MODULE 2

Introduction to Combinational Logic Circuits and Advanced Combinational Logic Circuits

Structure

2.1 Objevtive
2.2 Introduction
2.3 General approach
2.4 Decoders-BCD decoders, Encoders.
2.5 Digital multiplexers-using multiplexers as Boolean function generators & Design methods of building blocks of combinational logics
2.6 Adders and Subtractors-Cascading full adders
2.7 Look ahead carry
2.8 Binary comparators. .
2.9 Outcome
2.10 Future Readings

2.1 Objevtive

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

2.2 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 circuits called multiplexer

2.3 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 two-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:

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

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

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

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

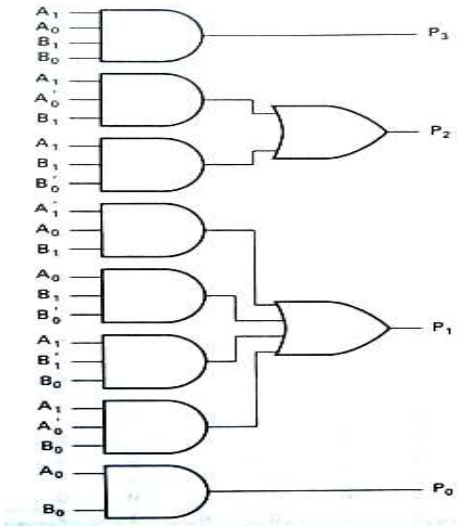
The individually simplified equations are

$P_3=A_1A_0B_1B_0$

$P_2=A_1A_0'B_1+A_1B_1B_0'$

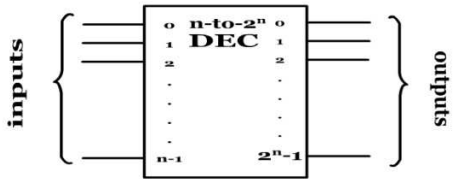
$P_1=A_1'A_0B_1+A_0B_1B_0'+A_1B_1'B_0+A_1A_0'B_0$

$P_0=A_0B_0$



2.4 Decoders-BCD decoders, Encoders.

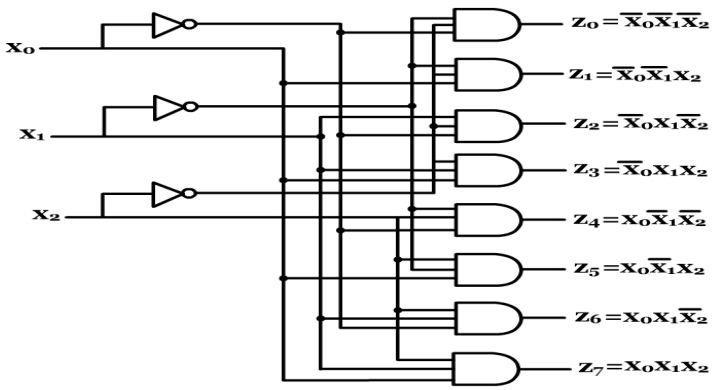
A Decoder is a multiple input ,multiple output logic circuit.The block diagram of a decoder is as shown below.



An n-to-2^nline decoder symbol.

The most commonly used decoder is a n –to 2^n decoder which ha n inputs and 2^n Output lines .

3-to-8 decoder logic diagram



A 3-to-8 decoder Logic diagram

Inputs			Outputs							
x_2	x_1	x_0	z_0	z_1	z_2	z_3	z_4	z_5	z_6	z_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Truth table.

