APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

KTU SPECIAL

We can together dream the B.tech

APJ ABDUL KALAM
TECHNOLOGICAL UNIVERSITY
സാങ്കേതിക സർവ്വകലാശാല

• KTU STUDY MATERIALS	• SYLLABUS
• KTU LIVE NOTIFICATION	• SOLVED QUESTION PAPERS

JOIN WITH US

WWW.KTUSPECIAL.IN KTUSPECIAL t.me/ktuspecial1

SYLLABUS

Module 1: Introduction to digital Systems ;-Digital abstraction: Number Systems – Binary, Hexadecimal, grouping bits, Base conversion; Binary Arithmetic – Addition and subtraction, Unsigned and Signed numbers; Fixed-Point Number Systems; Floating-Point Number Systems Basic gates- Operation of a Logic circuit; Buffer; Gates - Inverter, AND gate, OR gate, NOR gate, NAND gate, XOR gate, XNOR gate; Digital circuit operation - logic levels, output dc specifications, input dc specifications, noise margins, power supplies; Driving loads - driving other gates, resistive loads and LEDs.

Module 2: Combinational Logic Design: –Boolean Algebra - Operations, Axioms, Theorems; Combinational logic analysis - Canonical SOP and POS, Minterm and Maxterm equivalence; Logic minimization - Algebraic minimization, K-map minimization, Dont cares, Code convertors.
Modeling concurrent functionality in Verilog:- Continuous assignment - Continuous Assignment with logical operators, Continuous assignment with conditional operators, Continuous assignment with delay

Module 3: MSI Logic and Digital Building Blocks
MSI logic - Decoders (One-Hot decoder, 7 segment display decoder), Encoders, Multiplexers, Demultiplexers; Digital Building Blocks - Arithmetic Circuits - Half adder, Full adder, half subtractor, full subtractor;Comparators.
Structural design and hierarchy - lower level module instantiation, gate level primitives, user defined primitives, adding delay to primitives.

Module 4: Sequential Logic Design :- Latches and Flip-Flops- SR latch, SR latch with enable, JK flipflop, D flipflop, Register Enabled Flip-Flop, Resettable Flip-Flop. Sequential logic timing considerations; Common circuits based on sequential storage devices - toggle flop clock divider, asynchronous ripple couter, shift register. **Finite State Machines** :-Finite State Machines - logic synthesis for an FSM,FSM design process and design examples; Synchronous Sequential Circuits -Counters;**Verilog (Part 2) :** -Procedural assignment; Conditional Programming constructs; Test benches; Modeling a D flipflop in Verilog; Modeling an FSM in Verilog.

MODULE -3

MSI Logic and Digital Building Blocks

MSI logic - Decoders (One-Hot decoder, 7 segment display decoder), Encoders, Multiplexers, Demultiplexers; Digital Building Blocks - Arithmetic Circuits - Half adder, Full adder, half subtractor, full subtractor;Comparators. Structural design and hierarchy - lower level module instantiation, gate level primitives, user defined primitives, adding delay to primitives.

MSI logic

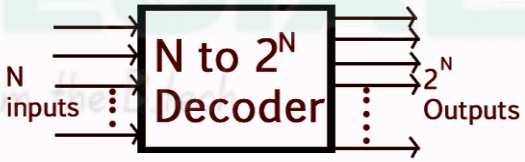
MSI or Medium scale integrated circuits are a set of basic, elementary logic circuits. They are available as ICs (integrated chips) and implement specific. MSIs can be used as a stand-alone IC or in combination with other ICs in order to implement combinational circuits in various applications. In this module, you will study many of the common types of MSI devices.

1. Decoders

A binary decoder is a digital circuit that converts a binary code into a set of outputs. The binary code represents the position of the desired output and is used to select the specific output that is active. Binary decoders are the inverse of encoders and are commonly used in digital systems to convert a serial code into a parallel set of outputs.

The basic principle of a binary decoder is to assign a unique output to each possible binary code. For example, a binary decoder with 4 inputs and 2^4 = 16 outputs can assign a unique output to each of the 16 possible 4-bit binary codes.

A **decoder** is a **combinational circuit** that converts binary information from **n input**



lines to a maximum of **2^n unique output lines**.

1.1. One-Hot decoder

One Hot Encoding is a *method for converting categorical variables into a binary format.* It creates new columns for each category where 1 means the category is present and 0 means it is not. The primary purpose of One Hot Encoding is to ensure that categorical data can be effectively used in machine learning models.

Importance of One Hot Encoding

1. Eliminating Ordinality: Many categorical variables have no inherent order (e.g., "Male" and "Female"). If we were to assign numerical values (e.g., Male = 0, Female = 1) the model might mistakenly interpret this as a ranking and lead to biased predictions. One Hot Encoding eliminates this risk by treating each category independently.
2. Improving Model Performance: By providing a more detailed representation of categorical variables. One Hot Encoding can help to improve the performance of machine learning models. It allows models to capture complex relationships within the data that might be missed if categorical variables were treated as single entities.
3. Compatibility with Algorithms: Many machine learning algorithms particularly based on linear regression and gradient descent which require numerical input. It ensures that categorical variables are converted into a suitable format.

How One-Hot Encoding Works: An Example

To grasp the concept better let's explore a simple example. Imagine we have a dataset with fruits, their categorical values and corresponding prices. Using one-hot encoding we can transform these categorical values into numerical form. For example:

- Wherever the fruit is "Apple," the Apple column will have a value of 1 while the other fruit columns (like Mango or Orange) will contain 0.
- This pattern ensures that each categorical value gets its own column represented with binary values (1 or 0) making it usable for machine learning models.

Fruit	Categorical value of fruit	Price

apple	1	5
mango	2	10
apple	1	15
orange	3	20

The output after applying one-hot encoding on the data is given as follows,

Fruit_apple	**Fruit_mango**	**Fruit_orange**	**price**
1	0	0	5
0	1	0	10
1	0	0	15
0	0	1	20

Advantages of Using One Hot Encoding

1. It allows the use of categorical variables in models that require numerical input.
2. It can improve model performance by providing more information to the model about the categorical variable.

3. It can help to avoid the problem of ordinality which can occur when a categorical variable has a natural ordering (e.g. "small", "medium", "large").

Disadvantages of Using One Hot Encoding

1. It can lead to increased dimensionality as a separate column is created for each category in the variable. This can make the model more complex and slow to train.
2. It can lead to sparse data as most observations will have a value of 0 in most of the one-hot encoded columns.
3. It can lead to overfitting especially if there are many categories in the variable and the sample size is relatively small.

Best Practices for One Hot Encoding

To make the most of One Hot Encoding and we must consider the following best practices:

1. Limit the Number of Categories: If you have high cardinality categorical variables consider limiting the number of categories through grouping or feature engineering.
2. Use Feature Selection: Implement feature selection techniques to identify and retain only the most relevant features after One Hot Encoding. This can help reduce dimensionality and improve model performance.
3. Monitor Model Performance: Regularly evaluate your model's performance after applying One Hot Encoding. If you notice signs of overfitting or other issues consider alternative encoding methods.
4. Understand Your Data: Before applying One Hot Encoding take the time to understand the nature of your categorical variables. Determine whether they have a natural order and whether One Hot Encoding is appropriate.

Alternatives to One Hot Encoding

While One Hot Encoding is a popular choice for handling categorical data there are several alternatives that may be more suitable depending on the context:

1. Label Encoding: In cases where categorical variables have a natural order (e.g., "Low," "Medium," "High") label encoding can be a better option. This method assigns a unique integer to each category without introducing the same risks of hierarchy misinterpretation as with nominal data.

2. Binary Encoding: This technique combines the benefits of One Hot Encoding and label encoding. It converts categories into binary numbers and then creates binary columns. This method can reduce dimensionality while preserving information.
3. Target Encoding: In target encoding, we replace each category with the mean of the target variable for that category. This method can be particularly useful for categorical variables with a high number of unique values but it also carries a risk of leakage if not handled properly.

1.2. 7 segment display decoder

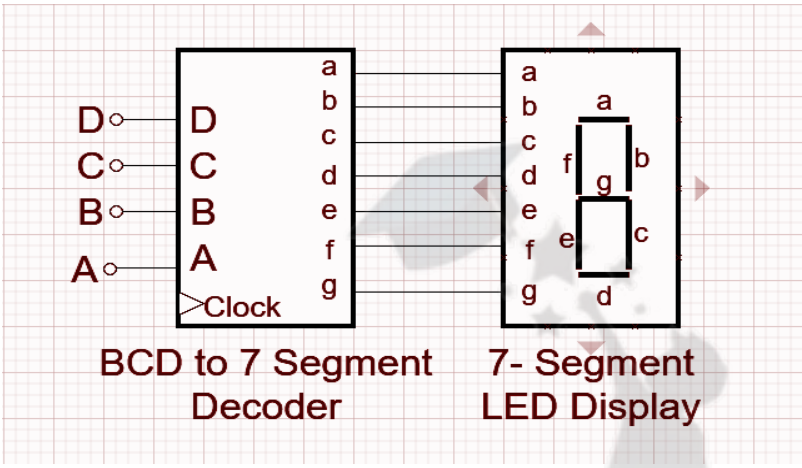
Binary Coded Decimal (BCD)

BCD is the encoding scheme where each of the decimal numbers(0-9) is represented by its equivalent binary pattern(which is generally of 4-bits).

Seven segment

Seven Segment display is an electronic device which consists of seven Light Emitting Diodes (LEDs) arranged in a some definite pattern (common cathode or common anode type), which is used to display Hexadecimal numerals(in this case decimal numbers, as input is BCD i.e., 0-9). Two types of seven segment LED display:

1. Common Cathode Type: In this type of display all cathodes of the seven LEDs are connected together to the ground or -Vcc(hence, common cathode) and LED displays digits when some 'HIGH' signal is supplied to the individual anodes.
2. Common Anode Type: In this type of display all the anodes of the seven LEDs are connected to battery or +Vcc and LED displays digits when some 'LOW' signal is supplied to the individual cathodes.



