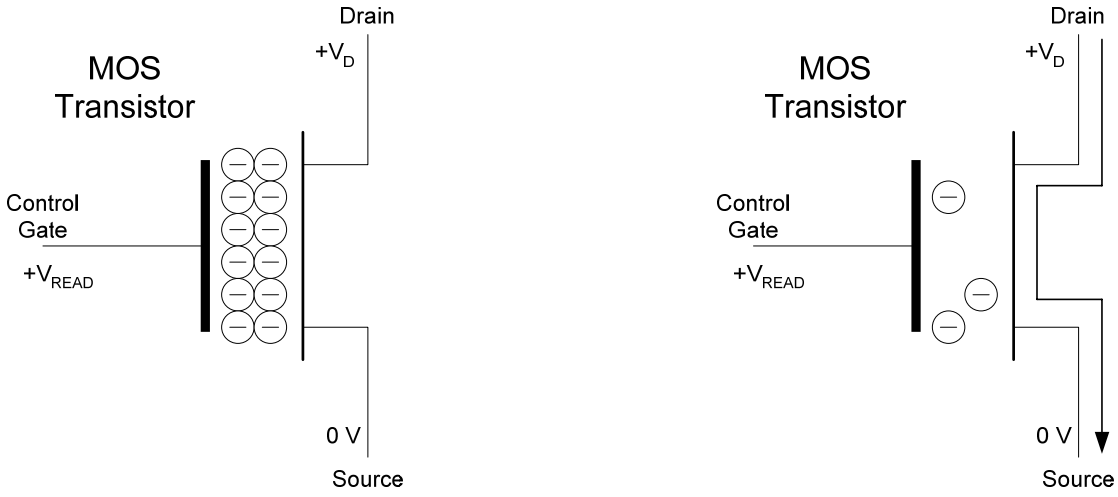


Figure 41.11 Read operation to read a logic 0 and a logic 1

Erase Operation

During the erase operation charge is removed from the memory cell. A sufficiently large positive voltage is applied at the source with respect to the control gate. The voltage applied across the control gate and source is opposite to the voltage applied during programming. If charges are present on the gate, the positive voltage supply at the source attracts the electrons depleting the gate. A FLASH memory is erased prior to programming. Figure 41.12

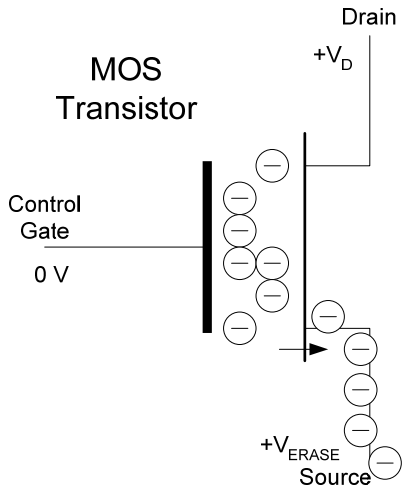


Figure 41.12 The Erase Operation of the FLASH Memory Cell

Lesson No. 42

FLASH MEMORY ARRAY

The FLASH Memory array is arranged in the form of rows and columns. The row line is connected to the Control Gate of each MOS transistor which implements a single bit storage cell. The number of such MOS transistors in a row generally depends upon the size of the data value stored at each location. A byte value stored at a location requires 8 cells activated by a single row. Only a single row is selected at a time to select the appropriate cells. The Source terminals of all the transistors arranged in a column are connected to a column line. Similarly, the transistors in the second column have their Source terminals connected together. Generally, if the size of the data stored is a byte, the memory has eight column lines connecting the Source of the corresponding transistor cells in each column together. Figure 42.1.

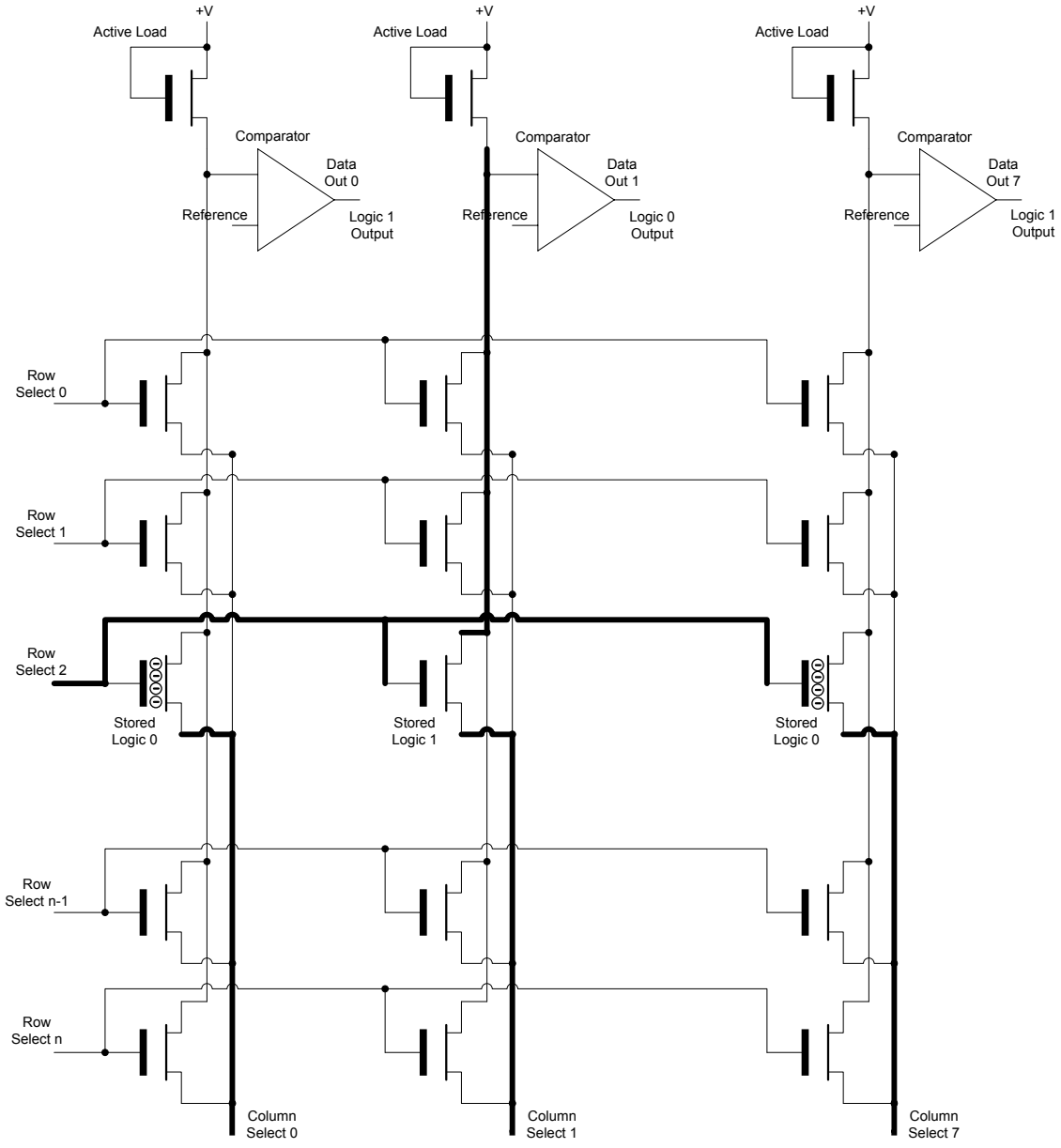


Figure 42.1 FLASH Memory Array

When a row is selected, (row 2 is shown to be selected in the figure) all the transistors which have their Gate terminals connected to the row select line are activated. A current flows through the selected transistor if the corresponding column select line is activated and the transistor Gate has no charge stored. In the figure, the column select 0 line is activated however no current flows through the selected transistor as the transistor gate has a stored charge. The column select 1 line is activated and a current flows through the transistor in the second column as the transistor gate has no stored charge. Similarly, no current flows through the transistor in the last column when the column select 7 line is activated as the transistor gate has a stored charge.

Thus no current flows through a column line if the selected cells has a charge stored on the transistor gate. If some of the selected cells in a row have charges stored while others do not have charges stored then current will not flow in columns corresponding to the cells which have charges stored, while columns having current flowing through them correspond to cells having no charge. The presence of current in a column produces a voltage drop across an active load connected at the end of each column line, while an absence of current doesn't produce a voltage drop across the active load. The voltage drop produced across each active load is separately compared with a reference voltage by a comparator circuit. If there is a voltage drop across the active load due to flow of current, the comparator output is a 0. If the voltage drop across the active load is 0 volts due to absence of current the comparator output is a 1. The presence or absence of current in a column line is based on the binary 1 and 0 stored in the cell. Thus the comparator output is opposite to the information stored.

Memory Summary

A summary of memory types and their characteristics are shown. Table 42.1. The Static Ram (SRAM) is non-volatile and is not a high density memory as a latch is required to store a single bit of information. Implementation of a latch requires almost six transistors. The Dynamic Ram is also non-volatile however it offers high density memories as each storage cell requires a single transistor and a capacitor. ROMs and PROMs retains information permanently even if the supply voltage is removed. Since a single transistor is used to store a logic 0 or 1 therefore ROMS and PROMs are high density memories. EEPROMs allow data to be read or written however the ability to change the data without having to remove the EEPROM chip from a circuit board requires extra logic. Thus EEPROM memories are not high density memories.

Memory Type	Non-Volatile	High Density	One-Transistor Cell	In-System Write ability
SRAM	No	No	No	Yes
DRAM	No	Yes	Yes	Yes
ROM	Yes	Yes	Yes	No
EPROM	Yes	Yes	Yes	No
EEPROM	Yes	No	No	Yes
FLASH	Yes	Yes	Yes	Yes

Table 42.1 A summary of Memory Types

Special Type of Memories

Two types of memories namely the first in-first out (FIFO) memory and last in-first out (LIFO) are implemented using shift registers. These memories are used in specific applications.

First In-First Out (FIFO) Memory

Digital systems receive data or transfer data to devices that are operating at different data rates. A Computer (microprocessor), for example, receives data from the Keyboard as a user types in the information. The keyboard is a very slow device which generates data at a rate of few bytes per second. The microprocessor on the other hand is very fast and can processes information at very high data rates. Devices that operate at different data rates can not be connected to each other directly through their data lines because the devices that operate at very high data rates are slowed down to the data rate of the slow device. For example, if a microprocessor is connected directly to a keyboard it would be waiting for data from the keyboard. During the waiting period the microprocessor would not be processing any information and would be lying idle. Practically, a microprocessor is connected to a keyboard through a keyboard buffer which is a temporary memory where the keyboard writes its data (the keystrokes). The microprocessor instead of waiting for the keyboard is kept busy processing information. When ever the microprocessor needs to use the information typed through the keyboard it accesses the keyboard buffer and reads the necessary information. Two devices operating at the same data rates are shown to be connected directly through their data lines. Figure 42.2a. Since, device A produces data at the same rate at which the device B consumes the data, the two devices can be connected together through their data lines without the use of the buffer. Device A is configured to write data to the data lines and device B is configured to read the data from the data lines.

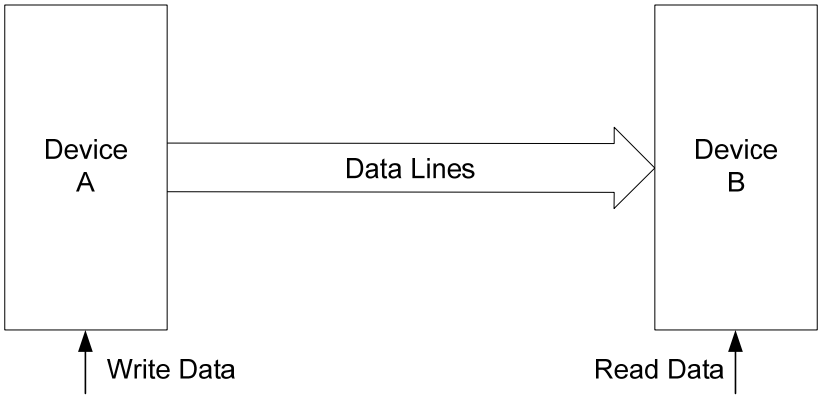


Figure 42.2a Devices A and B operating at same data rates

The keyboard buffer is an example of a FIFO memory. In the FIFO scheme data is not accessed randomly from any location as in RAM and ROM memories where any location can be accessed by specifying the location address. In the FIFO memory the data which is written into the memory first is the first one to be read out. As mentioned above, FIFO memories are used to connect two digital devices that produce and consume data at different rates. Assuming that Device A produces data at a certain rate and device B consumes the data at a different rate. A FIFO memory is connected between the output of device A and the input of device B, the data produced by device A is written into the FIFO memory. The device B reads the data from the FIFO memory in the same sequence as was written by the device A. Figure 42.2b.

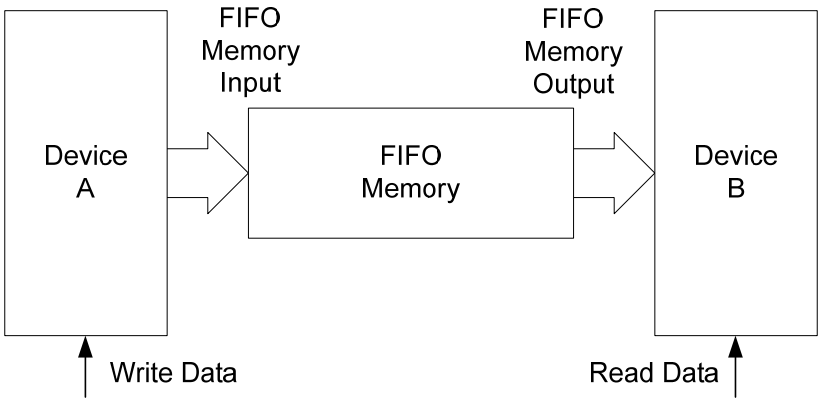


Figure 42.2b FIFO Memory connecting two communicating devices

The FIFO memory is implemented using shift registers with a control circuitry that allow the data entered at the FIFO input to be stored at the FIFO output when the FIFO memory is empty. Addition data that is entered at the FIFO input is shifted to the appropriate location in the FIFO memory. When the data at the FIFO output is consumed by a device, the stored data within the FIFO memory is shifted forward so that the second data to be input into the FIFO memory is placed at the FIFO memory output. Figure 42.2c.