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 specifc 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

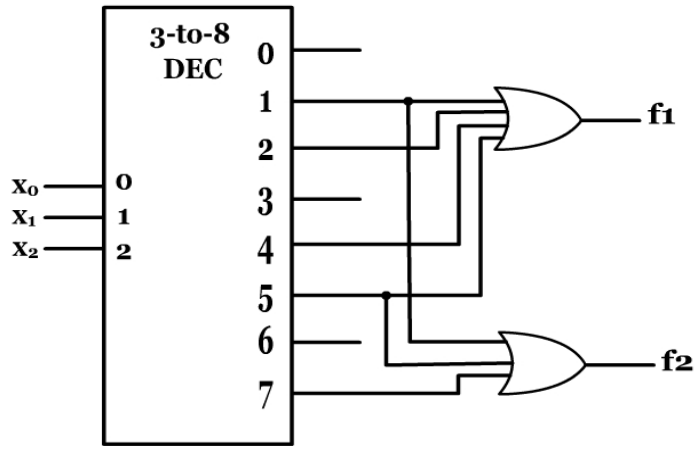
Realization of boolean expression using Decoder and OR gate

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

P1: to realize the Boolean functions given below using decoders…

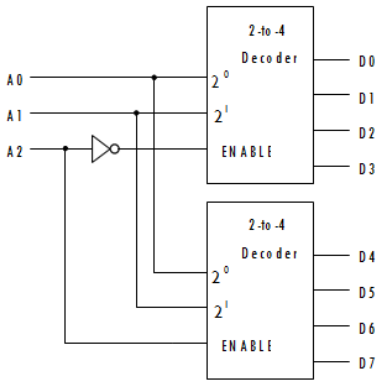
•F1=Σm(1,2,4,5)

•F2=Σm(1,5,7)



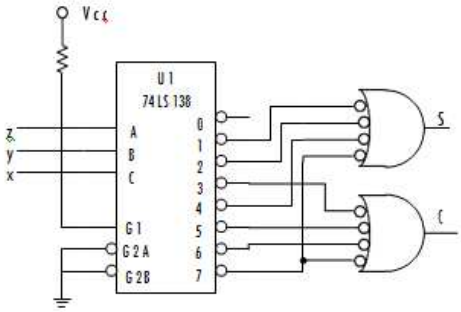
Realisation of boolean expressions

P2: A 3-to-8 Decoder constructed



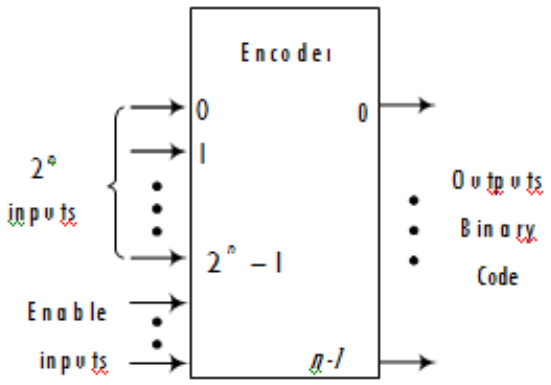
P3: Design a binary 3-bit adder with a 74xxx138 and NAND gates.

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



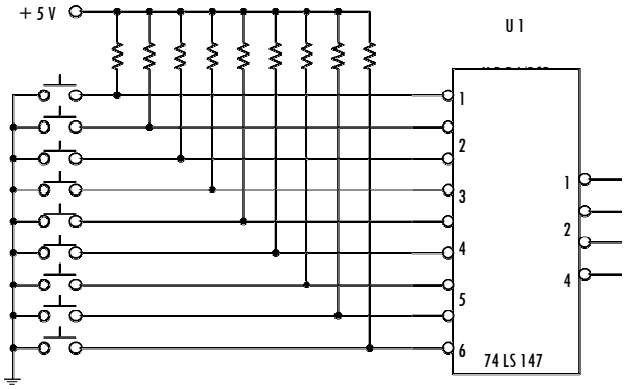
Encoder

It is a inverse of decoder having 2^n input and n output.



