

# **ATME COLLEGE OF ENGINEERING**

**13th KM Stone, Bannur Road, Mysore - 570 028**



## **DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING**

### **NOTES**

**Course Title : Digital Logic Circuits**

**Course CODE: BEE306A**

**SEMESTER: III**

**Academic Year - 2024-25**

# **INSTITUTIONAL VISION AND MISSION**

## **VISION:**

- Development of academically excellent, culturally vibrant, socially responsible and globally competent human resources.

## **MISSION:**

- To keep pace with advancements in knowledge and make the students competitive and capable at the global level.
- To create an environment for the students to acquire the right physical, intellectual, emotional and moral foundations and shine as torchbearers of tomorrow's society.
- To strive to attain ever-higher benchmarks of educational excellence.

## **Department Vision and Mission**

### **Vision:**

To create Electrical & Electronics Engineers who excel to be technically competent and fulfill the cultural and social aspirations of the society.

### **Mission:**

- To provide knowledge to students that builds a strong foundation in the basic principles of electrical engineering, problem solving abilities, analytical skills, soft skills and communication skills for their overall development.
- To offer outcome based technical education.
- To encourage faculty in training & development and to offer consultancy through research & industry interaction.

## **Program Educational Objectives (PEOs)**

### **PEO1:**

To produce competent and ethical Electrical and Electronics Engineers who will exhibit the necessary technical and managerial skills to perform their duties in society.

### **PEO2:**

To make Graduates continuously acquire and enhance their technical and socio-economic skills.

### **PEO3:**

To aspire Graduates on R & D activities leading to offering solutions and excel in various career paths.

### **PEO4:**

To produce quality engineers who have the capability to work in teams and contribute to real time projects.

## **Program Outcomes (POs)**

**Engineering Graduates will be able to:**

**PO1: Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.

**PO2: Problem Analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

**PO3: Design / Development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

**PO4: Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

**PO5: Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

**PO6: The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

**PO7: Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

**PO8: Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

**PO9: Individual and team work:** Function effectively as an individual and as a member or leader in diverse teams, and in multidisciplinary settings.

**PO10: Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

**PO11: Project management and finance:** Demonstrate knowledge and understanding of the engineering management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

**PO12: Life-long learning:** Recognize the need for and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change.

### **Program Specific Outcomes (PSOs)**

The students will develop an ability to produce the following engineering traits:

PSO1: Apply the concepts of Electrical & Electronics Engineering to evaluate the performance of power systems and also to control industrial drives using power electronics

PSO2: Demonstrate the concepts of process control for Industrial Automation, design models for environmental and social concerns and also exhibit continuous self- learning

# Digital Logic Circuit

				Academic Year: 2024-2025			
Department: Electrical and Electronics Engineering							
Course Code	Course Title	Core/Elective	Prerequisite	Contact Hours			Total Hrs/ Sessions
				L	T	P	
<b>BEE306A</b>	<b>Digital Logic Circuit</b>	<b>Elective</b>	<b>Basic Electronics</b>	<b>3</b>	<b>-</b>	<b>-</b>	<b>40 theory</b>
<b>Topics Covered as per Syllabus</b>							
<b>Module-1</b>							
Definition of combinational logic, canonical forms, Generation of switching equations from truth tables, Karnaugh maps-3,4,5 variables, Incompletely specified functions (Don't care terms) Simplifying Max term equations, Quine-McCluskey minimization technique, Quine-McCluskey using don't care terms, Reduced prime implicants Tables.							
<b>Module-2</b>							
General approach to combinational logic design, Decoders, BCD decoders, Encoders, digital multiplexers, Using multiplexers as Boolean function generators, Adders and subtractors, Cascading full adders, Look ahead carry, Binary comparators							
<b>Module-3</b>							
Basic Bistable elements, Latches, Timing considerations, The master-slave flip-flops (pulsetriggered flip-flops): SR flip-flops, JK flip-flops, Edge triggered flip-flops, Characteristic equations							
<b>Module-4</b>							
Registers, binary ripple counters, synchronous binary counters, Counters based on shift registers, Design of a synchronous counter, Design of a synchronous mod-n counter using clocked T, JK, D and SR flip-flops.							
<b>Module-5</b>							
Mealy and Moore models, State machine notation, Synchronous Sequential circuit analysis, Construction of state diagrams, counter design.							
Memories: Read only and Read/Write Memories, Programmable ROM, EPROM, Flash memory.							
<b>List of Text Books</b>							
<b>"Digital Logic Applications and Design"</b> by John M Yarbrough, 2011 edition. <b>"HDL Programming (VHDL and Verilog)"</b> by Nazeih M. Botros, 1 st Edition <b>"Digital Principles and Design "</b> , Donald D Givone, Tata McGraw Hill Edition,2002.							
<b>List of Reference Books</b>							
<b>"Logic Design"</b> by RD Sudhaker Samuel							
<b>List of URLs, Text Books, Notes, Multimedia Content, etc:</b> <a href="https://www.youtube.com/watch?v=VnZLRrJYa2I">https://www.youtube.com/watch?v=VnZLRrJYa2I</a>							

## MODULE 2

### Analysis & Design of Combinational Logic

#### Structure

- Objective
- Introduction
- General approach
- Decoders-BCD decoders, Encoders.
- Digital multiplexers-using multiplexers as Boolean function generators & Design methods of building blocks of combinational logics
- Adders and Subtractors-Cascading full adders
- Lookahead carry
- Binary comparators..
- Outcome
- Future Readings

#### Objective

- Ability to understand, analyze and design various combinational circuit.
- 

#### Introduction

The complex combinational circuits can be designed using fundamental circuits, this fundamental circuits mean the we have considered half adder, full adder, the decoder. Now, we will read how the combinational circuits can be designed using another fundamental circuit called multiplexer

#### General approach

Combinational Circuits A combinational circuit consists of logic gates whose outputs, at any time, are determined by combining the values of the inputs. A combinational circuit consists of logic gates whose outputs, at any time, are determined by combining the values of the inputs. For  $n$  input variables, there are  $2^n$  possible binary input combinations. For  $n$  input variables, there are  $2^n$  possible binary input combinations. For each binary combination of the input variables, there is one possible binary value on each output. For each binary combination of the input variables, there is one possible binary value on each output.