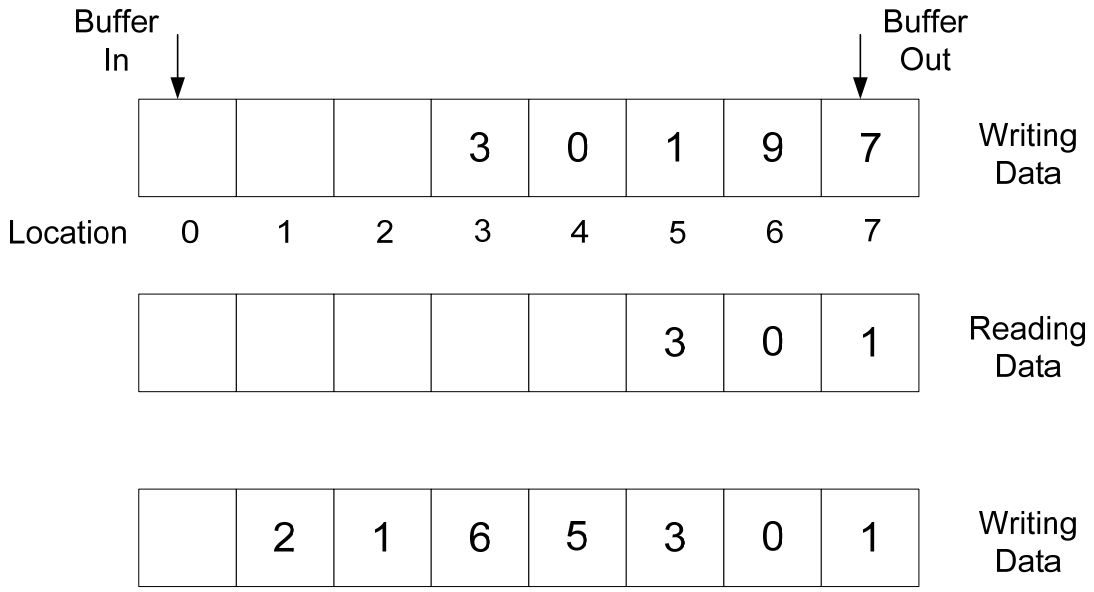


Figure 42.2c Writing and Reading from FIFO Buffer

In the diagram data is produced by the device in the sequence 7, 9, 1, 0 and 3. The data is written in the FIFO buffer maintaining the sequence in which the data values are produced. Device B consumes the first two values 7 and 9 leaving the values 1, 0 and 3 in the buffer. The data values in the buffer are shifted towards the head of the buffer to create space for more data values to be written. Device A produces four new data values 5, 6, 1 and 2 which are appended into the buffer in the available empty locations.

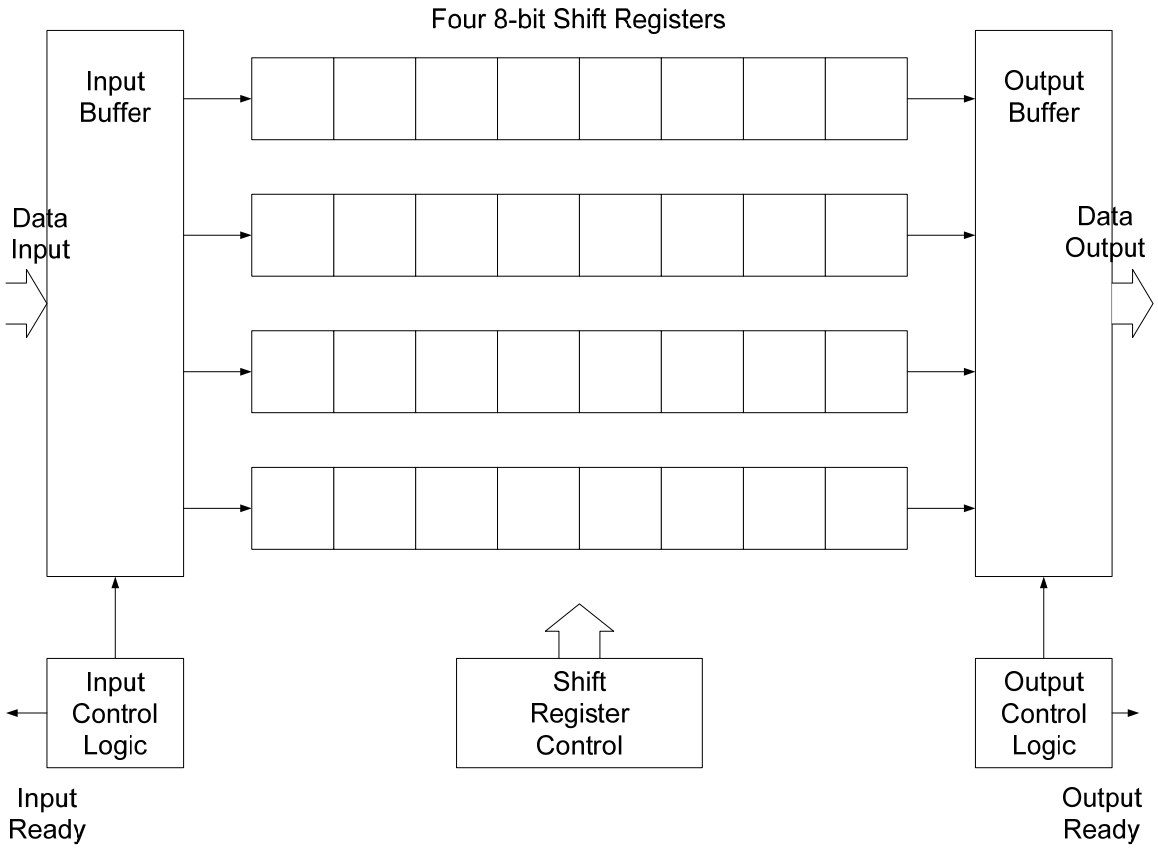


Figure 42.3 FIFO Implementation using four 8-bit Shift Registers

The FIFO Implementation using four, 8-bit shift registers is shown. Figure 42.3. Data (4-bit data) to be written into the FIFO buffer is placed in the Input Buffer which is shifted to appropriate location by the Shift Register Control circuit. When the data is stored in the FIFO buffer at the appropriate location the Input Buffer is ready to accept more data for temporary storage in the FIFO buffer. The Input Control Logic circuit indicates the availability of the Input Buffer for latching new data values by activating the Input Ready control signal. The data is read out from the FIFO buffer through the Output Buffer. Data at the Buffer Out location is latched in by the Output Buffer from where the device can read the data. Once the data is read the Shift Control circuitry updates the buffer by shifting the buffer contents towards the right. The right most data value in the buffer is moved to the Output Buffer latch for reading by the device. The Output Ready signal is activated to indicate the availability of data for reading.

Implementing FIFO memory using RAM

Shift register based FIFO memory is used in digital systems designed for specific applications where small sized buffers are used to allow transfer of data between two devices operating at different data rates. Such digital systems either have no RAM or very small RAM for storing variables. Computers implement FIFO memory by reserving a part of their RAM memory for use as buffers. The Keyboard buffer for example is implemented by reserving a part of the RAM. When RAM is used as FIFO memory, two registers are used to point to the FIFO Buffer Out and Buffer In respectively. The two registers hold the addresses of the locations of the Buffer Out and Buffer In respectively, which are updated as new data is written into the buffer and previous data is read out from the FIFO buffer. Implementation of the FIFO

© Virtual University of Pakistan

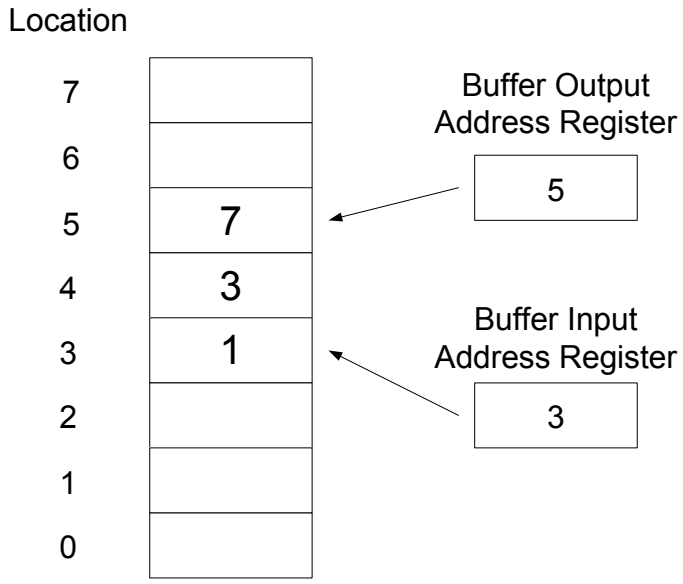


Figure 42.4 Implementation of a FIFO buffer using RAM

Lesson No. 43

LAST IN-FIRST OUT (LIFO) MEMORY

Last In-First Out Memory finds applications in computer systems where it is used to implement a stack. The operation of a stack can be understood by viewing a stack of plates. In a stack of plates the first plate is placed at the bottom the next plate placed is placed on the top, the third plate is placed on the top of the second plate and so on. Plates are removed one at a time from the top of the stack, thus the last plate placed on the stack top is the first to be removed followed by the second plate and then the plate at the bottom which was placed first. In a register based LIFO memory implementation a set of Parallel In/Parallel Out registers are connected together such that data is pushed down or pulled up when data is stored or removed from the memory respectively. Figure 43.1.

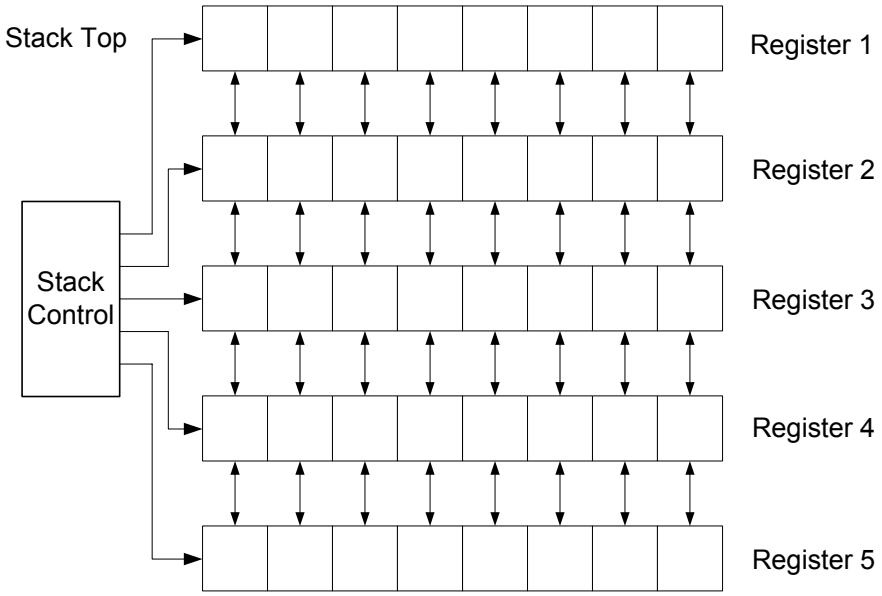


Figure 43.1 A five byte LIFO Memory

In the LIFO memory shown, the first 8-bit data value is stored in the first register Reg. 1. To store the next value, the first value stored in Reg. 1 is pushed down (shifted) to the second register Reg. 2. The second 8-bit data value is written into the first register Reg. 1. The third data value can only be stored when both the previous values are pushed (shifted) down to the Registers 2 and 3. A maximum of five, 8-bit data values can be stored in the LIFO register. The fifth and the last value stored in the first register Reg. 1 is the first value to be read out. The remaining four values in the memory are pulled (shifted) up. At any time new data can be added to the LIFO memory or the stored data can be read out.

Shift Register based Stack implementation finds use in specialized digital systems. A practical way to implement the program stack which a program under execution uses to access variables is by means of the RAM memory. The stack is known as a RAM Stack. A special purpose register known as the 'Stack Pointer Register' stores the address of the top of the stack, a reserved area in the RAM memory. As data values are written or read from the RAM stack, the Stack Pointer Register increments or decrements its contents always pointing to the stack top. Figure 43.2.

© Virtual University of Pakistan

Initially, the Stack Pointer Register has the contents 0, which is address of memory location of the stack. A value '7' is stored in the Stack by writing it to memory location 0 pointed to by the Stack Pointer Register. To store the next value 4 in the stack the contents of the Stack Register are incremented so that the next vacant location in the Stack is accessed. The new data value 4 is written at the new location. Similarly, data values 9 and 8 are stored in the next two consecutive locations in the Stack. The Stack Pointer Register points to the Stack Top (location 3) which has the data value 8 stored. A data value can be read from the Stack Top by reading the data value from the address pointed to by the Stack Pointer Register. After reading the data value the contents of the Stack Pointer Register are decremented to point to the new Stack Top.

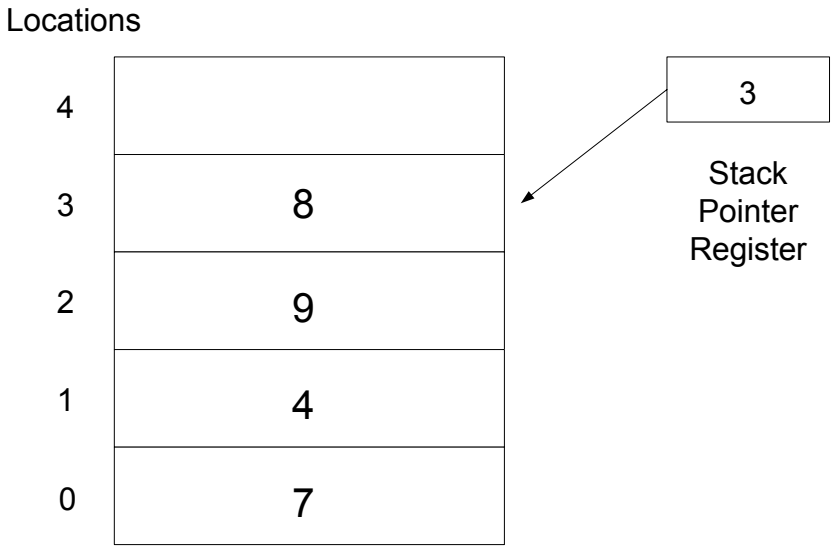


Figure 43.2 Memory Based Stack

Memory Expansion

Digital systems require different amounts of memory in the form of RAM and ROM Memory depending upon specific applications. A computer requires large amounts of RAM memory to store multiple application programs, data and the operating system. In a computer, part of the RAM is reserved to support the Video Memory, Stack and I/O buffers. The ROM used by a computer is relatively very small as it stores few bytes of code used to Boot the Computer system on power up. Micro-controller based digital system designed for specific applications do not have large memory requirement, in fact the total memory requirement of such micro-controller systems is met by on-board RAM and ROM having a total storage capacity of few hundred of kilobytes. Computer and Digital systems have the capability to allow RAM memory to be expanded as the needed arises by inserting extra memory in dedicated memory sockets on the computer motherboard.

The total amount of memory that is supported by any digital system depends upon the size of the address bus of the microprocessor or a micro-controller. A microprocessor having an address bus of 16 bits can generate 2^{16} or 65536 unique addresses to access 65536 locations which allows either a single 65536 location RAM or a combination of RAM and ROM totalling 65536 memory locations to be connected to the microprocessor. It is also possible to initially have a 32768 location RAM connected to the microprocessor with the remaining 32768 address locations unoccupied allowing the microprocessor to execute a program that can be

another 32768 location RAM. Microprocessors used in computer systems have memory spaces of the order of 2^{32} and larger.