P4: **Decimal-to-BCD Encoder (74xxx147)**

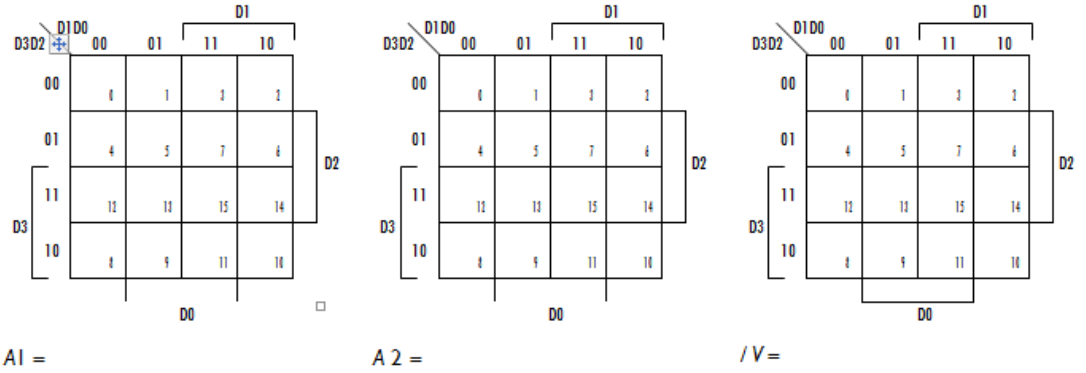
Inputs									Outputs			
1	2	3	4	5	6	7	8	9	D	C	B	A
1	1	1	1	1	1	1	1	1				
0	1	1	1	1	1	1	1	1				
x	0	1	1	1	1	1	1	1				
x	x	0	1	1	1	1	1	1				
x	x	x	0	1	1	1	1	1				
x	x	x	x	0	1	1	1	1				
x	x	x	x	x	0	1	1	1				
x	x	x	x	x	x	0	1	1				
x	x	x	x	x	x	x	0	1				
x	x	x	x	x	x	x	x	0				



priority encoder

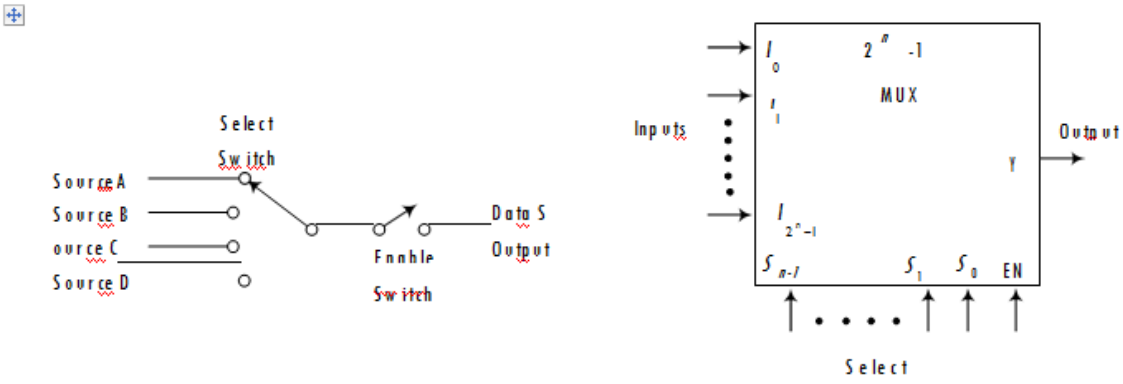
Several possible events may occur in an industrial system, and you want to identify an event and assign and transmit a code to the control unit based on some priority.

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

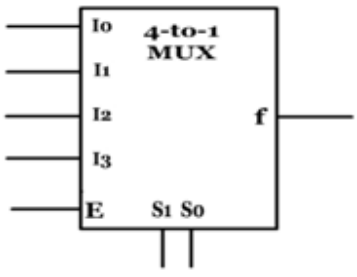


2.5 Digital multiplexers-using multiplexers as Boolean function generators. & Design methods of building blocks of combinational logics.

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



P1: 4-to-1 Line Multiplexer



A 4-to-1 line multiplexer symbol.