#### 1. Design a combinational circuit that will multiply two 2-bit binary values

Solution:

1. input variables ( $A_1, A_0, B_1, B_0$ )  
output variables ( $P_3, P_2, P_1, P_0$ )

Construct a truth table

Inputs				Outputs			
$A_1$	$A_0$	$B_1$	$B_0$	$P_3$	$P_2$	$P_1$	$P_0$
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

**The output SOP equations are:**

$$P_3 = f(A_1, A_0, B_1, B_0) = \sum(15)$$

$$P_2 = f(A_1, A_0, B_1, B_0) = \sum(10, 11, 14)$$

$$P_1 = f(A_1, A_0, B_1, B_0) = \sum(6, 7, 9, 11, 13, 14)$$

$$P_0 = f(A_1, A_0, B_1, B_0) = \sum(5, 7, 13, 15)$$

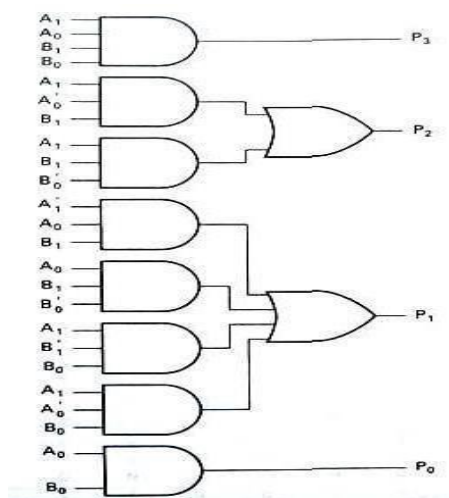
**The individually simplified equations are**

$$P_3 = A_1 A_0 B_1 B_0$$

$$P_2 = A_1 A_0' B_1 + A_1 B_1 B_0'$$

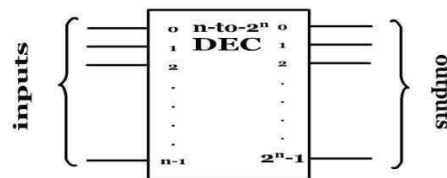
$$P_1 = A_1' A_0 B_1 + A_0 B_1 B_0' + A_1 B_1' B_0 + A_1 A_0' B_0$$

$$P_0 = A_0 B_0$$



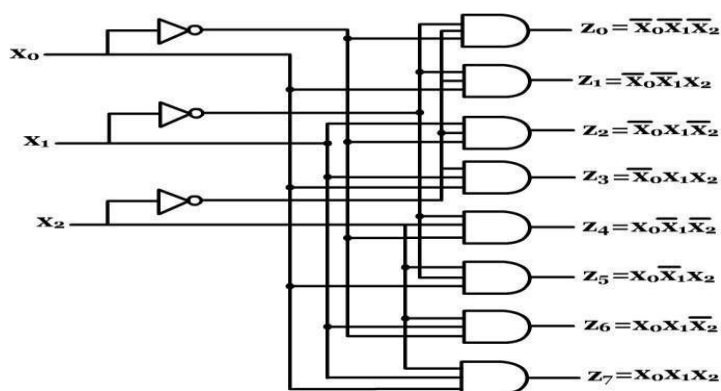
**Decoders-BCD decoders, Encoders.**

A Decoder is a multiple input, multiple output logic circuit. The block diagram of a decoder is as shown below.



**An n-to-2<sup>n</sup> line decoder symbol.**

The most commonly used decoder is an n-to-2<sup>n</sup> decoder which has n inputs and 2<sup>n</sup> output lines.

**3-to-8 decoder logic diagram**

**A 3-to-8 decoder Logic diagram**

Inputs			Outputs							
X <sub>2</sub>	X <sub>1</sub>	X <sub>0</sub>	Z <sub>0</sub>	Z <sub>1</sub>	Z <sub>2</sub>	Z <sub>3</sub>	Z <sub>4</sub>	Z <sub>5</sub>	Z <sub>6</sub>	Z <sub>7</sub>
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

**Truth table.**



In this realization shown above the three inputs are assigned  $x_0, x_1$ , and  $x_2$ , and the eight outputs are  $Z_0$  to  $Z_7$ .

Function specific decoders also exist which have less than  $2^n$  outputs. Examples are 8421 code decoder also called BCD to decimal decoder. Decoders that drive seven segment displays also exist.