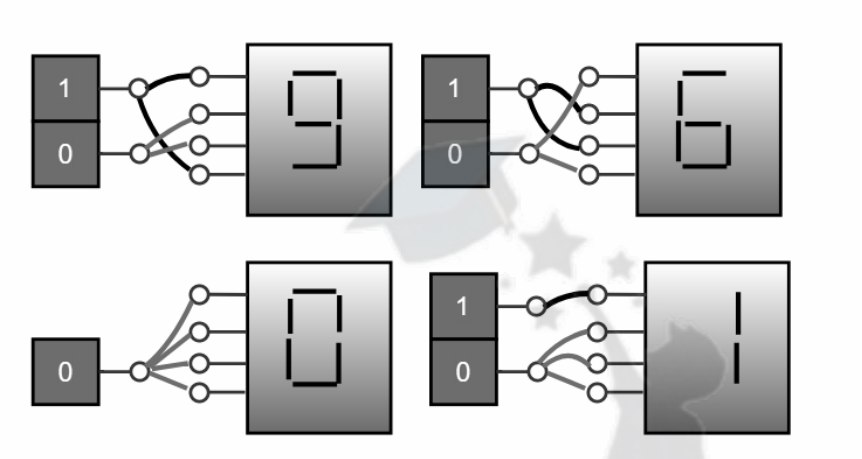
Truth Table

For common cathode type BCD to seven segment decoder:

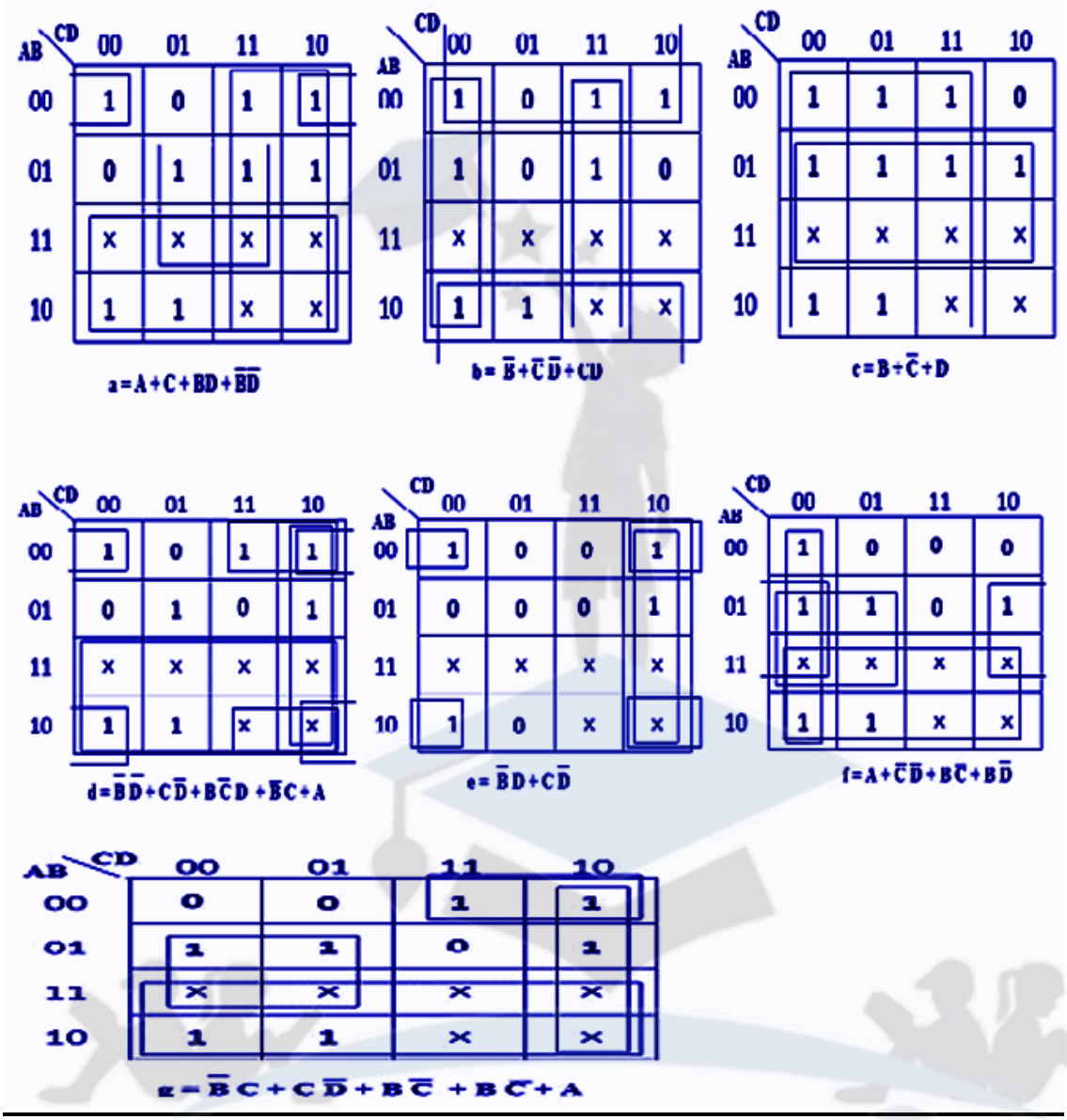
A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

Note -

- For Common Anode type seven segment LED display, we only have to interchange all '0s' and '1s' in the output side i.e., (for a, b, c, d, e, f, and g replace all '1' by '0' and vice versa) and solve using K-map.
- Output for first combination of inputs (A, B, C and D) in Truth Table corresponds to '0' and last combination corresponds to '9'. Similarly rest corresponds from 2 to 8 from top to bottom.
- BCD numbers only range from 0 to 9,thus rest inputs from 10-F are invalid inputs.



For a combination where all the inputs (A, B, C and D) are zero (see Truth Table), our output lines are a = 1, b = 1, c = 1, d = 1, e = 1, f = 1 and g = 0. So the 7 segment display shows 'zero' as output. Similarly, for a combination where one of the input is one (D = 1) and the rest are zero, our output lines are a = 0, b = 1, c = 1, d = 0, e = 0, f = 0 and g = 0. So only LEDs 'b' and 'c' (see diagram above) will glow and the 7 segment display shows 'one' as output.



Advantages

- Diminished Part Count: Limits the quantity of parts expected to control a 7-portion show.
- Further developed Dependability: Gives a strong and solid change from BCD to show portions.
- Simplicity of Mix: Smoothes out the method involved with incorporating mathematical presentations into advanced frameworks.

Disadvantages

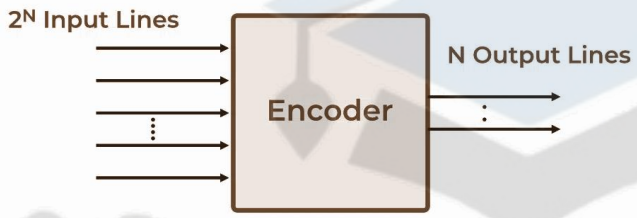
1.Cost for Straightforward Applications: Utilizing a dedicated decoder IC may be more costly than less complex answers for essential necessities.

2.Speed Limits: Execution is limited by the decoder's handling speed, which is inadmissible for fast applications.

3.Size Imperatives: The actual size of the decoder IC may not fit in minimized plans.

2. Encoders

An Encoder is a combinational circuit that performs the reverse operation of a Decoder. It has a maximum of 2^n input lines and 'n' output lines, hence it encodes the information from 2^n inputs into an n-bit code. It will produce a binary code equivalent to the input, which is active High. Therefore, the encoder encodes 2^n input lines with 'n' bits. Encoders are widely used in digital systems to convert parallel inputs into serial codes.

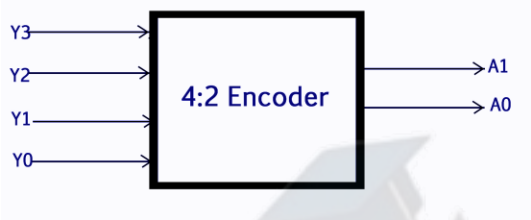


Types of Encoders

- 4 to 2 Encoder
- Octal to Binary Encoder (8 to 3 Encoder)
- Decimal to BCD Encoder
- Priority Encoder

4 to 2 Encoder

The 4 to 2 Encoder consists of **four inputs Y3, Y2, Y1 & Y0, and two outputs A1 & A0**. At any time, only one of these 4 inputs can be '1' in order to get the respective binary code at the output. The figure below shows the logic symbol of the 4 to 2 encoder.



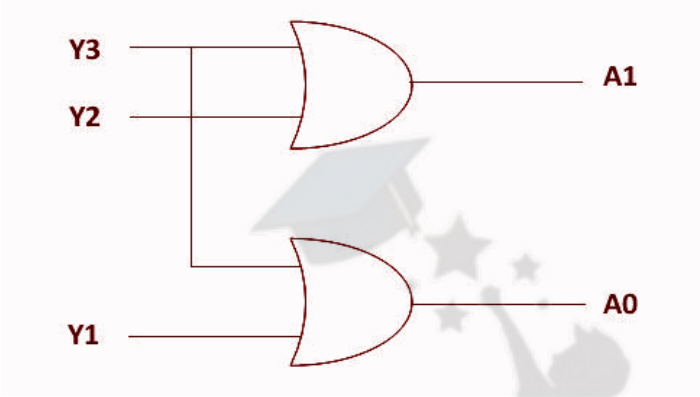
The Truth table of 4 to 2 encoders is as follows:.

INPUTS				**OUTPUTS**	
Y3	**Y2**	**Y1**	**Y0**	**A1**	**A0**
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

Logical expression for A1 and A0:

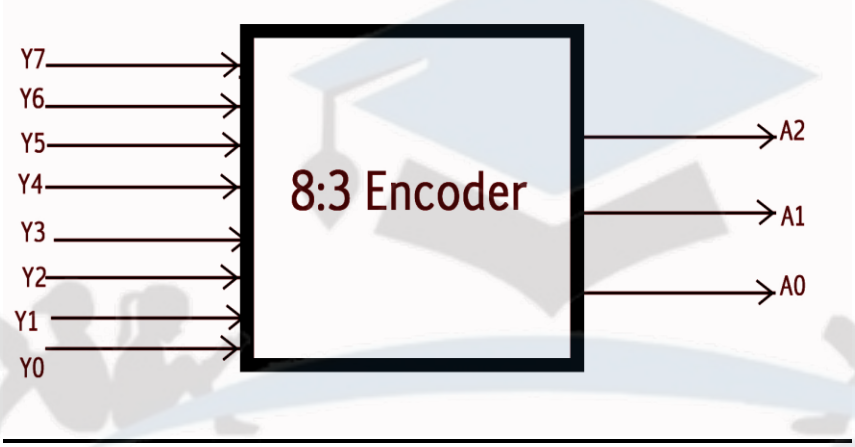
A1 = Y3 + Y2
A0 = Y3 + Y1

The above two Boolean functions A1 and A0 can be implemented using two input OR gates :



Octal to Binary Encoder (8 to 3 Encoder)

The 8 to 3 Encoder or octal to Binary encoder consists of 8 inputs: Y7 to Y0 and 3 outputs: A2, A1 & A0. Each input line corresponds to each octal digit value and three outputs generate corresponding binary code. The figure below shows the logic symbol of octal to the binary encoder.



The truth table for the 8 to 3 encoder is as follows.

