Signals - Signal is the variation in physical parameters.
We can say temperature variation is a signal, video is a signal as the light intensity varies.

Any type of signal need to be processed or need to be stored. We can do any type of processing of the signal in real life.

Let us take an example of microphone. The speech signal given to the microphone need to amplify and then given out to the speaker. So, its typical signal processing scenario.

Basically, signal in the electrical is the variation in the electrical quantity. The voltage and current as a function of the time. Signal is defined as the variation of the electrical quantity slowly a voltage or current with time. Of course, we can have the signal which does not change with time, it can have constant value or D.C. values.

Now once it has the variation, any signal that naturally occurs, such as speech or temperature variation, first we need to convert into the electrical variation. So there is a transducer, a short of a device, this device converts the non-electrical quantity into the electrical quantity. So output of the transducer shows the variation of parameters in electrical domain.

Now Signal can take any value within the given limits and at different instants of time varies continuously.

We can have the time variation of the signal.
So, any number of values, the signal can take as a function of time. But of course, there is a limit 0 to \(V_{\text{max}}\).

Two values are defined and we can say signal values in between them. This is the range of the signal. At any instant of time, it can take any value. This signal can take any value within the given range.

This type of signal is called as Analog Signal. Hence Analog Signal can take any value at any instant of time within the given interval of the signal range.

But some time we need to process the signal at discrete intervals of time. Suppose, I want to measure the signal every minute, or hour. Hence we discrete the time. Hence, it is clear that signal values at the time continuously, but we are taking the signal at discrete instants of times.

Such type of signal:

![Discrete time signal diagram](image)

So, I monitor the value of signal \(V(t)\) at instant of time \(t_0, t_1, t_2, t_3, t_4\) as long as I am interested in the parameters. This type of signal is called as Discrete time signal.

But what is the problem in monitoring these signals? When we monitor Continuous Signal, but what is continuous at least it is discrete. Even if I monitor with 1 Hz, then what is in between 1 Hz.

Physical parameters, however, can be an observer. The observation only can be discrete.
But whether the signal is continuous or discrete, there is no defined point where it can vary. It can acquire any value.

Suppose, I am defining the range as 0 to 5 volts as an example. I am allowing the signal only to vary from 0 to 5v. Within 0 to 5v it can take any value.

But in that case, we can have the difficulty in representing the numbers in a calculator or how can we represent any number after decimal beyond a certain number to be meaningful.

But theoretically, it can take infinite numbers of value between 0v and 5v. Now, when you are trying to transmitt any signal. Because the signal is having no meaning. If I can not process it.

So, as we speak, it is converted into electrical signal. Transmitted, amplified, and given to the converter and finally converted into speech signal. This is communication process. Basic purpose is to use the signal. Whether we can transmit it or store it.

Some time in order to compare the parameters, we need to store it. I.e. I take the signals, how accurately to store it. I.e. I take the signal between 0 and 5 volt because I can take the signal between 0 and 5 volt because we have infinite numbers of values. I may not limit by the channels. How can I be accurate while processing the signal. When I use it and process it and store it, I can be limited by the number of processed digit that is used.

Within the limit I can define various level that we can take. Actually, there is change in so many levels.

So hence, we want to introduce one more discretization. I only take the signal at specific levels. Specific numbers of discrete levels. So, process of recording at discrete levels. When we transmit it, we try to transmit at discrete levels.

We can have two levels or four levels.
On the basis of range of the signals I can decide choose the levels of the signals. So, more and more levels introduce, increases the accuracy and in processing and transporting the signals. But, still in discrete level.

So, if I can discretize in time and amplitude domain, then it is called as "Digital signal". That means

\[ v(t) \]

No. of steps is decided by the size of each step and maximum allowable range of the signal. In our example we always tries to round off the signal to the nearest lower level.

Each of this level can be store or represented by smaller numbers of digits. Then, suppose I have infinite numbers of levels, I have required infinite numbers of digits to represent. Now, each level is going to have discrete level.

But we require fewer level to represent and transmit and store for the representation.

Suppose, total I have 8 steps, between 0 and 5V. As I don't expect much variation in the temperature. So, 8 level can be represented by 8 different values.

Rather than.

If we are trying to more accurate and we can transmit or store more numbers of levels, then we can increase the number of levels.

In case of analog domain, we can only interpret that data is in between some range. Hence we can guess. But if we want to be more accurate then we have to make the large numbers of subdivisions.
Inaccuracy also generates in analog system because of the observation power and inaccuracy of instruments. Now in digital domain, we have approximated the signal to within a particular range. 

Why do we need to go for a digital system? Because in order to improve the accuracy of the analog system, we will have to use precise meter with the large range.

Like this, I can also improve the accuracy of the digital system by introducing more and more number of levels. So, that I can introduce more digits. But why digital?

**Why Digital?**

All the signals are analog. During processing of the signal, we need to convert it in electrical domain, processed it and finally, again converted into speech signal. So, why was required to introduce the digital signal, analog signal can also be more accurate by spending more and more money and digital signals also can be improved by introducing more and more numbers of levels.

Digital signals are more easier to store and manipulate without any error.

Suppose I am storing an analog value, we can store the voltage using capacitor. Suppose I charge the capacitor to 1.27 volts today. Tomorrow, somebody comes and measures it: it and finds 1.25 volts. Hence, there is always a leakage. Suppose I want to transmit this information across the radio. To the other end, it receives 1.25 volts across the radio. To the other end, it receives 1.25 volts. Hence, analog domain introduces more and more uncertainty in storing and processing the information.

Suppose I have 8 levels between 0 and 5 volts. So, I can store any value lying between 0 to 0.625 0.5

\[
\frac{5}{8} = 0.625. 
\]

Hence any voltage lying between 0.5 to 0.625 volts can transmit the level '0'. If it is in between 0.625 to 1.25 volts, then it is level '1'.

Suppose, I am sending the signal 0.75 and by some error, the receiver receives 0.8. Then still, it can be recorded as 1. The effect of noise in digital transmission is much lower than analog communication.
It is not perfect transmission, but it is better than analog in noise performance. It can store and reproduce the data more reliably. Why it is not accurately, accuracy depends upon the number of levels used. Digital Electronics provides the most reliable storing and transmitting of data.

Second reason is accuracy, as we said that accuracy can be improved by introducing more number of levels. In analog we can choose the devices that is more sensitive and devices which can be highly calibrated. But choosing more number of levels and high sensitive devices are two different things. To improve the accuracy, it is easier to increase the number of levels. So, reproduction of number of level is same effort. Analog accuracy improvement is more difficult in comparison to digital accuracy improvements.

The quality of the signal includes storing, measurement, and transmitting. The quality improves in the digital domain.

Now can I process everything in digital...

Why do we need analog?

All real life signals are analog signal. Whether it is temperature variation of a body, speech, light intensity, cost of the digital becoming lower and lower. So, first of all we peak up the analog signal by transducer, then before so it converting into digital we need to amplify the signals.

Analog Signal

\[\text{Preprocess (if required)}\]
\[\text{Convert to digital}\]
\[\text{Process}\]
\[\text{Convert back to analog}\]
Building Blocks of Digital Circuit and System

- Analysis
- Use and
- Design

Basically two types of circuits comes in this categories

- Combinational circuits
- Sequential circuits

Lecture 02

Most of the real life system having analog input.

The block diagram of Digital System:

As I have digital output, we require some filtering.

What is the digital system and what are the components of the digital system.

Example of Digital System:
1) Computer
2) Calculator
3) Digital Clock
4) Digital Instruments

All all types of signal goes through the digital processing.

- Control System
- Audio Video Processing

Depending on the complexity of the system depends upon its application.
Any digital system can be designed because of its building block.

Let us consider the components:

(i) CPU (Central Processing Unit) → Control logic Register

(ii) Memory

(iii) Input/output unit

Now, in the same manner, I can divide any digital system. Now, I can further divide the CPU unit: as Arithmetic logic unit. Similarly, I can further divide the Arithmetic logic unit as adder, subtractor, multiples, and so on. So the digital system can be described from top to bottom in the simple form of a block diagram.

In other words, we can divide the digital system into smaller and smaller subsystems.

System → Subsystems → Modules → Functional block

Gate level implementation

Actual circuits

(Active and passive elements)

We need to design these things as an electronics engineer. Digital System is a central processing unit in which desired task is performed. I need to know its central programming.

In digital systems, as we have discussed that a variable Information is defined at discrete interval. Suppose I want to define the system, in which the min value and max value is 0V and 5V. In case of analog system, the information can take value between 0V and 5V continuously. But in digital system, only certain discrete level are allowed to take inside the system.

But the question is: How can I make the system more accurate?
Let us considering a grade system. In digital only two systems are allowed. So, in order to improve the resolution, I need to introduce more number of levels. Since only two levels of data is transmitted, thus the system is more reliable. Suppose I want to improve accuracy; I can improve the data levels. So, if I have more numbers of data to represent the information, then I can represent it more accurately.

**Number Representation**

**Assumption:**

(i) We work with voltages
(ii) The voltage levels are 0 and 5 V

I can have two condition by applying the switches. In order to represent more level, I can use more.

"0" → 0V OFF (Absence of flag)

Symbol: "1" → 5V ON (Presence of flag)

**Four levels:**

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>OFF</th>
<th>OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>OFF</td>
<td>ON</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>ON</td>
<td>OFF</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>ON</td>
<td>ON</td>
</tr>
</tbody>
</table>

Now increasing the switches, I can represent the four levels.

Now we can map 0 → 5 V.

0V, 5V → In 1 Switch

0V, 1.25, 2.5, 3.75, 2

Hence by increasing the number of switches I can able to represent the data more accurately. So, if I have 10 switches then I can have 2^10 levels.

This is called as **Binary number system**.
Hence we can represent the data more accurately by increasing the number of bits, but still it is more discrete.

Lecture - 03

Since we have only two levels in binary system, so for more accurate representation of information or data, we require more levels and in turn we require more number of bits. So, any signal level can be represented by several number of binary bits.

Number of bits represents the precision or number of level of accuracy by which we want to represent the number.

As we have discussed that analog is the real world signal.

A to D conversion

Analogue signal

Multiple line (bits)

Since single line going represents the single variation or value between 0 and 5 in our taken example. But in case of digital.

Normally, 0 and 5V are conventional value.

Switches are electronic switches to represent the 0V and 5V. Today, due to advancement of technology, we
Operating at lower voltage so that we can consume less power \((V^2/R)\). Today we can talk about over 12V.

In some technologies where we require enough margin between "0" level and "1" level, then we require higher voltage also.

**Positive Logic**

\[
\begin{align*}
0 & \rightarrow 0V \\
1 & \rightarrow 5V \text{ (or any other nominal value)} \\
\end{align*}
\]

"1" level indicates the value of current in voltage, "0" level indicates the absence of signal represented by higher voltage.

**Negative Logic**

\[
\begin{align*}
0 & \rightarrow 5V \\
1 & \rightarrow 0V \\
\end{align*}
\]

"1" level value is < "0" level.

**Types of Functional Block (Logic)**

Basic of the circuits is logic gates because it operates on some logical concepts. Any complex circuits can be break it down to the logical gates.

- two types of logic function
  - Combinational
  - Sequential

In any practical Digital System, there will be Combinational and Sequential System. Sequential logic will have definitely Combinational logic, we will see why.

Combinational functional block is the functional block, both input and output. Output of the block depends upon the input at that time.
Output depends only on the current input. It cannot store the history of the circuit.

We can say I have set of switches and the particular combination of switches makes the light glow. So by changing the input, I can analyse the change in the output, without worrying what it was earlier.

On the other hand, the sequential logic will have

In \(\text{Seq Logic}\) \rightarrow \text{Out (Memory Element)}

In this output not only depends upon the current input but also depends upon the past input and output of the circuits.

So, output depends on the current inputs and past output. E.g. Counter (the process depends upon the earlier value). So, the Counter has to be a sequential value. Similarly, the microprocessor is also a sequential logic.

B: Basic Combinational Building Block

Gate + Gate is a logical function. It decides an output based on the input condition under desired functionality.

<table>
<thead>
<tr>
<th>Gates</th>
<th>AND</th>
<th>$F = A \cdot B = AB$</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Logical relationship</td>
<td>$F$ is true if both $A$ and $B$ are true.</td>
</tr>
</tbody>
</table>
|       |     |                      | $\begin{array}{c|c|c|}
| A & B & F \\
|----|----|----|
| 0 & 0 & 0  \\
| 0 & 1 & 0  \\
| 1 & 0 & 0  \\
| 1 & 1 & 1  \\
| \end{array}$ | $\Rightarrow$ Truth Table |
Any logic can be implemented using three logic gates AND, OR and NOT. But, it may not be optimum efficient. Because for any device size, energy and cost, speed, power consumption is important.

The two more important gates are NAND, \( F = \overline{A \cdot B} \) (Not \( A \) and \( B \)), and NOR, \( F = \overline{(A + B)} \).

These are the basic gates. But NAND and NOR gates are used for the construction of logic gates with small size, low power, high performance and low cost.

2) Exclusive-OR gate

\[ F = A \oplus B \]

3) Exclusive-NOR gate

\[ F = \overline{A \oplus B} \]

Any digital system can be designed using NAND and NOR logic. So, it is called as universal gate. AND, OR, NOT are fundamental gate because they require to make any operation system in digital processing.
Implementation of universal gates (NAND)

Inverter using NAND gate

\[
\begin{array}{c|c|c}
A & B & F \\
\hline
0 & 0 & 1 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
\end{array}
\]

\{ Inverter Performance \}

AND gate

\[
\begin{array}{c}
A \\
\hline
0 \\
1 \\
\end{array}
, \quad \begin{array}{c}
B \\
\hline
0 \\
1 \\
\end{array}
\begin{array}{c}
A \cdot B \\
\hline
0 \\
0 \\
0 \\
1 \\
\end{array}
\]

OR gate

\[
\begin{array}{c}
A \\
\hline
0 \\
1 \\
\end{array}
= \begin{array}{c}
\overline{A} \\
\hline
1 \\
0 \\
\end{array}
\begin{array}{c}
B \\
\hline
0 \\
1 \\
\end{array}
\begin{array}{c}
\overline{AB} \\
\hline
1 \\
0 \\
0 \\
0 \\
\end{array}
\]

It is always good to form a complex circuit with a very good elements and same types of element.