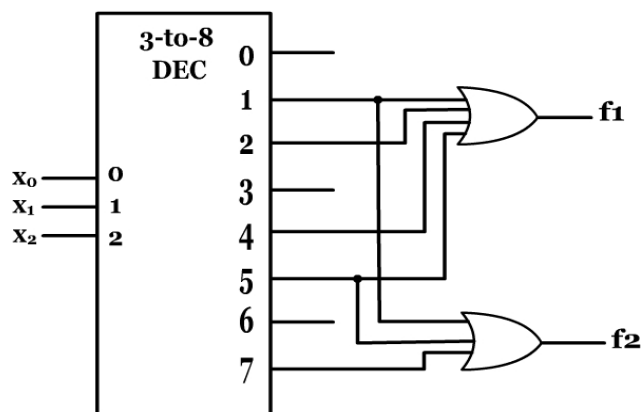
### Realization of boolean expression using Decoder and OR gate

We see from the above truth table that the output expressions correspond to a single minterm. Hence an  $n$ -to- $2^n$  decoder is a minterm generator. Thus by using OR gates in conjunction with a  $n$ -to- $2^n$  decoder boolean function realization is possible.

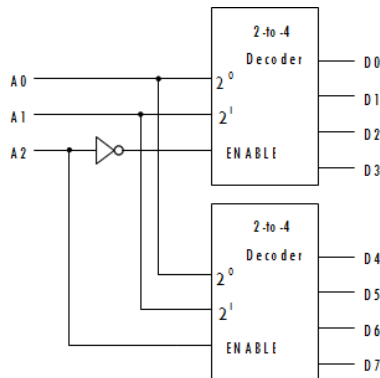
**P1:** to realize the Boolean functions given below using decoders...

$$F_1 = \sum m(1, 2, 4, 5)$$

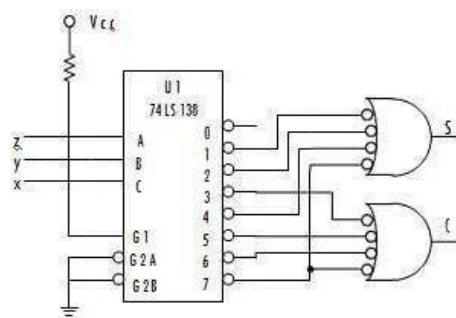
$$F_2 = \sum m(1, 5, 7)$$



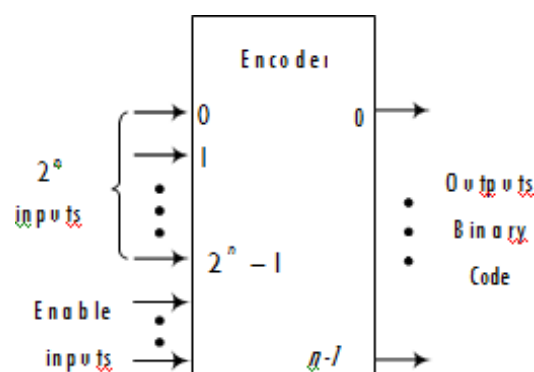
### Realisation of boolean expressions

**P2:A3-to-8Decoder constructed****P3:Design a binary 3-bit adder with a 74xx138 and NAND gates.**

$$S = f(x, y, z) = \sum m(1, 2, 4, 7), \quad C_f(X, Y, Z) = \sum m(3, 5, 6, 7)$$

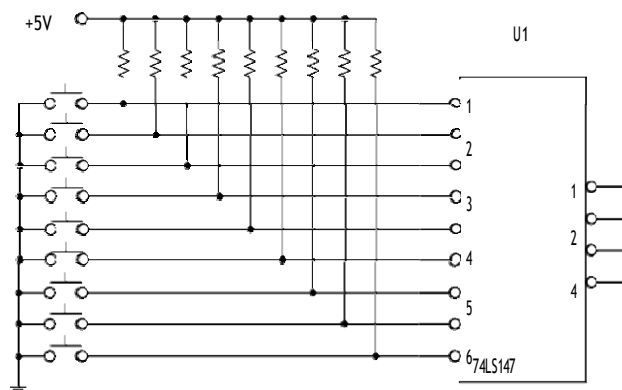
**Encoder**

It is an inverse of a decoder having  $2^n$  inputs and  $n$  outputs.



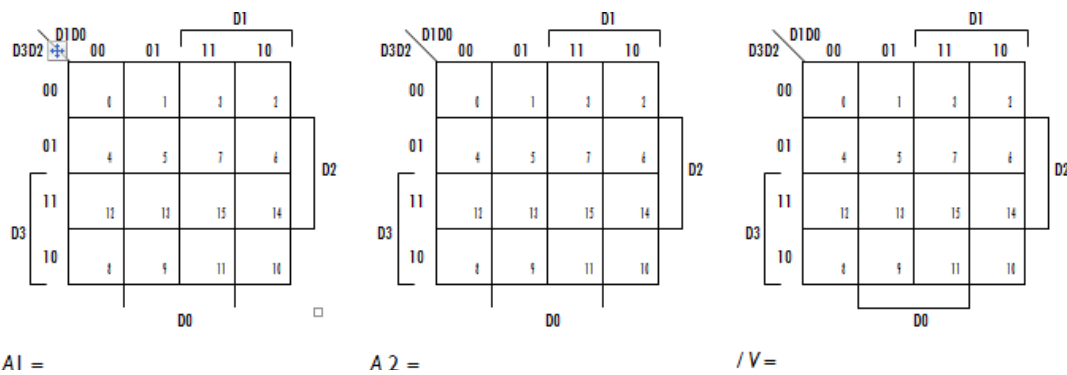
**P4:Decimal-to-BCDEncoder(74xxx147)**

Inputs									Outputs			
1	2	3	4	5	6	7	8	9	D	C	B	A
1	1	1	1	1	1	1	1	1				
0	1	1	1	1	1	1	1	1				
x	0	1	1	1	1	1	1	1				
x	x	0	1	1	1	1	1	1				
x	x	x	0	1	1	1	1	1				
x	x	x	x	0	1	1	1	1				
x	x	x	x	x	0	1	1	1				
x	x	x	x	x	x	0	1	1				
x	x	x	x	x	x	x	0	1				
x	x	x	x	x	x	x	x	0				