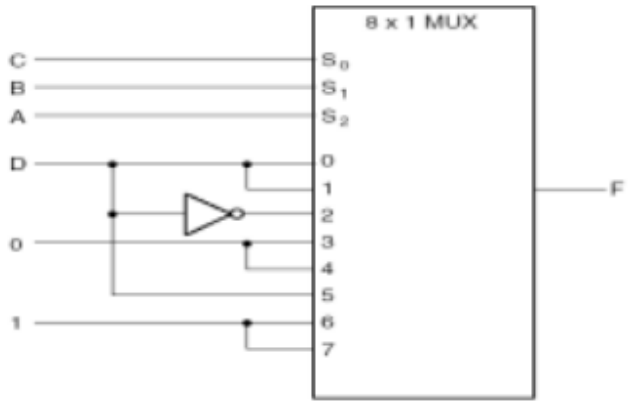
E	S1	So	Io	I1	I2	I3	f
0	x	x	x	x	x	x	0
1	0	0	0	x	x	x	0
1	0	0	1	x	x	x	1
1	0	1	x	0	x	x	0
1	0	1	x	1	x	x	1
1	1	0	x	x	0	x	0
1	1	0	x	x	1	x	1
1	1	1	x	x	x	0	0
1	1	1	x	x	x	1	1

Compressed Truth table

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

This function canbe implemented with an 8-to-1 line MUX (see Figure 7) A, B, and C are applied to the select inputs as follows: A ⇒ S2 , B ⇒ S1, C ⇒ S0

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	$F = \overline{D}$
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	



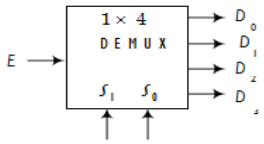
Demultiplexers

• 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: A 1-to-4 line Demux

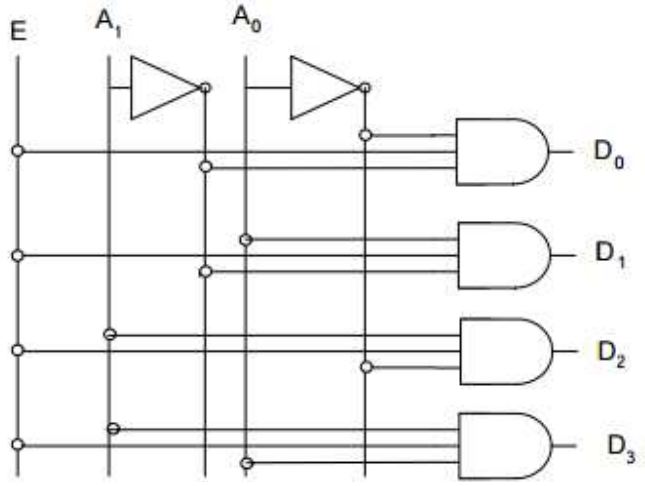
The input E is directed to one of the outputs, as specified by the two select lines S1 and S0. D0 = E if S1S0 = 00 ⇒ D0 = S1' S0' E

D1 = E if S1S0 = 01 ⇒ D1 = S1' S0 E

D2 = E if S1S0 = 10 ⇒ D2 = S1 S0' E

D3 = E if S1S0 = 11 ⇒ D3 = S1 S0 E

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

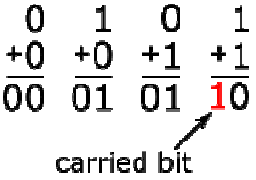


Decimal value	Enable	Inputs		Outputs			
	E	A_1	A_0	D_0	D_1	D_2	D_3
	0	X	X	0	0	0	0
0	1	0	0	1	0	0	0
1	1	0	1	0	1	0	0
2	1	1	0	0	0	1	0
3	1	1	1	0	0	0	1

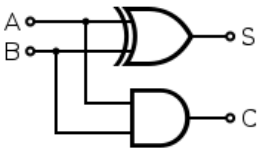
Table for 1-to-4 line demultiplexer

2.6 Adders and Subtractors-Cascading full adders

Consider adding two binary numbers together:



We see that the bit in the "two's" column is generated when the addition 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 xor B

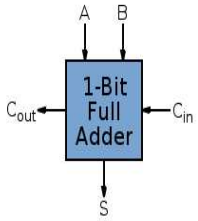
C=AB

Full Adder:

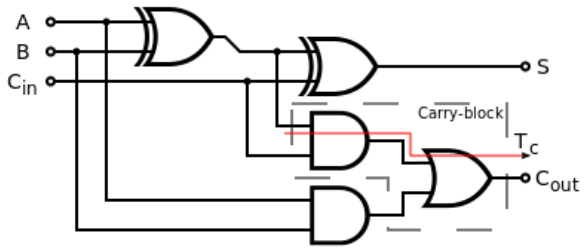
Half-adders have a major limitation in that they cannot accept a carry bit from a previous stage, meaning that they cannot be chained together to add multi-bit numbers. However, the two output bits of a half-adder can also represent the result A+B=3 as sum and carry both being high.

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

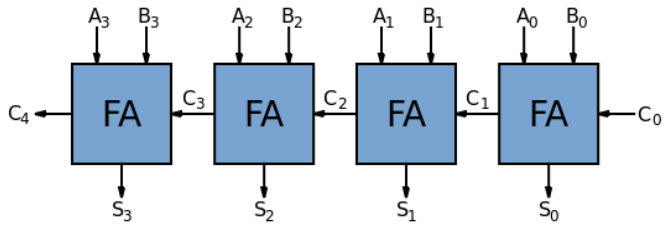


A	B	C_{in}	A+B+C_{in}	S	C_{out}
0	0	0	0	0	0
0	0	1	1	1	0
0	1	0	1	1	0
0	1	1	2	0	1
1	0	0	1	1	0
1	0	1	2	0	1
1	1	0	2	0	1
1	1	1	3	1	1

Ripple Carry Adder:

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



Propagation Delay in Full Adders

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:

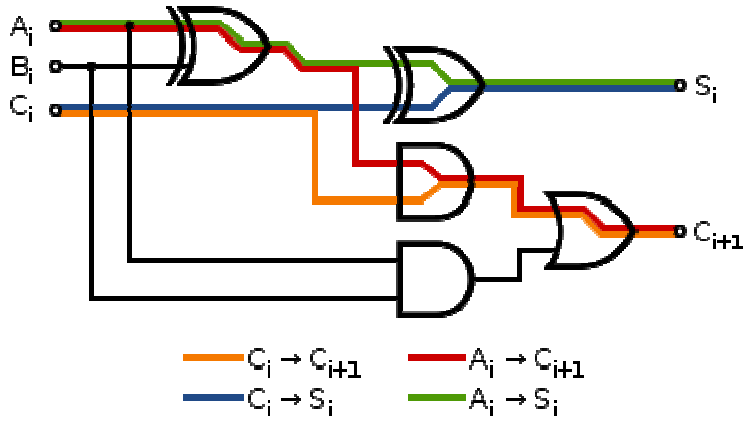
- From A_i or B_i to C_{i+1}: 4 gate-delays (XOR → AND → OR)
- From A_i or B_i to S_i: 4 gate-delays (XOR → XOR)
- From C_i to C_{i+1}: 2 gate-delays (AND → OR)
- From C_i to S_i: 2 gate-delays (XOR)

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.

2.7 Look ahead 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:

$G_i = A_i . B_i$

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 time of a single gate 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 ($P_i \cdot C_i = 1$)

$C_{i+1} = A_i B_i + A_i C_i + B_i C_i$

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

$C_{i+1} = G_i + P_i C_i$

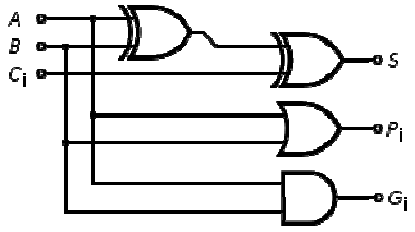
Truth table

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

$$
\begin{aligned}
c_{i+1} &= G_i + P_i c_i \\
c_{i+1} &= G_i + P_i\,(G_{i-1} + P_{i-1} c_{i-1}) \\
c_{i+1} &= G_i + P_i G_{i-1} + P_i P_{i-1}\,(G_{i-2} + P_{i-2} c_{i-2}) \\
c_{i+1} &= \quad \vdots \\
c_{i+1} &= G_i + P_i G_{i-1} + P_i P_{i-1} G_{i-2} + P_i P_{i-1} P_{i-2} G_{i-3} + \ldots + P_i P_{i-1} \cdots P_1 P_0 c_0
\end{aligned}
$$