NOR gate

\[ (i) \text{ Inverter} = \begin{array}{c|c}
A & F \\
\hline
0 & 1 \\
1 & 0 \\
\end{array} \]

\[ (ii) \text{ AND gate} = \begin{array}{c}
A \\
\hline
0 \\
1 \\
\end{array}
\begin{array}{c}
\overline{A} \\
\hline
1 \\
0 \\
\end{array}
\begin{array}{c}
B \\
\hline
0 \\
1 \\
\end{array}
\begin{array}{c}
\overline{A \cdot B} \\
\hline
1 \\
0 \\
0 \\
0 \\
\end{array} = \begin{array}{c}
\overline{A} \cdot \overline{B} \\
\hline
1 \\
0 \\
0 \\
0 \\
\end{array} \]

\[ (iii) \text{ OR gate} = \begin{array}{c}
A \\
\hline
0 \\
1 \\
\end{array}
\begin{array}{c}
\overline{B} \\
\hline
1 \\
0 \\
\end{array}
\begin{array}{c}
\overline{A \cdot B} \\
\hline
1 \\
0 \\
0 \\
0 \\
\end{array} = \begin{array}{c}
A + B \\
\hline
1 \\
0 \\
1 \\
1 \\
\end{array} \]

These two gives the flexibility in designing the various circuits with the same building blocks. Multiple signals are used to represent the simplest variation.

There is an Simple method for Analyzing the function using Simple block diagram.
Combinational Circuits

\[ F = (A + B\overline{C}) \]

Digitai
Circuit

Functional Block
diagram

Truth Table

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th></th>
<th>A</th>
<th>B\overline{C}</th>
<th>(A + B\overline{C})</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

It can be event occurring in a controller. So, in practical life situation etc. required the arbitrary logic that should take place due to the presence of certain input conditions.

So we can say: F: Go Home
A = Sunday
B = Holiday
C = Exam.

This logic states that I go home on Sundays, or Holiday and either there is no exam.

Can I get the minimized expression. We require a proper mathematical tool. The function is known as Boolean function or Logical Function.

Boolean Algebra is the Mathematics to Simplify the Complex Boolean function in the Simplest function, and they are performance wise identical.

K-map method provides the graphical solution. So that we can design a particular for minimum possible Hardware Solution.
Truth Table: Truth table is the list of all the possible combinations of the input and corresponding output.

It contain 2^n row, where 'n' is the no. of input. From each of the. If we did not have the Boolean expression, then from truth table we can conclude

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

This Form is known as the Sum of Product form. This particularly called Canonical Sum of Product. Each Product term contains all the input variable. After simplification we can get the minimum of sum of product. Each term is called Minterms.

**Lecture - 05**

As we have taken the example

$$ F = (A + BC) $$

But we are not sure that this is simplest possible circuit, in terms of no. of gates and no. of inputs. For saving the Hardware, it is preferable to apply smaller number of gates and inputs.

There is a simple technique to reduce the circuit in a simpler form and given form.

- Boolean Algebra
- Karnaugh’s Map (Graphical Approach)

Boolean Algebra is based on the logical identity. The same thing is applied in the systematized way in the graphical approach. In complex system we may have number of controllers. In that type of system, graphical approach is very important.
Boolean Algebra

Each row is called a minterm.

We can write the function in terms of sum of products.

Canonical sum of products:

\[ F = \overline{ABC} + \overline{A} \overline{BC} + ABC + ABC + \overline{ABC} \]

Now we can minimise these functions by two methods.

**Boolean Algebra Application**

\[ F = B \overline{C} (A + \overline{A}) + AC (B + \overline{B}) + \overline{A} BC \]

\[ = B \overline{C} + AC + ABC \]

\[ = B \overline{C} + AC (C + \overline{B}) \]

\[ = B \overline{C} + AC + AB \]

**Boolean Identity**

1. \( A + \overline{A} = 1 \)
2. \( A \cdot \overline{A} = 0 \)
3. \( A + AB = A(B + 1) = A \)
4. \( A + 1 = 1 \)
5. \( A \cdot 1 = A \)
6. \( A + AB = (A + B) \)

\[ F = \overline{B} \overline{C} + ABC + A \overline{B} C + \overline{A} BC + ABC \]

\[ = \overline{B} \overline{C} + AB (C + \overline{C}) + AB (C + \overline{C}) \]

\[ = \overline{B} \overline{C} + A \overline{B} + AB \]

\[ = A \overline{B} + \overline{A} BC \]

\[ = A + \overline{A} BC \]

\[ F = A + 1 \overline{B} \]
Demorgan Theorem

(i) \( \overline{AB} = \overline{A} + \overline{B} \)

(ii) \( (A+B) = \overline{A} \overline{B} \)

or we can apply the approach

\[ F = \overline{A} \overline{B} \overline{C} + \overline{A} B \overline{C} + \overline{A} B C \]

Applying demorgan theorem

\[ F = \overline{A} \overline{B} \overline{C} + \overline{A} B \overline{C} + \overline{A} B C \]

\[ F = (A+B+C)(A+B+\overline{C})(A+\overline{B}+\overline{C}) \]

This is known as Product of Sum (POS form) and it is also known as Max term - (Canonical Product of Sum). It can be solved using KAR for Min POS.

\[ F = (A+AB+AC+AB+BC+AC+BC+CD)(A+B+C) \]

\[ = (A+AB+AC+A+B+BC+AC)(A+B+C) \]

\[ = (A+AB+AC)(A+B+C) \]

\[ = (A+B)A+(A+B)(B+C) \]

\[ = (A+B)A+(A+B)(B+C) \]

\[ = A(A+B+B+C) \]

\[ = A(A+B+B+\overline{C}) \]

\[ = A(A+B+B+\overline{C}) \]

\[ = A + A \overline{C} + B \overline{C} = A + \overline{C}(A+B) \]
Map and Lecture - 06

Map Method of Boolean Function

Let us take an example of 3-variables

3 Variable Map (A, B, C). Now we will draw the graph with 8 cells

<table>
<thead>
<tr>
<th>A</th>
<th>BC</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ABC</td>
<td>ABC</td>
<td>ABC</td>
<td>ABC</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>ABC</td>
<td>ABC</td>
<td>ABC</td>
<td>ABC</td>
<td></td>
</tr>
</tbody>
</table>

3-variable map

Advantage of this is that we allow only one variable to change from 0's to 1's or 1's to 0's. Other variable remains same. That means I can use 'A + A' cell of the map. Maps in either horizontal direction or vertical direction is having only one variable change and all other variables remain same. Adjacent cell differ in value of only one variable.

Let us map the Boolean function

F = A + B C

f = \{m_2, m_4, m_6, m_8, m_{10}\}

Given function is

F = \sum m_2, m_4, m_6, m_8, m_{10} = \Sigma m(2, 4, 6, 8, 10)

Since this method solves the complete circuit in the same manner that of Boolean expression, but we have combined all 1's in a proper manner. Hence it provides the efficient solution or most simplified form. In this case, we always try to form largest possible of 1's.
In this method we are more confident about the final result. We can extend the methods for the four variables.

Let us take an example of four variables.

Example:

Minimize using K-Map:

\[ F(PQRS) = \Sigma im(0, 2, 3, 7, 11, 13, 14, 15) \]

```

R S P Q
0 0 1 1 1 1
0 1 1 5 1 6
1 1 1 1 15 1
1 0 8 9 9...
```

\[ F = RS + \overline{P}QS + PQS + PQR \]

Each of this is called as prime implicant. A prime implicant is any group of 1's. Prime implicant is the largest possible group of 1's. For any problem prime implicants are very much essential for solving any problems.

Essential prime implicant: One prime implicant in which at least one 1 is not included in any other group.

Sometimes we can get non-essential prime implicants.
Lecture - 07

The main objective is to identify groups of 1's as large as possible, in order to satisfy the adjacent rule and remove or knock-off as many variables possible.

Let us take few more examples:

Example 1:

\[
\begin{array}{c|c|c|c|c}
  & 00 & 01 & 11 & 10 \\
\hline
  00 & 1 & 0 & 0 & 0 \\
  01 & 0 & 1 & 0 & 0 \\
  11 & 0 & 0 & 1 & 0 \\
  10 & 0 & 0 & 0 & 1 \\
\end{array}
\]

That means \( F = \Sigma m \{2, 6, 7, 9, 13, 15\} \). This is the map.

The idea is to group adjacent 1's as large as possible. We should not consider the smaller group that can be submerged into larger groups of 1's. Make sure that all 1's are covered.

For this example, only 1 out of groups 4 & 2 1's are possible. Most efficiently not covering more than 1's, as long as we can possible.

Hence we can find 3 groups of 2 1's each:

\[
F = A \& B \& D + B \& C \& D + A \& C \& D
\]

Instead of doing this only suppose, marked it slightly differently.

\[
\begin{array}{c|c|c|c|c}
  & 00 & 01 & 11 & 10 \\
\hline
  00 & 0 & 1 & 0 & 0 \\
  01 & 0 & 1 & 0 & 0 \\
  11 & 0 & 0 & 1 & 0 \\
  10 & 0 & 0 & 0 & 1 \\
\end{array}
\]

This example has been introduced to explain the implications. Prime implications, essential prime implications.
Implicant is any group of 1's.

A prime implicant is the largest possible group of 1's.

Essential implicant is the implicant whose at least one member 1's is not covered by prime implicants.

So, in first example, we can see essential prime implicants.

1 - Essential prime implicant
3 - Essential prime implicants because it cannot be covered otherwise.

In second example, all the four group are prime implicants and 3 groups in 1st example are prime implicants. But 1 and 3 are essential prime implicants. III is not an essential prime implicant. I can cover III ones in two other ways. So I is non-EPI.

So non-essential prime implicants can be used for the efficient computation. In this case, of course, we will choose 1st condition for efficient simplification.

Let us consider another example:

\[ Y^* = 00 \quad 01 \quad 11 \quad 10 \]

\[ Y = 00 \quad 01 \quad 11 \quad 10 \]

\[ F = \Pi m(0, 1, 2, 3, 5, 7, 8, 10, 11, 14) \]

Obviously in 1st example group I will be essential prime implicants. III will be essential prime implicants.

Note the prime implicants in which I is EPI, II is EPI, III is EPI. IV and V are non-essential prime implicants.

So we can write the expression as

\[ F = I + \Pi + III + IV \]

\[ = I + I + \Pi + Y \]

So whether we can include a non-EPI in \( F \) that should be decided by the minimum sum of product.
It is a system of four variables

\[ F = \overline{x}z + \overline{w}xz + wy + yz \]

Let us draw the circuit

It requires:
1 - 3 input AND gate
1 - 4 input OR gate
3 - 2 input AND gate
3 - NOT gate

System like a microprocessor in controller are very complex.

What will happen if I have 5 variables?
Let us say, we have 5 variable map.
5 variable K'-map

A, B, C, D, E

A = 0

One way to implement is the draw two map

During mapping we should take consideration that cell no '0' and cell no. 16 are adjacent.

So, we have 5 EPI.

F = I + III + IV + Y

F = \Sigma E + BD + \Sigma DE + \Sigma BCE + \Sigma BCE

Can I extend this concept for 6 variable map
the number of variable is extremely large, then we go for the tabulation method.

Lecture 08

Min terms and Max term is having the central importance. We can represent the function in terms of sum of the product or product of the sum.

Sometimes, if we have a large number of 1’s, the large number of groups of 1’s. That’s they may contain many numbers of prime implicants or essential prime implicants. If the two large 1’s in compare to 0’s. It is possible to get the F instead of F. Finally, we can apply the Demorgane Theorem to obtain the F.

Let us take an example, in which we call use the 0’s function for the logic minimization.

Example: \( F = A + B \bar{C} \)

\( \text{K-Map} \)

\[
\begin{array}{cc|cc}
A \times C & 00 & 01 & 11 & 10 \\
0 & 0 & 0 & 1 & 1 \\
1 & 1 & 0 & 0 & 1 \\
\end{array}
\]

Prime Implicants:

\( \overline{F} = I + II \)

\( \overline{F} = (A \bar{C} + \bar{A}C) \)

\( F = \overline{\overline{F}} = (A \bar{C} + \bar{A}C) = (A + B)(A + \bar{C}) = A + A \bar{C} + B \bar{C} \)

Positive forms:

\( F = A + A \bar{C} + B \bar{C} \)

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

The second thing is:

Depending upon the topology, we can apply any form.
Example 2. 1 Variable Map

\[ F = 2 \{0, 1, 3, 4, 5, 8, 11, 14, 15\} \]

Thus using 1's,

\[ F = \overline{BD} + ABC + \overline{A} \overline{B} \overline{C} \]

min SOP

Using 0's

\[ F = (B + D)(\overline{A} + \overline{B} + \overline{C})(A + \overline{B} + \overline{C}) \]

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

\[ = 0 + o + \overline{A}B \]

\[ F = (\overline{A}B + 0 + 0 + \overline{A}D + \overline{B} + \overline{D} + \overline{C}) \]

Concept of Don't Care Condition

In general, output is asserted with either 1's or 0's.

On the other hand, I may have the system:

\[ A \]
\[ B \]
\[ C \]

Dig System of F.

For certain combination of input, the F will be true and for certain another combination of input F will be false. There will be some combination which can not comes in either categories. For some region there may be some combination can not be defined...in may not occurs in the particular system. Or it can be occurs but it does not makes any sense in the original algorithm. It can be 1's or 0's.
These types of States are called Don't Care State.

Let us take an example:

Don't Care State are used to reduce the logic. It is used to reduce the literals. The Final idea is to reduce the Hardware.