The data unit size accessed by a microprocessor when it issues an address to either read or write from or to a memory also depends upon the microprocessor architecture more specifically the number of the data lines. A microprocessor having an 8-bit data bus can access a byte of information from any unique memory location. A microprocessor having a 16-bit data bus allows two bytes to be accessed from a memory location. Practically, microprocessors used in computer systems have up to 64 bit wide data buses allowing up to 8 bytes of data to be accessed simultaneously. A microprocessor that accesses 64-bits of data simultaneously requires RAM to be organized in such a way that allows 8 bytes of data to be accessed when ever any unique address is selected. On the other hand a microprocessor having a data bus of only 8-bits requires RAM that allows only a single byte of data to be accessed when ever any single address location is selected.

The total memory requirement of a computer or digital system is determined by the size of the address and data bus of a microprocessor. Microprocessors which have small address bus and a data bus have a small memory space. Microprocessors which have wide address and data buses have very large memory spaces which are rarely fully occupied by RAM and ROM devices.

Memory, both RAM and ROM are implemented in fixed data unit sizes of 1, 4 or 8 bits. Similarly, these memory devices are implemented having sizes in terms of total addressable locations which are restricted to address ranges between few hundred kilobytes to megabytes. The memories that are available in fixed sizes have to be connected together to form larger memories having appropriate data unit sizes and total number of addressable locations to fulfil the memory space requirements of a digital or computer system. Another important aspect of the RAM and ROM memories that are manufactured are the addresses of each memory location. For example, two 32Kbyte RAM chips have 2^{15} locations each. The first addressable locations in both the RAM chips have an address 0. Similarly, the second and third locations in both the memory chips have addresses 1 and 2 respectively. If the two RAM chips are connected together to form a 64 Kbyte RAM then one of 32Kbyte memory chips should respond to the address between 0 and 32767 and the other 32Kbyte memory chip should respond to the address 32768 and 65535. The two memory chips have bases address 0 and 32768 respectively.

Memory Map

The Memory Map of any digital system specifies the total memory space that can be accessed by the microprocessor and the distribution of the total addressable space amongst RAM, ROM, stack and buffers. The memory map shown in the figure shows the division of 1 MByte of addressable space into ROM, RAM for storage of data, RAM for storage of program code, vacant space which can be used in the future and a stack. Figure 43.3. The 1 MByte address space is divided into 16 equal blocks of 64 Kbytes each. The first 64Kbyte block having a base address 00000H is reserved for ROM memory. A maximum 64Kbyte sized ROM chip can be connected in the memory space. If a smaller ROM chip is connected in the memory space, the remaining unoccupied addresses can be utilized in future to expand the ROM memory by connecting extra ROM chips. The next block of 64KByte is reserved for storage of data by connecting a 64KByte RAM chip. The base address of the block is 10000H. The third block of 64KByte is used to store program code by connecting a 64KByte RAM chip. The base address of the third block is 20000H. The last 64KByte block having a base address

base address F0000H to support the Stack. Twelve blocks starting from base address 30000H are left unoccupied. These blocks can be used to connect additional RAM to increase the total amount of Memory RAM.

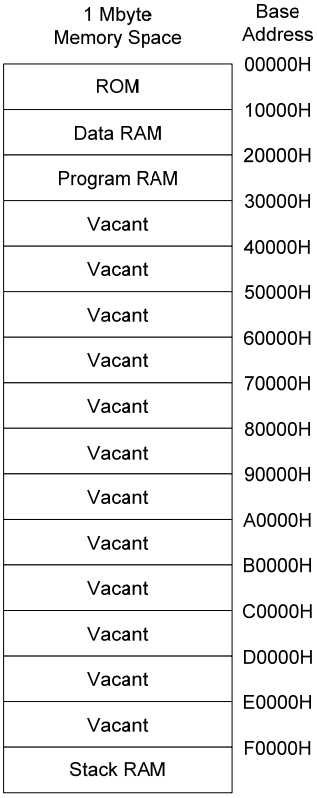


Figure 43.3 1 MByte Memory Map

Expanding Data Unit Size

Memories are implemented in 1, 4 and 8 bit data unit sizes. A processor that accesses 16-bit of data at each address location requires memory to be connected such that each address location allows access to 16-bits of data. In the example shown, two 4 K byte RAM chips are connected together to form a 4K Word (16-bit) memory or 8K Byte memory. Figure 43.4. A 4KByte RAM memory chip has A_0 – A_{11} address lines to address 4K locations. The address lines of both the 4KByte RAM chips are connected together so that the same address is used to select identical memory locations in both the memory chips. Each 4KByte RAM chip has 8 data lines to allow access to 8-bits of data at each memory location. The address lines of both the memory chips are kept separate. The memory chip shown on the right stores the least significant byte of the 16-bit data and the chip on the left stores the most significant byte of the 16-bit data value. The least significant byte is accessed through data lines D_0 – D_7 and the most significant byte is accessed through data lines D_8 – D_{15}. The $R/\overline{W}$ control line of both the memory chips are also connected together so that a word (16-bit) value is read or written to the selected location. The $\overline{CS}$ pins of both the chips are also connected together so that both the memory chips are selected simultaneously when ever a read or write memory operation is carried out.

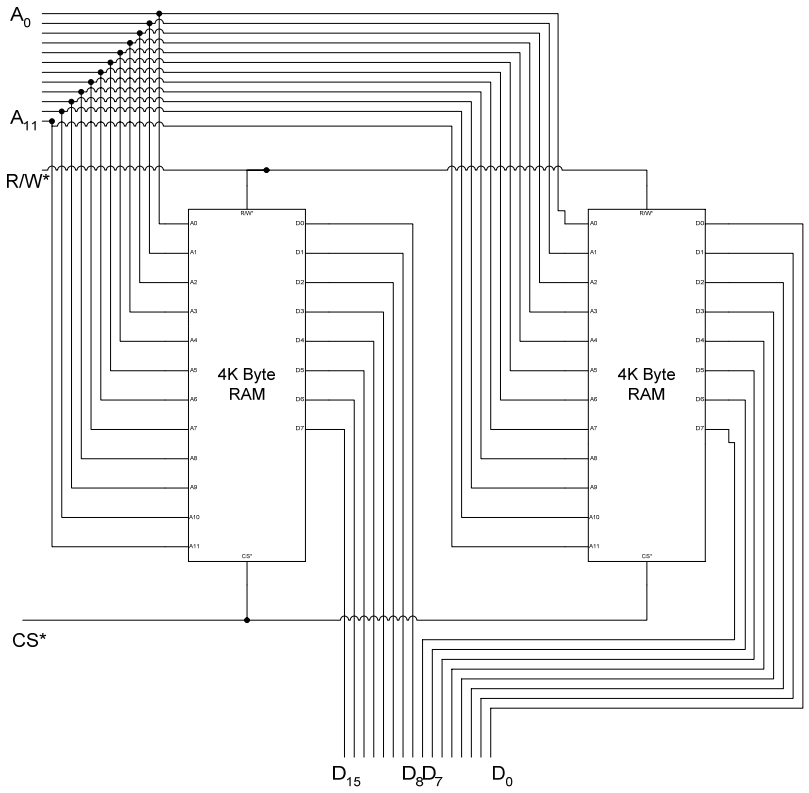


Figure 43.4 Implementing 4K Word RAM using two 4K Byte RAM chips

Expanding Memory Locations

The two 4KByte memory chips can be connected together to form an 8 KByte memory thereby doubling the total number of memory locations. Addressing 8KByte of memory requires 13 address lines. The first 12 address lines A_0 – A_{11} of the two memory chips are connected together, the data lines of both the chips are also connected together. Since the data lines are shared, therefore at any given instant data can be read or written to one of the two chips. Selection of either of the two memory chips is done through the $\overline{CS}$ signal. The first memory chip which maps the address range from 0 to 4K is selected when the $\overline{CS}$ signal is set to logic 0. The second memory chip which maps the memory range 4K to 8K is connected to the $\overline{CS}$ through a NOT gate therefore it is selected when the signal is set to logic high. The $\overline{CS}$ line is connected to the A_{12} address line which selects the first memory chip when it is logic 0, the second memory chip is selected when the A_{12} signal is set to logic 1. Figure 43.5.

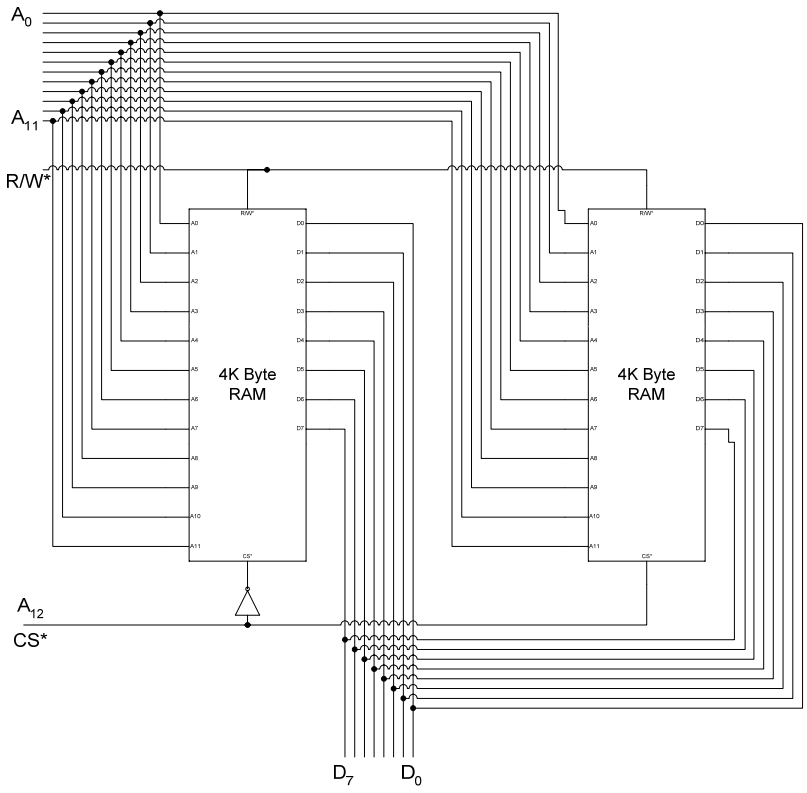


Figure 43.5 Implementing 8K Byte RAM using two 4K Byte RAM chips

Expanding Data Unit Size and Memory Locations

Memory chips can be connected together in different manners to increase the total size (locations) or the size of the data unit stored. Four 4K Byte chips can be connected together to implement an 8K x Word memory. Figure 43.6. Chips A and B are connected to provide 4K x 16 bit of memory space and the chips C and D are connected to provide another 4K x 16 bit of memory space. The RAM chips A and B are selected simultaneously when A_{12} address line is set to logic 0. The RAM chips C and D are selected simultaneously when A_{12} address line is set to logic 1. The $R/\overline{W}$ control line is connected to all the four RAM chips. RAM chips A and C provide access to upper byte of the 16-bit data and RAM chips B and D provide access to the lower byte of the 16-bit data.

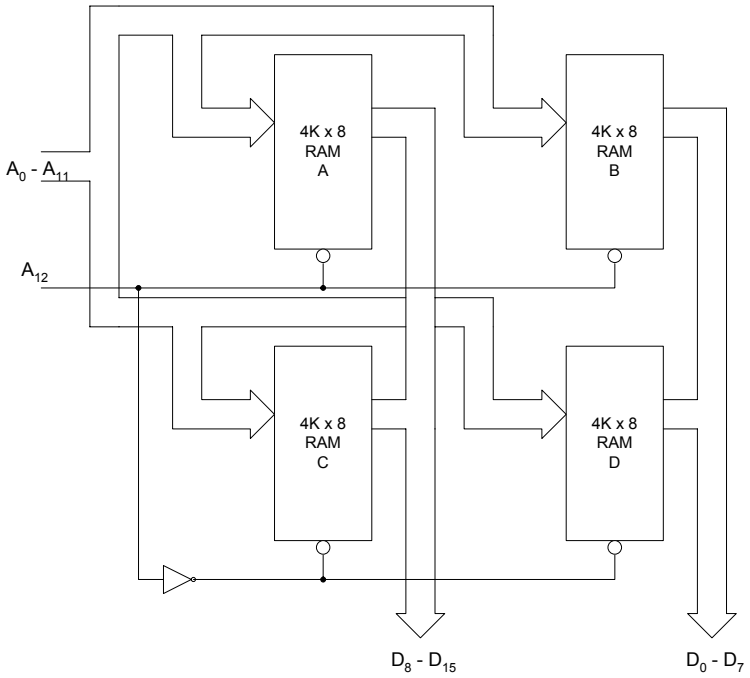


Figure 43.6 An 8K x 16 RAM implemented using four 4K x 8 memory chips

Address Decoders

All memory chips have the first location identified by address zero. The next location has the address one and successive memory locations have addresses assigned in an ascending order. When these memory chips are connected to a microprocessor at the specified location represented in the memory map the memory chips are connected such that the memory chip has the start address specified by a Base Address and the successive memory locations are selected by ascending addresses with respect to the Base address. Thus the first memory location is accessed by the Base Address and the next successive location is accessed by Base Address +1 and so on. In the 8KByte memory implemented in figure 43.5 the first 4KByte memory has the base address 0000H and the second 4KByte memory chip has a base address 1000H. A memory chip is connected at the Base Address by selecting it when the specified Base Address is generated by the microprocessor. An Address Decoder detects the generated Base Address and selects the desired memory chip.

Three 4KByte memory chips are shown to be connected at Base Addresses 0000H, 1000H and 2000H respectively. Figure 43.7. A 2 x 4 decoder is used for address decoding. The most significant address lines A_{12} and A_{13} are connected to the two input lines of the 2 x 4 decoder. When both the address lines are logic 0, the base address is
0000 and the first 4K memory chip RAM1 is selected. When A_{12} address line is logic 1 it indicates the Base Address 4K which selects the second 4K memory chip RAM2. When address line A_{13} is set to logic 1 it indicates a Base Address 8K selecting the third 4K memory chip RAM3.