INPUTS	OUTPUTS

Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	A2	A1	A0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

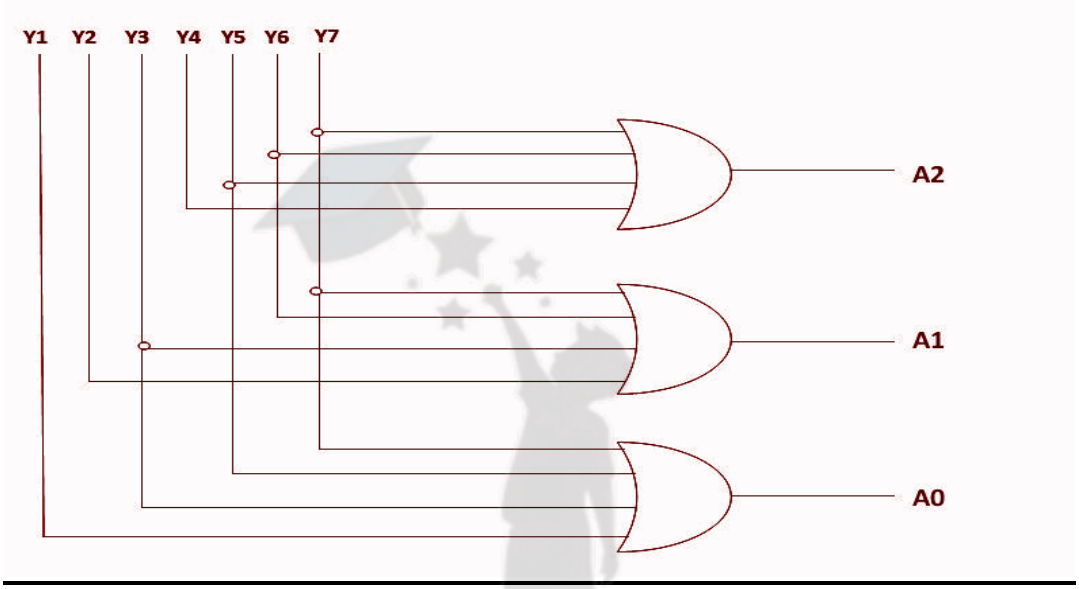
Logical expression for A2, A1, and A0.

$$A2 = Y7 + Y6 + Y5 + Y4$$

$$A1 = Y7 + Y6 + Y3 + Y2$$

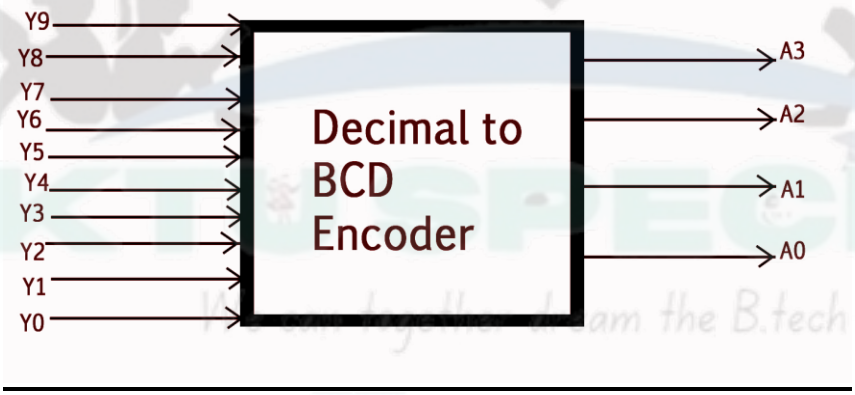
$$A0 = Y7 + Y5 + Y3 + Y1$$

The above two Boolean functions A2, A1, and A0 can be implemented using four input OR gates.



Decimal to BCD Encoder

The decimal-to-binary encoder usually consists of 10 input lines and 4 output lines. Each input line corresponds to each decimal digit and 4 outputs correspond to the BCD code. This encoder accepts the decoded decimal data as an input and encodes it to the BCD output which is available on the output lines. The figure below shows the logic symbol of the decimal to BCD encoder :



The truth table for decimal to BCD encoder is as follows:

INPUTS										OUTPUTS			
Y9	**Y8**	**Y7**	**Y6**	**Y5**	**Y4**	**Y3**	**Y2**	**Y1**	**Y0**	**A3**	**A2**	**A1**	**A0**
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	1	0
0	0	1	0	0	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	1

Logical expression for A3, A2, A1, and A0.

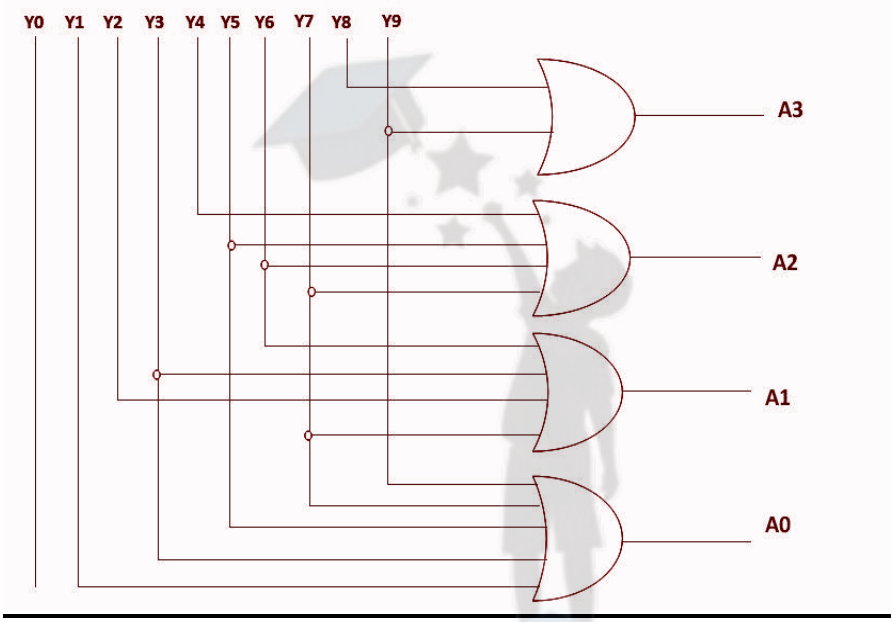
A3 = Y9 + Y8

A2 = Y7 + Y6 + Y5 +Y4

A1 = Y7 + Y6 + Y3 +Y2

A0 = Y9 + Y7 +Y5 +Y3 + Y1

The above two Boolean functions can be implemented using OR gates.



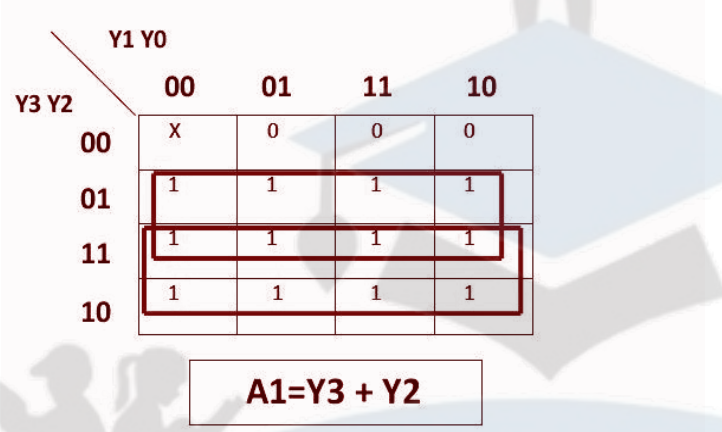
Priority Encoder

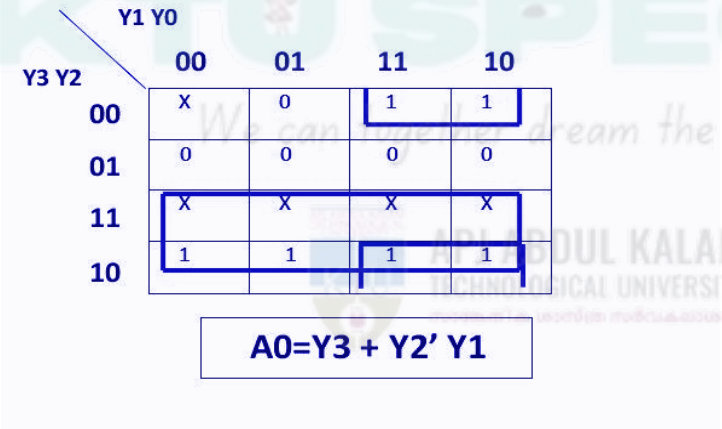
A 4 to 2 priority encoder has **4 inputs**: Y3, Y2, Y1 & Y0, and **2 outputs**: A1 & A0. Here, the input, Y3 has the **highest priority**, whereas the input, Y0 has the **lowest priority**. In this case, even if more than one input is '1' at the same time, the output will be the (binary) code corresponding to the input, which is having **higher priority**. The truth table for the priority encoder is as follows.

INPUTS				**OUTPUTS**		
Y3	**Y2**	**Y1**	**Y0**	**A1**	**A0**	**V**
0	0	0	0	X	X	0

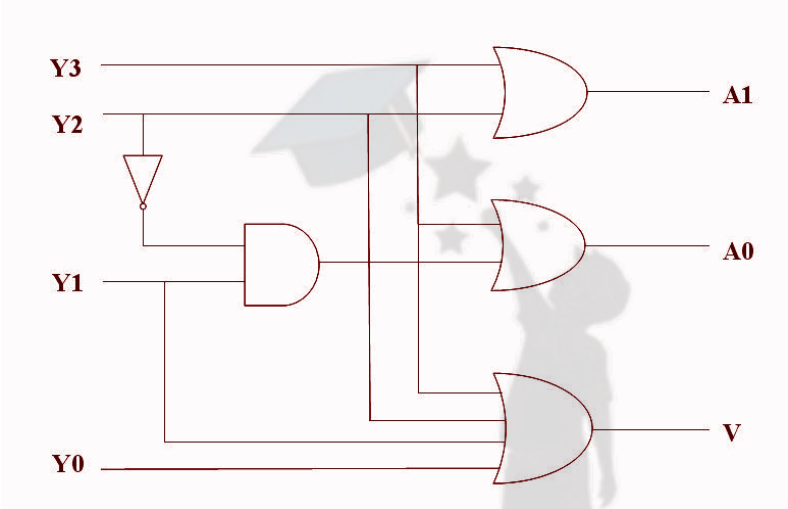
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

The logical expression for A1 , A0is shown below.





The above two Boolean functions can be implemented as.



There are some errors that usually happen in Encoders are mentioned below.

- There is an ambiguity, when all outputs of the encoder are equal to zero.
- If more than one input is active High, then the encoder produces an output, which may not be the correct code.

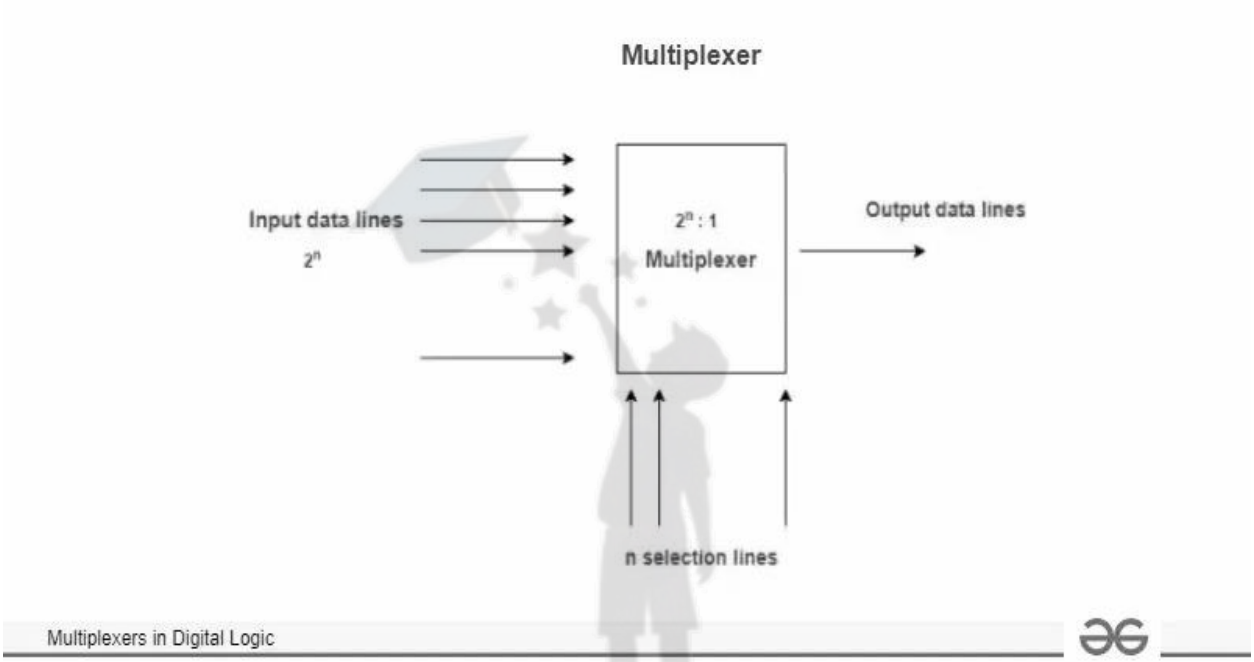
So, to overcome these difficulties, we should assign priorities to each input of the encoder. Then, the output of the encoder will be the code corresponding to the active high inputs, which have higher priority.