Let us take an example

\[ F = \Sigma m(1, 3, 4, 6, 9, 11, 14, 15) \]
\[ + \Sigma d(2, 6, 9, 18) \]

\( \rightarrow \) Can be replaced by \((X, D, \Phi)\) \( \Phi\) Symbol of Don't Care

We can draw the Map

![Modified Map with Don't Care](image)

\[ F = AB + BC + BD \]

No need to cover all Don't Care.

Basically Logic gates can be used for the Arithmetic operations.
Lecture 09

CODE CONVERTER

In previous lectures we studied about the formulation of and realization of simple and complex logic function.

Design of the Combinational Logic

Code Converter

Code represents data using code. Such as binary representation of any decimal number is a code. Because, we have two numbers only 0's and 1's. But in order to represent the larger decimal numbers, we use same 0's and 1's in different patterns. At it is known as code.

So, the basic purpose is to control the information and it can be controlled using basic functions: gate (AND, OR on INVERTER). In the same way it can be implemented using NAND gate and NOR gate (Universal gate).

Such as in order to processing the special character, we require ASCII code.

ASCII (American Standard Code for Information Interchange). Any representation in binary number, it may be small numbers, large numbers, special characters are code. Even graphics are also example of code. Some times it is used for the simplification purpose. Some times it can be used for the security purpose.

So basically Code Converters are required for the security reasons. Some times it is used for the Hardware Simplification.

The Class of Converters which comes in the Categories of the Hardware Simplification are

- Excess 3 Code
- Gray Code
Excess 3 Code: Excess 3 code is used to convert a set of number into another set of number, which makes easier to complement the number.

For many hardware performances increases due to excess 3 code.

Gray Code: Gray code is used for improving the switching activity. Suppose I have 4 bit no. 0111, T.E (7). Its decimal is 1000, suppose I keep counting, count from 0 - 7, 7 represented by 0111 and next count 8 (1000). Hence all the 0's and 1's value & are changed. This is called as switching. Hence 4 Switches is operated when 1 changes from 7 to 8.

Suppose, 0 I have to change only one switching in order to go from one number to another number, that reduces the switching activity. That will enhance the speed of operation. Switching activity represents the Power Consumption.

Code Conversion are the Standard operation in the digital System. There are many good reason.

Some times it is used for the encryption (Cryptography).
### Decimal to Excess-3 Code

<table>
<thead>
<tr>
<th>$b_3$</th>
<th>$b_2$</th>
<th>$b_1$</th>
<th>$b_0$</th>
<th>$e_3$</th>
<th>$e_2$</th>
<th>$e_1$</th>
<th>$e_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Table for Excess-3 Code

- $b_3$, $b_2$, $b_1$, $b_0$: Decimal digits
- $e_3$, $e_2$, $e_1$, $e_0$: Excess-3 code digits
Now, we have to design a binary to excess-3 converter.

So, I need to convert 0-9 decimal numbers into binary and it should be converted into excess-3 code.

So, actually it is BCD to excess-3 converter. So anything happens after '3' can be considered as a Don't Care Condition. The input is not going to change the output, but we can use this in the minimization of logic.

Now, we have to derive the logic combination for $e_0$, $e_1$, $e_2$, and $e_3$.

So, we have to calculate $e_0$, $e_1$, $e_2$, $e_3 = f(b_0, b_1, b_2, b_3)$.

---

$k$-map for $e_0$

```
00 01 11 10
00 1 1 X X
01 1 X L X X
11 1 X X X
10 1 X X X
```

$e_0 = \overline{b_0}$

---

$k$-map for $e_1$

```
00 01 11 10
00 1 0 1 0
01 1 0 1 0
11 X X X X
10 1 0 1 0
```

$e_1 = \overline{b_1} \overline{b_0} + b_1 b_0$

$e_1 = b_1 \overline{b_0}$
K-map of \( e_2 \):

\[
e_2 = b_0 \overline{b}_2 + b_1 \overline{b}_2 + b_1 b_0 b_2
\]

K-map for \( e_3 \):

\[
e_3 = b_3 + b_0 b_2 + b_1 b_2
\]

Logic gate formation:

\( BCD \) to \( Excess-3 \) code.
Gray Code Converter

Binary to Gray code:

As we know that the Gray code is designed to reduce the switching action. As counting proceeds from one place to another, consecutive no. their corresponding Gray code changes by one bit.

<table>
<thead>
<tr>
<th>b3</th>
<th>b2</th>
<th>b1</th>
<th>b0</th>
<th>d3</th>
<th>d2</th>
<th>d1</th>
<th>d0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Diagram:

- 0110
- 0101
Now K-map for $g_0$

\[
g_0 = \overline{b_1}b_0 + b_1\overline{b_0} = b_1 \oplus b_0
\]

K-map for $g_1$

\[
g_1 = b_2\overline{b_1} + \overline{b_2}b_1 = b_1 \oplus b_2
\]

K-map for $g_2$

\[
g_2 = \overline{b_3}b_2 + b_3\overline{b_2} = b_2 \oplus b_3
\]

K-map for $g_3$

\[
g_3 = b_3
\]
Logic gate minimization

Parity generator

We sent digital data in the form of 0's and 1's. And at the receiving end we try to get 0's and 1's in the same pattern. But, there may be some error. 0's can be converted into 1's and 1's can be converted into 0's.

So, one bit error can be checked by parity. Suppose the numbers of 1's present in a particular stream is called parity.

Suppose I am sending the 4 bit data.

- Data = 1011 (3 ones!) → Odd parity (No. of 1's is odd)
- 1001 → Even parity.

Suppose ASCII code 12 (4 bit)

- Odd parity

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

How to generate a parity bit
Suppose we are sending the groups of 4 bits. Then 5th bit is 8th and which is called parity bit. If the no. of bit is odd, we sent an extra bit '1', and if the no. of bit is even, then we will send 0's representing the no. of bit is even.

Transmitter and receiver must know the protocol, whether it is odd parity bit or even parity bit. But it is used to detect if the Single bit error.

<table>
<thead>
<tr>
<th>b3</th>
<th>b2</th>
<th>b1</th>
<th>b0</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

K-map for P

<table>
<thead>
<tr>
<th>b2 b0</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td></td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\[ P_{odd} = b_0b_1b_2b_3 + \overline{b_0}b_1\overline{b_2}b_3 + \overline{b_1}b_0b_3\overline{b_2} + b_1b_0\overline{b_3}\overline{b_2} \\
+ b_1b_0b_2b_3 + \overline{b_0}b_1b_3b_2 + \overline{b_1}b_0b_3b_2 + b_1b_0b_3b_2 \\
= b_0b_2 \{ b_0 \oplus b_1 \} + \overline{b_0}b_2 \{ b_1 \oplus b_0 \} + b_2b_3 \{ b_1 \oplus b_0 \} \\
+ \overline{b_0}\overline{b_2} \{ b_1 \oplus b_0 \} \\
= (b_0 \oplus b_1) \{ b_0 \oplus b_2 \} + (b_1 \oplus b_0) \{ b_0 \oplus b_2 \} \\
= (b_1 \oplus b_0) (b_0 \oplus b_3) + (b_2 \oplus b_3) \overline{b_0} \overline{b_3} \\
\Rightarrow b_0 \oplus b_1 \oplus b_2 \oplus b_3 \oplus b_1 \oplus b_0 \oplus b_2 \oplus b_3 \]
7 Segment Display Decoder (BCD to 7 Segment)

In this section I want to design such types of system in which input is binary but the output should be the corresponding decimal number.

So 4 bit binary number 0 to 9 has to be displayed as a decimal number. This requires an Hardware.

\[\begin{array}{c}
\text{BCD to} \\
\text{7 Segment display decoder}
\end{array}\]

\[\begin{array}{cccccccc}
\text{b}_3 & \text{b}_2 & \text{b}_1 & \text{b}_0 & \text{a} & \text{b} & \text{c} & \text{d} & \text{e} & \text{f} & \text{g}
\end{array}\]

\[\begin{array}{cccccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\end{array}\]

\(\text{BCD (0-9)}\)
\(\text{(0000 - 1001)}\)
\[ a = b_3 + b_1 \bar{b}_0 \bar{b}_2 \]

\[ d = b_3 + b_2 \bar{b}_1 + b_1 \bar{b}_2 \]
Arithmetic unit have to be control. So far we have used logic gates in order to form the circuits, like parity generators and code converters. But gates are basically logic devices. How can we transform the logical operations into mathematical operations? However, any number need to transfer into binary format (0 and 1). And of course it will give the output in 0's and 1's. Simplesst arithmetic operation is Adder.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>S</th>
<th>A+B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

So, we can locate

\[
S = A \oplus B \\
C = AB
\]

But we required to add larger numbers. Regionally assume that A and B are multibit number.

Let us assume that A and B are multi-bit numbers.

(4-bit) \[ \begin{array}{c}
A = A_3 A_2 A_1 A_0 \\
B = B_3 B_2 B_1 B_0
\end{array} \]

\[ \begin{array}{c}
\Rightarrow C_0, C_1, C_2, C_3, C_4, \text{Output Carry}
\end{array} \]

Since we try to add first bit A_0 and B_0, naturally no carry is generated. Now we can add the first term by the above explained hardware. But if we try to add A_1 and B_1 then we require to add A_1, B_1, and Carry obtained from the A_0 and B_0.
So, this Hardware can add only 2 bit at a time.
Such a Hardware is called Half adder. It does not take account the Carry from the previous addition.
Hence, we need to implement:

![Half Adder Diagram]

\( \text{Cin} \rightarrow \text{Input Carry} \)
\( \text{Cout} \rightarrow \text{Output Carry} \)

The adder that also accounts for the previous event's carry is called Full adder. It is having more practical use.

**Full Adder Truth Table**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Cin</th>
<th>Sum</th>
<th>C0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**K-map for Sum**

\[
\begin{array}{c|cccc}
\text{BCin} & 00 & 01 & 11 & 10 \\
\hline
A & 0 & 1 & 1 & 1 \\
\end{array}
\]

\[
\text{Sum} = ABCin + \overline{A}BCin + ABCin + \overline{A}BCin
\]
\[
= (A \overline{B} + \overline{A}B)Cin + (AB + \overline{A}B)Cin
\]
\[
= (A \oplus B)Cin + (A \oplus B)Cin
\]
\[
= A \oplus B \oplus Cin
\]

**K-map for C0**

\[
\begin{array}{c|cccc}
\text{BCin} & 00 & 01 & 11 & 10 \\
\hline
A & 0 & 1 & 1 & 1 \\
\end{array}
\]

\[
\text{C0} = AB + BCin + ACin
\]

\[
\text{C0} \rightarrow \overline{AB} + Cin (A+B)
\]

\[
\Rightarrow AB +
\]

\[
\text{Sum} \rightarrow A \oplus B \oplus Cin
\]
Hardware Minimization

\[
\begin{array}{c|cccc}
A & 00 & 01 & 11 & 10 \\
\hline
\text{B}_C & 0 & 0 & 1 & 1 \\
\end{array}
\]

\[\text{AB} = \text{A} \times \text{B}_C + \text{A} \times \overline{\text{B}_C} \]

\[\text{C}_{\text{out}} = \text{AB} + \overline{\text{C}_T} (\text{A} \oplus \text{B}_T)\]

4-bit adder

Results can have 5-bit
This is called as 4-bit full adder.

8-bit full adder

\[\text{B}_8 \rightarrow \text{A}_8 \rightarrow \text{A}_7 \]

\[\text{C}_0 \rightarrow \text{S}_4 \rightarrow \text{S}_3 \rightarrow \text{S}_2 \rightarrow \text{S}_1 \rightarrow \text{S}_0 \]

\[\text{B}_8, B_7, B_6, B_5, B_4, B_3, B_2, B_1, B_0, A_8, A_7, A_6, A_5, A_4, A_3, A_2, A_1, A_0 \]

\[\text{C}_{\text{out}} \]
The adder is commercially available. What is the efficiency of these types of logic gates? Speed is one of the important factors for measuring the performances of the devices. Actually, the gates themselves take time. Delay depends upon the type of the gate and one of the input propagation delay of the full adder, considered in the two parts, whether sum comes first or carry comes first. But we are interested in the speed of carry, because it behaves as the input for the next section. As soon as carry comes, the sum of the next section will come and carry go on.

**Lecture-12**

every Hardware has delay. So, when we design a hardware, apply some input, then we can not expect output as soon as the input is given to the Hardware. But, there is a finite time delay for a particular Hardware. This time is called as propagation delay. Of course it also depends upon the length of the signal path. Basically, it is switching activity. So, the total propagation delay depends upon the length of the sub-system and switching activity.

So, when we want to design a fast responding circuit, then improving technology helps.

Now, can we reduce the path of the transmission. Now for a particular technology, we can reduce the path length and cut-off the propagation. Consider the case of Parallel adder, each carry has to go from Stage to Stage.

Naturally, for higher level of circuit, the propagation delay increases from bit to bit and finally, the final output will depend upon the no of each delay.
There are some techniques depending upon the predictions of carry bit, that do not require the occurrence of carry bit from the previous stage. That can help in improving the speed.

We can have some sort of predict or look ahead what type of carry can be generated and then feed those carry, directly in the respective stages. Instead of loading the carry from each stage, feed the carry at each stage simultaneously. Which means, we require some additional logic.

So, more hardware can increase the speed of operation.

**Carry Look Ahead Adder (Speed Improvement)**

Set us take an example of 4 bit adder

\[
\begin{align*}
A &= A_3 A_2 A_1 A_0 \\
B &= B_3 B_2 B_1 B_0 \\
\overline{C} &= C_3 C_2 C_1 C_0 \\
\end{align*}
\]

Set us take the example of input for which carry is '1'.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

When both A and B are ‘1’ we have carry, irrespective of whether C_3 is 1 or 0.

Basic purpose is to understand the carry mechanism.

Either condition is that either A or B is ‘1’, but carry in 1 is always 1.