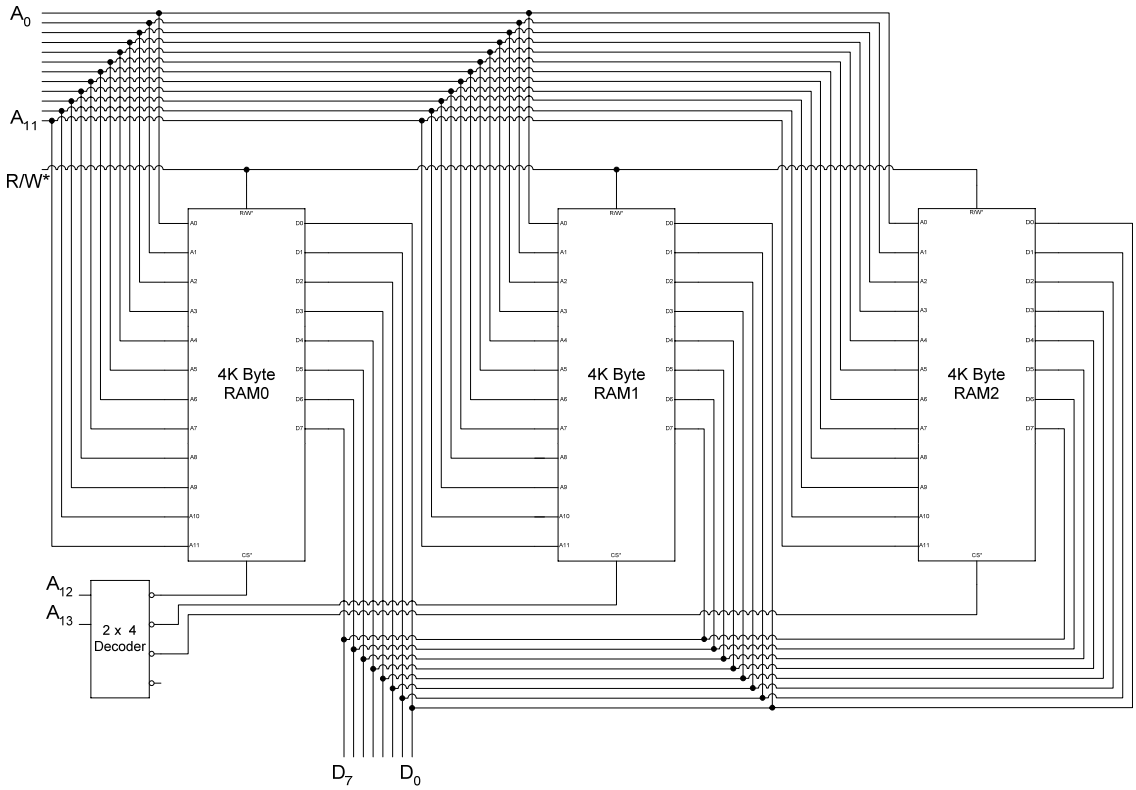


Figure 43.7a Address Decoding of three 4KByte memory chips

The memory map of the memory configuration shown in figure 43.7a is shown in figure 43.7b.

A_{13}	A_{12}	Output
0	0	CS0
0	1	CS1
1	0	CS2
1	1	CS3

4K RAM0
4K RAM1
4K RAM2
Vacant

Figure 43.7b Memory Map for the three 4K RAM chips

Memory Decoders can be implemented in different ways. The simplest method to implement is by using logic gates. The other method is to use m x n decoders. Both decoders are shown. Figure 43.8 and 43.9. In the logic based address decoder a combination of OR,

NAND and NOT gates are used to select four memory devices at Base Addresses 000H, 200H, 400H, 600H respectively. A 2 x 4 Decoder is used to decode the same memory space. A 3 x 8 Decoder divides the 64K memory space into eight equal blocks of 8K.

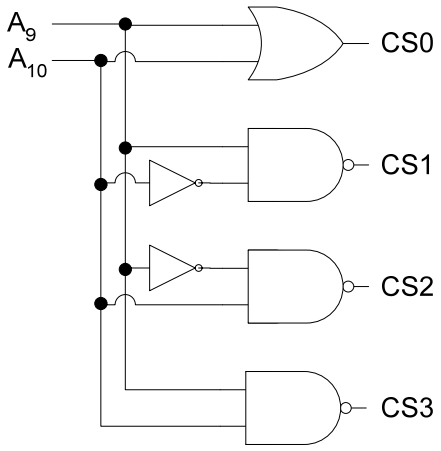


Figure 43.8 Logic Gate based Address Decoder

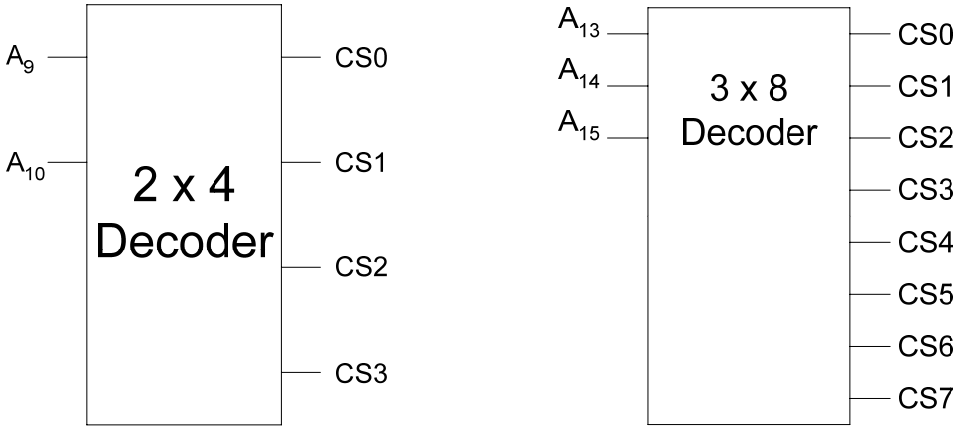


Figure 43.9 2 x 4 and 3 x 8 Decoder based Address Decoders

Introduction to FPGAs

Programmable Logic Devices are based on a programmable AND-OR gate array which are programmed to implement any function in the SOP form. The output of the AND-OR gate array can be directly used as a combinational circuit output. Provision is there to connect the output of the AND-OR gate array to a D-flip-flop for Sequential circuit operation. An FPGA is a more flexible device than PLDs as instead of a single AND-OR gate array, an FPGA device contains multiple logic blocks that can be individually programmed to perform different functions. Each Logic Block is connected to other blocks through row and column interconnects that can be programmed to connect any Logic block to another. The Logic blocks are connected to the outside world through programmable I/O blocks. The block diagram of a Field Programmable Gate Array FPGA is shown. Figure 43.10.

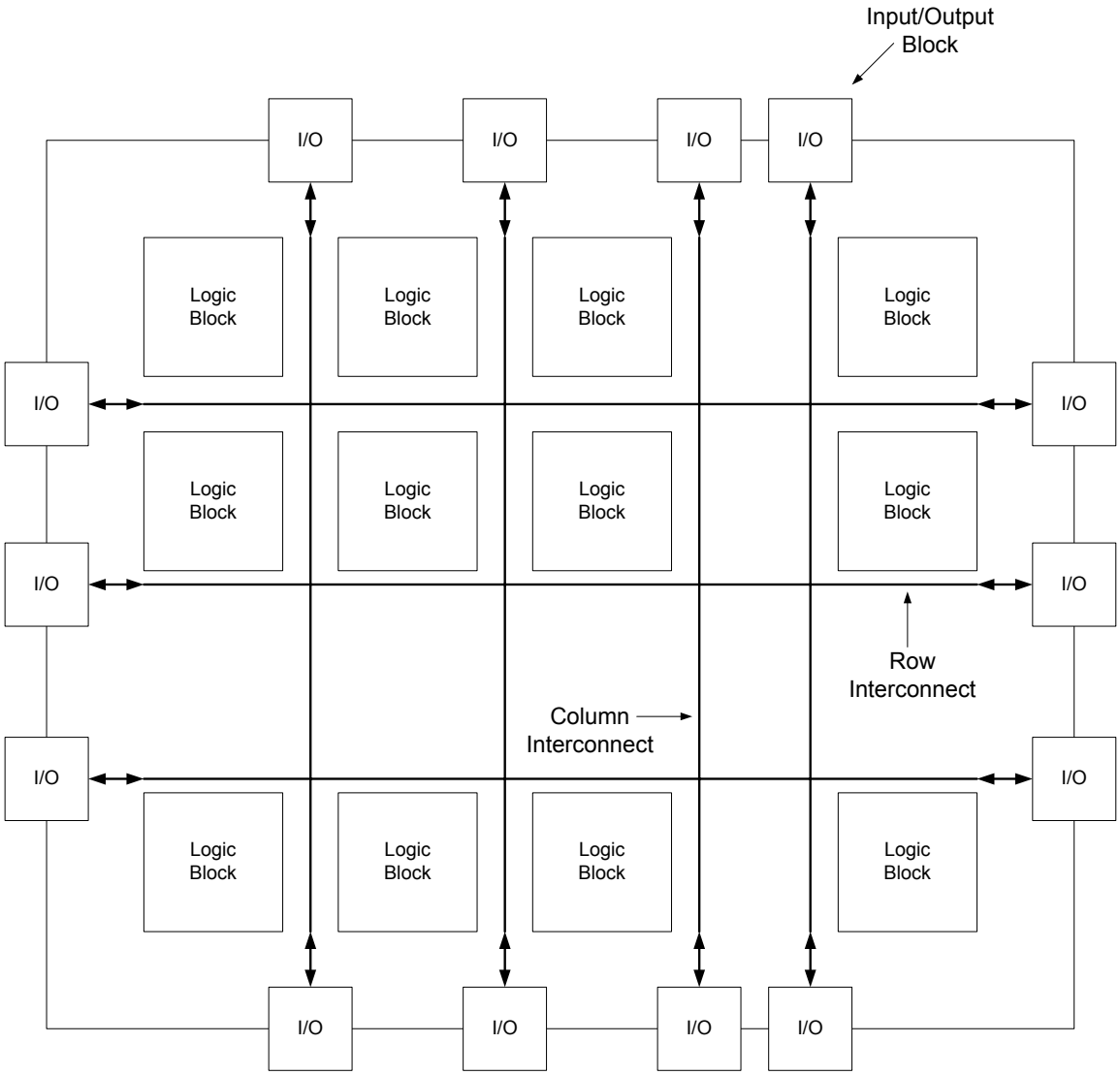


Figure 43.10 Block diagram of a FPA

Lesson No. 44

THE LOGIC BLOCK

Each Logic Block consists of a several Logic elements. The details of the Logic Element are shown in figure 44.1.

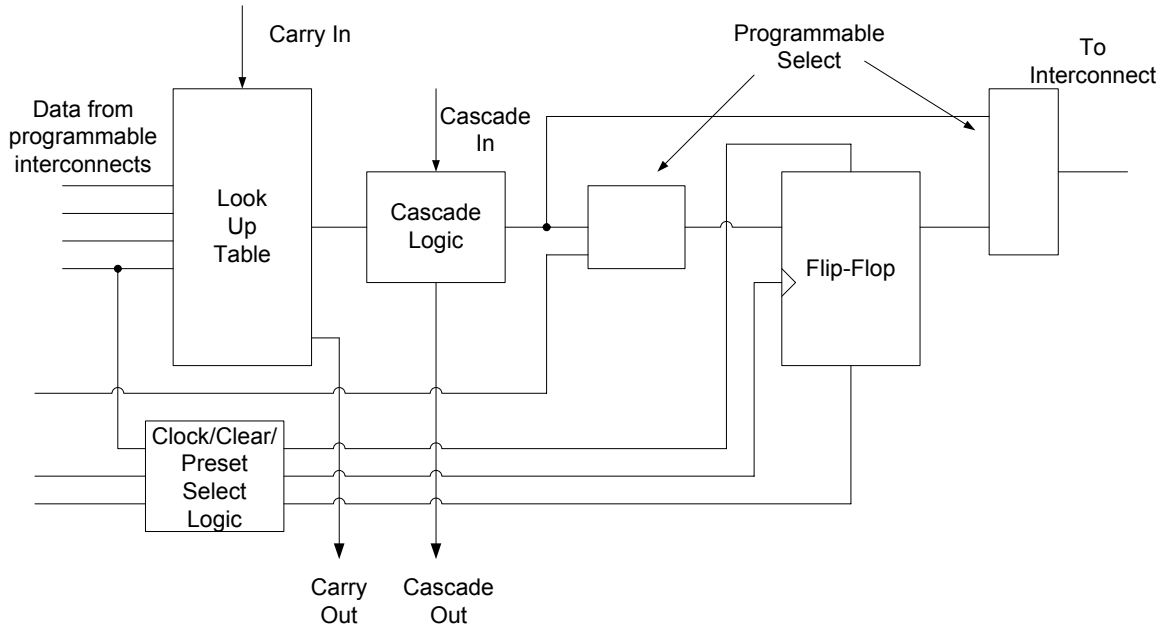


Figure 44.1 Block diagram of a Logic Element

The Logic Element

The Look-Up Table LUT has 4-inputs and can be programmed as a logic function generator. The LUT can be programmed to produce SOP functions or logic functions such as adders and comparators. When the LUT is configured as an adder the Carry In and Carry Out inputs and outputs allow for adder expansion by connecting more adders. The Cascade Logic allows one LUT to be cascaded with another LUT in other logic units. There are two Programmable selects, the first Programmable selects allows selection of either the combinational functions from the LUT output or a direct input to be connected to the input of a flip-flop. The second Programmable select allows selection of the combinational function from the LUT output or the registered functions from the flip-flop output. The clock/Clear/Preset Select Logic controls the operation of the flip-flop through the Clear and Select Asynchronous signals and the Clock Synchronous signal.

The Look-Up Table

The Look-Up Table shown in the Logic Element is implemented using a memory element that can be programmed to implement different logic functions. Two examples of LUT programmed to generate a logic function and to implement a full-adder are shown. Table 44.1

In the first example illustrated by Table 44.1a a three variable SOP function $F = \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC$ is implemented. A memory having 8 locations and storing a single bit value at each location is required to implement the three variable function. The three bit address lines which are used to select one out of eight memory locations represents the three

are represented by memory locations that store logic 1 binary value. The product terms or the Min-terms that are missing from the function are represented by memory locations that store logic 0 value. In the function shown, memory locations 2, 5 and 7 which represent the product terms $\overline{A}B\overline{C}, A\overline{B}C$ and ABC respectively have logic 1 values stored. Thus when ever addresses 2, 5 or 7 are issued, the data output is a 1 for all other addresses the data output is a 0.

Address Input			Data Output
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

$$F = \overline{A}B\overline{C} + A\overline{B}C + ABC$$

Table 44.1a LUT programmed to generate a function

Address Input			Data Output	
A	B	C_{in}	Sum	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$Sum = \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC$$
$$C_{out} = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

Table 44.1b LUT Programmed as Full-Adder

A Single bit Full-Adder can be similarly implemented by using a memory which has three storage locations, each location storing two bits. A single bit full-adder has three input variables A, B and C_{in} and two output variables Sum and C_{out}. The three address lines which select one out of eight memory locations are connected to the three input variables A, B and C_{in}. The eight two bit memory locations are programmed to represent the Sum and C_{out} functions. An address 110 represents the variables A=1 and B=1 and C_{in} = 0. The Sum output

should be 0 and the C_{out} should be 1 which is represented by the data output 01 corresponding to the address 110.

Analogue to Digital Conversion

Digital systems process digital information. The input and the output to the digital systems is in digital binary format. Real world quantities are in analogue form, which are converted into digital format for processing by the digital system. The processed digital output has to be converted back into analogue format. Mobile phones convert the analogue speech signal into a digital signal which is processed digitally. The digital signal which is received is converted back to an analogue form which one hears. Digital thermometers measure temperature which is in analogue form. The analogue signal is converted into digital format which is displayed in the form of numbers representing the temperature value. Measuring instruments such as digital voltmeters also sample an electrical signal in its analogue form. The analogue samples are digitized and displayed in the form of numbers representing voltage values. CDs which store digital audio and video files have the original audio and video analogue signals converted into digital format for storage on CDs. To replay the audio or video file the digital information is converted back into analogue form. Industrial controller system sample analogue values, digitize the sampled values, process the digitized data and convert the digitized processed information into corresponding analogue outputs.