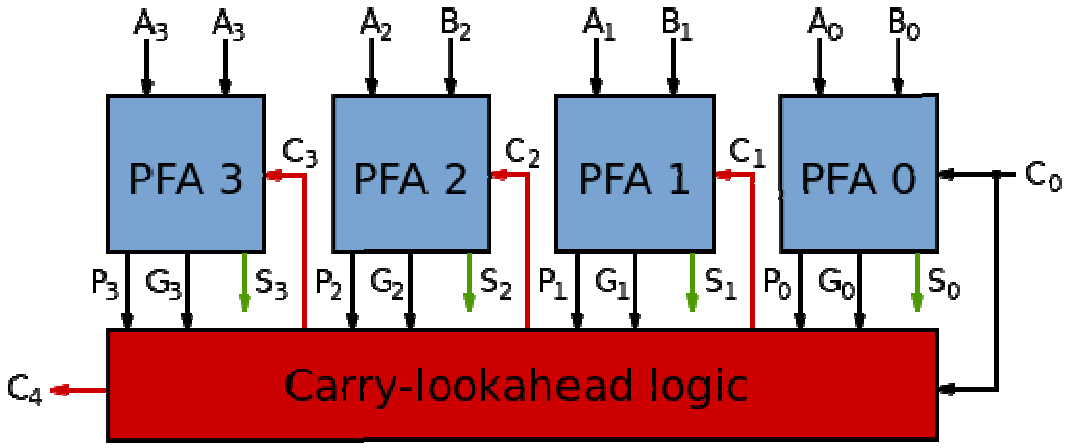
Note that this does not require the carry-out signals from the previous stages, so we don't have to wait for changes to ripple through the circuit. In fact, a given stage's carry signal can be computed once the propagate and generate signals are ready with only two more gate delays (one AND and one OR).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

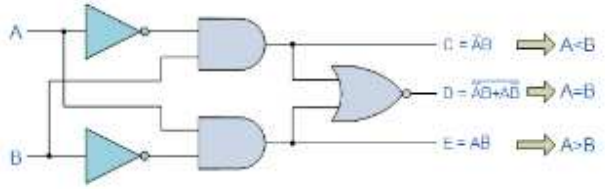


Two numbers can therefore be added in constant time, O(1), of just 6 gate delays, regardless of the length, *n* of the numbers. However, this requires AND and OR gates with up to *n* inputs. If logic gates are available with a limited number of inputs, trees will need to be constructed to compute these,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

2.8 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

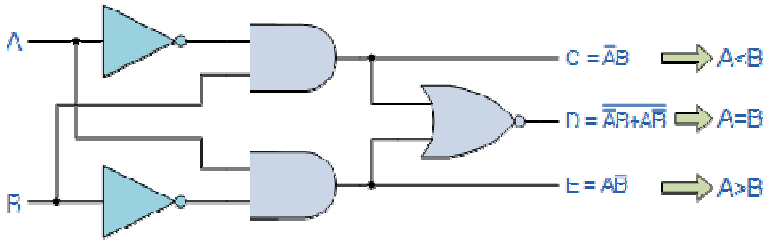
The purpose of a **Digital Comparator** is to compare a set of variables or unknown numbers, for example A (A1, A2, A3, …. An, etc) against that of a constant or unknown value such as B (B1, B2, B3, …. Bn, etc) and produce an output condition or flag depending upon the result of the comparison. For example, a magnitude comparator of two 1-bits, (A and B) inputs would produce the following three output conditions when compared to each other.