So, sum rep can be written as

\[
G = A B + (A \oplus B) C_3
\]

\[\downarrow\] carry generate

\[\downarrow\] carry propagate
So AB is Carry generator irrespective the C0 Cin is 0 or 0.

No need to look out these what type of Carry is pushed through Cin.

The only condition is that if A or B either one of them is '1' the Carry will be generated if the input Carry is one.

This condition completely defines the generation of mechanism of Carry for 1 bit.

\[ C_0 = G + P \]

\[ C_i = G_i + P_i C_{i-1} \]

\[ C_0 = G_0 + P_0 \text{ Cin} \]

\[ C_1 = G_1 + P_1 C_0 = G_1 + P_1 (G_0 + P_0 C_{0}) = (G_1 + G_0 P_1 + P_0 C_0 - 1) \]

\[ C_2 = G_2 + P_2 C_1 = G_2 + P_2 (G_1 + G_0 P_1 + P_0 C_0 - 1) \]

\[ C_3 = G_3 + P_3 C_2 = G_3 + P_2 G_1 + G_0 P_1 P_2 + P_0 C_0 - 1 P_2 \]

So, \[ A_i \rightarrow \text{Fo} \quad \text{B}_i 

\[ \rightarrow G_i \]

By implementing the individual Carry in parallel manner to each FA I can improve the Speed.
Hence CLA required extra two level Hardwary. It can increase the speed of the performance at the cost of delay. Sum is available after four stage of gating.

Before CLA the full adder was

So, circuit is modular, it is easy to design. We can pack the circuit nicely in the smaller space. It is called as modularity.

But Carry look ahead adder is more irregular structure of carry. It will cost of course, structural optimization of Space, Packaging etc.
Second thing, for more bit adder, the complexity of the circuit will increase. Can I have any number inputs. Then actually, it depends upon the voltage and current level inside the gate. Actually, if we go below logic level, the logic gates are nothing but the "Transistors", capacitors and resistors. So, the maximum no. of inputs basically depends upon how much loss takes place in the subsequent stages, without the signal level completely dropped to less than the recognised level value.

The number of input that a gate can sustain is called fan in. Number of output that a gate can have is called fan out.

Fan out is the no. of output, that can from a gate to feed similar gate without degradation in the performance of the output. i.e. we can maintain '0' level and '1' level as per the specification.

So, I can not design say 24 bit Carry look ahead adder.

So, there are some advantages and disadvantages of the Carry look ahead adder. First, so we do not need to count for the Carry for previous stages. But disadvantage is non-uniformity and non-modularity.
Lecture - 18

SUBTRACTORS

The other arithmetical operation is Subtractors - and
Multiplication. When we can design in the
Same manner half Subtractor and full Subtractor
When we need to Subtract two bit without Considering
the previous bit, the one have half Subtractor. But
When we have Series of bit. In this case, we
have to take care of borrow in. Similarly we
can have borrow out.

One of the schemes of Subtraction is Complement Scheme
of Subtraction. Actually the basic objective is to
Compact the Hardware. So, if we can design adder
and Subtractor in one unit, that is short of the
desirable.

Subtraction by Complement Number

What is the Complement of number, depends upon the
Number of System. In the Decimal number System, any
Number have base. The 10 Decimal number have base
10, the Binary number have base 2. This is also called
as Radio.

Any general Radio \( r \)

1's Complement of a \( n \)-digit number

\[ = r^{n-1} - N \]

\( r = 10 \) (Decimal System)

\( n = 4 \)

10’s Complement of 4 digit number = \((10^4 - N)\)

For Ex. Example; \( N = 1234 \)

10’s Complement \( N = \underline{10000} - \underline{1234} = 8766 \)

\( r^{n-1} \)'s Complement \& (1's Complement - 1)

For \( N \) digit,

\( (r-1)^n \)’s Complement = \((r^n - 1) - N\)

For \( N = 1234 \), \( r = 10 \), \( n = 4 \)

9/10's Complement 18 = \underline{9999} - \underline{1234} = 8765
Hence 1's Complement is easier to calculate, in comparison of 10's Complement, because it does not involve the borrow operation.

10's Complement = 1's Complement + 1

Actually, Subtraction using Complement method can be done using same hardware used for addition. Hence, it improves the hardware compactness.

But really we are interested in binary number $n$ if $n < 2^n$.

So, we can have Only 2's Complement and 1's Complement.

If $n = 4$ (4 bit number)

$N \Rightarrow 0101$

2's Complement

1's Complement

For 1's Complement, we have written the largest possible numbers for that particular bit.

1's Complement of $N = \begin{array}{c} 1111 \\ 0101 \\ \underline{0100} \\ 1010 \end{array}$

Again do not require any borrow

$
\begin{array}{c} 2's \ Complement + 1010 \\ \underline{1011} \end{array}$

$2^n \Rightarrow \begin{array}{c} 10000 \\ 0101 \\ \underline{0101} \\ 1010 \end{array}$

Subtract Subtraction is same as the adding the Complement of the number.
Subtraction using Two's Complement:

\[ A - B = \overline{A + B} \]

Given:

\[ A = 1100 \]
\[ B = 0101 \]

1's complement of \( B \): 1010

\[ 1100 + 1010 = 10110 \]

2's complement:

\[ 10110 + 1 = 10111 \]

Another case where \( A < B \):

\[ A = 0101 \]
\[ B = 1100 \]

\[ A - B = 0101 - 1100 = 0100 \]

If last position bit is '0' as discarded, represents positive result. But if last position bit is '1' and '0's as discarded, then it is negative number. Then we have to calculate the 2's complement of 0100.

2's complement of \( A - B \): \( A - B = 10110 \),
\[ \overline{10110} = 0111 \]

So, if MBS is 0, then it is positive number, and if MBS is 1, then it is negative number. Need to calculate 2's complement.

Negative number is represented by 2's complement.

\( B \) has MBS 1.

\[ 12 - 5 = +7 \Rightarrow 0111 \]
\[ 5 - 12 = -7 \Rightarrow 1001 \]
Now, how can I convert the given binary numbers in its 2's Complement form using Hardware.

**Subtraction = 2's Complement addition**

\[ A_t \rightarrow \bar{A}_t \]

Then we must have some controlling unit that should decide whether we want to calculate Adder or Subtractor.

\[ S = 1 \]

S = 1 (Subtraction)

\[ \begin{array}{c|c|c|c|c|c} \hline S & A_t & \bar{A}_t & \bar{S} & \text{Inverted for Subtraction will take place.} \\
\hline 1 & 1 & 0 & \text{1} & S = 1 (\bar{A}_t) \\
0 & 1 & 1 & \text{1} & \text{Does as it is} & \text{Addition will take place when } S = 0 \\
0 & 0 & 0 & \text{0} & \text{(A)} \\
\hline \end{array} \]

Let us say 4 bit Complement:

\[ \begin{array}{c|c|c|c|c|c} \hline B_4 & B_3 & B_2 & B_1 & S & \text{Complement} \\
\hline 0 & 0 & 0 & 0 & 0 & S \\
0 & 0 & 0 & 1 & 0 & \bar{S} \\
0 & 0 & 1 & 0 & 1 & \bar{S} \\
0 & 1 & 0 & 0 & 1 & \bar{S} \\
1 & 0 & 0 & 0 & 1 & \bar{S} \\
\hline \end{array} \]
Lecture 14

We discussed Subtraction is nothing but adding 2's Complement. We should be careful about the number of bits working with it.

E.g.: 6 is represented by 111, the MSB is 1, it is negative, i.e.

Suppose I have 4 bit number, and 6 is represented by 111, and 0's MSB is 0, thus positive.

Hence, a particular hardware is designed for specific bit of operation. And we required to fill all the position before performing the arithmetic operation.

4 bit number of representation

\[
\begin{array}{c}
\text{Sign Bit} \\
\text{Magnitude}
\end{array}
\]

It can represent only 3 position

MSB 0 - Positive
1 - Negative

Suppose 1001 - (11) 3

\[
\begin{array}{c}
0 \to 0000 \\
7 \to 0111 \\
-7 = 1001
\end{array}
\]

Hence positive go + 0 to 7
0 to -8

So, if 1 and 1 is Ca

Negative

\[
\begin{array}{c|c|c}
1000 & 1111 \\
1001 & 1110 \\
1010 & 1101 \\
1100 & 1100 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
0111 & 0111 \\
1000 & 0111 \\
1001 & 0111 \\
0110 & 0111 \\
0111 & 0111 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
1000 & 1111 \\
1001 & 1101 \\
1010 & 1100 \\
1100 & 1100 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
0111 & 0111 \\
1000 & 0111 \\
1001 & 0111 \\
0110 & 0111 \\
0111 & 0111 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
1000 & 1111 \\
1001 & 1101 \\
1010 & 1100 \\
1100 & 1100 \\
\end{array}
\]
General 4 bit Adder Subtractor using full adder $A + B$

3 = 0011
-3 = 1100

<table>
<thead>
<tr>
<th>Number</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1001</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>7</td>
<td>1001</td>
</tr>
<tr>
<td>6</td>
<td>1010</td>
</tr>
<tr>
<td>1</td>
<td>0111</td>
</tr>
</tbody>
</table>

$C_{in}$ add extra one in order to obtain the 2's complement.

Now if number is negative $S_3$ will be one, and if the number is positive $S_3$ will be zero. Now in terms of the sign bit $A$ and $B$ can take any value between 0 to 7 and -8 to -1, and the result can be any thing between 0 to 7 and -8 to -1.
If I want the magnitude of obtained result again I need to again pass the obtained output result through the Complementer. That will be decided by $B_0$ bit. If $B_0$ is 1 then that means we need to calculate the 1's Complement and if $B_0$ is 0 the number will pass as it is.

When negative number representation is required, it is necessary to know how many to told no. of bit is required and whether it is Signed bit or not.

**ECD (Binary Coded Decimal)**

Suppose I want to use the decimal number system. Because, the computer deals with binary numbers. So I need to apply the binary. But we are not comfortable about the one above the no. '7' because we are using decimal number system. I want to enter the input in decimal, the Hardware should convert the input decimal into binary and finally after all arithmetic operation, I want to find out the final result in the form of decimal number. By means of decimal.

**ECD Adder**

```
0000 - 1001
  A

000 - 1001
  B
```

But ECD Adder includes Binary Adder. So, I need to convert in the binary form up to 9. No problem.

Suppose the result is $11_{10}(1011)$

Can be LED glow for 10 to 12
No glow for 0 to 9
What can I do to represent BCD as a binary?

Sum

<table>
<thead>
<tr>
<th>Sum</th>
<th>BCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>0100</td>
</tr>
<tr>
<td>1010</td>
<td>1000</td>
</tr>
<tr>
<td>1011</td>
<td>1001</td>
</tr>
</tbody>
</table>

Set the levels of the full truth table for

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0001</td>
</tr>
<tr>
<td>0010</td>
<td>0011</td>
</tr>
<tr>
<td>0100</td>
<td>0101</td>
</tr>
<tr>
<td>0110</td>
<td>0111</td>
</tr>
<tr>
<td>1000</td>
<td>1001</td>
</tr>
<tr>
<td>1010</td>
<td>1011</td>
</tr>
<tr>
<td>1100</td>
<td>1101</td>
</tr>
<tr>
<td>1110</td>
<td>1111</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0100</td>
<td>0101</td>
</tr>
<tr>
<td>0110</td>
<td>0111</td>
</tr>
<tr>
<td>1000</td>
<td>1001</td>
</tr>
<tr>
<td>1010</td>
<td>1011</td>
</tr>
<tr>
<td>1100</td>
<td>1101</td>
</tr>
<tr>
<td>1110</td>
<td>1111</td>
</tr>
</tbody>
</table>
Multiplication is most widely used arithmetic operation. Of course, there are different types of multipliers depending upon speed and hardware. More hardware, faster computation of multiplication, but we need to spend more money.

Basically, the multiplication process involves shift and add method. When we talk about operands, one is called multiplicand and other is called multiplier. Each digit of multiplier is multiplied with multiplicand, to write the result, which is called the partial product, then we take the second digit of multiplier and multiply the same digit, write it down from and multiply the same digit, write it down from the first partial product, but this time we do the first partial product, but this time we do the first partial product, but this time we shift the result (left shift). This is done for all the digits of multiplier. Finally, we add all the series, to get the final product, this is the algorithm used to calculate the multiplication. This algorithm is called as shift and add algorithm. It is the most efficient method used in computers.

Actually, we have one adder. Calculate one partial product. Keep it in the register, then get the second partial product. Shift it and add it using one adder. Then result is again stored in the register. So using one adder, if we have enough storage facility, then store the partial multiplication and final multiplication, then I can compute the final multiplication, without using any other of addition. One after another. But if I have series of adders and all partial fractions is added in one time then multiplication would be much faster.

So one is serial operation, one after the other. Other all partial products are fed into the adder at the same time.
Such types of multipliers are expensive in terms of hardware but faster.

**Array Multipliers**

It is called array multipliers because it requires array of adders. It is used to shift and add algorithm. I have to array put an array with proper position in order to get the final result which is correct.

Let us take an example of 2 4 bits

$$A \rightarrow a_3 \ a_2 \ a_1 \ a_0 \ (\text{multiplier}) \quad (\text{all } a \text{ and } b \text{ are } 1 \text{ or } 0 \text{'s})$$

$$B \rightarrow b_3 \ b_2 \ b_1 \ b_0 \ (\text{multiplier})$$

So for 4-bit multipliers, result can be 8 bits depending upon carry. We can say that even when we multiply two 4-bit array the final result will have maximum sum of numbers of bits of two operands. For this we required array of adder.

$$a_0 b_0, a_1 b_1 \quad \text{it is an result of 'and' gate.}$$

For 4x4 multiplication, 16 product term need to be generated. For this we required 16 'AND' gate.
So we can draw the structure of array multiplier.

Here we can see $a_0 b_0$ directly appears in the form of $P_0$. In second column I need to add $a_1 b_0$ and $a_0 b_1$. I put it an adder. This is an adder. Half adder, because it has only two inputs and no carry. Carry from the second column has to go to the next column. So, the carry travels horizontally to the next column.