**priorityencoder**

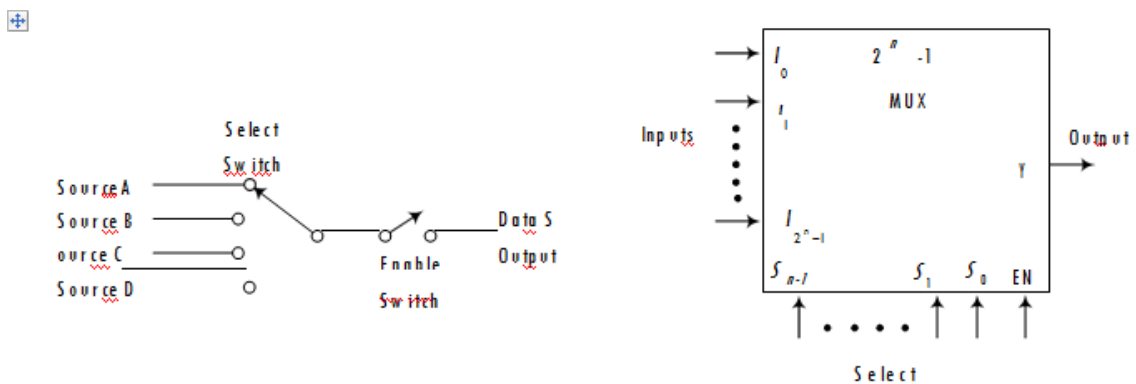
Several possible events may occur in an industrial system, and you want to identify an event and assign and transmit a code to the control unit based on some priority.

Inputs				Outputs		
D3	D2	D1	D0	A1	A0	/V
0	0	0	0			
0	0	1	1			
0	1	X	X			
1	X	X	X			

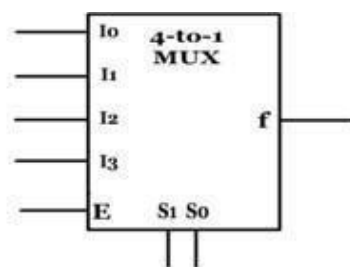


### Digital multiplexers-using multiplexers as Boolean function generators. & Design methods of building blocks of combinationallogics.

Multiplexers also called data selectors are another MSI devices with a wide range of applications in microprocessor and their peripherals design. The following diagram show the symbol and truth table for the 4-to-1 mux.



#### P1:4-to-1LineMultiplexer



A 4-to-1 line multiplexer symbol.

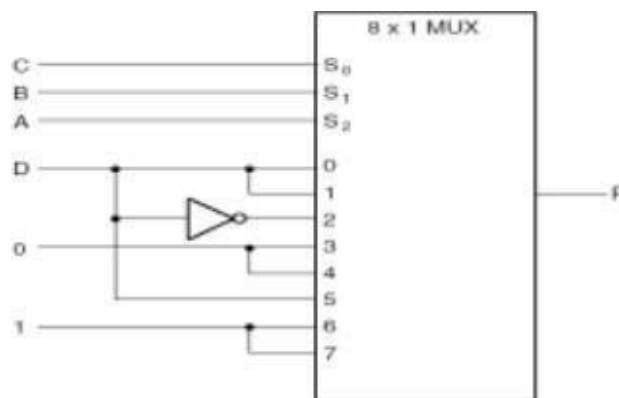
E	S <sub>1</sub>	S <sub>0</sub>	I <sub>0</sub>	I <sub>1</sub>	I <sub>2</sub>	I <sub>3</sub>	f
0	x	x	x	x	x	x	0
1	0	0	0	x	x	x	0
1	0	0	1	x	x	x	1
1	0	1	x	0	x	x	0
1	0	1	x	1	x	x	1
1	1	0	x	x	0	x	0
1	1	0	x	x	1	x	1
1	1	1	x	x	x	0	0
1	1	1	x	x	x	1	1

Compressed Truth table

P2: Consider the function  $F(A,B,C,D) = \sum(1,3,4,11,12,13,14,15)$

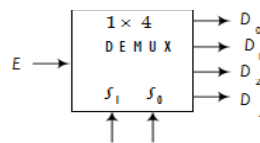
This function can be implemented with an 8-to-1 line MUX (see Figure 7). A, B, and C are applied to the select inputs as follows:  $A \Rightarrow S_2$ ,  $B \Rightarrow S_1$ ,  $C \Rightarrow S_0$

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



### Demultiplexers

- Perform the opposite function of multiplexers
- Placing the value of a single data input onto one of the multiple data outputs
- Same implementation as decoder with enable
- Enable input of decoder serves as the data input for the demultiplexer