Application of Encoders

- Encoders are very common electronic circuits used in all digital systems.
- Encoders are used to translate the decimal values to the binary in order to perform binary functions such as addition, subtraction, multiplication, etc.
- Other applications especially for Priority Encoders may include detecting interrupts in microprocessor applications.

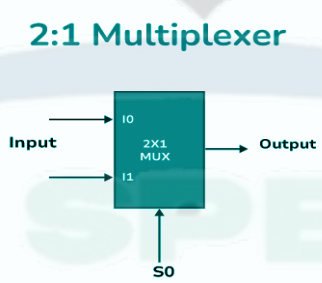
3. Multiplexers

A multiplexer is a combinational circuit that has many data inputs and a single output, depending on control or select inputs. For N input lines, $\log_2(N)$ selection lines are required, or equivalently, for 2^n input lines, n selection lines are needed. Multiplexers are also known as "N-to-1 selectors," parallel-to-serial converters, many-to-one circuits, and universal logic circuits.



2x1 Mux

The 2x1 is a fundamental circuit which is also known 2-to-1 multiplexer that are used to choose one signal from two inputs and transmits it to the output. The 2x1 mux has two input lines, one output line, and a single selection line. In this Block Diagram where I0 and I1 are the input lines, Y is the output line and S0 is a single select line.



Truth Table

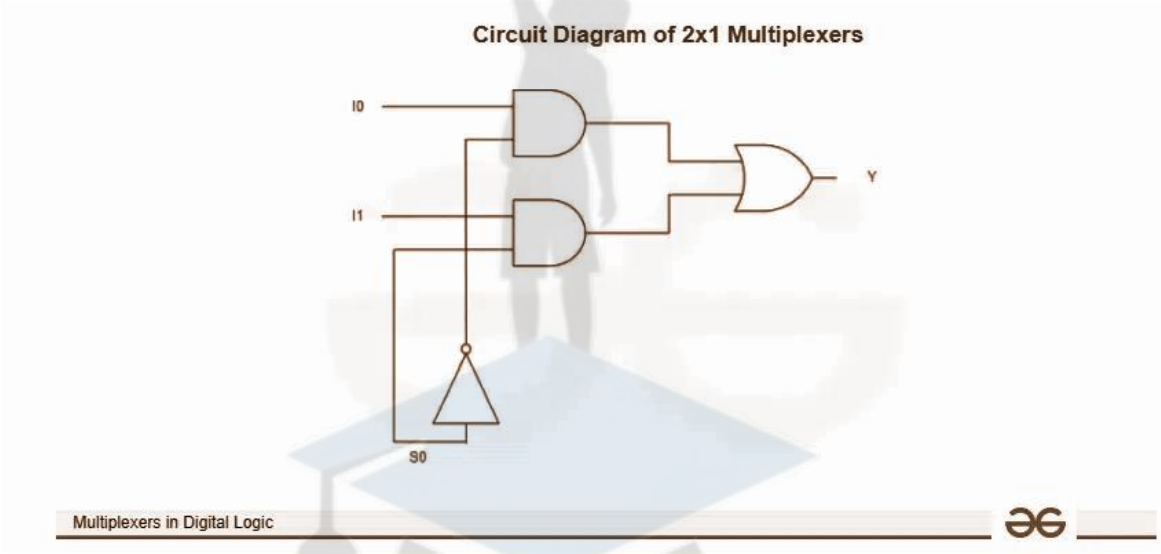
S_0	I_0	I_1	Y
0	0	X	0
0	1	X	1
1	X	0	0
1	X	1	1

Block Diagram of 2:1 Multiplexer with Truth Table

The output of the 2x1 Mux will depend on the selection line S0,

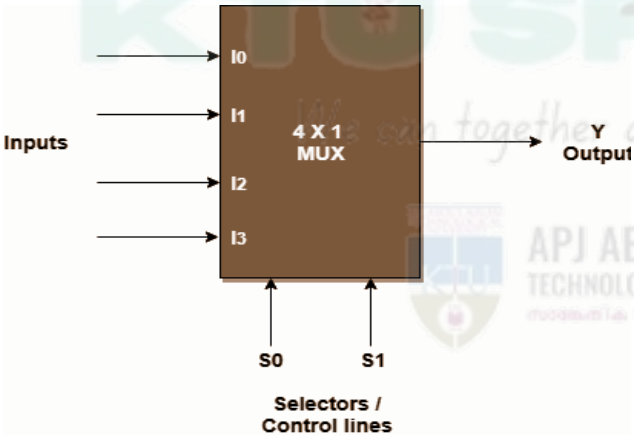
- When S is 0(low), the I0 is selected
- when S0 is 1(High), I1 is selected

$$Y=S_0'.I_0+S_0.I_1$$



4x1 Mux

The 4x1 Multiplexer which is also known as the 4-to-1 multiplexer. It is a multiplexer that has 4 inputs and a single output.
In the Given Block Diagram I0, I1, I2, and I3 are the 4 inputs and Y is the Single output which is based on Select lines S0 and S1.



The output of the multiplexer is determined by the binary value of the selection lines

When S1S0=00, the input I0 is selected.

When S1S0=01, the input I1 is selected.

When S1S0=10, the input I2 is selected.

When S1S0=11, the input I3 is selected.

Truth Table

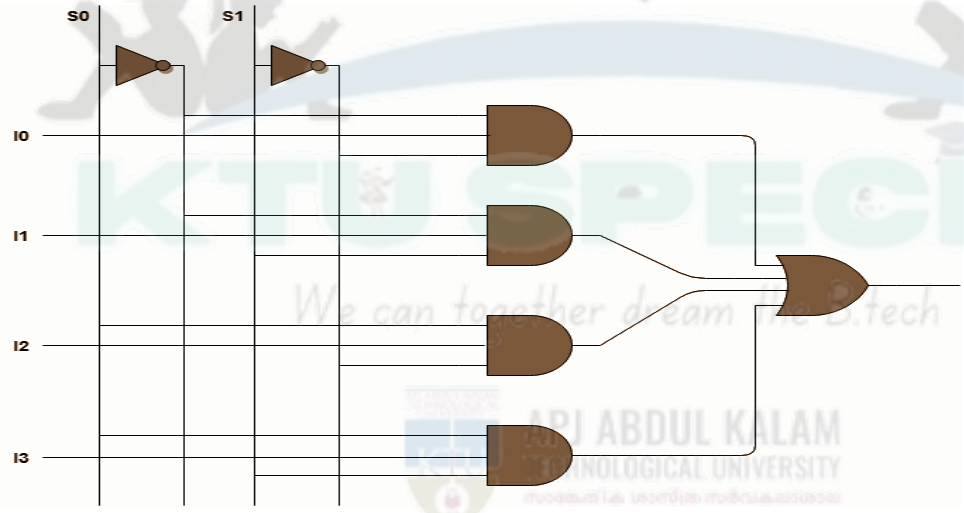
S0	S1	Y
0	0	I0
0	1	I1
1	0	I2
1	1	I3

So, final equation,

$Y = S0'.S1'.I0 + S0'.S1.I1 + S0.S1'.I2 + S0.S1.I3$

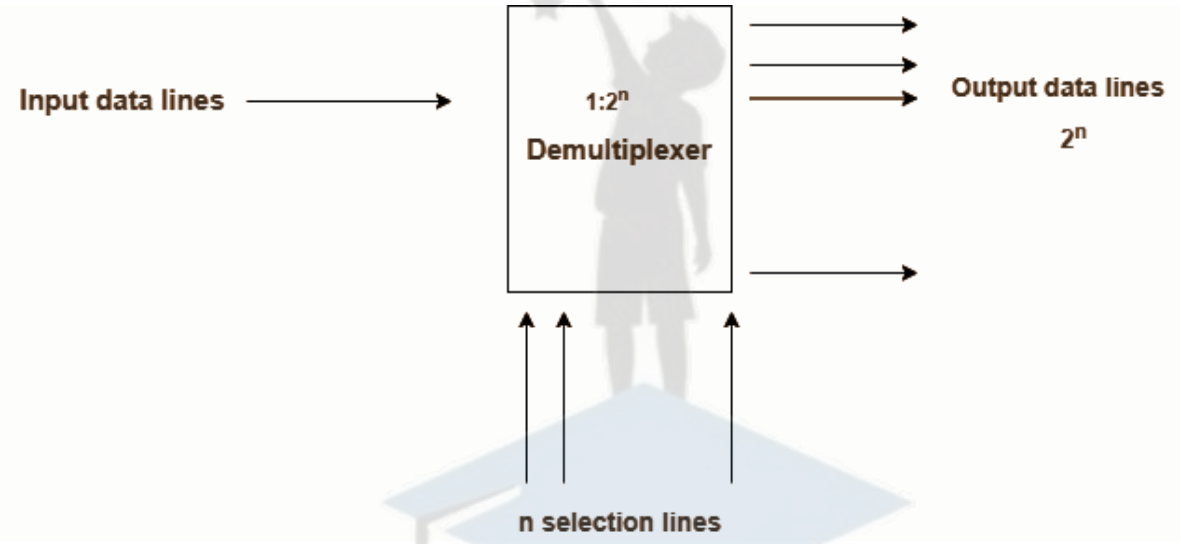
Circuit Diagram of 4x1 Multiplexers

Using truth table the circuit diagram can be given as

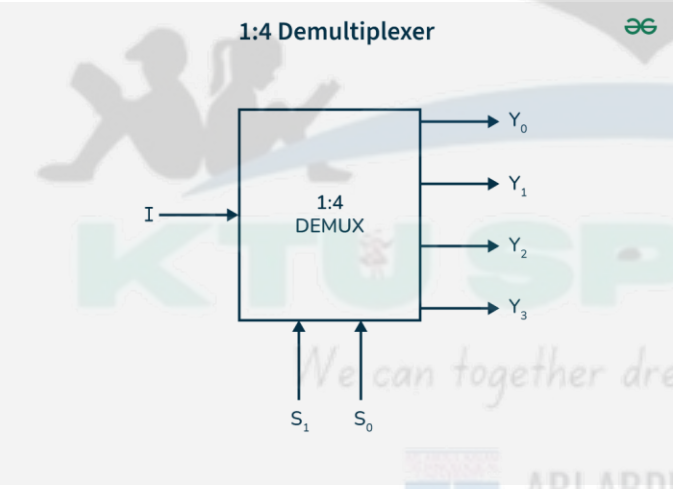


4. Demultiplexers

The DEMUX is a digital information processor. It takes input from one source and also converts the data to transmit towards various sources. The demultiplexer has one data input line. The demultiplexer has several control lines (also known as select lines). These lines determine to which output the input data should be sent. The number of control lines determines the number of output lines.