Analogue signals are converted into Digital signals by Analogue to Digital (A/D) converters. The conversion of the analogue signal into a corresponding digital signals is done by first sampling the analogue signal and holding it stable for the A/D converter to convert the signal into a digital value.

Sample and Hold Operation

A sample and hold circuit performs two important operations. The analogue signal is sampled at regular sampling intervals to take sufficient number of discreet values at points on the analogue waveform. The more samples are taken the more accurate is the representation of the original analogue signal. The sampling frequency according to the Nyquist Criteria should be twice the maximum frequency of the highest analogue frequency. A sampled value has to be held stable for a certain minimum time to allow the A/D converter to convert the sample values into equivalent digital values. Figure 44.2a shows an input analogue signal representing temperature varying with time. To convert the analogue signal it is sampled at regular intervals. The analogue signal which is sampled at 15 equal intervals is shown in figure 44.2b. The number of samples determine the accuracy of the digitized signal. If the 15 samples are plotted the resulting signal represents reconstructed analogue signal based on the 15 samples. Figure 44.2c. The reconstructed signal is not an exact replica of the original analogue signal but is similar. The exactness of the reconstructed signal depends upon the number of samples. If the number of samples are few then the reconstructed signal losses its resemblance to the original signal. Figure 44.2d. The signal with fewer samples then the desired number of samples that are required to accurately reconstruct the signal is known as an under sampled signal. The under sampled signal is shown to lose the information in the original signal when it is reconstructed. If the original signal is over sampled by increasing the number of samples beyond 15, the reconstructed signal will be a very accurate representation of the original signal however processing too many samples may require too much processing time.

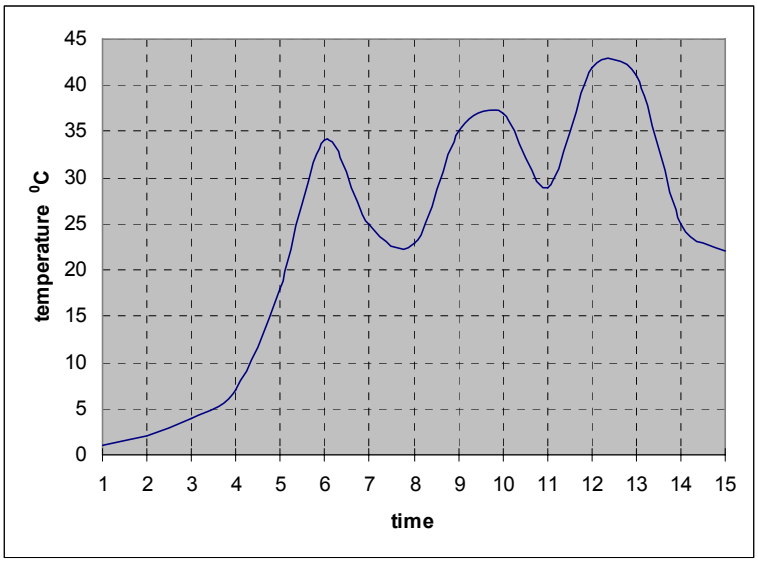


Figure 44.2a Continuous signal showing temperature varying with time

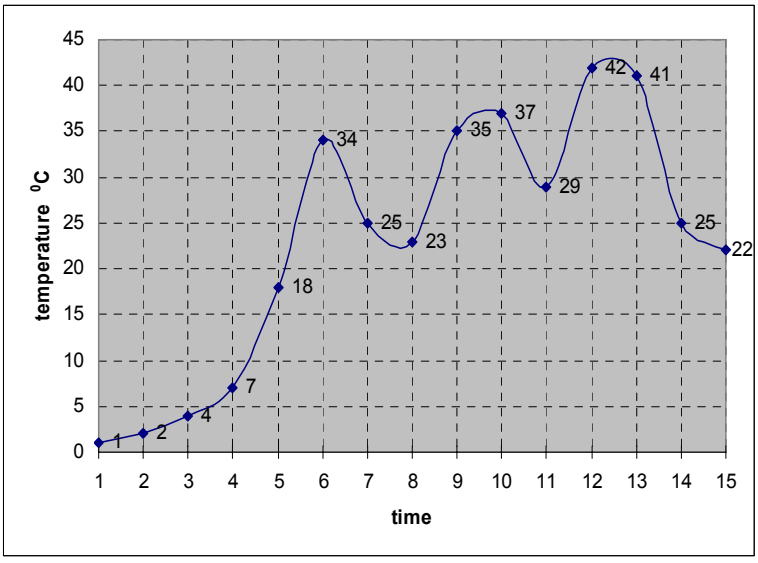


Figure 44.2b Sampling the Continuous Signal at 15 equal intervals

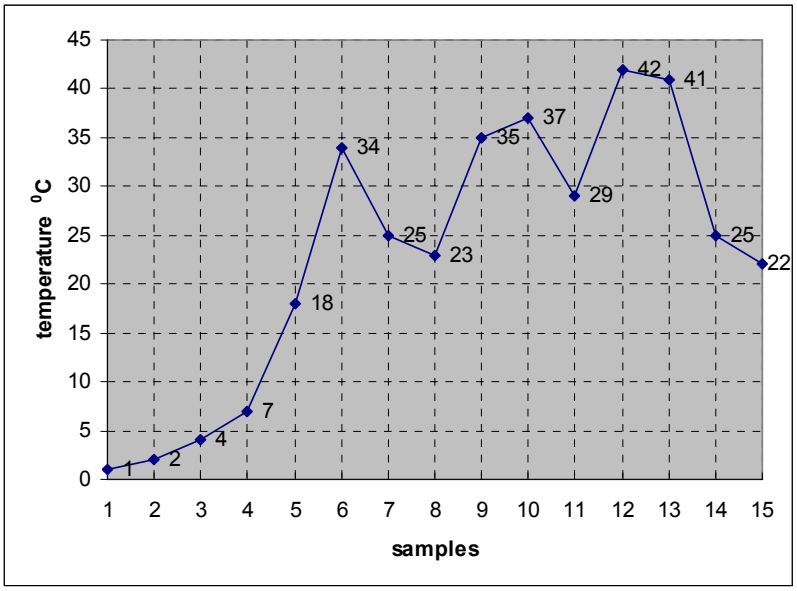


Figure 44.2c Reconstructed Signal by plotting 15 sampled values

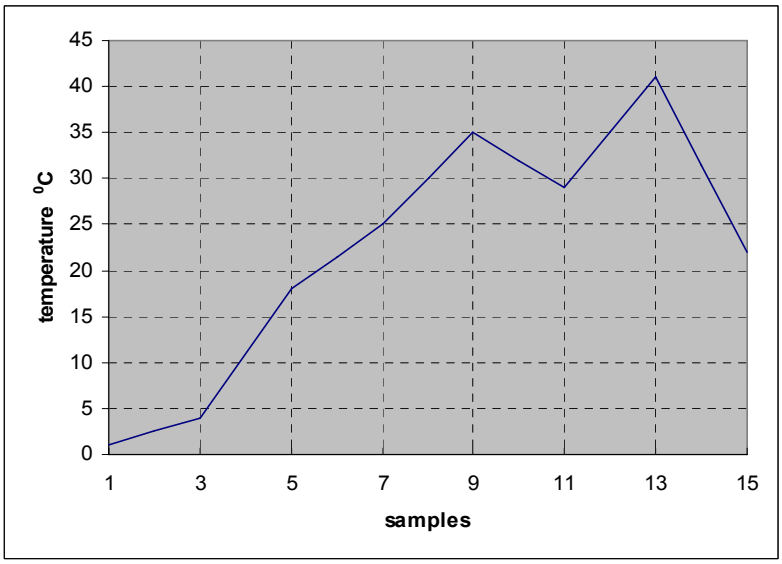


Figure 44.2d Reconstructed Signal by plotting 7 sampled values

The number of samples that are essential to accurately represent the original signal is determined by the Nyquist Criteria which requires that the sampling frequency should be twice the frequency of the sampled signal. Assuming the original signal to have a frequency of 50 Hz, the sampling which allows accurate reconstruction of the signal should be carried out at 100 Hz.

The sampled signal at Nyquist frequency have to be held stable for a minimum time period to allow the A/D converter to convert the analogue sample into a digital value. If the sampled signal is not held stable, the A/D converted would not have enough time to accurately convert the signal into a digital value. The samples that are held stable for converting into digital values by the sample and hold circuit are depicted by a staircase signal shown in figure 44.3.

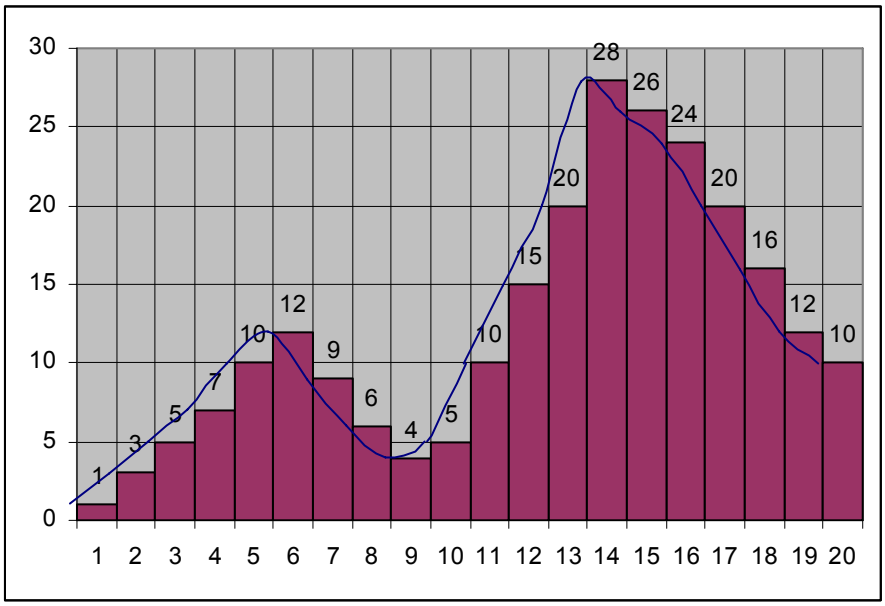


Figure 44.3a Sample and Hold signal

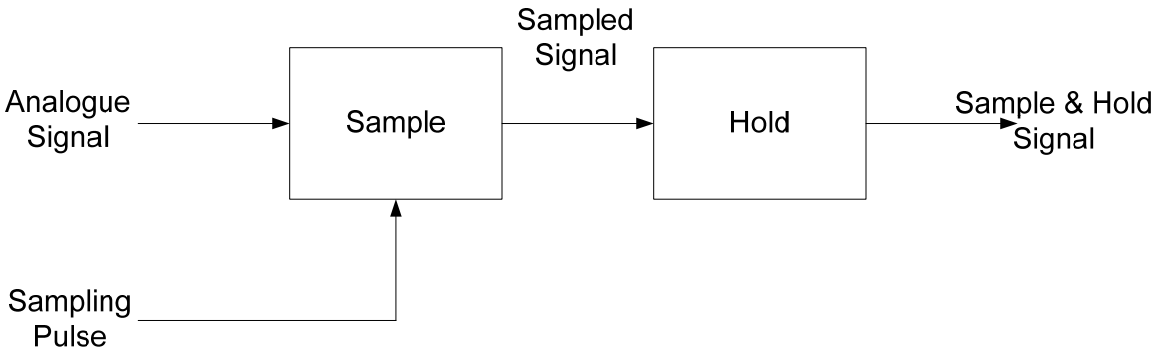


Figure 44.3b Sample and Hold Circuit

Quantization

The process of converting the analogue signal into a digital representation (code) is known as quantization. The number of bits that are used to represent the digital code determine the accuracy of the digitized signal. An analogue 220 volt signal can be represented in digital terms by a 2-bit binary number. The four possible digital values 00, 01, 10 and 11 represent four levels of the input 220 volt analogue signal. The binary value 00 represents 0 volts, 01 represents 73 volts, 10 represents 146 volts and 11 represents 220 volts. Since four values can be represented, therefore analogue voltages in the ranges 37 to 109 are represented by binary 01. Similarly voltage ranges between 110 to 183 are represented by binary 10. If a three bit representation is used then the range of analogue signals represented by the eight, 3-bit values is reduced thereby increasing the accuracy.

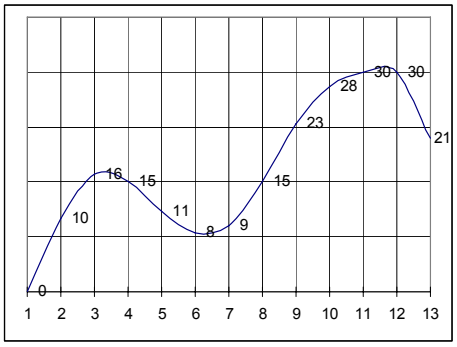


Figure 44.4a Analogue Signal

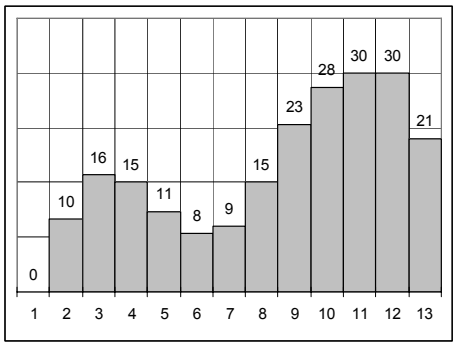


Figure 44.4b Sample & Hold Signal

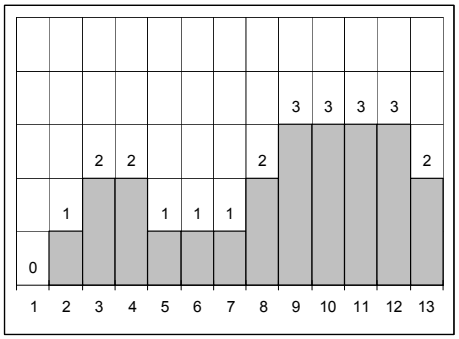


Figure 44.4c Digitized Signal

In figure 44.4 the representation of an analogue signal using 2 bits or four quantization levels is shown. The original signal has analogue value range from 0 to a peak value of 31. Figure 44.4a. The analogue signal is sampled, the output of the Sample and Hold circuit is shown in figure 44.4b. The sampled values range between 0 and 30. The sampled signal is digitized using four quantization levels or 2-bits. Figure 44.4c. The original signal having values in the range 0 to 7.5 are represented by a digital representation 00. Analogue values in the range 8 to 14.5 are represented by a digital representation 01. Analogue values in the range 15 to 22.5 are represented by digital representation 10 and the values ranging between 23 and 30 are represented by digits 11. If the quantization level is quadrupled to 16 levels the digitized representation of the analogue signal becomes more accurate. Figure 44.5.