### P1: A1-to-4lineDemux

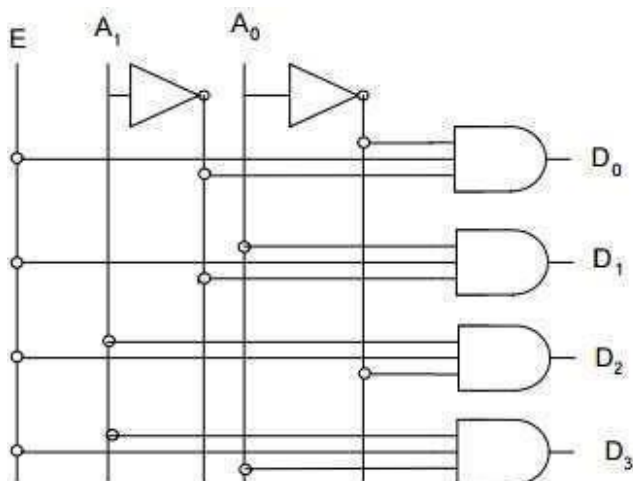
The input  $E$  is directed to one of the outputs, as specified by the two select lines  $S_1$  and  $S_0$ .  $D_0 = E$  if  $S_1 S_0 = 00 \Rightarrow D_0 = S_1' S_0' E$

$D_1 = E$  if  $S_1 S_0 = 01 \Rightarrow D_1 = S_1' S_0 E$

$D_2 = E$  if  $S_1 S_0 = 10 \Rightarrow D_2 = S_1 S_0' E$

$D_3 = E$  if  $S_1 S_0 = 11 \Rightarrow D_3 = S_1 S_0 E$

A careful inspection of the Demux circuit shows that it is identical to a 2 to 4 decoder with enable input.



Decimal value	Enable	Inputs		Outputs			
	$E$	$A_1$	$A_0$	$D_0$	$D_1$	$D_2$	$D_3$
	0	X	X	0	0	0	0
0	1	0	0	1	0	0	0
1	1	0	1	0	1	0	0
2	1	1	0	0	0	1	0
3	1	1	1	0	0	0	1

Table for 1-to-4 line demultiplexer

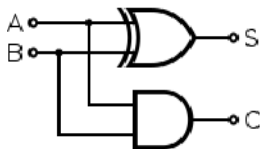
### Adders and Subtractors - Cascading full adders

Consider adding two binary numbers together:

$$\begin{array}{r}
 0101 \\
 +0011 \\
 \hline
 00010
 \end{array}$$

carry bit

We see that the bit in the "two's" column is generated when the addition is carried over. A half-adder is a circuit which adds two bits together and outputs the sum of those two bits. The half-adder has two outputs: **sum** and **carry**. Sum represents the remainder of the integer division  $A+B/2$ , while carry is the result. This can be expressed as follows:



A	B	A+B	S	C
0	0	0	0	0
0	1	1	1	0
1	0	1	1	0
1	1	2	0	1

$$S = A \oplus B$$

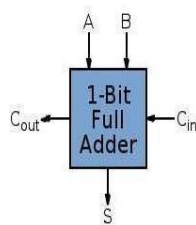
$$C = AB$$

### Full Adder:

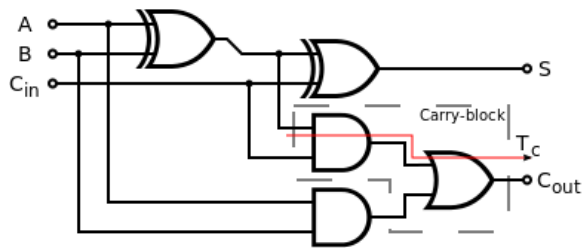
Half-adders have a major limitation in that they cannot accept a carry bit from a previous stage, meaning that they cannot be chained together to add multi-bit numbers. However, the two output bits of a half-adder can also present the result  $A+B=3$  assuming both are high.

As such, the full-adder can accept three bits as an input. Commonly, one bit is referred to as the carry-in bit. Full adders can be cascaded to produce adders of any number of bits by daisy-chaining the carry of one output to the input of the next.

The full-adder is usually shown as a single unit. The sum output is usually on the bottom on the block, and the carry-out output is on the left, so the devices can be chained together, most significant bit leftmost:



Logic Symbol of Full adder

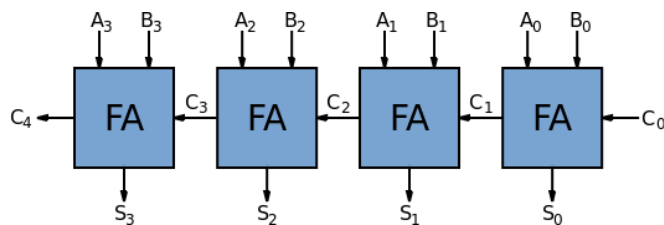


A	B	C <sub>in</sub>	A+B+C <sub>in</sub>	S	C <sub>out</sub>
0	0	0	0	0	0
0	0	1	1	1	0
0	1	0	1	1	0
0	1	1	2	0	1
1	0	0	1	1	0
1	0	1	2	0	1
1	1	0	2	0	1
1	1	1	3	1	1

### RippleCarryAdder:

A ripple carry adder is simply several full adders connected in a series so that the carry must propagate through every full adder before the addition is complete. Ripple carry adders require the least amount of hardware of all adders, but they are the slowest.

The following diagram shows a four-bit adder, which adds the numbers A[3:0] and B[3:0], as well as a carry input, together to produce S[3:0] and the carry output



### PropagationDelayinFullAdders

Real logic gates do not react instantaneously to the inputs, and therefore digital circuits have a maximum speed. Usually, the delay through a digital circuit is measured in gate-delays, as this allows the delay of a design to be calculated for different devices. AND and OR gates have a nominal delay of 1 gate-delay, and XOR gates have a delay of 2, because they are really made up of a combination of ANDs and ORs.