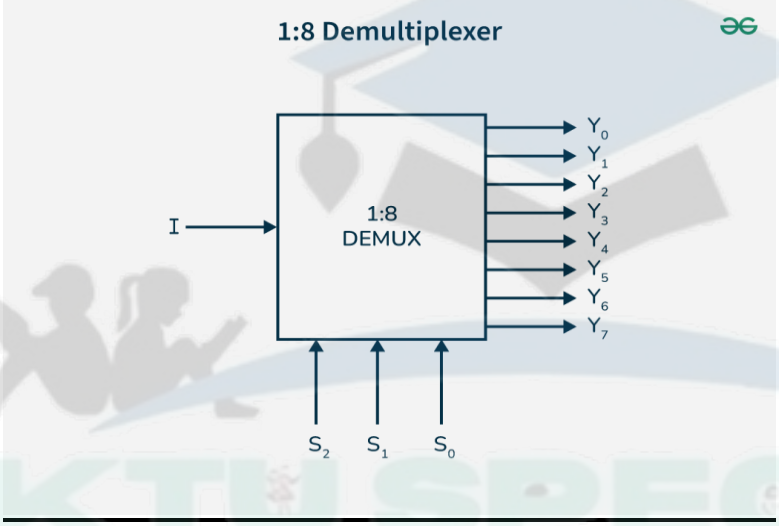
1X4 DEMUX



Truth table of the 1x4 DEMUX as mentioned below.

Selection Inputs		Outputs			
S_1	**S_0**	**Y_3**	**Y_2**	**Y_1**	**Y_0**
0	0	0	0	0	**I**
0	1	0	0	**I**	0
1	0	0	**I**	0	0
1	1	**I**	0	0	0

1x8 DEMUX

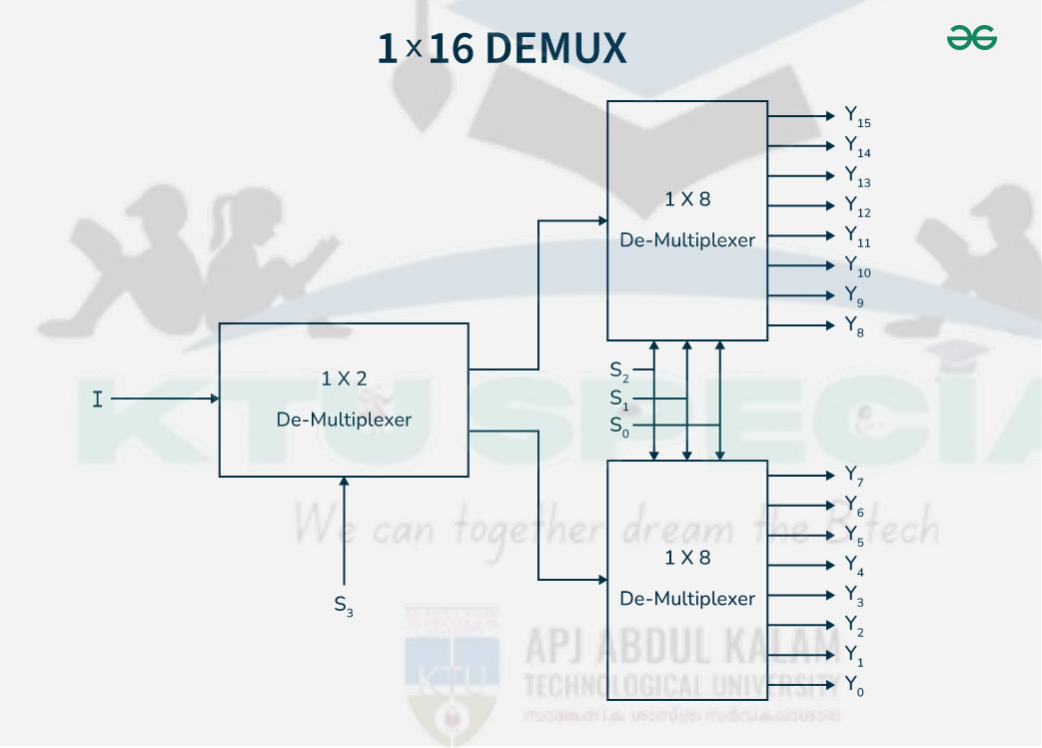


1x8 DEMUX truth table as mentioned below.

Selection Inputs			Outputs							
S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	I

0	0	1	0	0	0	0	0	0	I	0
0	1	0	0	0	0	0	0	I	0	0
0	1	1	0	0	0	0	I	0	0	0
1	0	0	0	0	0	I	0	0	0	0
1	0	1	0	0	I	0	0	0	0	0
1	1	0	0	I	0	0	0	0	0	0
1	1	1	I	0	0	0	0	0	0	0

1x16 DEMUX



Advantages of the DEMUX

- The DEMUX increases the efficiency of the particular communication system as it takes data from a specific input source and distributes it to different sources.
- The DEMUX helps to separate the different signals from the mixed data sources. Then it distributes these data to different sources.
- DEMUX can decode the signal outputs of the multiplexer, as the system works in a reverse way of the MUX.

Disadvantages of the DEMUX

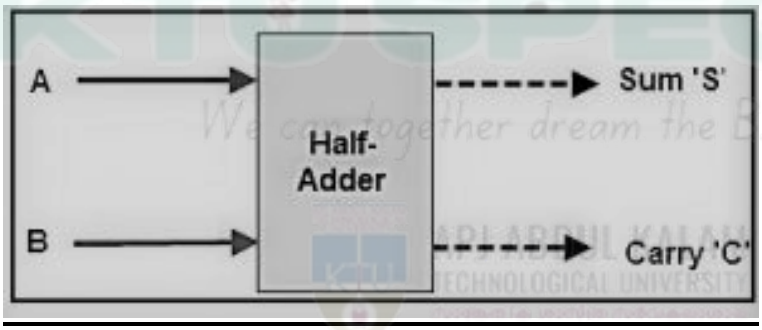
- The DEMUX may cause a wastage of bandwidth as it distributes the refined data in different channels. These channels can overlap with each other which leads to the loss of signal.
- The DEMUX may cause problems in the synchronization of signals. The data channels can overlap with each other which leads to the delay in the whole process.

Digital Building Blocks - Arithmetic Circuits

Arithmetic circuits are fundamental blocks in digital systems and are used for arithmetic operations such as addition, subtraction, multiplication and division. Arithmetic circuits can perform seven different arithmetic operations using a single composite circuit.

1. Half adder

The circuit adds two binary digits where the input bits are termed as augend and addend and the result will be two outputs one is the sum and the other is carry. To perform the sum operation, XOR is applied to both the inputs, and AND gate is applied to both inputs to produce carry.

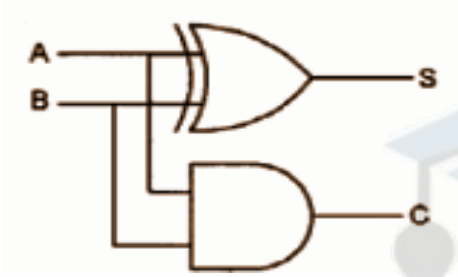


The 2-bit half adder truth table is as below:

0+0 = 00
0+1 = 01
1+0 = 01
1+1 = 10

The output ‘1’of ‘10’ is carry-out. ‘SUM’ is the normal output and ‘CARRY’ is the carry-out.

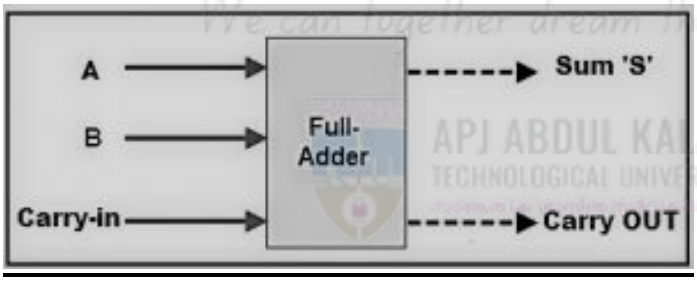
Inputs		Outputs	
A	B	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Sum=A.B+A.B’
Sum= A XOR B

Carry = A AND B= A.B

2. Full adder



The full-adder has three inputs and two outputs, whereas the half adder has only two inputs and two outputs. The first two inputs are A and B and the third input is an input carry as C-IN.

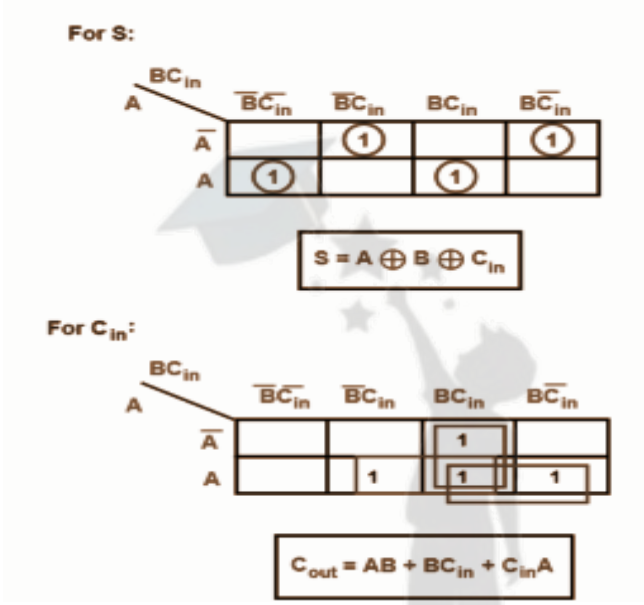
A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

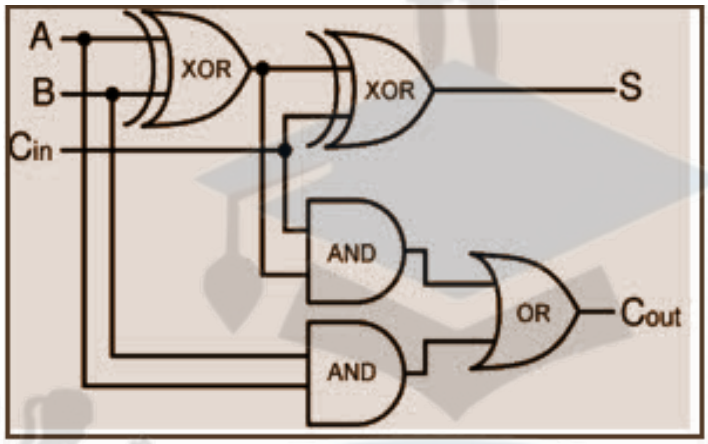
The output carry is designated as C-OUT and the normal output is represented as S which is 'SUM'. With the above full adder truth-table, the implementation of a full adder circuit can be understood easily. The SUM 'S' is produced in two steps:

- By XORing the provided inputs 'A' and 'B'
- The result of A XOR B is then XORed with the C-IN

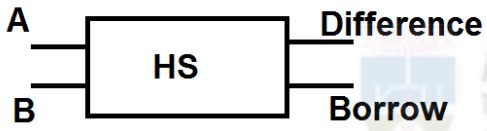
Sum= A'B'Cin + A' B CCin' + A B'Cin'+ AB Cin
= Cin (A'B'+ AB) + Cin' (A'B +A B')
= Cin EX-OR (A EX-OR B)

C_{out}= A'B Cin + AB'Cin + AB Cin' + ABCin
= AB + BCin + ACin





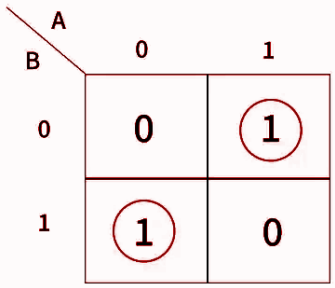
3. Half subtractor



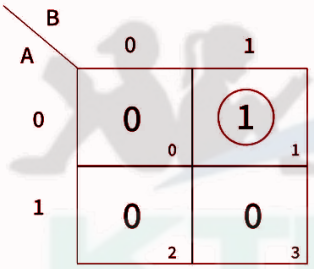
Half Subtractor

Half subtractor is a digital logic circuit that performs the binary subtraction of two single-bit binary numbers. It has two inputs, A and B, and two outputs, Difference and Borrow. The Difference output represents the result of subtracting B from A, while the Borrow output indicates whether a borrow is needed when A is smaller than B.