© Virtual University of Pakistan

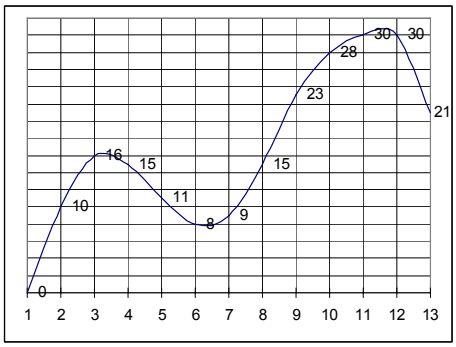


Figure 44.5a Analogue Signal

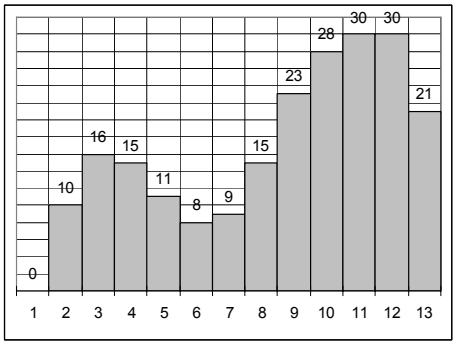


Figure 44.5b Sample & Hold Signal

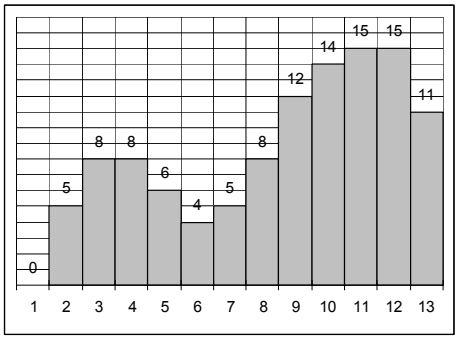


Figure 44.5c Digitized Signal

In figure 44.5, the analogue signal is shown to be digitized using a 16 level quantization. The digitized signal shown in figure 44.5c closely resembles the analogue signal as compared to the 4 level quantized signal shown in figure 44.4c.

Operational Amplifier (Op-Amp)

Operational Amplifier is a linear amplifier which has two inputs (inverting and non-inverting) and one output. It has a high voltage gain, high input impedance and low output impedance. The Op-Amp amplifies the difference signal between its inverted and non-inverted inputs. Figure 44.6a

The Op-Amp is used as an inverting amplifier and as a comparator. When the Op-Amp is used as an inverting amplifier, the input signal is applied at its Inverted input through a series resistance R_i. The output of the Op-Amp is connected to the inverted input through a feedback resistance R_f. Figure 44.6b. The voltage gain of the Inverting Amplifier is given by the relation

$$V_{out}/V_{in} = - R_f/R_i$$

When the Op-Amp is used as a comparator two voltages are applied at the inputs, when these voltages differ by a very small amount the output of the Op-Amp is driven into one of its two saturated output sates logic high or low depending upon which of the two input voltages is higher. Figure 44.6c

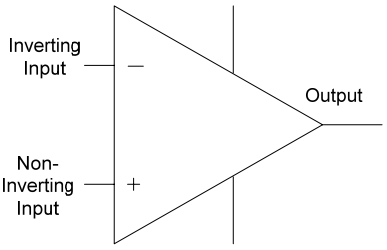


Figure 44.6a Op-Amp

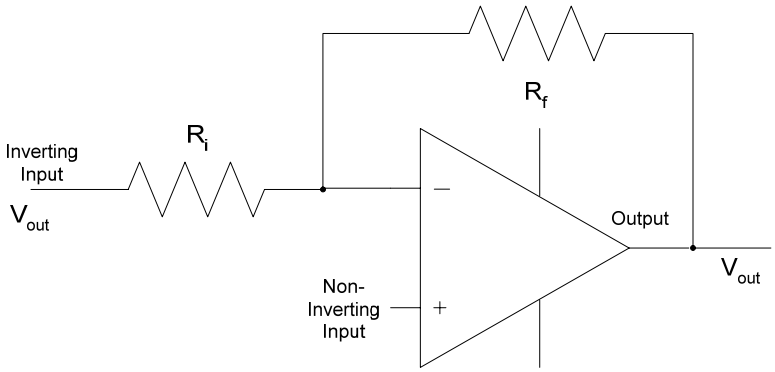


Figure 44.6b Op-Amp as an Inverting Amplifier

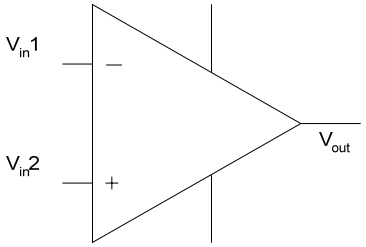


Figure 44.6c Op-Amp as a Comparator

Analogue to Digital converters use Op-Amps as an Integrator and Comparator. An Integrator integrates the input voltage. An Integrator is implemented by replacing the feedback resistance R_f by a Capacitor.

Flash Analogue-to Digital Converter

The Flash A/D converter is based on a resistor potential divider, where multiple resistors of identical value form a voltage divider. A reference voltage is applied at one end of the potential divider which divides the voltage equally across all the resistors. The input analogue voltage is applied at the non-inverting inputs of a set of Op-Amp based comparators. The inverting input of each comparator is connected to the resistive voltage divider which provides reference voltages for all the comparators. If the input voltage is larger than the reference voltage the output of the comparator is logic high otherwise it is logic low. The outputs of all the comparators are connected to the input of a priority encoder which converts the comparator outputs to a binary coded equivalent value. Figure 44.7a.

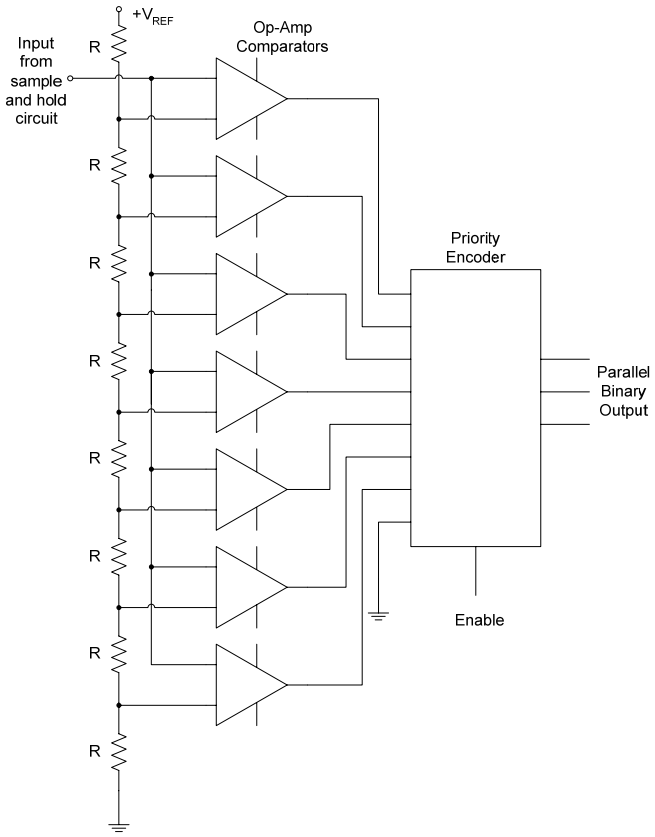


Figure 44.7a Flash A/D Converter

The analogue input sampled signal is applied at the input of the seven comparators. The inverted input of each of the seven comparators is connected to voltage divider circuit made up of eight resistors having the same value R. A reference voltage $+V_{REF}$ is connected at the top end of the voltage divider circuit and the lower end of the voltage divider is connected to the ground. The voltage drops across the eight resistors starting from the top most resistor are V_{REF}, $7/8V_{REF}$, $6/8V_{REF}$, $5/8V_{REF}$, $4/8V_{REF}$, $3/8V_{REF}$, $2/8V_{REF}$ and $1/8V_{REF}$ respectively. If the input sampled voltage input is greater than the reference input voltage for any comparator, the comparator output is logic 1, otherwise the output is logic 0. The inputs of an eight-to-three Priority Encoder are connected to the comparator outputs. The lowest priority input of the Encoder is grounded. The priority encoder is enabled at each sampled input and a 3-bit code representing the value of the input sample appears at the output. Consider an example, the input sample is 4.2 volts. The reference voltage V_{REF} is equal to 8 volts, the seven reference voltages applied at the inverted inputs of the seven comparators starting from the first comparator are 7, 6, 5, 4, 3, 2 and 1 volts respectively. With an input of 4.2 volts the outputs of the first three comparators are set to logic 0 and the outputs of the lower four comparators are set to logic 1 which sets the encoders first three inputs to inactive-low and the next four inputs to active high. The highest priority input the encoder outputs 100 which is binary equivalent of 4. Figure 44.7b.

© Virtual University of Pakistan

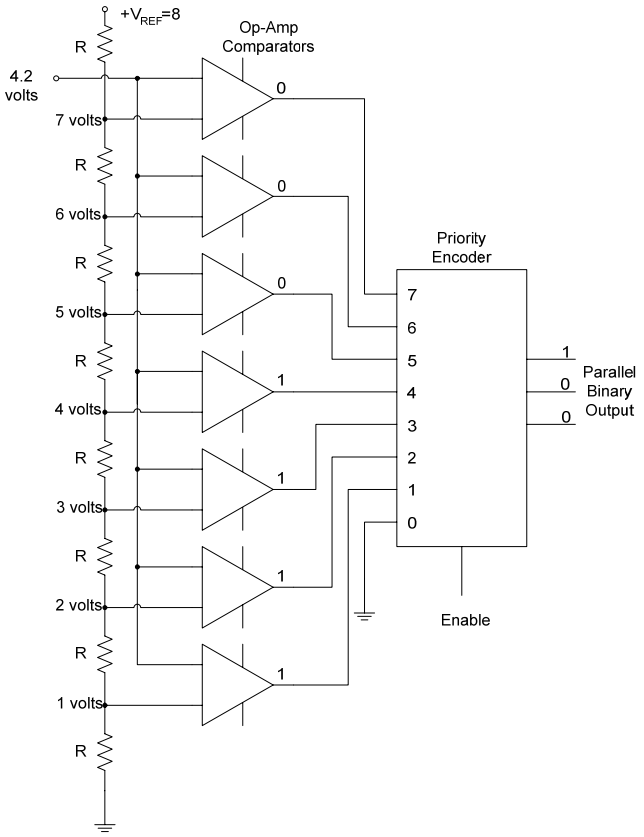


Figure 44.7b 3-bit FLASH A/D Converter

The 3-bit Flash converter requires seven comparators, a 4-bit Flash converter requires fifteen converters. A large number of comparators are required to implement a reasonable-sized converter. The advantage is that the conversion is done in parallel and the binary equivalent value is available at the output of the converter almost instantaneously. Flash converters are used for high speed conversion applications such as conversion of analogue video signals into digital signals. For accurate reproduction of the digital signal, Flash A/D converters are based on high number of Quantization levels which requires the use of many Op-Amp based Comparators which makes the Flash converters expensive and power hungry

Figure 44.8 shows a set of sampled analogue voltage inputs applied at the input of the Flash converter shown in the figure 44.7. The reference voltage of the Flash converter is set to 8 volts. At each sampling interval an enable pulse allows the Flash converter to convert the corresponding analogue input voltage sample to be converted and represented in its binary form. Figure 44.9.

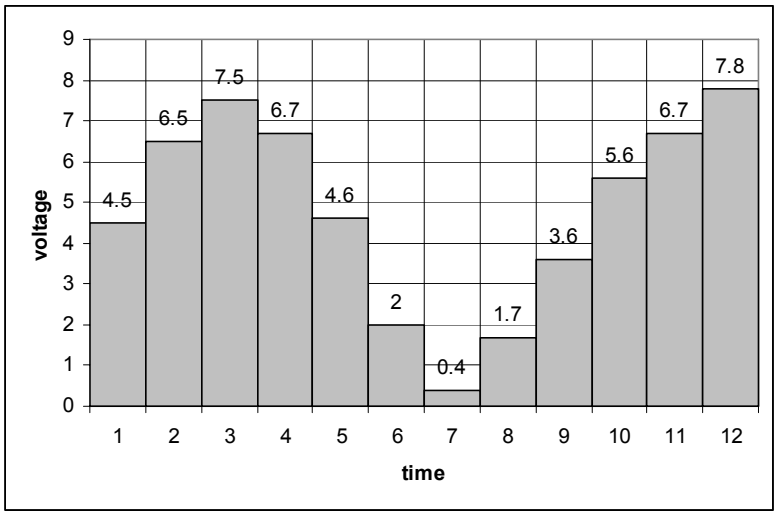


Figure 44.8 Input analogue voltage samples

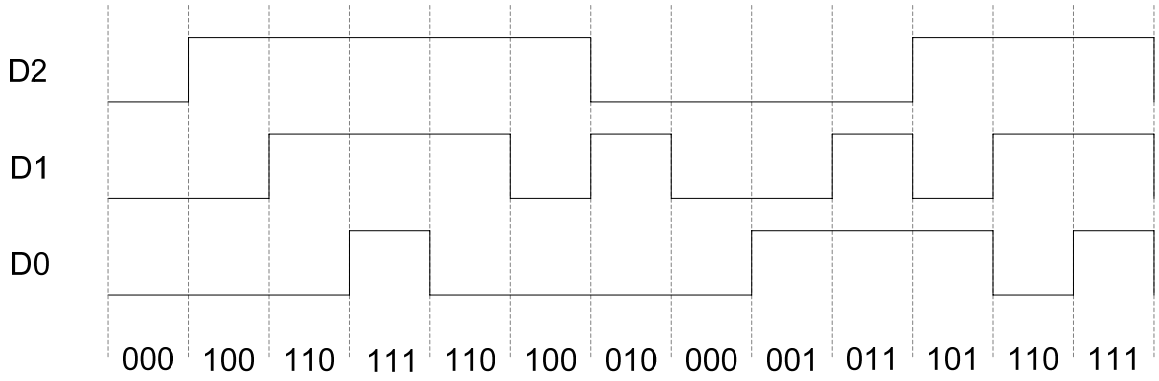


Figure 44.9 Binary output representing input analogue voltage samples

Dual-Slope Analogue to Digital Converter

The Dual-Slope A/D converter is used in digital voltmeters and other types of measuring instruments. A Dual-Slope A/D converter is slower than the Flash Converter. The circuit diagram of the converter is shown. Figure 44.10. The converter consists of a switch. Initially, the switch connects the circuit to the input analogue voltage which is to be converted into its corresponding binary representation. During the conversion process the switch connects the circuit to a negative reference voltage. The switching between the input voltage and the reference voltage allows a capacitor connected between the Op-Amp output and input to be charged and discharged. An Op-Amp based Integrator integrates the analogue input voltage over a fixed period of time. An Op-Amp based Comparator compares the output of the Integrator with the ground voltage to enable or disable a counter circuit. A counter and latch counts the binary output corresponding to the analogue input value. A logic control circuit is used to switch between analogue voltage input and the reference voltage. It also enables/disables the latch.