In the third column we have to have to add $a_2 b_0$ and $a_1 b_1$ along with the carry obtained in previous column, and that sum add to $a_0 b_2$. So, carry can be generated in any section, and all carry need to transfer in the 4th column.

In the same manner, Sum proceeds in vertical direction and carry in horizontal direction. This is the most elementary form of the multiplier.
This elementary algorithm of 4x4 array multipliers.
The hardware requirement 16 AND gates.
4 Half adders, 8 full adders.
In general, if I want to design mxn array multipliers,
m x n = m+1 \times \text{bits of a},
\quad n+1 \times \text{bits of b}.

mxn AND gates
Half adder \Rightarrow n
Full adder \Rightarrow (nm-2n)
Total adder \Rightarrow n(m-1)

Here m and n need not be same.

No. of Carry, that is need to proceed faster is 1 to 6.
If \( T_c \) \text{ Carry Propagation time},
\( T_a \) \text{ Carry Propagation time of an adder}.
\( T_a \) \text{ AND gate time}.

If \( T_c \geq TC \)
Total add time \( T_a = 8T_c \)
Multiplication time \( T_a + [n(n-1)(n-1)T_c] \)
\( T_c < T_c \)
Multiplication time \( T_a + 3T_c + 3T_c \)
\geq T_a + n(n-1)T_c + (n-1)TC \)
Introduction to Sequential Circuits

Basic S.R. Batch

Until now we have seen the Combinational Circuit. The Combinational circuits are the Circuit in which output only depends upon the present input. On the other hand we have another class of circuit called as Sequential Circuit.

As the name implies there must be some sequencing phenomena. That means output of course depends upon the input, but not only on the input but also the past behaviour of the circuit.

E.g. A Counter, I give the instruction Count, That means you have to start from 1 to what it is right now. Hence, it should count from where it left. Present input along with the past output because, some times we required to study the pattern of behaviour, such as traffic control signal.

For this, the system requires Memory for history has to remember.

So, the Sequential circuit no doubt contains the Combinational Circuit, as soon as history is remembered, it is taken as the input. Hence previous output is also given as the input.
Hence Sequential circuit is nothing but the Combinational circuit with some feedback.
Now One can ask a simple question: how to design Memory in Data Storage System.

The Most Common element to Store the binary data.
It is important to design a system that can be able to store a Single Bit.
That means '0' should be 'Zero' as long as we want and 1 should be 1 as long as we want.

Basic Storage element is called a Latch

\[ R \rightarrow P \times 1 \rightarrow Q \times 1 \rightarrow L \]

Memory is nothing but a feedback concept. It should act as Latch. Output will be always Complementary we will see that.

Suppose, earlier output can acquire any value.

\[
\begin{array}{c|c|c|c}
8 &=& 1, \quad R = 0, \quad Q = 1, \quad \bar{Q} = 0 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
8 &=& 0, \quad R = 0, \quad Q = 1, \quad \bar{Q} = 0 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
8 &=& 0, \quad R = 1, \quad Q = 0, \quad \bar{Q} = 1 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
8 &=& 0, \quad R = 0, \quad Q = 0, \quad \bar{Q} = 1 \\
\end{array}
\]

Note for \( R = 0, \quad \bar{Q} = 0 \), output the Storage element carries the previous information.

So, this condition when \( R = 0, \bar{Q} = 0 \), the output remains unchanged, what it was earlier. This is Memory.

When you want to leave the information, removes this excitation, still retains.

So, I have put the Information 1, 0, 0, and Switched off, still memory retains the Information.
If I want to store the 1-bit information latch is the memory element.

Now what will happen?

\[ \begin{align*}
S = 1, & \quad R = 1, & \quad Q = 0, & \quad \bar{Q} = 0 \\
S = 0, & \quad R = 0, & \quad Q = 0, & \quad \bar{Q} = 1 \\
S = 0, & \quad R = 0, & \quad Q = 1, & \quad \bar{Q} = 0
\end{align*} \]

Not a reliable (Unreliable Operation Combination)

Table of S-R Latch (Characteristics Table) (Nor gate)

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>\bar{Q}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Latch means hold on the data.

Using NAND gate also we can form the latch.

Characteristics table (S-R Latch) with NAND gate

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>\bar{Q}</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Not used</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Memory</td>
</tr>
</tbody>
</table>
Clocked Inputs

S-R Flip-Flops and J-K Flip-Flops

A simple latch can store one bit of information. Today we will make some valuable changes in the latch. One is called a Clocked Latch. One need to apply the pair of proper input, if we want to store the proper bit. When it changes the output changes.

We do not want the system in which stored data changes as the input combinations changes. When we want to retain the data then it should be available.

We must have proper control when the data should be stored. In other words I required a storage mechanism. This control is called as Clock. It is not enough, if you apply the inputs. The inputs are present, if and only if enabling input is present on the clock input.

Let us take a simple NAND Latch.

\[ \text{\begin{array}{c}
S \\
\downarrow \\
D \hspace{1cm} \downarrow \\
R \\
\end{array}} \]

I want a clocking mechanism, that is an input enabling signal. That means when enabling signal present, then only change in input should affect the output, otherwise there should not be any effect in the change in input.
This can be implemented as follows:

If clock is high then S and R gets reflected as S* and R* respectively. If clock is zero, S and R are both 0, and the system will keep storing the data.

<table>
<thead>
<tr>
<th>CLK</th>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>( \bar{Q} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>MEMORY</td>
<td>MEMORY</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Some time this controlling mechanism is essential in order to perform the sequential operation in control mechanism. We can give a very high clock signal. So, speed of mechanism basically depends upon the frequency of the clock signal.

Clock Signal +
The Minimum period of Clock should be later required to process the input. This type of clocked latch is known as Flip-Flop "S-R Flip-Flop". Output changes from Stage to Stage

Now another modification

\[
S=1; \quad R=0; \quad Q=1; \quad \bar{Q}=0
\]

\[
S=0; \quad R=1; \quad Q=0; \quad \bar{Q}=1
\]

Note Clock is used for just storing the data. In order to store the data, we can apply the Clock Signal enable, that according to store the data according to input condition and
To keep the data in the memory we required to remove the clock. This type of Flip-Flop is called D-Flip-Flop and can be simply represented as

![D-Flip-Flop Diagram]

This Flip-Flop is most widely used for data storage.

If I can remove the not-permitted state from S-R latch, then it will be beneficial. In S-R state also there may be some functionality.

Let us analyse the S-R Flip-Flop again.

![S-R Flip-Flop Circuit Diagram]

Thus as long as S=1, R=1 with CLK high, the Q changes to 0, 1, 0, 0, 1, 0, 0, 1 if CLK=High(1) 
R changes to 1, 0, 1, 0, 1, 0, 1, 0 if S=R=1

The output change continues called Flapping. Now it is predictable but not used. It is also called 'RACING' Condition. We can use this condition.

8. The Speed of flapping depends upon the propagation delay.
CLOCK ON TIME < PROP DELAY of FF
Racing condition can be avoided.

If Flip-Flop is made to toggle only once in a Clock period, this type of condition may be usefull.

Lecture 18

Summary of Lecture
- Master-Slave Operation of J-K Flip-Flop
- T Flip-Flop
- Edge Trigger Flip-Flops

Very frequent change in Output with S=1, R=1. Here, toggling is repeated; actually we do not count this type of behavior. RACING not a desirable thing. On the other hand, it permissible if only one toggling takes place in one Clock cycle.

So, we were thinking how to prevent the RACING condition. One way to make the Clock high period much smaller then the propagation delay which are not having the much practical application.

One way I can achieve the thing using the Master Slave Concepts.
Now if I have Condition of two Flip-Flops. This behavior exactly like the feedback flip-flop. This output changes only once in a clock cycle.

As, when the clock is high, the output changes according to the input combination and when the clock remains low, the output does not change, and remains independent of the input combination and preserves the previous output data.

Now in this case when the clock is high for first flip-flop, remains low for second flip-flop.

So, the data will change for second flip-flop, but feedback is given from second flip-flop for which data will not change. So, change due to feedback from the second flip-flop, will only take place during the second clock pulse, not within the one clock pulse. Hence, we are preventing the continuous variation of data by the one clock.
This Flip Flop is called as J-K Flip Flop.

<table>
<thead>
<tr>
<th>CLK</th>
<th>J</th>
<th>K</th>
<th>Qn</th>
<th>Qn+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>MEMORY STATE</td>
<td>Qn+1 = Qn</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>MEMORY STATE</td>
<td>Qn+1 = Qn</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>(Reset)</td>
<td>Qn+1 = 0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>(Set)</td>
<td>Qn+1 = 1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td>Qn+1 = Qn</td>
</tr>
</tbody>
</table>

Sometimes we want only toggling action.

\[ \text{T-Flip Flop} \]

\[ \text{Toggling Flip Flop} \]

**Characteristics Table**

<table>
<thead>
<tr>
<th>CLK</th>
<th>T</th>
<th>Qn</th>
<th>Qn+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>Qn</td>
<td>Qn</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Qn</td>
<td>Qn+1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Qn</td>
<td>Qn+1</td>
</tr>
</tbody>
</table>

**Edge Trigger Technique (Flip Flop)**

Some small modification in the flip-flop circuit can speed up the operation. By this method, the change will stay for the entire duration of clock.

When I said clock is enabling function, when clock is high, flip-flop should work normally, and when clock is low, the flip-flop is in the memory stage. Note: When the clock is high, the output can change several number of times. If I want to avoid this condition, the change in the output should not be level sensitive. It should be sensitive to the transition of the gate function.

So, in order to use non-master slave flip-flop for the output change only once in the clock period.
The Flip-Flop will sample the input during the transition from 0 to 1 and correspondingly reflects the output. Any change in the input will not be recognized until the next transition takes place. Such type of operation is called edge triggering. Hence, it can be classified as positive edge or negative edge.

I can make my Flip-Flop using positive edge alone. Sensitive.

Similarly we make negative edge triggered flip-Flop.

In this case we can avoid the RACING CONDITION.

This requires some extra logic.
Lecture 10

We are required to study the output behavior due to different types of triggering.

Generally, regular clock pulse generated due to the standard circuit contains the duty cycle of 50%. As we know, that duty cycle is the ratio of the on-time period to the total time period. This clock is applied to different types of Scheme with D-Flip-Flop for different triggering mechanism.

First the data is applied to the Master Slave Flip-Flop. As we know that Master Slave Flip-Flop is Level Sensitive. If the clock is high, any change in input will change the output. Similarly, if the clock is low, change in the input will not affect the output. So in Master Slave Flip-Flop, the Master is Simple Level Sensitive Flip-Flop, but, hence, sudden in the output level reflects in Slave. And if the Master is not reflected in Slave. The change takes place only after one clock period. In Other Conditions, it affects the diminish the effect of change in pulse within one clock period.
In positive edge trigger method, the output data remains constant for the whole clock period. Here, we should not change the input when the clock is high, otherwise it may affect the output. In negative edge, the data is only identified when the clock pulse makes the transition from high to low and remains constant until the next clock period. Actually, a Master Slave Flip-Flop works as a negative edge triggered flip-flop.

As we know that D-Flip-Flop is simply used for storage. JK-Flip-Flop is used for Data Storage and it can also be used for the toggling mode.

Is there any use of Toggling of Data using J-K Flip-Flops?

Let us considering a negative edge Flip-Flop in Master Slave Clock.

Now:
So, we can analyze the reduction in clock frequency by factor $2^2$ in increment in the clock period by factor of $2$. So, if I want to increase the clock frequency of the CLk signal, then I can feed the clock in J-K Flip-Flop keeping the flip-flop in the toggling mode. Toggle Flip-Flop is divide by 2 flip-flop.

Now, if I take another flip-flop.

Now suppose I have 4 flip-flops in J-K or negative edge triggered Flip-Flops.
Number of Pulse  |  \( \begin{array}{cccc} R & R & R & R \end{array} \)  |  Decimal Equivalent
---|---|---
0  | 0 0 0 0 | 0
1  | 0 0 0 1 | 1
2  | 0 0 1 0 | 2
3  | 0 0 1 1 | 3
4  | 0 1 0 0 | 4
5  | 0 1 0 1 | 5
6  | 0 1 1 0 | 6
7  | 0 1 1 1 | 7
8  | 1 0 0 0 | 8
9  | 1 0 0 1 | 9
10 | 1 0 1 0 | 10
11 | 1 1 0 0 | 11
12 | 1 1 0 1 | 12
13 | 1 1 1 0 | 13
14 | 1 1 1 1 | 14
15 | 0 0 0 0 | 15
16 | 0 0 0 1 | 16

Hence, the above arrangement counts the number of clock pulses.

So, if we have \( P \) flip-flops then we can have \( 2^P \) flip-flops. That can allow to count 0 to \( (2^P - 1) \). After 15 in case of 4 flip-flops, it will become 0000. So, we can not count after 15.

It is also called Modulo Counter. That means after 16, it will become 0, again in Modulo 24 Counter.

So, we have been two uses of the flip-flop. It divides the frequency of clock signal by the factor 2. Other thing, we can use it as a counter.

Suppose, I want to count the no. of people in this room. Then I can make an arrangement. Suppose like this, that any person comes in the room, the clock pulse goes from 0 to 1. Then by this arrangement we can count the no. of people. It is also called Binary Counter.
we were discussing about the application of Flip-Flops. No. of Counts depends upon the number of
Flip-Flops.
The Counter also can perform Down Counting. Actually, Upcounting and Downcounting is not a very different.

Because.

Let us say its a 3 bit counter. Let us assume
the outputs of 3 flip-flops are Qa Qb Qc

<table>
<thead>
<tr>
<th>Qa</th>
<th>Qb</th>
<th>Qc</th>
<th>Q(do)</th>
<th>Q(b)</th>
<th>Q(c)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

If we take the output Qc instead of Q, then it gives the down counter.