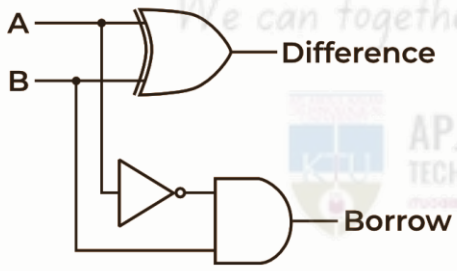
A	B	Diff	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0



Difference = A'B+AB'



Borrow = A'B



Advantages of Half Subtractor

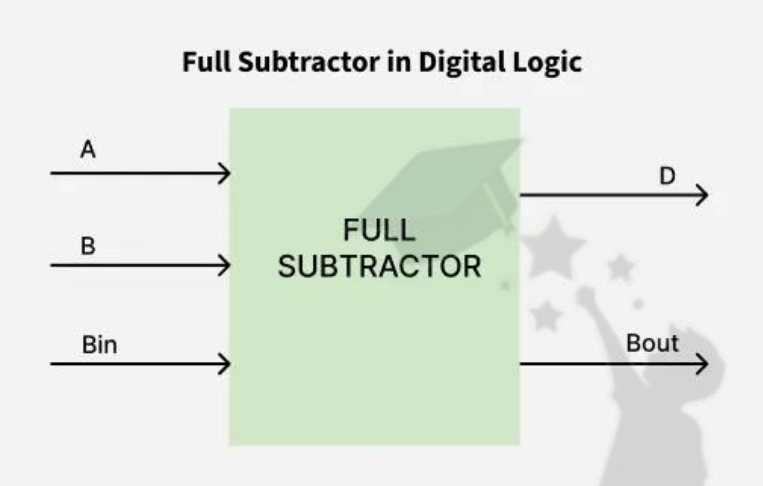
- **Simplicity:** The half subtractor circuits are simple and easy to design, implement, and debug compared to other binary arithmetic circuits.
- **Building blocks:** The half subtractor is basic building block that can be used to construct more complex arithmetic circuits, such as full subtractors, multiple-bit subtractors.
- **Low cost:** The half subtractor circuits use only a few gates, which reduces the cost and power consumption compared to more complex circuits.
- **Easy integration:** The half subtractor can be easily integrated with other digital circuits and systems.

Disadvantages of Half Subtractor

- **Limited functionality:** The half subtractor can only perform binary subtraction of two single-bit numbers, respectively, and not suitable for more complex arithmetic operations.
- **Inefficient for multi-bit numbers:** For multi-bit numbers, multiple half subtractors need to be cascaded, which increases the complexity and decreases the efficiency of the circuit.
- **High propagation delay:** When cascaded for multi-bit operations, the cumulative propagation delay of half subtractors becomes higher compared to dedicated multi-bit subtractors (e.g., using look-ahead borrow).

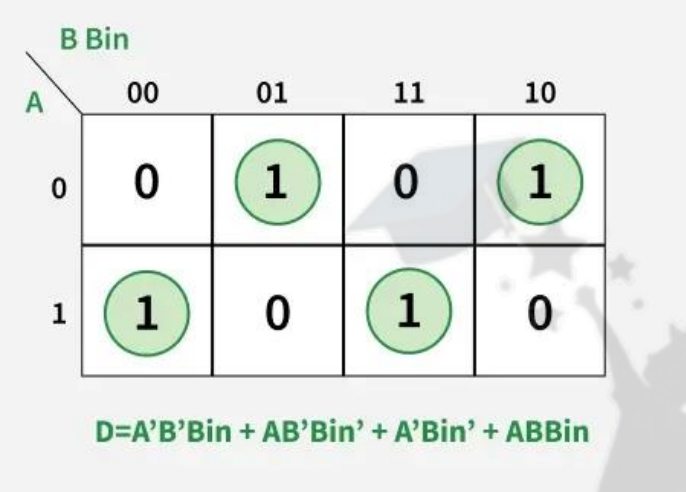
4. Full subtractor

The full subtractor is essential because a half-subtractor can only subtract the least significant bit (LSB) of binary numbers. However, if a borrow is generated during the subtraction of the LSBs, it will affect the subtraction in the next stages. A full subtractor handles this situation by considering the borrow from the previous stage, ensuring accurate subtraction even when a borrow is present.The full subtractor is used to subtract binary numbers with borrow handling, making it suitable for multi-bit subtraction in digital circuits like **Arithmetic Logic Units (ALUs)**.



Input			Output	
A	**B**	**B_{in}**	**D**	**B_{out}**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

From above table we can draw the K-Map as shown for "difference" and "borrow".



D = A'B'Bin + A'BBin' + AB'Bin' + ABBin

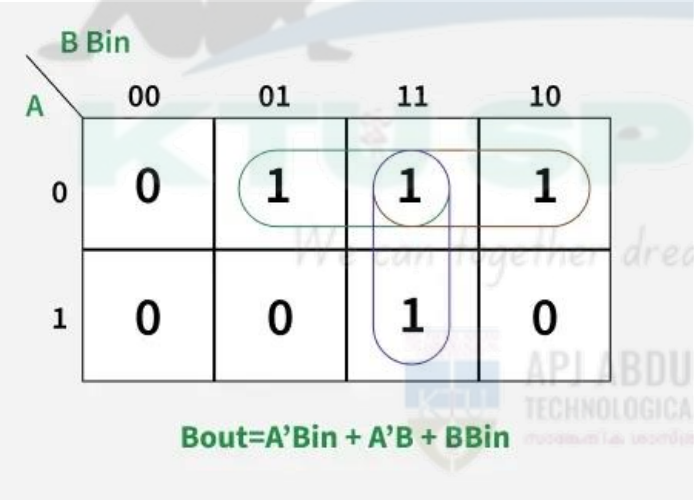
D = Bin(A'B' + AB) + Bin'(AB' + A'B)

A'B' + AB = A XNOR B
AB' + A'B = A XOR B

D = Bin(A XNOR B) + Bin' (A XOR B)

D = Bin ⊕(A ⊕B)

D = (A ⊕B) ⊕Bin



Bout = A'B'Bin + A'BBin' + A'BBin + ABBin

Bout = A'Bin(B + B') + A'B(Bin + Bin') + BBin(A + A')

Bout = A'Bin + A'B + BBin

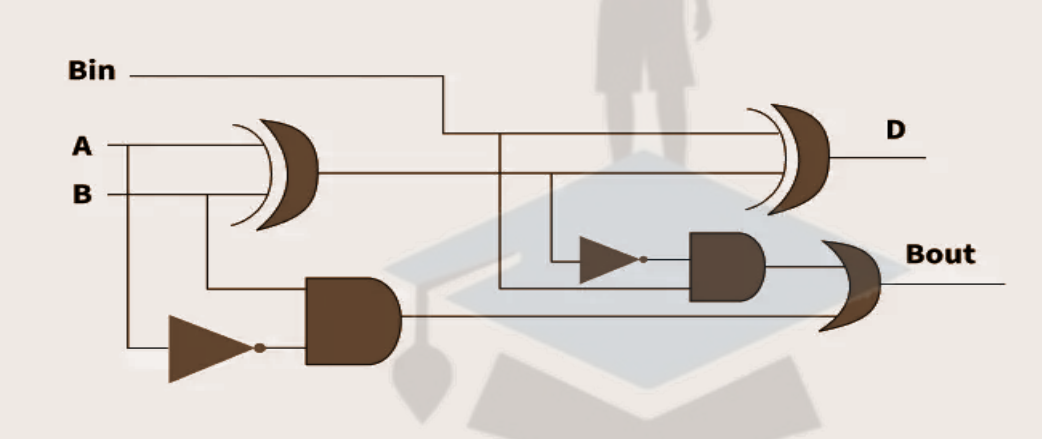
Bout = A'B'Bin + A'BBin' + A'BBin + ABBin

Bout = Bin(AB + A'B') + A'B(Bin + Bin')

AB + A'B' = A XNOR B

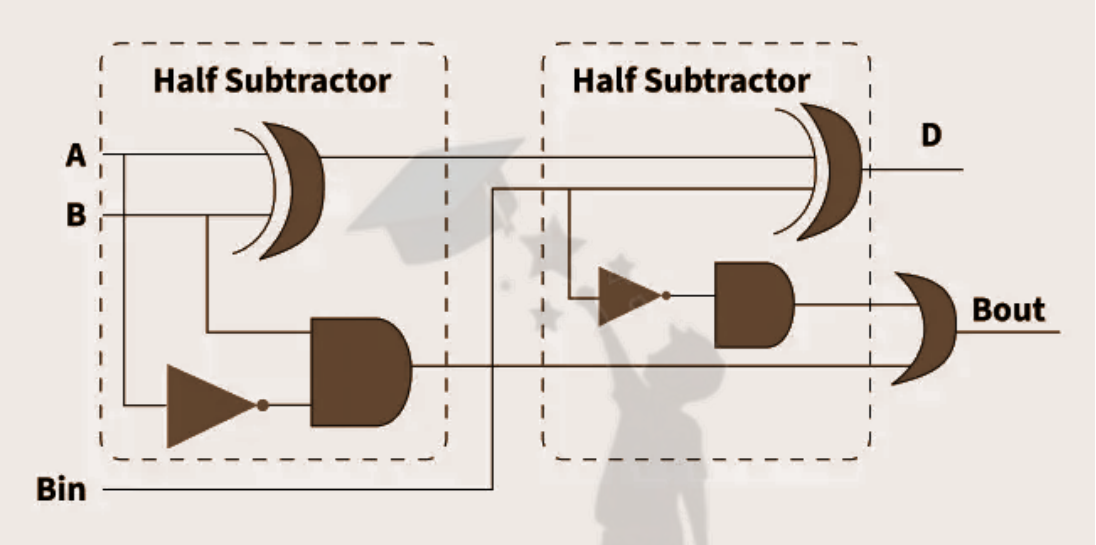
Bout = Bin(A XNOR B) + A'B

Bout = Bin (A XOR B)' + A'B



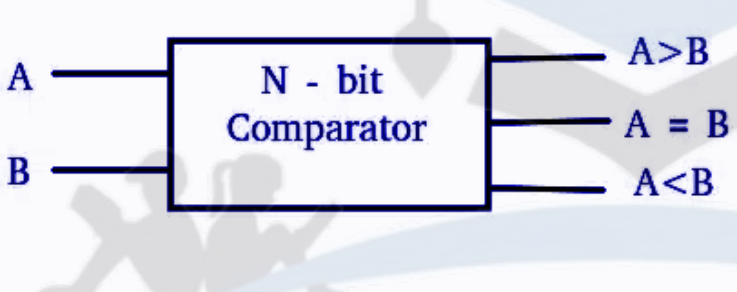
Implementation of Full Subtractor using Half Subtractors

2 Half Subtractors and an OR gate is required to implement a Full Subtractor.



Comparators

Magnitude comparator is a type o\ It Basically compares two binary numbers and determines their relative magnitude. It gives output whether one number is greater than the other, or less than or equal. These comparators are used in digital systems, such as for sorting networks, and decision-making circuits to handle numerical comparisons perfectly without any error.