A full adder block has the following worst case propagation delays:

- From  $A_i$  or  $B_i$  to  $C_{i+1}$ : 4 gate-delays (XOR  $\rightarrow$  AND  $\rightarrow$  OR)
- From  $A_i$  or  $B_i$  to  $S_i$ : 4 gate-delays (XOR  $\rightarrow$  XOR)
- From  $C_i$  to  $C_{i+1}$ : 2 gate-delays (AND  $\rightarrow$  OR)
- From  $C_i$  to  $S_i$ : 2 gate-delays (XOR)

Because the carry-out of one stage is the next's input, the worst case propagation delay is then:

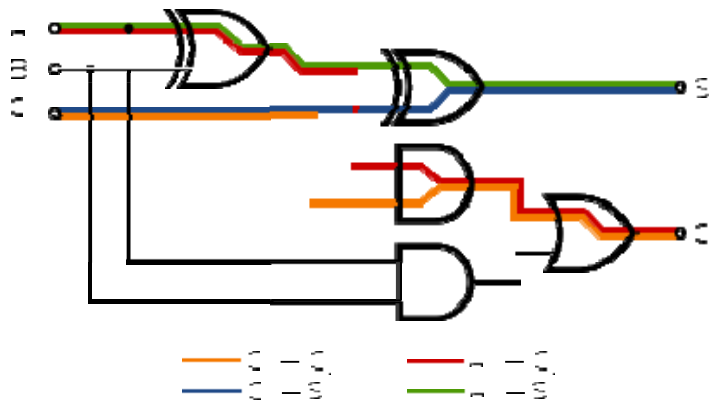
- 4 gate-delays from generating the first carry signal ( $A_0/B_0 \rightarrow C_1$ ).
- 2 gate-delays per intermediate stage ( $C_i \rightarrow C_{i+1}$ ).



- 2 gate-delays at the last stage to produce both the sum and

Carry-out outputs ( $C_{n-1} \rightarrow C_n$  and  $S_{n-1}$ ).

So for an  $n$ -bit adder, we have a total propagation delay,  $t_p$  of:



$$t_p = 4 + 2(n - 2) + 2 = 2n + 2$$

This is linear in  $n$ , and for a 32-bit number, would take 66 cycles to complete the calculation. This is rather slow, and restricts the word length in our device somewhat. We would like to find ways to speed it up.

### Lookahead carry

A fast method of adding numbers is called carry-lookahead. This method doesn't require the carry signal to propagate stage by stage, causing a bottleneck. Instead it uses additional logic to expedite the propagation and generation of carry information, allowing fast addition at the expense of more hardware.

In a ripple adder, each stage compares the carry-in signal,  $C_i$ , with the inputs  $A_i$  and  $B_i$  and generates a carry-out signal  $C_{i+1}$  accordingly. In a carry-lookahead adder, we define two new functions.

The generate function,  $G_i$ , indicates whether that stage causes a carry-out signal  $C_i$  to be generated if no carry-in signal exists. This occurs if both the addends contain a 1 in that bit:

$$G_i = A_i \cdot B_i$$

The propagate function,  $P_i$ , indicates whether a carry-in to the stage is passed to the carry-out for the stage. This occurs if either the addends have a 1 in that bit

$$P_i = A_i + B_i$$

Note that both these values can be calculated from the inputs in a constant delay. Now, the carry-out from a stage occurs if that stage generates a carry ( $G_i = 1$ ) or there is a carry-in and the stage propagates the carry ( $P_i \cdot C_i = 1$ )

$$C_{i+1} = A_i B_i + A_i C_i + B_i C_i \quad C_{i+1} = A$$

$$B_i + C_i (A_i + B_i) C_{i+1} = G_i$$

$$+ P_i C_i$$

### Truth table

$A_i$	$B_i$	$C_i$	$G_i$	$P_i$	$C_{i+1}$
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	0	1	0
1	0	1	0	1	1
1	1	0	1	1	1
1	1	1	1	1	1

$$c_{i+1} = G_i + P_i c_i$$

$$c_{i+1} = G_i + P_i (G_{i-1} + P_{i-1} c_{i-1})$$

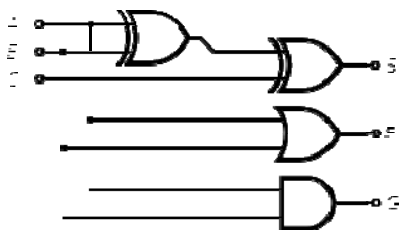
$$c_{i+1} = G_i + P_i G_{i-1} + P_i P_{i-1} (G_{i-2} + P_{i-2} c_{i-2})$$

$$c_{i+1} = \vdots$$

$$c_{i+1} = G_i + P_i G_{i-1} + P_i P_{i-1} G_{i-2} + P_i P_{i-1} P_{i-2} G_{i-3} + \dots + P_i P_{i-1} \dots P_1 P_0 c_0$$