So I can have 0 to 2^3-1 using 3 flip-flops

That is the major use of any flip-flops.

Sometimes we may not count till the last stage possible. Total no. of State in a Counter is
called modulus of counter. Suppose, I want to terminate the count at a number less then the maximum number count.

Suppose I want to terminate at 9 in a 4 bit counter. i.e I want to design decimal Counter
or Suppose, I want to start the count larger then the number 0's. Suppose I want to start from 3. For this extra mechanism 13 required.
In any flip-flop (whether it is J-K flip-flop or D-flip-flop).

Of course, this device is active component, therefore, it requires Vcc and ground. Leaving those 2 inputs, the two more inputs are define, these are called PRESET and CLR.

The signal that activates or effects the circuit in 'low' condition, then it is called as active low Signal. Similarly, that activates the circuit at high condition of signal, then it is called as active high.

**Preset** Present input + \( PST = 0 \), makes \( Q = 1 \), irrespective of J, K, and CK at that time.

As long as preset is equal to 0, output continues to be 1. Such type of input is called override input.

Likewise \( CLR = 0 \), \( Q = 0 \), irrespective of J, K and CK.

This requires some extra logic. So we can make table:

<table>
<thead>
<tr>
<th>PST</th>
<th>CLR</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>X (Should not used)</td>
</tr>
</tbody>
</table>

(Normal functioning of flip-flop) Behavior depends upon J, K, CLR)
Suppose I want to terminate at some intermediate point. This can be done by identifying the particular point, resetting all the data at that point.

Let us take a simple example of a 3-bit counter. Hence it will contain 3 flip-flops. Asynchronous or Ripple Counter

Now let us say I want to 1, 2, 3, 4, 5, and back to ’0’. I do not want to count the 6 and 7. Now, we have to do, we have to detect the 5, and then clear it again. So, all these have Clear and Preset 5 but I do not want the immediate change in the data after 5. I want 5 should stay for 1 clock period.

Assuming that is no delay in clearing the data we should keep for 5. As soon as 6 is reached, it should clear all the data. The reset takes place in very small glitches.
State '6' is given by 110 00

Suppose I want to start at count '8'.

<table>
<thead>
<tr>
<th>Qa</th>
<th>Qb</th>
<th>Qc</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Start

<table>
<thead>
<tr>
<th>Qa</th>
<th>Qb</th>
<th>Qc</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

End

So I want to count 8, 3, 4, 5, 6

So, we can design up-count, down-count and arbitrary designed point. One disadvantage, we feed a clock in very first stage, so closed is passed from flip-flop to flip-flop.
Each of the flip-flop will have some propagation delay. Suppose we are counting 011. Application of second clock, the next count is 100. Hence each digit makes a transition from 0 to 1 with $t' = 0$. It will take a finite amount of time ($\approx 3 \times T_{ph}$) more no. of flip-flops the delay will increase.

Hence, in order to resolve this problem, there is a new concept of Synchronous counters.

**Synchronous Counters**

In this case, the crystal clock signal is given to all the flip-flops.

---

**Section - 22**

**Synchronous Counters and Shift Register**

Basically, delay of the flip-flop decides how fast we can count. Which is not a very desirable thing because as the number of flip-flops increases, we can think of some other concept of counters, in which the processing of counters are independent of the number of intermediate stages. That can happen only if all the flip-flops contain same clock at the same time. Because, any change in output based on input only take place when clock goes from high to low or low to high.

Now a question how to control the count; as all the flip-flops will toggle simultaneously, then of initially all the flip-flop is set 0. Then after one clock pulse all the flip-flop will toggle to 1.

In this case, the flip-flop will make a transition from '0000' to '1111'.

So, what we have to do is the concept of Synchronous Counters.
Set us start with 3 bit counter.

\[ \text{Clock} \]

To count the count:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

C needs to toggle after every clock pulse. So, no problem for first J-K flip-flop.

But in present situation B would also change in each clock pulse, which is not desirable. I want toggling of data after every 2 clock pulse. Now

\\[ \text{Clock} \]

B will become '1' after every alternate pulse. Now either A or C will become '1', then only the QA is allowed to toggle. I can keep the going...
Let us take an example of 4-bit Synchronous Counters.

<table>
<thead>
<tr>
<th>Q3</th>
<th>Q2</th>
<th>Q1</th>
<th>Q0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

4-bit Synchronous Counter

So, when all the previous flip-flop remains 1 only it requires to log 4. Thus the simple concept.
Hence, in this mechanism we have allowed the controllable toggling of flip-flops. Now we can see that change in input stage requires only one propagation delay. Hence, synchronous mode of counting is more useful. Only disadvantage we required some extra hardware.

What are the major use of flip-flops then counting?

Initially, we have discussed that flip-flops can be used as a storage element. We can have the several bits stored at the same time. Such type of combination is called register. The register is nothing but a bunch of flip-flops.

Let us take an example of a 4-bit flip register. It is a combination of 4-flip flop used for data storage.

![Diagram of 4-bit flip-flop register]

Normally, these flip-flop can be controlled by single clock pulse. Now, I can put the information and make the clock pulse '1'. But I don't count, the storage is removed by the next clock pulse. Since next clock pulse will overwrite the data. Hence once stored data will be corrupted. So, I want the control mechanism so that, when I want to store the data, I can store the data.

So, I need a control which is called as load control. So internal circuitry will be such that when the load is active, then at another clock pulse the input data will reach to the output port and then when load is inactive then output will not be changed, even if the input combination changes.
I can say that, load will put the data in. I can have two modes of operation, Synchronous and Asynchronous.

Synchronous load means, the load has to be high and clock needs to be activated. Asynchronous, the Data should be loaded on enabling load signal irrespective of the clock pulse.

Like wise, we can clear the flip-flop using Clear Control. Clear can be Synchronous and Asynchronous.

Clear makes the data zero. So, I want the control so that any time I can remove the data or make the data zero. I can use the clear data immediately all the data will be zero.

Again it can be done with the clock pulse or independent of the clock pulse. Independent of clock pulse, which is called as Asynchronous. But when the clearing of data depends upon the Clear Signal as well the edge of the clock pulse is called as Synchronous clearing of data.

In short, we have registers which have bunches of flip-flops. With several things we can put the data in as long as I want and take it off.
One of the most important uses of Register is called as Shift Register.

In this case after every clock pulse the data will shift to the next output flip-flop.

Initially:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

1st Clock: \( \text{IN}_1 \) 1 0 0 0

2nd Clock: \( \text{IN}_2 \) \( \text{IN}_2 \) 1 0

It will keep shifting the data from one flip-flop to the another flip-flop. Such type of arrangement of flip-flop is called the Shift Registers.

Again I can draw block diagram.

4 bit SR

I can have the option Serial input and Parallel input. So, I can have the Serial out and Parallel output
So I can have 4 different combinations

(i) EISO - Menbar operator (Delay element)

(ii) SIPO

(iii) PIPO

(iv) PIPO

SIPO used for serial to parallel converters. Suppose I am taking the data for ASCII code. Unfortunately, there is a device that can give data one bit at a time. That allows for serial connection, but I can not proceed until ASCII code is built. Then I can store in another SIPO such as telephone modem.

Finally, EISO is similar to the register. Just load it to and take. There is no question of shifting.

In the same manner we can perform the left shift.
Today we will see some important application of shift register. Shift register is often used for other applications.

**Shift Register Application**

Let us take an example of 4-bit shift register.

4-bit shift right 30, R10

In this figure, the output of Ex-OR gate is loaded at the serial input.

Note assume that initial pattern is 1000

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>Initial</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1st Clock</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2nd Clock</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3rd</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4th</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5th</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>6th</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>7th</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8th</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>9th</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10th</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>11th</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>14</td>
</tr>
</tbody>
</table>

After 15 clock pulse original pattern is stored. We should not read 0000. Hence such types of circuit is called **Pseudo Random Noise Generators**. It is called **Pseudo Random Sequence Generators**.
Even all the 15 pattern will arise, but the sequence of pattern depends upon the initial input. Hence pseudo random means semi random that means 15 data will come but each have unknown pattern. It can be used for testing. So pseudo random sequence generator is one of the important application. It is also used for establishing the reliability of the communication system.

Let us consider the another application

![Diagram](image)

Let us say,

<table>
<thead>
<tr>
<th>Initially</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2nd</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3rd</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4th</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5th</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6th</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
So, we can divide the timer period by n times. So,

Period of this clock is n times the period of original clock, where n is number of stages the number of flip-flops. There is always a phase difference of 1/3rd of the clock period.

Clock period = 2n x System Clock period.

There is phase difference of two successive clock clock = 1. Clock period (original) successive clock

It is called a multi-phase clock. So... It is also called Counter. In case of Counter every time the clock of the subsequent stages of flip-flop is divided by

Similar things happen in this case also.
Recirculating Shift Register

Ring Counter

We can take an example of

<diagram>

It is called the Twisted ring Counter. It is also called as Johnson Counter.

If I want only one Clock period at different phases, then this requires the different mechanism.

Suppose I want pulses like this.
Let us considering the previous table.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>(-\bar{A})</th>
<th>(-\bar{B})</th>
<th>(-\bar{C})</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>(\bar{A})</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>(\bar{B})</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>(\bar{C})</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>(A)</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>(B)</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>(C)</td>
</tr>
</tbody>
</table>

Suppose \(E\).

Gates are the basic building block in combinational
Flip-Flops are the basic building blocks in sequential.

Now using this \(E\) can make Second Order of
Complexity. Then using this \(E\) we have to build the
System. Technologically, these are having the same
Complexity. All these are integrated circuits.

Even a gate is an integrated circuit.

A general integrated circuit is any component the which
Active and passive component

SSI \{Small Scale integrated circuit. \(10\) gates in one chip\}
MSI \{10-100\}
LSI \{100-1000\}
VLSI \{>1000\}
ULSI

With this functional complexity, hence all combinational
and sequential building blocks come in the
Category of SSI.

Now in MSI, we will study about mux, demux.
Content of the Lecture

→ Introduction to State Machines
→ Generalized Model for State Machine
→ Difference between Synchronous and Asynchronous Sequential Circuits.

Now, we have been building blocks of Combinational and Sequential logic. Most of the practical circuits are Sequential in nature. Of course, Sequential circuits also have the Combinational parts. Very rarely, we can find the circuits which are Combinational in nature. But, most of the circuits are a Combination of Combinational and Sequential circuits. The Sequential circuit consists of Sequential logic which are designed to store the information. Information need not be a data. Information may be a State. Let us take an example of Counter. Counter is an Sequential Circuit. It counts the input from State to State. A particular Counter has to retain the State. Input comes and goes to next State. Which is the next Count. Then, let us take another example. Traffic controller.

A circuit or system need not be huge. Like a Computer. Digital television or flight simulator. So Simple video game. Counters. It is also systems. So, in order to design a system, we need to know what are various stages. The System permitted to be in, and what gets of input to occur. for a System for transition from State to State. Generally, these are called State Machine.
State Machines

A state machine is a circuit or system that describes the flow of a set of input signals that enable the process to go from one state to another state. And each state also produces an output.

Let us take a simple circuit, in which there are four states:

![State Machine Diagram](image)

This is the circuit of the system we are talking about.

Let us call a single input 'x', which decides what state the circuit goes from the previous stages and it also produces the output 'z'. Both input and output are variables that can get the value of '0' and '1'.

But the condition is if the circuit is in S0, it will remain in S0, even when x = 0. That means if input remains x = 0, then system remains in the state S0.
Now, when \( x=1, g \) goes to state \( S_1 \). Based on the system too, we can define the others possibility.

So, in each of state of the circuit for the look, what out the various possibility of input and what out the various possibility of input and what about the particular happened to the circuit, when the particular input occurs.

This is the concept of state machine. State machine is a machine, of course it is a machine. Which takes the input and gives the output. Actually, the output depends upon the input and the state, in which the machine exists.

Only by deciding the state of the machine, we can decide, what will be the output of the machine for a particular input.

Sometimes the input is not required for changing the output, such as counter counts, we don't need any input except the clock. Except initial next state. The circuit initial input is in "0000". It is also a state machine. As it moves from state to state based on the previous state. But we are sure that next state is determined by the previous state. In order to determine the behavior of the machine, or the system, we need to know, in what state the machine is present now. And what type of input can be accepted from the particular state. That input and present state, determines the output.

Now State machine can also contain the external output depending upon the state; in addition to the state in which it is in. The basic requirement of state-machine is there, should be one state more than the state, in which machine can exist.
Machine can have several states. And the behavior of the machine can be determined from the state to state. This state can be determined not only on the present state or previous state but also the current input.

We can represent the state in the form of the binary variables:

\[ S_0 = '00' \]
\[ S_1 = '01' \]
\[ S_2 = '10' \]
\[ S_3 = '11' \]

Natural Binary Sequence.

So, A, B are state variables. Now when both A and B are zero, then state machine acquires the value S0.

Representation of state machine is called state graph or state diagram.

If state graph represents this system, needs four states. Four state B can be represented by two binary variables. The transition of state can take place depending upon the one clock period. When clock indicates the transition, input determines the transition.

That we need to variables A and B coming out of the system. A and B is required to determine the state in next clock period depending upon the \( S_0 \) input.

Let us now assume, that circuit has reached in the state \( S_0 \). Now, we need to remember the \( S_0 \) for entire clock period. Only then the correct transition can take place.

I need two flip-flop to store the value. The whole scheme contains the two parts. Sequential part remembers the present state and makes it available for the next present state and makes it available for the next present state and makes it available for the next present state and makes it available for the next present state and makes it available for the next present state and makes it available for the next present state and makes it available for the next present state and makes it available for the next present state and makes it available for the next present state.
Let us draw the Counter State graph. Let us consider the 3-bit Counter.

We are required to deduce the State graph from the behavior of any particular digital system. Given the behavior of the circuit, we require to model the graph.