If the bit in the first number is greater than the corresponding bit in the second number, **the A>B output is set to 1**,and the circuit immediately determines that the first number is greater than the second. Similarly, if the bit in the second number is greater than the corresponding bit in the first number, **the A<B output is set to 1**, and the circuit immediately determines that the first number is less than the second.If all the bits are equal, the circuit generates an **A=B output**, indicating that the two numbers are equal.

1-Bit Magnitude Comparator

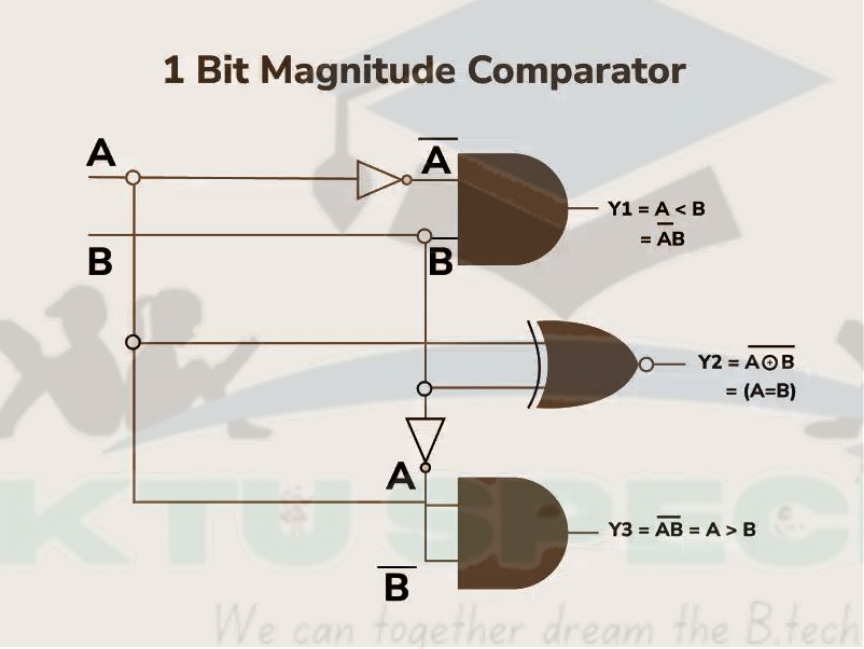
A	B	A<B	A=B	A>B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

From the above truth table logical expressions for each output can be expressed as follows.

A>B: AB'

A<B: A'B

A=B: A'B' + AB

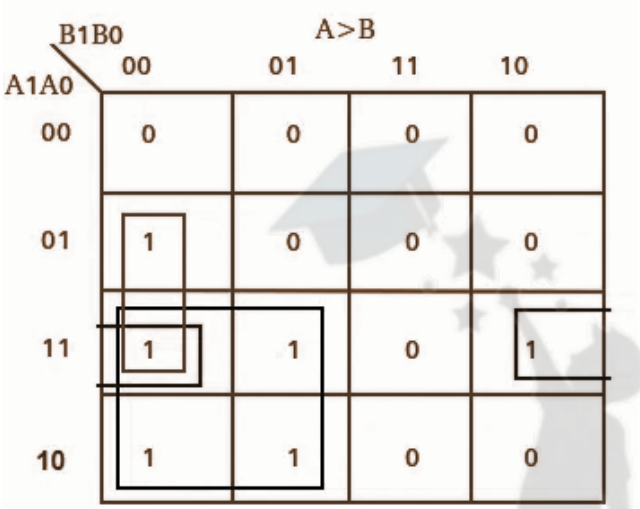


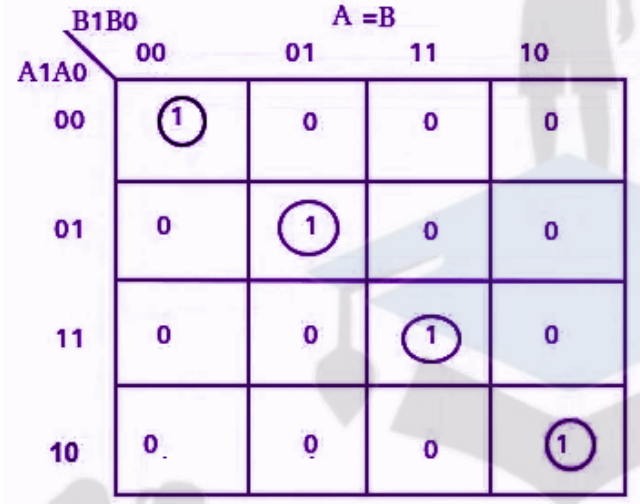
2-Bit Magnitude Comparator

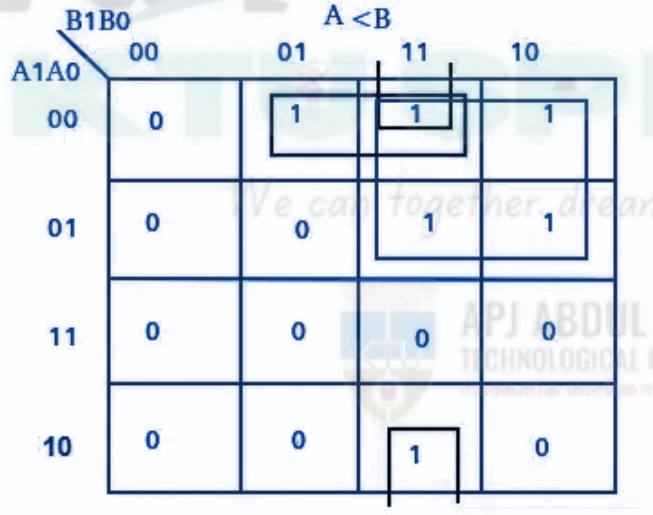
A comparator used to compare two binary numbers each of two bits is called a 2-bit Magnitude comparator. It consists of four inputs and three outputs to generate less than, equal to, and greater than between two binary numbers.

INPUT				OUTPUT		
A1	**A0**	**B1**	**B0**	**A < B**	**A = B**	**A > B**
0	**0**	**0**	**0**	**0**	**1**	**0**
0	**0**	**0**	**1**	**1**	**0**	**0**
0	**0**	**1**	**0**	**1**	**0**	**0**
0	**0**	**1**	**1**	**1**	**0**	**0**
0	**1**	**0**	**0**	**0**	**0**	**1**
0	**1**	**0**	**1**	**0**	**1**	**0**
0	**1**	**1**	**0**	**1**	**0**	**0**
0	**1**	**1**	**1**	**1**	**0**	**0**
1	**0**	**0**	**0**	**0**	**0**	**1**
1	**0**	**0**	**1**	**0**	**0**	**1**

1	**0**	**1**	**0**	**0**	**1**	**0**
1	**0**	**1**	**1**	**1**	**0**	**0**
1	**1**	**0**	**0**	**0**	**0**	**1**
1	**1**	**0**	**1**	**0**	**0**	**1**
1	**1**	**1**	**0**	**0**	**0**	**1**
1	**1**	**1**	**1**	**0**	**1**	**0**







From the above K-maps logical expressions for each output can be expressed as follows.

A>B: A1B1' + A0B1'B0' + A1A0B0'

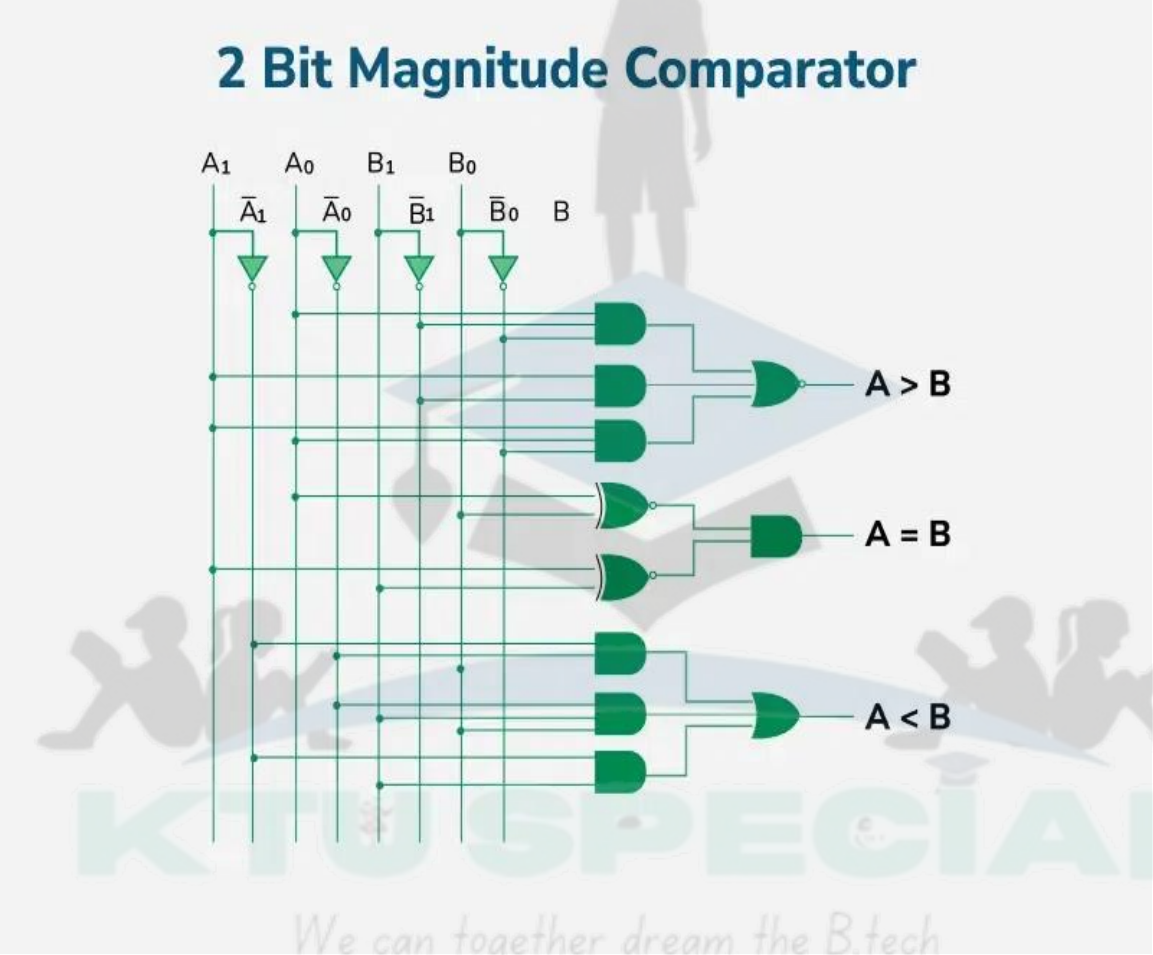
A=B: A1'A0'B1'B0' + A1'A0B1'B0 + A1A0B1B0 + A1A0'B1B0'

A1'B1' (A0'B0' + A0B0) + A1B1 (A0B0 + A0'B0')

(A0B0 + A0'B0') (A1B1 + A1'B1')

(A0 Ex-Nor B0) (A1 Ex-Nor B1)

A<B: A1'B1 + A0'B1B0 + A1'A0'B0



Applications of Comparators

- Comparators are used in central processing units (CPU s) and microcontrollers (MCUs).