© Virtual University of Pakistan

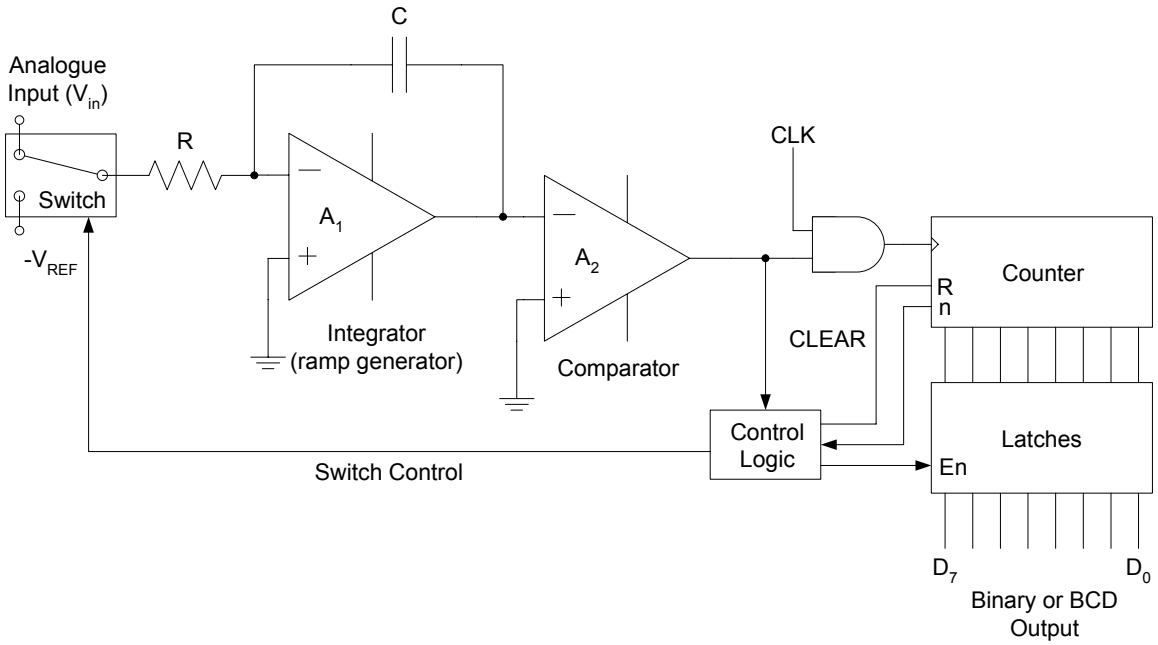


Figure 44.10 Dual-Slope A/D Converter

The first Op-Amp is connected as an Integrator. Initially, the counter is reset and has a zero count. The Input switch is connected to the Analogue input which is to be converted into equivalent binary value. The counter is reset to count zero by the Control Logic circuit. It also sets the switch to the Analogue input voltage. The Input analogue voltage is assumed to be constant for the duration of the conversion process. Due to the high input impedance of the Integrator, the current from the Analogue Input source flows through the Resistor R and the Capacitor C. The Capacitor will charge and there will be a negative-going linear voltage ramp at the output of A_1. The non-inverted input of the Comparator is connected to the ground, therefore as the inverted input of the comparator becomes negative, the output changes to logic 1. The Logic 1 output triggers the Control Logic which in turn resets the counter. The logic 1 output enables the AND gate which allows the clock signal to be applied at the counter clock input which increments the counter at each clock pulse. The Integrator output remains at negative voltage as the negative-going linear ramp continues the integration process. As the counter count reaches its maximum count value (terminal count), it rolls over and sends a signal to the control Logic circuit which switches the switch to $-V_{REF.}$ The Capacitor which is charged to a positive input voltage discharges resulting in a positive going slope at the output of integrator. When the voltage at the inverted input of the comparator reaches zero volt, the comparator output become logic 0 disabling the AND gate and therefore inhibiting the counter from counting. The Control Logic circuit sends a pulse which loads the latch with the count value.

Table 44.2 depicts the working of the Dual-Slope A/D converter. At interval t = 0 the converter is switched to the V_{in} input which is assumed to remain constant during the conversion operation. The capacitor starts charging at a constant rate. The output of the Integrator (voltage output) decreases at constant slope (-V). The output of the comparator is set to 1 enabling the clock signal and incrementing the counter. The converter remains connected to V_{in} for a fixed duration determined by time interval t = n. The duration of the interval is determined by the maximum count value of the counter. During this interval the capacitor has charged to a maximum value determined by the input voltage. At time interval t =

V_{ref}. At interval t=n+1 the capacitor begins to discharge as now it is connected to a negative voltage $-V_{ref}$. The Integrator output starts increasing towards a 0 voltage at a constant rate. The output of the Comparator is logic high allowing the counter to count. At interval t = n+m the capacitor has completely discharged and the comparator inputs become equal setting its output to 0. The clock signal is disabled, disabling the counter from counting. The count value represents the input voltage. Interval m is determined by the magnitude of the charge stored on the capacitor. Higher the voltage stored on the capacitor longer it will take to discharge to 0 volts, thereby allowing the counter to a larger count value. If the input analogue voltage is small, the capacitor will be charged to a smaller voltage. It will therefore discharge in a shorter interval of time allowing the counter to count to a small value.

Time interval t	Input signal	Output of Integrator	Output of Comparator	Clock Input	Counter
0	V_{in}	-V	1	enabled	Counting
1	V_{in}	-V	1	enabled	Counting
n	V_{in}	-V	1	enabled	Terminal count reached. Counter reset. Switched to $-V_{ref}$
n+1	$-V_{ref}$	-V	1	enabled	Counting
n+2	$-V_{ref}$	-V	1	enabled	Counting
n+m	$-V_{ref}$	0	0	disabled	Binary value representing $V_{analogue}$

Table 44.2 Operation of Dual-Slope A/D Converter

Lesson No. 45

SUCCESSIVE –APPROXIMATION ANALOGUE TO DIGITAL CONVERTER

The most commonly used A/D for converting analogue values to corresponding binary values is the Successive-Approximation A/D converter. It has a fixed conversion time and is faster than the Dual-Slope A/D converter. The Successive-Approximation converter is however slower than the Flash converter. The main components of a Successive-Approximation A/D Converter are the Successive Approximation Register (SAR), a Digital to Analogue Converter and a Comparator. Figure 45.1.

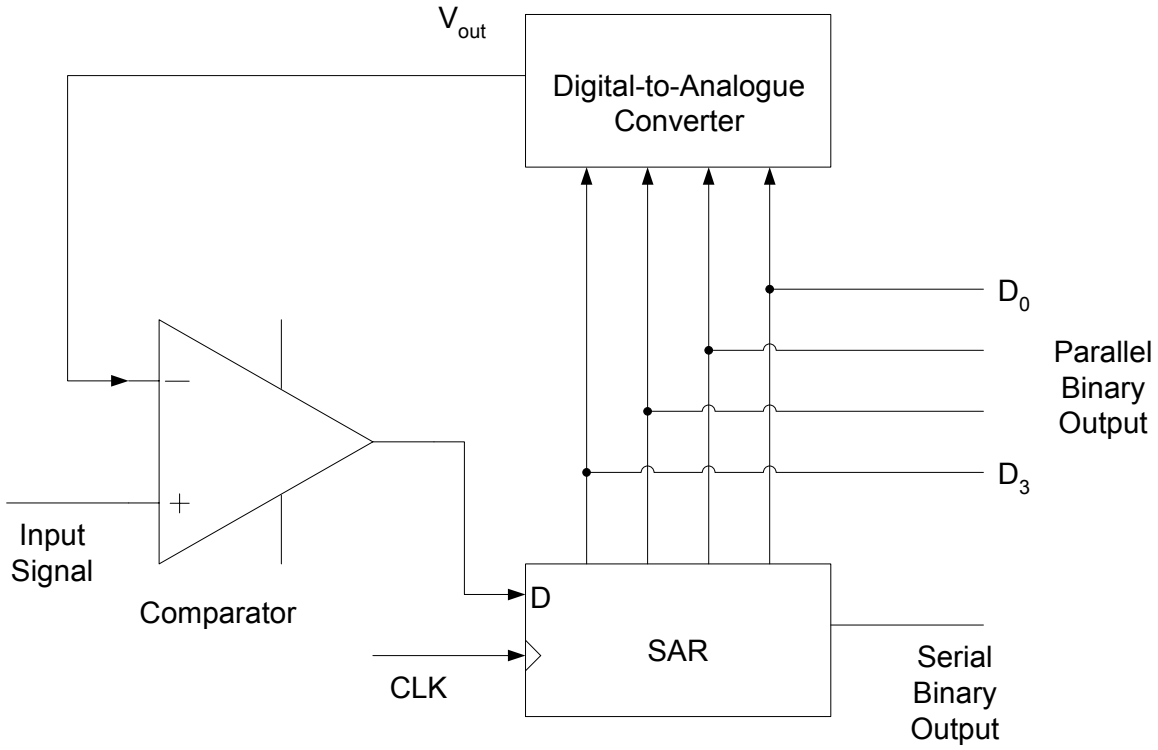


Figure 45.1 Successive-Approximation 4-bit A/D Converter

The analogue signal sample which is to be converted into its corresponding binary value is applied at the non-inverted input of the Comparator. Initially, the most-significant bit D_3 is set to logic 1 by the Successive Approximation Register (SAR). The Digital-to-Analogue converter converts the binary input digit 1000 to it equivalent analogue value. The output analogue value is connected to the inverted input of the comparator. If the applied Input signal is larger than the signal generated by the D/A converter the output of the comparator is logic 1 which sets the most significant bit D_3 of the SAR to logic 1. The next most significant bit is set to 1 and the new binary number 1100 is applied at the input of the D/A converter. The analogue output is applied at the comparator input which generates logic 1 or 0 depending upon the magnitude of the two signals applied at the inputs of the comparator. Depending upon logic 0 or 1 produced at the output of the comparator, the SAR sets or resets the next most significant digit. This procedure is repeated for all the binary digits.

Assuming an analogue signal of 5.2 volts applied at the input of the 4-bit A/D converter. Initially, the SAR sets its 4 bits to 1000, which is converted to 8 volts by the D/A converter. Since the applied signal (5.2 V) is less than 8 V signal, therefore the SAR resets the most significant bit and sets the next most significant bit 0100. The D/A converts the 4 bit number to

© Virtual University of Pakistan

next most significant bit. (0110). The 4-bit number is converted into an analogue value of 6 volts by the D/A converter. Since the analogue value is greater than the input signal therefore the SAR resets the bit and sets the least significant bit to 1 (0101). The number is converted into an analogue value by the D/A converter. The converted value (5 V) is less than the input signal, therefore the four bits are retained (0101). Since the Successive-Approximation A/D converter is a 4-bit converter therefore the 0101 represents the final value at the end of the conversion process. Table 45.1. The duration of the conversion depend upon the quantization level, a 6-bit converter completes its conversion in 6 time periods.

V_{in}	SAR output	D/A output	Comparator output
5.2	1000	8	0
5.2	0100	4	1
5.2	0110	6	0
5.2	0101	5	1

Table 45.1 Successive-Approximation D/A Conversion

Analogue-to-Digital Converter Errors

Analogue to Digital converters exhibit three different types of errors during their conversion operation. The three errors encountered during the conversion operation are the Missing Code, Incorrect Code and the Offset error. The three errors are represented through graphs. Figure 45.2. A test signal which is an ideal linear ramp is assumed for testing for the three errors.

1. Missing Code

In the graph illustrated to highlight the missing code '1001' a linearly increasing analogue voltage is applied at the input of an A/D converter and the binary output is plotted. Ideally, a staircase output should be obtained showing the linearly increasing binary values. Figure 45.2a. The graph shows a missing binary code 1001, represented instead by 1000. The missing code in the case of a Flash converter is due to the failure of a comparator which fails to provide an appropriate input to the Priority Encoder. The Priority Encoder therefore outputs the same code for analogue values 8 and 9.

2. Incorrect Code

Incorrect Code at the output of the A/D converter is due to a particular bit stuck at some fixed logic value. In the graph illustrating an example of Incorrect Code, the bit next to the least significant is permanently stuck at logic 0. Therefore, for analogue values 2, 3, 6, 7, 10, 11, 14 and 15 the binary output is 0, 1, 4, 5, 8, 9, 12 and 13 respectively. Figure 45.2b.

© Virtual University of Pakistan

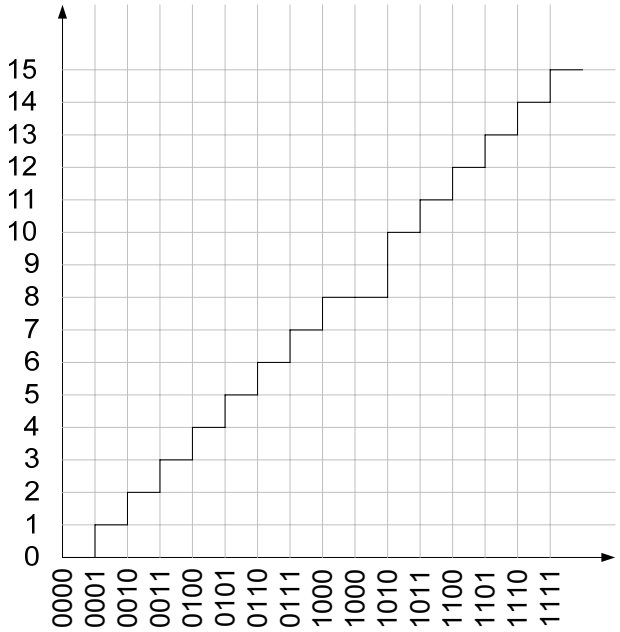


Figure 45.2a Missing Code '1001'

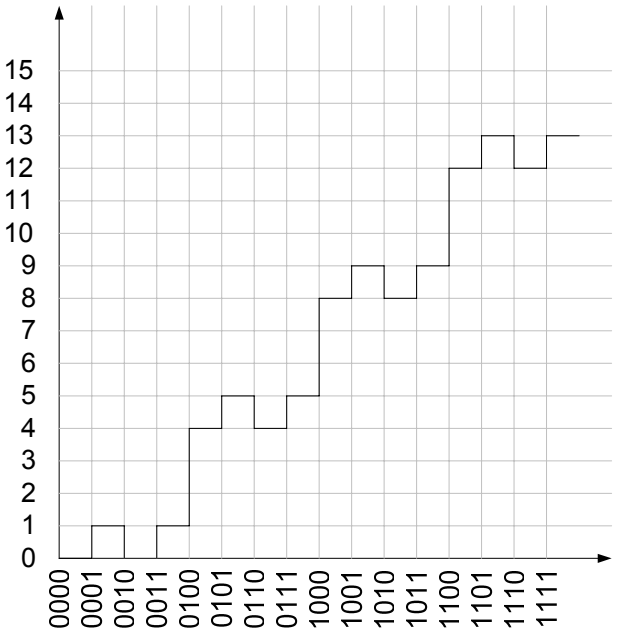


Figure 45.2b Incorrect Code

3. Offset Error

The offset error occurs when the binary output of the A/D converter represents a value which is greater than the actual analogue input signal value. The error is due to a fault in the comparator circuit. The offset error can be compensated by adjusting the output with respect to the amount of offset error. The missing and incorrect code however can not be

compensated. Figure 45.2c. The diagram illustrates an offset error of 2 volts, as the corresponding output value for each analogue input exceeds by 2 volts.