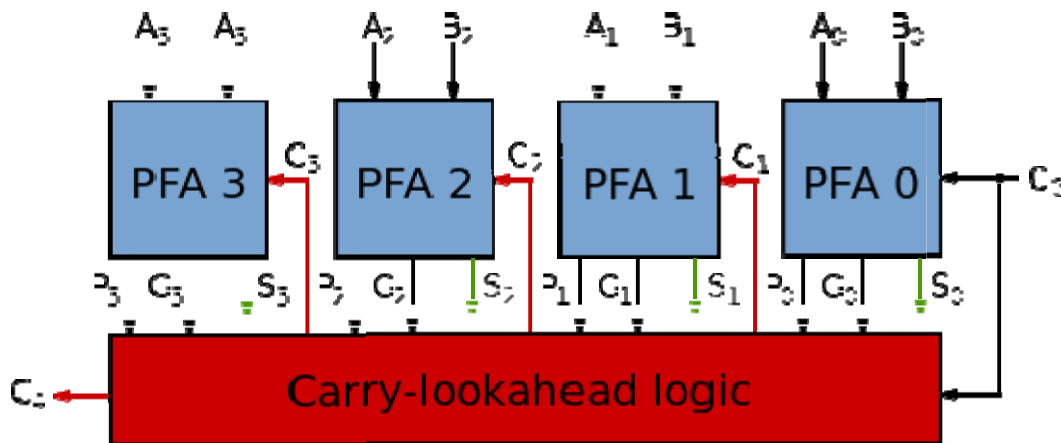
Note that this does not require the carry-out signals from the previous stages, so we don't have to wait for changes to ripple through the circuit. In fact, a given stage's carry signal can be computed once the propagate and generate signals are ready with only two more gate delays (one AND and one OR). Thus the carry-out for a given stage can be calculated in constant time, and therefore so can the sum.

The  $S$ ,  $P$ , and  $G$  signals are all generated by a circuit called a "partial full adder" (PFA), which is similar to a full adder.



For a slightly smaller circuit, the propagate signal can be taken as the output of the first XOR gate instead of using a dedicated OR gate, because if both  $A$  and  $B$  are asserted, the generate signal will force a carry. However, this simplification means that the propagate signal will take two gate delays to produce, rather than just one.

A carry lookahead adder then contains  $n$  PFAs and the logic to produce carries from the stage propagate and generate signals:

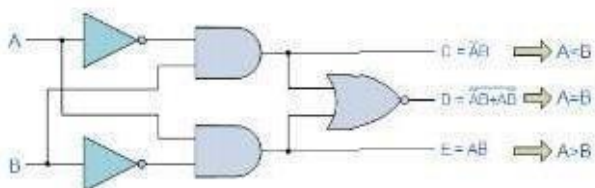


Two numbers can therefore be added in constant time,  $O(1)$ , of just 6 gate delays, regardless of the length,  $n$  of the numbers. However, this requires AND and OR gates with up to  $n$  inputs. If logic gates are available with a limited number of inputs, trees will need to be constructed to compute these, and the overall computation time is logarithmic,  $O(\ln(n))$ , which is still much better than the linear time for ripple adders.

### Binary comparators

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

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



For example, along with being able to add and subtract binary numbers we need to be able to compare them and determine whether the value of input  $A$  is greater than, smaller than or equal to the value at input  $B$  etc. The digital comparator accomplishes this using several logic gates that operate on the principles of *Boolean Algebra*.

There are two main types of **Digital Comparator** available and these are.

- 1. Identity Comparator – an *Identity Comparator* is a digital comparator that has only one output terminal for when  $A = B$  either “HIGH”  $A = B = 1$  or “LOW”  $A = B = 0$
- 2. Magnitude Comparator – a *Magnitude Comparator* is a digital comparator which has three output terminals, one each for equality,  $A = B$  greater than,  $A > B$  and less than  $A < B$

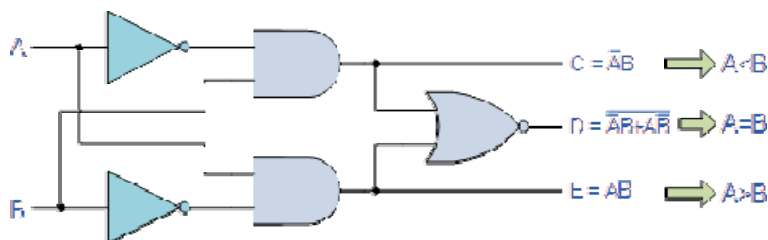
The purpose of a **Digital Comparator** is to compare a set of variables or unknown numbers, for example A ( $A_1, A_2, A_3, \dots A_n$ , etc) against that of a constant or unknown value such as B ( $B_1, B_2, B_3, \dots B_n$ , etc) and produce an output condition or flag depending upon the result of the comparison. For example, a magnitude comparator of two 1-bits, (A and B) inputs would produce the following three output conditions when compared to each other.

$$C = \bar{A}B \quad D = \bar{A}B + A\bar{B} \quad E = AB$$

Which means: A is greater than B, A is equal to B, and A is less than B

This is useful if we want to compare two variables and want to produce an output when any of the above three conditions are achieved. For example, produce an output from a counter when a certain count number is reached. Consider the simple 1-bit comparator below.

### 1-bit Digital Comparator Circuit



Then the operation of a 1-bit digital comparator is given in the following Truth Table.