- These are used in control applications in which the binary numbers representing physical variables such as temperature, position, etc. are compared with a reference value.
- Comparators are also used as process controllers and for Servo motor control.
- Used in password verification and biometric applications.

Advantage of Comparator

- Comparators are simple and efficient for comparison of binary values.
- Fast decision-making in Digital Circuits.
- This comparator can be easily integrated in to complex systems like processors and arithmetic units.
- Design of comparator are modular, which allow them to scale solutions for comparing multi-bit numbers

Disadvantages of Comparator

- Comparators have limit of bits for comparison.
- It requires more complex circuit for large bits.
- Power consumption increase with increase in the complexity of the circuit.

Structural design and hierarchy

In digital electronics, structural design involves defining how various components are interconnected to create a functional circuit or system. This process focuses on the physical arrangement and connections of logic gates, flip-flops, and other fundamental building blocks to achieve desired functionality

Key aspects of structural design in digital electronics:

- Component Interconnection:
  Structural design determines how different components like logic gates (AND, OR, NOT, etc.) are connected to perform specific logical operations.
- Gate-Level Modeling:
  At the structural level, designs are described in terms of interconnected logic gates and their outputs.
- Building Blocks:
  Structural design relies on combining basic building blocks (like flip-flops and logic gates) to create larger, more complex circuits.

- VHDL and Verilog:
  Hardware Description Languages (HDLs) like VHDL and Verilog are used to describe and model these structural designs.
- Simulation and Synthesis:
  After structural design, simulations are performed to verify the functionality, and then the design is synthesized into a physical implementation.

Structural design is one of the three major design styles in digital hardware description (the others being **behavioral** and **dataflow**). **Structural design** describes a digital system in terms of **interconnections between components** (gates, multiplexers, flip-flops, ALUs, etc.).

Characteristics:

- Specifies what components are used and how they're connected.
- Used in **HDL (Hardware Description Languages)** like Verilog or VHDL.
- Low-level and close to actual hardware implementation.
- Mirrors how physical circuits are built.

Example:In Verilog:

```
module AND_GATE(output Y, input A, B);
  assign Y = A & B;
endmodule

module TOP_MODULE(output Y, input A, B, C);
  wire temp;
  AND_GATE U1(temp, A, B);
  AND_GATE U2(Y, temp, C);
endmodule
```

Hierarchy in Digital Electronics

Hierarchy refers to organizing a system in **multiple levels**, from high-level blocks down to basic gates.

Hierarchical Design:

- Breaks a large system into **subsystems** or **modules**.
- Promotes **modularity**, **reusability**, and **easier testing/debugging**.
- Each module can be designed, simulated, and verified independently.

Common Hierarchical Levels:

1. **System Level** – e.g., a CPU
2. **Subsystem Level** – e.g., ALU, Control Unit, Register File
3. **Module Level** – e.g., Full Adder, Decoder, Multiplexer
4. **Gate Level** – e.g., AND, OR, NOT gates
5. **Transistor Level** – CMOS implementation (optional at ASIC level)

Benefits:

- **Scalability**: Design can grow in complexity without becoming unmanageable.
- **Abstraction**: Higher levels don't need to know internal details of lower levels.
- **Collaboration**: Teams can work on different modules independently.

Lower level module instantiation

Module instantiation is the process of including one module inside another. In structural design, lower-level modules (like adders, multiplexers, gates, etc.) are instantiated in higher-level modules to build complex systems.

Instantiating a Lower-Level Module

Example: **Step 1: Define the Lower-Level Module**

This is a basic 2-input AND gate module:

```
module and_gate (
        input A,
        input B,
        output Y
);
        assign Y = A & B;
endmodule
```

Step 2: Instantiate and gate in a Higher-Level Module

Now, create a module that uses two and gate modules to build a 3-input AND logic.

```
module three_input_and (
        input A,
        input B,
        input C,
        output Y
);

        wire temp;  // Intermediate wire
```

```
// First AND gate: temp = A & B
and_gate u1 (
.A(A),
.B(B),
.Y(temp)
);

// Second AND gate: Y = temp & C
and_gate u2 (
.A(temp),
.B(C),
.Y(Y)
);
```

endmodule

This is structural design with hierarchical instantiation:

- And gate is the **lower-level module**,
- Three input and is the **higher-level module**.

Gate level primitives

Here's a table of the most commonly used Verilog gate-level primitives:

Primitive	Function	Description
and	AND gate	Output is high only if all inputs are high
or	OR gate	Output is high if any input is high
nand	NAND gate	Inverse of AND
nor	NOR gate	Inverse of OR
xor	XOR gate	Output is high if inputs are different
xnor	XNOR gate	Output is high if inputs are same
not	Inverter (NOT gate)	Output is logical complement of input
buf	Buffer gate	Passes input to output unchanged

Syntax for Gate-Level Instantiation

gate_type instance_name (output, input1, input2, ...);

Example: Simple AND Gate

```
module gate_example (
        input A, B,
        output Y
);
        and G1 (Y, A, B);  // G1 is the instance name
endmodule
```

Example:2-Input XOR Gate

```
module xor_example (
        input A, B,
        output Y
);
```

```
    xor G2 (Y, A, B);
endmodule
```

Example:Combining Gates – NAND and NOT

```
module nand_not_example (
    input A, B,
    output Y
);
    wire temp;
    nand G3 (temp, A, B);  // NAND gate
    not  G4 (Y, temp);    // Inverter
endmodule
```

Many gate primitives support more than two inputs:

```
and G5 (Y, A, B, C, D);  // Y = A & B & C & D
```

User defined primitives(UDPs)

UDPs are *custom logic blocks* defined using truth tables in Verilog. Unlike modules, which can contain multiple components and outputs, UDPs are limited but offer a compact way to model:

- Basic combinational logic (like gates, MUXes)
- Simple sequential elements (like latches and flip-flops)

Special Symbols in UDP Truth Tables:

Symbol	Meaning
0, 1	Literal logic levels

?	Don’t care
-	No change
(01)	Rising edge trigger
(10)	Falling edge trigger

Combinational UDP:

```
primitive <name> (output, input1, input2, ...);
        output <output>;
        input <input1>, <input2>, ...;
          table
        // inputs : output
        val val : val;
        ...
        endtable
endprimitive
```

Example 1: Combinational UDP (2-input AND gate)

```
primitive and_udp (Y, A, B);
        output Y;
```

```
    input A, B;
     table
    // A B : Y
    0 0 : 0;
    0 1 : 0;
    1 0 : 0;
    1 1 : 1;
    endtable
endprimitive
```

Example 2: Combinational UDP (2:1 MUX)

```
primitive mux2_1_udp (Y, A, B, Sel);
    output Y;
    input A, B, Sel;
    table
    // A B Sel : Y
    0 ?  0  : 0;
    1 ?  0  : 1;
    ? 0  1  : 0;
    ? 1  1  : 1;
    endtable
endprimitive
```

Sequential UDP:

```
primitive <name> (output, data, clock);
        output reg <output>;
        input <data>, <clock>;
         table
        // data clock : current_state : next_state
        val  (01)   :    ?        : val;
        ...
        endtable
endprimitive
```

Example 1: Sequential UDP (D Flip-Flop with Positive Edge Trigger)

```
primitive d_flip_flop_udp (Q, D, clk);
        output reg Q;
        input D, clk;
          table
        // D clk : Q : Qnext
        0 (01) : ? : 0;
        1 (01) : ? : 1;
        ? (0?) : ? : -;  // no change
        ? (1?) : ? : -;
        endtable
endprimitive
```

Adding delay to primitives

Delays are specified with a # symbol and are usually used for **gate-level modeling** or **testbench simulation**.

```
module delay_example (output Y, input A, B);
  wire temp;
   and #5 G1 (temp, A, B); // 5 time unit delay for AND gate
  not #3 G2 (Y, temp);        // 3 time unit delay for NOT gate
endmodule
```

- #5 means the AND gate takes 5 time units to produce its output.
- #3 means the NOT gate takes 3 time units to invert.

These are **unit delays** used in simulation — they don't affect synthesis.

Delays in Testbenches

```
initial begin
  A = 0; B = 0;
  #10 A = 1;
  #20 B = 1;
end
```

Here, #10 and #20 cause time-based changes to the inputs for simulating behavior over time.

Gate-Level Modeling with Delay

Verilog includes gate primitives like and, or, not, etc., and you can add delay directly:

```
and #(2) g1 (out, in1, in2); // simple delay

and #(2:3:4) g2 (out, in1, in2); // min:typ:max delays
```

n VHDL – Adding Delay

In VHDL, delay is specified using the after keyword.

```
Y <= A and B after 5 ns;
```

This means the AND operation produces an output 5 nanoseconds after the inputs change.

QUESTION BANK

1. What is a decoder? Name any two common types.
2. Draw the truth table of a 2-to-4 line decoder.
3. What is the function of a 7-segment display decoder?
4. Define encoder. Give an example.
5. What is the function of a multiplexer?
6. Draw the logic diagram of a half adder and write its truth table.
7. What is the difference between a half subtractor and a full subtractor?
8. What does a 1-bit comparator do?
9. What is a module in Verilog?
10. Write a simple Verilog module for an AND gate using `assign`.
11. Design a 3-to-8 decoder using two 2-to-4 decoders.
12. Implement a 4:1 MUX using basic logic gates.
13. Write the truth table and logic diagram for an 8-to-3 priority encoder.
14. Design a 1-to-4 demultiplexer and explain its working.
15. Design a 4-bit ripple-carry adder using full adders.
16. Write the Boolean expressions for a full subtractor and draw the logic circuit.
17. Compare a 2-bit number using logic gates and write the output for “A > B”.
    Instantiate a 2-to-1 multiplexer module inside a top module in Verilog.
18. Explain the concept of gate-level modeling with an example.
19. Write a Verilog module that uses user-defined primitives (UDP) for an OR gate.

20. Design a 4-to-16 decoder using 2-to-4 decoders with enable lines. Show the block diagram.

21. Implement a decimal-to-7 segment decoder and explain how it works with BCD input.

22. Design a 16:1 multiplexer using only 4:1 MUXes. Show hierarchy and logic.
23. Design and write a Verilog module for a 4-bit comparator that outputs “equal”, “greater than”, and “less than”.

24. Write Verilog code for a 4-bit adder-subtractor using structural modeling.

25. Design a full subtractor using two half subtractors and one OR gate. Prove it algebraically and with truth tables.
26. Design a top-level module that instantiates two lower-level modules (e.g., adder and comparator) and connects their inputs and outputs.

27. Add delays to a gate-level Verilog model of a 2-input NAND gate using the `nand #` syntax.

28. Create a user-defined primitive (UDP) for a D latch and include timing delays.

KTU SPECIAL
CONNECT WITH US

WHATSAPP GROUP

FOLLOW US NOW

TELEGRAM CHANNEL

SUBSCRIBE NOW

www.ktuspecial.in

SUBSCRIBE

FOLLOW US

Visit Website