Let us take an example of traffic light controller:

\[ T_{m_1} \quad (M_G, G_R) \quad 1st \quad State \quad 00 \]
\[ T_{m_2} \quad (M_Y, G_R) \quad End \quad 0 \]
\[ T_{m_3} \quad (M_R, G_G) \quad 3rd \quad 0 \]
\[ T_{m_4} \quad (M_R, G_Y) \quad 4th \quad 0 \]

P, Q are State Variables.
depending upon the timer, each state must last for a particular time period. Let us say for every 30 sec it will change. In this case, state machine will be very simple.

State will change automatically in clock period 30 sec. Can have requirement, the main road state duration should be longer in comparison to side road. Then in this case, we can set a timer. Thus if the timer is '0' then it will remain in the present state, and if it will reach the value, then go to the next state.

Hence, we can draw the model.
Sequential Circuits Design using State Graph

We have introduced the concept of State machines. Any digital system can be modeled by a State machine. Unless it is pure combinational machines, the circuits regains the different states and the behavior of the system is characterized by State graph or State diagram. In other words, it describes the condition in which it flows from one state to another state. The State machine should not be very complex. We want to know the process of construction of State machine, then only, we can know about its proper application.

Let us take an example of Simple Counter.

Counter Design using State Graph

Let us take an example of arbitrary Counter. E.g.

I can have the Counter:

\[ \begin{array}{c}
1 & 4 & 3 & 5 & 2 & 6 \\
\end{array} \]

If we want to represent in binary:

\[ \begin{array}{c}
A & C \\
0 & 0 & 0 \\
1 & 0 & 0 \\
1 & 1 & 0 \\
1 & 1 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
1 & 1 & 0 \\
1 & 1 & 0 \\
\end{array} \]

State is defined as condition of a circuit in particular clock period. And next state basically depends upon the present input and State of circuit. So, circuit can remain many state which are nothing but the Condition of Flip-Flops.

For this system we require 3 flip-flops:

A & C = State Variable
I can represent the state graph easily. Since there is no any external input. Every clock period, we are assuming that system will go from one state to the next state. Clock transition triggers the state transition. Hence if the external input is not present then state transition depends upon the previous state.

Similarly, we can have the output different from the state we have decide. Now Counter is an example, in which there is no external input and no external output. State of the system itself behaves as a counter's output.

---

**State diagram of Arbitrary Counter**

---

**COMB Logic**

\[ A^+, B^+, C^+ \rightarrow \text{next value of } A, B, C \]
Combinaional Logic should be designed in such a manner that for a particular input, output should be next number present to count. Let us try to design the combinational logic.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>A⁺</th>
<th>B⁺</th>
<th>C⁺</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>X</td>
</tr>
</tbody>
</table>

This is state table:

\[
A⁺ \rightarrow A \quad B⁺ \rightarrow \overline{A} \\
B⁺ \rightarrow A \quad B⁺ \rightarrow \overline{A} \\
C⁺ \rightarrow A \quad C⁺ \rightarrow \overline{A} \land B⁺ \land \overline{A} \\
\]

It is generated truth table of input-output state.
Like wise we have

\[ \begin{align*}
B & \quad \rightarrow \quad c^+ \\
A & \quad \rightarrow \quad \text{3rd FF} \\
\overline{C} & \quad \rightarrow \quad C
\end{align*} \]

This is the Combinational Logic. This Combinational Logic is also called Gating Logic because the logic alters the flip-flop from State to State. So, Counters are general example of State machine.

Suppose I want to use the J-K Flip-Flops instead of D-Flip Flops. D flip-flops have advantage of being same as D. The next stage output value is same as the input.

But for case of J-K, the output depends upon two inputs J and K.

Now Here the condition is different, Here we can not feed the input directly to the flip. If the next State is \( \text{1} \), then for that particular flip-flop J=1, K=0, Similarly if the next State is \( \text{0} \) the particular flip-flop must have J=0, K=1.
Now, what should be the value of \( J \) and \( K \) for transition from one state to the another state.

\[
\begin{array}{cccccccc}
A & B & C & J_a & K_a & J_b & K_b & J_c & K_c \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
\end{array}
\]

For steering the machine from one state to the another state, we required to fill the proper values of \( J_a, K_a, J_b, K_b, J_c \) and \( K_c \).

So far we have understand the behavior of the \( \Phi \)-flip-flop or \( \Phi \)-flip-flop. So by Characteristic table.

Let us take an example

\[
\begin{array}{ccc}
J & K & C_t \\
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

This is Characteristic table or Behavior table.

Let us consider the Excitation table. It is expanded version of Characteristics. From this table we have 8 to derive the Excitation table. But it can changes to \( \Phi^t \). What should be the particular set of \( J \) and \( K \).
Lecture 36

We are discussing the implementation of a State graph. Using the state table, we need to design the Combinational Logic that can allow the device to move from one state to another state. We can use the flip-flops, so that state transition will occur after one clock pulse. Based on it, any circuit can be implemented.

Let us draw the excitation table of J-K Flip-Flop. Only 4 transitions can take place.

<table>
<thead>
<tr>
<th>Qn</th>
<th>Qn+1</th>
<th>J</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Now map will be simpler:

$J_A = D$

$J_B = A$

$J_C = B$

$K_A = L$

$K_B = \overline{L}$

$K_C = L$

$K_D = (A + C)$
Hence we can design the logic as

Hence, we require only one extra 'OR' gate. This is the proper logic for arbitrary sequence.

Let us taken an example of external input. Suppose we are having 4-State

In this case state table can be filled with 2 inputs: one present state and one extra input. Show my state table will look like.
Present State Variable

<table>
<thead>
<tr>
<th>P</th>
<th>Q</th>
<th>X</th>
<th>P^+</th>
<th>Q^+</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ p^+ = (P\bar{Q}X + \bar{P}QX) \]

\[ q^+ = \bar{Q}X + \bar{P}X + P\bar{Q}X \]

We can draw the circuit diagram.
This example tells us how to formulate the particular problem if some specification is given.
Lecture-26

Mealy and Moore Circuits

We are talking off state graph and its implementation using flip-flops and combinational logic. One thing we are not emphasizing until now, which is the output of state machine. You have discussed that state machine is having a state and the process goes from state to state depending upon the present state and the input condition. We can able to show a graph based on these information. Then we translate the state graph in state table, from which we draw the transition table, from which we draw the combinatorial and combinational and sequential circuits. So, we can able to draw the steering logic, that can able to steer the flip-flop. But, we never talked about output, i.e., because we have taken the simple example of actually counters, whether sequential or non-sequential.

Since counters is not having the extra output, because in state of counters itself acts as an output. But, there are some circuits, in which outputs are other than the state of the system. E.g., traffic controller. It has various states. States. E.g., traffic controller. It has various states: e.g., traffic controller. It has various states. Certain states allow certain light glow. So, there is a need of output, other than the state of a system.

Let us take an example:

Output of State Machine
In this lecture, we will see how to consider these outputs.

There are two types of outputs possible in a state machine. Based on this state machine, it is classified into two types:

→ Mealy Machine
→ Moore Machine

Mealy Machines are one in which outputs are defined for in a state on an input condition.

Suppose, I have a state machine:

```
State Graph
```

![Mealy Machine Graph](image)

Circuit can exist with one output

Note that we define output 'z' for it. But we have not made the provision for showing the output which is different from state 'S0'.

It can be shown in two ways. In Mealy machine, it is part of state and input condition.

But in part of Moore Machine, the output is defined on state, irrespective of the input condition.

In Mealy machine, for each arc we need to define output. So I can say, z=0, the output is zero when the machine is state 'S0' and input 'zero' for that circuit is going to sustain 'S0'. On the other hand, when the system goes from S0 to S1 when x=1, 0/1 may be 0 or 1, again it is required to define...
so long as machine is in State S0. The O/p is 0/1.
So in Mealy Machine each arrow not only defines the state transition but also the O/p based on the Input Condition.
This can be represented by Mealy State table.

<table>
<thead>
<tr>
<th>Present State</th>
<th>Input</th>
<th>Next State</th>
<th>O/p</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>S_0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S_1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>S_2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>S_3</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ Z = A' B + A B' \]

Because for undefined State we don't want the glitches. So in the output section we should take it 0.

We should not use don't care for O/p.

Because for undefined State we don't want the glitches. So in the output section we should take it 0.

This could be additional Hardware to produce O/p along the A' and B'. So, O/p need to be taken into account. This is called Mealy Machine.

But if the output depends upon the State and purely independent of the Input, both types of Machine is called pure Moore Machine.

Let us take an example
It is similar to Counter. The State itself defines the O/p, but it can be defined in other manner. Suppose all counts less than the '5' light to be on, and for all the counts greater then '5' light to be off. It is an extra output depending upon the count.

Let us take the example of 3 State

![State Transition Diagram]

Hence, output only depends upon the state irrespective of the input condition.

Let us draw the Moore state graph:

<table>
<thead>
<tr>
<th>Present State</th>
<th>Input</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

It is easier to design a diagram, because we do not want the glitches.
Moore Map for 'z':

\[
\begin{array}{c|c|c|c|c}
X & 00 & 01 & 11 & 10 \\
\hline
Y & 0 & 1 & 1 & 0 \\
Z & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[Z = AB\]

So, outputs are also important in designing the state machine. State variables only indicate the particular state in which the machine exists but do not indicate the output in that particular stage. Outputs may be independent or dependent on the state variables.

Let us take an example of a state graph:

Now, independently, how to draw a state graph. Suppose I give you specification of the problem. The circuit acquires the particular behavior. Once we know the state graph, that can be easily understood.

E.g. Suppose my purpose is to detect a pattern: data is coming serially. When a particular pattern occurs, I want to know.

Let us design a simple pattern detector.

At present, I don't know the one of state. Hence, we do not know the state variables. But I know that input is coming one bit at a time.

For example:

In each clock pulse:

\[x = 0110\]  \(\text{(say)}\) → \(z = \text{Pattern to be detected}\)

Let say when pattern \(010\) in sequence occurs, I should know it, and it should be indicated by output high.

The output will be '0' normally.

\[z = 0000\]  \(010\)
So, every point should be cleared. Only erased pattern need to be detected. We should always understand what the person wants. Otherwise, there may be some ambiguity.

Let us now assume that overwritten pattern also should be detected. Now we are required to draw the state graph.

Next step is to draw the state graph.

Let us take an example Mealy Machine. I cannot overlapping pattern also need to be detected.

State graph

\[
\begin{align*}
S_0 & \rightarrow \text{Reset State} \\
S_1 & \rightarrow \text{State when changing power on} \\
S_2 & \\
S_3 & \\
S_4 & \\
S_5 & \end{align*}
\]

Hence if first bit is '1' then it should be at 'S0' state because we need to see a pattern as if the entered bit is '0', then it proceeds to 'S1' remembering the state 'O1=0'. Now at 'S1', if upcoming bit is '0' then it will stay on the same state and if goes to '1' then process fails remember 'O1=1'. Now on 'S2' if the entered bit is '0' then process jumps on to remembered state 'S1'. Otherwise, fails go to the state 'S0'.

Now we can define the state:

\[
\begin{align*}
S_0 & \rightarrow \text{00} \\
S_1 & \rightarrow \text{01} \\
S_2 & \rightarrow \text{10} \\
S_3 & \\
S_4 & \\
S_5 & \end{align*}
\]
Examples of Mealy, Moore Pattern detector

We are talking about the Mealy and Moore State machine. We discussed that in case Mealy machine, the output depends upon the present state and input. So, in any case, state more than one flip is possible, because we can apply the more than one input. But on the other hand, in case Moore machine, the flip only depends upon the present state irrespective of any input. Thus in case of Moore machine, a particular may contain only one output. And flip changes only when machine changes its state from one to another.

In this lecture, we will try to draw some more machines. First of all, we need to understand the problem clearly. There should not be any ambiguity or uncertainty. First step is to draw the state graph.

As Designer's job to implement the best state graph depending upon the specific condition. Once the state graph is prepared, the next steps remain very easier.

Let us take an example of another system.

Mealy State Graph

→ Draw a Mealy state graph for a circuit with a serial input 'x' and an output z.

```
     x -->  z
         |     |
         v     v
     Clock
```

This indicates that with every clock pulse, a new bit is sent into the hardware. So outputs and inputs are series of 1's and 0's.

Again I want to generate the pattern detector. It should be one (1). Coinciding with the last bit and the pattern [1011] or [010]. Otherwise it's overlapping of the two pattern is allowed.

Into account.
In this case, consider a set of bit
Sample + \( x = 01100010101010100 \)
\( z = 0000001011010000 \)
We should always start with zero state or reset state.

\[ S_0 \rightarrow \text{Reset} \]
\[ S_1 \rightarrow '0' \]
\[ S_2 \rightarrow '1' \]
\[ S_3 \rightarrow '01' \]
\[ S_4 \rightarrow '10' \]
\[ S_5 \rightarrow '1000' \]

Example of pattern detector.

Like Boolean algebra, similar concepts are possible in reducing the state graph. Converting the state, transform the symbol into binary is the state variable.

**Step of Reduction of State Graph**

- State assignment: \( PQR \)
  - \( P \land R \)
  - \( S_0 \rightarrow 0000 \)
  - \( S_1 \rightarrow 0011 \)

\( S_0 \rightarrow 101 \)

Finally, hardware requires the steering logic that can able to steer the logic from one state to the another state. What is the best possible assignment of \( S_0 \) and \( S_1 \) are possible. So that continuous hardware is required. This is also a problem of interest how to reduce hardware.
There must be some provision of optimally assigning the state rather than random assignment. Can I make an efficient assignment? Hardware, efficient assignment. Can I apply some short of a rule. There are some concepts of adjacent state. Like Karnaugh map, '000' and '001' are like a adjacent set. In this section also, adjacent state are differ by only one variable. Hence, we have to use some adjacency rule. State reduction is more meaningful, if I can reduce the number of flip-flops. If I can introduce more number of don't care in state table, then it will be simpler.