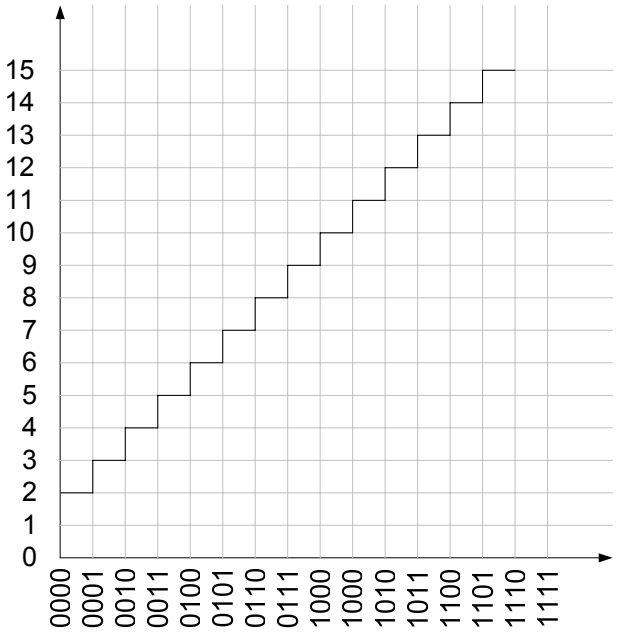


Figure 45.2c Offset Error

Digital to Analogue Conversion

Digital binary signals are converted into analogue signals using Digital to Analogue Converters. Generally two types of D/A Converters are used, the Binary-Weighted-Input D/A Converter and R/2R Ladder D/A Converter.

Binary-Weighted-Input Digital to Analogue Converter

The Binary-Weighted-Input D/A converter is based on a summer circuit which sums the input currents based on the binary input and represents it as a voltage output. In the Binary-Weighted-Input Method a resistor network is used with resistor values representing the binary weights of the input bits of the digital code. The binary (digital input) is applied at the resistor inputs. A current will flow through the resistor if the input voltage applied is logic high. No current flows through a resistor if the input voltage applied is logic low. The magnitude of the current flowing through each resistor depends upon the value of the resistor. The total current flowing through each resistor adds up and flows through the feedback resistor R_f which is connected between the output and the inverting input of the Op-Amp. The output voltage of the Op-Amp is determined by the voltage drop across the R_f resistance. Figure 45.3.

For a D/A converter with weighted resistors 8K, 4K, 2K and 1K respectively and the feedback resistor of 2K ohms. The output voltages for binary inputs 0000 to 1111 are shown. Table 45.2.

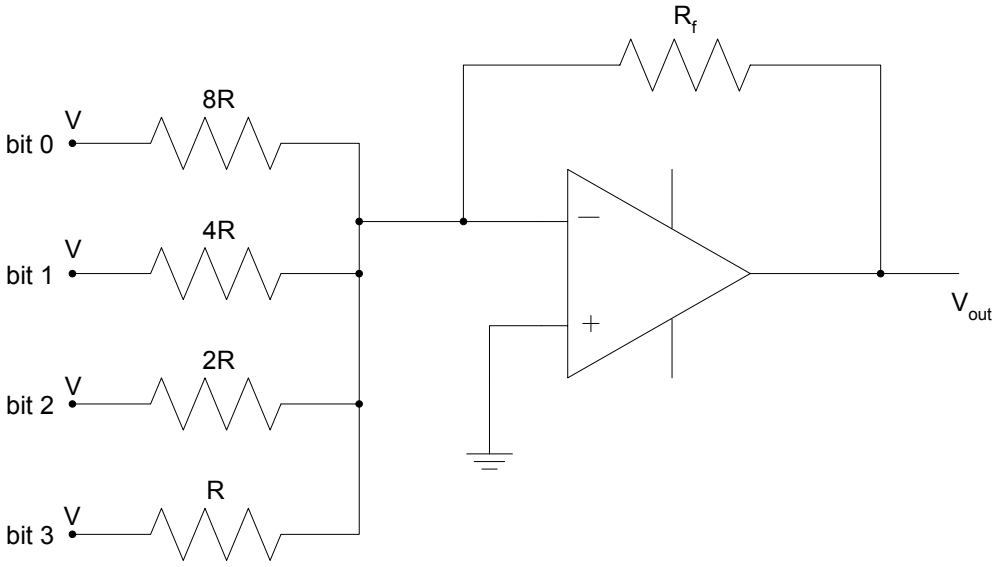


Figure 45.3 A 4-bit Binary-Weighted-Input D/A Converter

Input	Current through (mA)					V_{out} (volts)
	R	2R	4R	8R	R_f	
0000	0	0	0	0	0	0
0001	0	0	0	0.625	0.625	-1
0010	0	0	1.25	0	1.25	-2
0011	0	0	1.25	0.625	1.875	-3
0100	0	2.5	0	0	2.5	-4
0101	0	2.5	0	0.625	3.125	-5
0110	0	2.5	1.25	0	3.75	-6
0111	0	2.5	1.25	0.625	4.375	-7
1000	5	0	0	0	5.0	-8
1001	5	0	0	0.625	5.625	-9
1010	5	0	1.25	0	6.25	-10
1011	5	0	1.25	0.625	6.875	-11
1100	5	2.5	0	0	7.5	-12
1101	5	2.5	0	0.625	8.125	-13
1110	5	2.5	1.25	0	8.75	-14
1111	5	2.5	1.25	0.625	9.375	-15

Table 45.2 D/A Output voltages for binary inputs 0000 to 1111

The disadvantage of the converter is the resistors having weighted values that are required to implement the converter. An 8-bit converter requires eight weighted resistors which have exact values otherwise the output of the converter is not accurate. Resistors which have values which are exact multiples of each other are difficult to implement therefore these D/A converter are not used for applications where multiple bit binary numbers are to be converted into corresponding analogue values.

The R/2R Ladder Digital to Analogue Converter

The R/2R D/A Converter derives its name from the resistor network having values R and 2R. This converter also overcomes the problem of having multiple resistors having weighted values. The circuit diagram of the R/2R converter is shown. Figure 45.4.

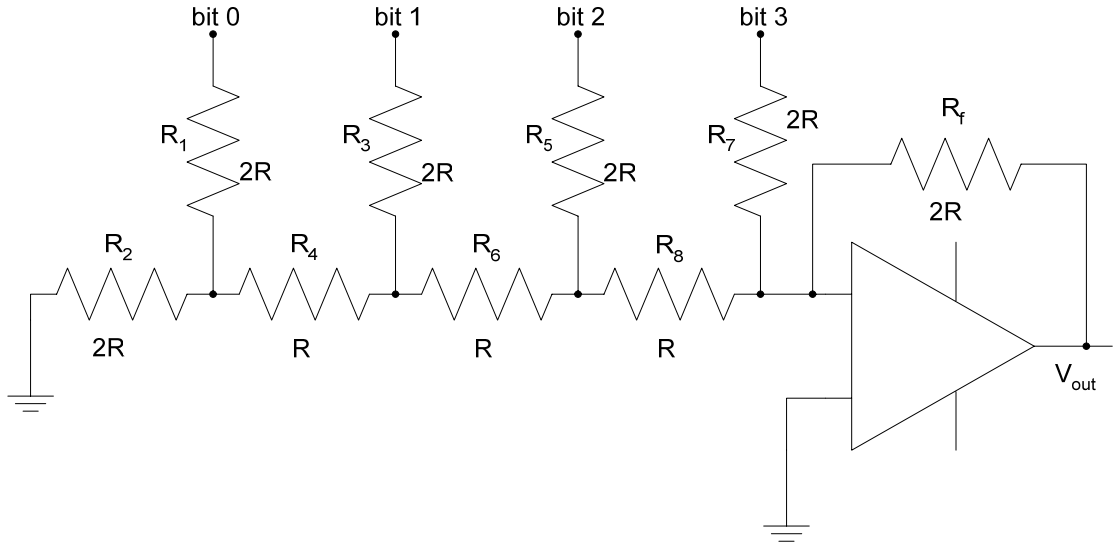


Figure 43.2 R/2R Ladder D/A Converter

Depending upon the binary input, the R/2R resistor network simplifies to an equivalent network which determines the total current flowing through the feedback resistance R_f. For example, when 0001 binary is applied the R/2R circuit simplifies to the circuit shown. Figure 45.5a. Simplifying the circuit further reduces it to the Thevenin's equivalent circuit. Figure 43.5b. The current flowing through the feedback resistance R_f is defined by the equation

$$I = \frac{0.625V}{2R}$$

The output voltage V_{out} is represented by the voltage drop across R_f. Table 45.3 gives a summary of the total current and the output voltage for each of the 16 combinations of 4-bit binary input.

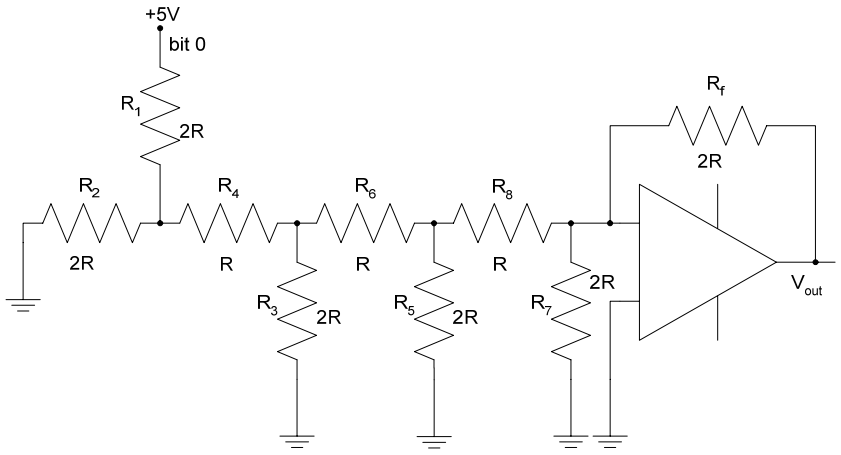


Figure 45.5a The R/2R resistor network with binary 0001

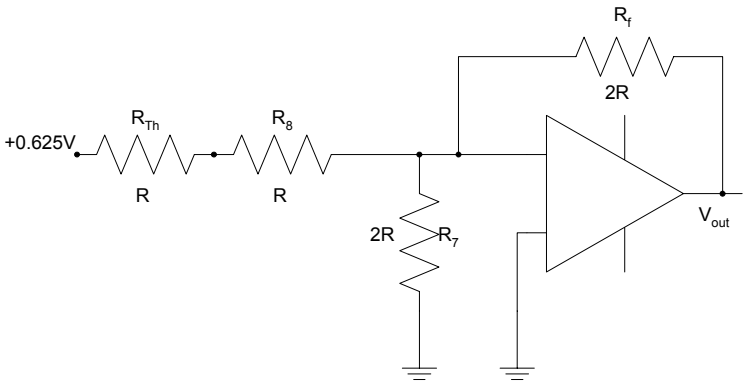


Figure 45.3b The equivalent R/2R resistor network with binary 0001

Input	V_{th} (volts)	Current through R_f	V_{out}
0000	0	0	0
0001	0.625V	0.625V/2R	-0.625V
0010	1.25V	1.25V/2R	-1.25V
0011	1.875V	1.875V/2R	-1.875V
0100	2.5V	2.5V/2R	-2.5V
0101	3.125V	3.125V/2R	-3.125V
0110	3.75V	3.75V/2R	-3.75V
0111	4.325V	4.325V/2R	-4.325V
1000	5V	5V/2R	-5V
1001	5.625V	5.625V/2R	-5.625V
1010	6.25V	6.25V/2R	-6.25V
1011	6.875V	6.875V/2R	-6.875V
1100	7.5V	7.5V/2R	-7.5V
1101	8.125V	8.125V/2R	-8.125V
1110	8.75V	8.75V/2R	-8.75V
1111	9.325V	9.325V/2R	-9.325V

Table 45.3 D/A Output voltages for binary inputs 0000 to 1111

Performance characteristics of Digital-to-Analogue Converters

1. Resolution

Resolution is defined as the number of bits that are converted. It is also defined as a reciprocal of the number of discrete steps in the output represented as percentage. The resolution of a 4-bit D/A converter is therefore represented as (1/15) 100 = 6.67%. An 8-bit D/A converter has a resolution of (1/63) 100 = 1.59%.

2. Accuracy

Accuracy of a D/A converter is determined by comparing the actual output of a D/A converter with the expected output. It is expressed as a percentage of the full-scale or maximum output voltage. If for example, the accuracy is ±0.1 %, for a D/A converter which has a maximum output of 20 volts, the maximum error for any output voltage is (20)(0.001) = 20 mV. Ideally, the accuracy should not be worse than ±1/2 of the least significant bit. For an 8-bit D/A converter, the least significant bit is 0.39% of the full-scale. The accuracy should be one-half of 0.39%, that is ±0.195%. In terms of voltage, consider that the full-scale output of the 8-bit D/A converter is 64 volts. The maximum error that is allowed is (64)(0.00195) = 0.1248 volts = 125 mV. Each discrete step of the D/A converter is equal to 0.25 volts = 250 mV. Assuming that 00000001 is applied at the input of the D/A converter, the exact analogue output should be 250 mV. If there is an error in the output voltage then the acceptable range of voltages representing 00000001 are from 125 mV to 375 mV. A voltage output which is less than 125 mV represents the binary value 00000000 and a voltage output which exceeds 375 mV represents the binary value 00000010. Thus the error should be within ±1/2% of the least significant bit.

3. Linearity

The output of the D/A converter should be a straight line when the binary input is varied between its minimum and maximum values. An offset error is determined by the output voltage when the binary input bits are all set to logic 0.

4. Monotonicity

The output of the D/A converter should give an increasing analogue voltage output when the binary input is varied between its minimum and maximum values. However, if the D/A converter outputs a lower voltage than its preceding output voltage the converter is said to exhibit non-monotonic behavior.

5. Settling Time

Ideally, the D/A converter should immediately result in an analogue output corresponding to the input binary value. A D/A converter however takes a finite amount of time to output an analogue value corresponding to the binary input. The settling time of a D/A converter is defined as the time the D/A converter takes to settle within ±1/2 least significant bit of its final value when a change occurs in the input. Assume that the input to an 8-bit D/A converter is 00000101 which is represented by 1.250 Volts. The binary input changes to 00000001 which is represented by 250 mV. The output of the D/A converter changes form 1.250 volts to 375 mV (±1/2 least significant bit of the final value 250 mV) in 20 msec. Thus the settling time of the D/A converter is 20 msec.