$$A > B, \quad A = B, \quad A < B$$

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

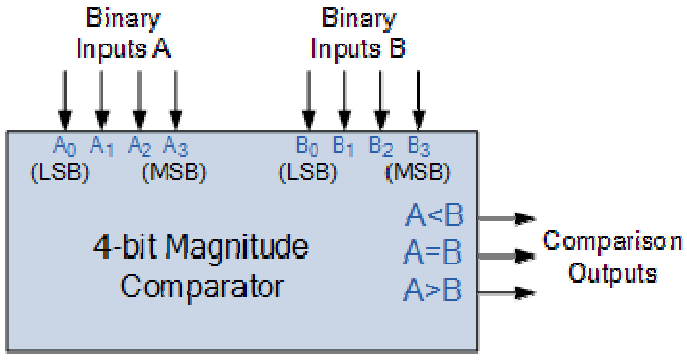
You may notice two distinct features about the comparator from the above truth table. Firstly, the circuit does not distinguish between either two "0" or two "1"'s as an output A = B is produced when they are both equal, either A = B = "0" or A = B = "1". Secondly, the output condition for A = B resembles 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.

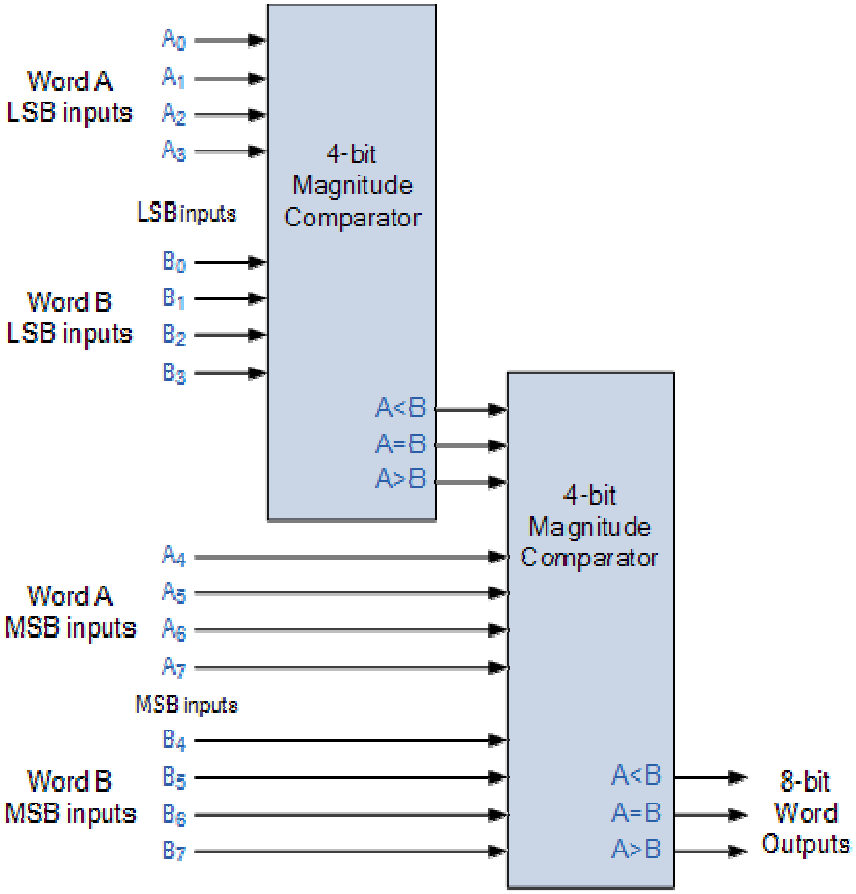
A very good example of this is the 4-bit **Magnitude Comparator**. Here, two 4-bit words ("nibbles") are compared to each other to produce the relevant output with one word connected to inputs A and the other to be compared against connected to input B as shown below.

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.

2.9 Outcome

- **Can create a appropriate truth table from the description of 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 circuits are reduced.**

2.10 Future Readings

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

https://www.youtube.com/watch?v=VnZLRrJYa2I

“Logic Design” by RD Sudhaker Samuel

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