### Digital Comparator Truth Table

Inputs		Outputs		
B	A	$A > B$	$A = B$	$A < B$
0	0	0	1	0
0	1	1	0	0

1	0	0	0	1
1	1	0	1	0

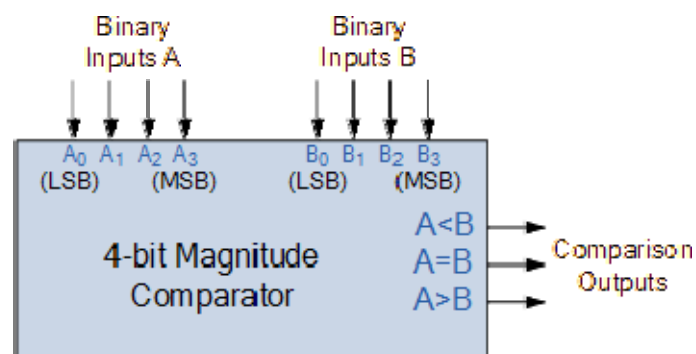
You may notice two distinct features about the comparator from the above truth table. Firstly, the circuit does not distinguish between either two “0” or two “1”’s as an output  $A = B$  is produced when they are both equal, either  $A = B = “0”$  or  $A = B = “1”$ . Secondly, the output condition for  $A = B$  resembles that of a commonly available logic gate, the Exclusive-NOR or Ex-NOR function (equivalence) on each of the  $n$ -bits giving:  $Q = A \oplus B$

Digital comparators actually use Exclusive-NOR gates within their design for comparing their respective pairs of bits. When we are comparing two binary or BCD values or variables against each other, we are comparing the “magnitude” of these values, a logic “0” against a logic “1” which is where the term **Magnitude Comparator** comes from.

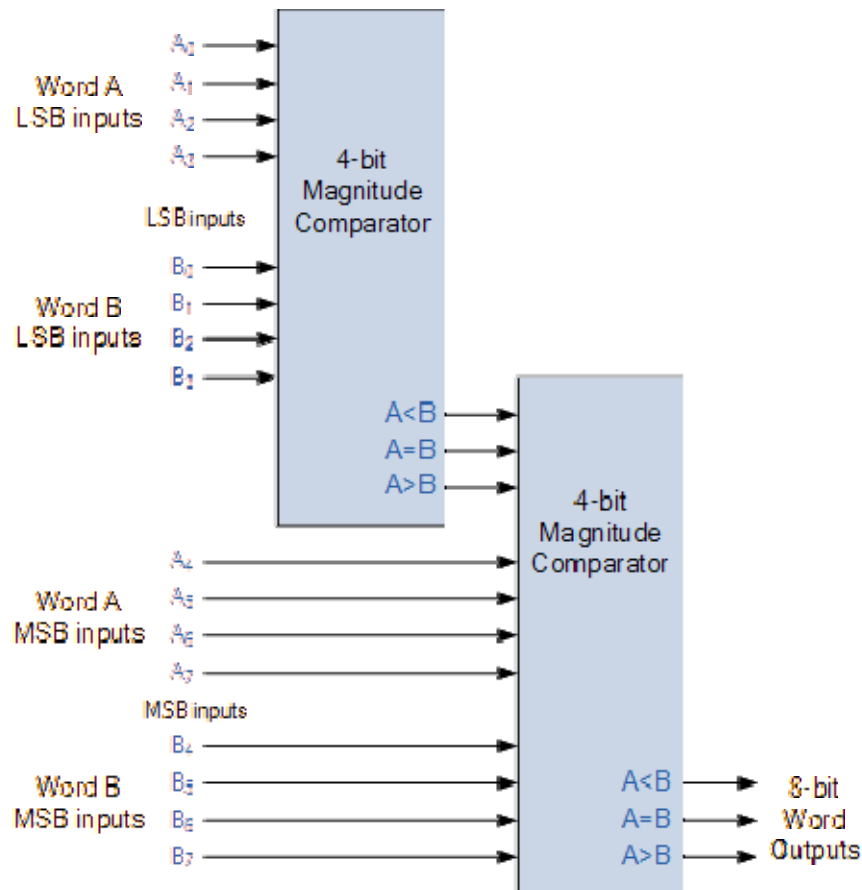
As well as comparing individual bits, we can design larger bit comparators by cascading together  $n$  of these and produce a  $n$ -bit comparator just as we did for the  $n$ -bit adder in the previous tutorial. Multi-bit comparators can be constructed to compare whole binary or BCD words to produce an output if one word is larger, equal to or less than the other.

A very good example of this is the 4-bit **Magnitude Comparator**. Here, two 4-bit words (“nibbles”) are compared to each other to produce the relevant output with one word connected to inputs A and the other to be compared against connected to input B as shown below.

#### 4-bit Magnitude Comparator



Some commercially available digital comparators such as the TTL 74LS85 or CMOS 4063 4-bit magnitude comparator have additional input terminals that allow more individual comparators to be “cascaded” together to compare words larger than 4-bits with magnitude comparators of “ $n$ ”- bits being produced. These cascading inputs are connected directly to the corresponding outputs of the previous comparator as shown to compare 8, 16 or even 32-bit words.

**8-bit Word Comparator**

When comparing large binary or BCD numbers like the example above, to save time the comparator starts by comparing the highest-order bit (MSB) first. If equality exists,  $A = B$  then it compares the next lowest bit and so on until it reaches the lowest-order bit, (LSB). If equality still exists then the two numbers are defined as being equal.

If inequality is found, either  $A > B$  or  $A < B$  the relationship between the two numbers is determined and the comparison between any additional lower order bits stops. **Digital Comparator** are used widely in Analogue-to-Digital converters, (ADC) and Arithmetic Logic Units, (ALU) to perform a variety of arithmetic operations.

**Outcome**

- Can create an appropriate truth table from the description of a combinational logic function.
- Able to design any logic circuit using MUX, DEMUX, encoders and decoders based on the application such that the gates used in a circuit are reduced.

**Future Readings**

<http://nptel.ac.in/courses/117105080/>

<https://www.youtube.com/watch?v=VnZLRrJYa2I> “Lo

gic Design” by RD Sudhaker Samuel

“Digital Logic Applications and Design” by John M. Yarbrough, 2011 edition