<table>
<thead>
<tr>
<th>Present State</th>
<th>Input</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>R x</td>
<td>0 0</td>
<td>R+ R+ 0 x</td>
<td></td>
</tr>
<tr>
<td>R x</td>
<td>0 1</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>R x</td>
<td>1 0</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>R x</td>
<td>1 1</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>R x</td>
<td>1 0</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>R x</td>
<td>1 1</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>R x</td>
<td>0 1</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>R x</td>
<td>0 0</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>R x</td>
<td>0 1</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>R x</td>
<td>1 0</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>R x</td>
<td>1 1</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

By filling this table, we can design the gate as possible.
Again we will do one more example of Moore Machine state graph.

Example: In this question we will not allow the overlapping pattern. We have to identify the patterns $[010]$ and $[1001]$.

Let us considering the sequence:

$$x = 001011$$

$$z = 000001$$

$$x = 0[010][010][010]$$

$$z = 000001$$

$$70000100$$

---

**Diagram:**

Graph showing states and transitions with labels $S_0, S_1, S_2, S_3, S_4, S_5, S_6$.

- $S_0 \rightarrow$ Reset
- $S_1 \rightarrow 0$
- $S_2 \rightarrow 1$
- $S_3 \rightarrow 01$
- $S_4 \rightarrow 10$
- $S_5 \rightarrow [010]$ (underlined)
- $S_6 \rightarrow 1000$

**Note:** $S_5$ is a state of $S[10]$.
In Section 2.9.1 will be some design, using Medium, medium-scale and Large-scale integration circuits.

First, we will look at the Combinational Circuit, then we will try to include the Sequential part. We have discussed, efficiency of any circuit is measured in terms of time, cost and hardware. So, Gate based structure can be used for small logic and design. But as the operation becomes more complex, we required more logic to design. So, Large no of gates may introduce the reliability problems. External connection can be unreliable in compare to the packaged circuits. Integrated circuits. Now, Silex is the another factor.

So, the large numbers of logic gates are required to put in a package.

Let us take an example, if IC electrode performance is equivalent to the 10 gates. The cost of the circuit is not proportional to the Complexity of the Circuit. Although, the gate level study is essential for understanding and minimizing the problem and reducing the complexity. Hence it provides the functional level to understanding. But, very few IC contain large no. of gates according to the given specification. Similarly lots of power is wasted under switching activity. Hence, there are very less chance of failure for MSI and LSI.
Combinational Logic Design using MSI and LSI

It is always possible to map any logic gate using MSI and LSI.

MSI: Multiplexer is one of the most important MSI circuits. As the name suggests it multiplexes. If I have several inputs and I need to select one of them, then we can use multiplexers. Multiplexers are nothing but a selector.

![Multiplexer Diagram](image)

Hence the function description of the multiplexer circuit is:

\[
F = I_0 \overline{S_0} + I_1 S_0 + I_2 \overline{S_1} + I_3 S_1
\]

Hence, it is just the selection process. Now, what should be the application of this type of circuit? It can be used in local telephone exchange. When I have several signals, and I required to select one of them, then I can use the multiplexer circuit. Of course, this is the combinational logic. This particular combination is called as 4x1 multiplexer. Similarly, I can have 2x1, 16x1, 8x1 mux. We can make larger mux using smaller mux.

Hence this particular function can be implemented as follow
(4x1 MUX using Logic gates)

So I will go towards MUX using logic gates, that could require 1 (4 input OR gate), 4 (3 input AND gate) and 2 (NOR gate).

Again one package of Inverter contains 6 inverters.

So, hence logic level implementation requires 4 ICs.

But this function can be available in one IC.

1 bit Full Adder Using MUX

Let us draw the truth table of full adder

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Cin</th>
<th>Sum (S)</th>
<th>Co</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>L</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>L</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>L</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>L</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>L</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>L</td>
<td>L</td>
</tr>
</tbody>
</table>
Let us draw the Karnaugh Map for this

Now, I have three variable A, B and C. Out of these we require proper selection. We are going to use 4x1 MUX

We know that if A = 0, B = 0 output is connected to I0. So, if I make

Now by using one 4x1 MUX and 1 NOT gate, I can make 'Sum' of full adder
Now let us implement the MUX logic for

Using 4x1 MUX, we make complete full adder.
In previous lectures, we studied about the Multiplexer. Multiplexers are basically 1-of-16 based design. In this lecture, we will analyze some Multiplexer-based design.

Implement $F = A$

$$F(A, B, C, D) = \Sigma m\{1, 2, 5, 6, 9, 12, 13\}$$, using

4x1 Mux.

First of all, we can draw the K-Map for this:

<table>
<thead>
<tr>
<th>AB</th>
<th>CD</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>01</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>01</td>
</tr>
</tbody>
</table>

Now I have 4x1 Mux.

Optimized graph can be represented as follows.
With C-D as selector variable.

In this case too, required only one gate (AND) and one inverter.
Let us use the 8x1 MUX for these applications.
We can make any order of MUX's using lower order Structure in tree structure.

**MUX TREES**

![MUX Diagram](image)

F(A,B,C) Now, it is restricted to use any higher order MUX.

Let us take the same example of Full Adder.

![Full Adder Truth Table](image)

I want to implement this logic using MUX TREES of 2x1 MUX's.

We can take another example of MUX Trees.
Hence, we have numbers of options using variable multiplexers.

Let us take some implement example:

Implement the first example using 2x1 MUXs, and inverters if necessary.

\[ F(A, B, C, D) = \Sigma m \{ 1, 4, 15, 7, 9, 12, 13 \} \]

![Logic circuit diagram]

This one is the implementation of 2x1 MUXs.

So I should able to reduce the given function using series of MUXs.
**Lecture - 30**

**Encoders and Decoders**

Content of this Lecture:
- Encoders and Decoders
- Combinational Design using Decoders
- High-impedance State and Tri-State Logic

In previous section we studied about Multiplexers and its efficient application to implement the logic gates. In this lecture we will study about the next two most important MSI circuits known as

**1. Encoders and Decoders**

As the name suggests, one codes the various combinations into smaller no. of bits and second one expands it back.

Suppose I have 8 inputs. Assuming only one input is active.

Only one input can be active at a given time. Hence at a given time we shall know which bit input is active. Hence, we require only 3 bits.

The combination of 3 bits, we determine which bit is active and all other bits are inactive at a given time. So, this is the process of Encoding.

Encoding reduces the possibility or compressing the data from different possibilities to smaller number of bits.

Sometimes when more than one input is high, then we do not know which one should get the priority. In that case priority Encoder is established. We can make some rule about that.
Suppose if $I_0$ is high and all other inputs are low, the output could be $101$. But if more than one input is high, in that case, we must have some scheme. One scheme is called Priority encoder. It can be the input with higher binary value or lowest binary value, must get the priority. It can be priority (goes from $0$ to $1$). Hence if $I_0$ and $I_7$ are both high, then $I_0$ must code, so $I_7$ must be ignored. Hence we can make the desired selection.

Decoding is the process of reducing larger no. of bits into smaller number of bits.
The complement of this is called Decoder.

Now why this is required? Because I have 8 possibilities of input and 8 possibilities of output, but I don't want to operate both 8 lines. Because, I know only one lane of the line is active all the time.

Hence instead of running 8 lines to know which one is active high, I can have 3 lines, with the coded information, at the receiving end with decoder, the 3 bits, to find which one was exactly high.

This is process of compression of data, because of length requirement, we can do it with three bits.

In case of Decoder, there is no concept of Priority scheme.

Hence it is helpful in designing Combinational logic just like Multiplier. Basically three are gate implemented logic.
Combination logic design using 8-Decoder

It is straightforward if you have 3 inputs and 8 possibilities, then its truth table is equivalent to 3 inputs and 8 outputs. Each of the outputs are being a minterms, and

Suppose, I want to realize the function

$$ F(A, B, C) = \sum m(0, 1, 2, 4, 7) = \sum m(3, 5, 6, 9) $$

![Diagram of 3x8 Decoder]

It is the Heuristic method. Here, too, we required the Hardware availability, and then too we required to use. We are building most of the thing conceptually. In these cases too must be careful of type of LCD, the current levels and voltage thresholds.

It also contains the enable signal. It is just like the preset and clear as the extra signal inside the flip-flop. Here, we can introduce the enable signal.

Enables are usually the active low signal. That means the signals are activated when the low signal is applied. When it is high, it is not activated. That means if the enable signal is kept in ground, then only the circuit works. But if enable is high, the output is not reflected in a truth table. Sometimes, it is having more than enable signal. So

$$ \overline{E_N} = 0 \quad \text{Normal Operation} $$

$$ \overline{E_N} = 1 \quad \text{Output Disable (High Impedance)} $$
High impedance is like output is disconnected from the Input section.

In practical Decoder like 3X8 Decoder generally they contain three Decoder.

For Normal Operation:
- \( G_1 = 1 \), \( G_2A = G_3B = 0 \), Otherwise, 0/1 as 'Z' State.

Then why two required 'z'. I want to control it with different times of users. Hence we can use all the combinations when all the three occur as required. Otherwise output occurs at high impedance 'Z' State.

The outputs are not active high. Output are active low.

That means after enabling properly connected, it particular combination occurs. So, assuming \( G_1 = 1 \), \( G_2A = G_3B = 0 \), by properly connecting the enables this is practical decoder, which is available.

**Tristate gates**

<table>
<thead>
<tr>
<th>Low</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>1</td>
</tr>
<tr>
<td>High Imp</td>
<td>Z</td>
</tr>
</tbody>
</table>

We know \( A \rightarrow F \)

\( EN \rightarrow 0 \)  \( 1 \)

Now we are going to use third concept, either 1 or 0 or 'Z'. Now, how can I make the high impedance.

That requires the involvement of an extra input and the third input is called
This is very important thing developed in Hardware.
The same thing can be operated for any gate. We can give an example with the AND-OR gate.

I want the combination, but in one time I want only one source of data.

Lecture 31

- Programmable Logic Array (PLA)
- Programmable Array Logic
- Programmable Array Logic (PAL) Based design

Programmable Logic Array (PLA).

As we know that as the number of inputs increases, the numbers of minterms also increases, not linearly but in a power of two. The size of the ROM depends on the number of inputs and outputs. As the number of input increases, the size of the devices increases exponentially, and as the number of output increases, the size of the devices increases linearly.

Hence what happens, system appears like a complex. Due to large number of inputs and outputs, the size of ROM becomes very large. Earlier time size was very important because size was responsible for deciding the cost of the device. Now, size also takes lots of space in the ICs. So, the basic challenge was to reduce the size of ROM for larger number of inputs.
But if the number of inputs is large, then there is no way one can reduce it. So we have to generate all the possible minterms, and find out either one of the minterms will combine for each of the outputs. But in practice, it has been seen that any output, depends upon few numbers of minterms.

Then we can say, even though we are generating all the minterms, for a given number of inputs, each of the outputs uses only few of them. Hence, many of the minterms goes unutilised and are unused.

Hence, the concept of Programmable Logic Array comes.

In Programmable logic array, the AND array. In PROM based design the AND array produces all possible minterms, and OR array is used to programme the minterms, in order to obtain the particular outputs. Now, if we make the AND array also programmable, that is, four AND array generates all the possible minterms.

Programmable AND array. Then we can reduce the OR minterms depending on the gain function. Hence, we can generate only those inputs that effect the output.

This is the combination of Programmable logic array.

Question is I need to reduce the given function. In order to generate the required function. In case of ROM, we required to check all possible minterms and combines only those minterms that gives the output. In that case, I do extra Step-up Simplification by K-map and generate only smaller numbers of minterms which are required.
Po, Pi, P2, P3 = f(a, b, c).

From implementation directly goes to the hardware from truth table, but PLA implementation goes to the implementation of logic minimization and reduces the Set of minterms or product terms, which are necessary to implement the required function. The other advantage is, if f1, f2, f3 are common then need not generate two times, one can generate only once. In fact, if we can create efficient designing if more numbers of common terms are available. Then we can share the output or terms.

Let us take simple example

\[ f_1, f_2, f_3 \text{ with } a, b, c \]

\[ f_1(a, b, c) = \Sigma m(0, 2) \]

\[ f_2(a, b, c) = \Sigma m(0, 3, 4) \]

\[ f_3(a, b, c) = \Sigma m(2, 4, 5, 7) \]

If I draw the Karnaugh Map for each one of them

For \( f_1 \):

\[ f_1 = \overline{a} \overline{b} \]

For \( f_2 \):

\[ f_2 = \overline{b}c + \overline{a}bc \]

For \( f_3 \):

\[ f_3 = \overline{a}b + bc \]
In order to implement these logic functions, we require 5 minterms (product terms), but it is generally easier to generate 8 minterms instead of generating 8 minterms and utilizing the minterms for generating $f_1, f_2, f_3, f_4$.

It is simpler to generate the 5 minterms required for $f_1, f_2, f_3$, and $f_4$. Hence, using PLA, the size of the PLA will be smaller.

Now consider the PLA table.

Now, size of the Hardware is smaller. Decoder does the mean level of the combinations of AND array.

Now it is required to add one more function $f_4$.

$$f_4(a,b,c) = \Sigma m(0, 3, 4, 7)$$

Hence $f_1$ and $f_4$ is already generated. To real design, we can have very large numbers of minterms and product terms.

3 inputs 5 product terms
5 outputs
<table>
<thead>
<tr>
<th>P0</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

For a given system inputs are given
P0, P1, P2, P3, P4 are function of 3 variables. Even though you may not use all the 3 variables.

**Programmable Array Logic (PAL)**

What is the need of Programmable Array Logic? Here, for ROM, we have one programmable feature. Once I know the size of the design, no of inputs and no of outputs. One can ask for required size of ROM, and all I know that all the minterms are available. By combining the required minterm, I can get the desired output. Hence, ROM solution is costly and having more area.

PLA gives 2 degrees of freedom. 1st it gives the freedom for AND gate selection and second for OR gate selection. So, there is a concept that gives 1 degree of freedom. Hence, ROM is not good because it is wastage of hardware. The other concept that gives freedom to select AND gate but it contains the constant OR gate.

**Programmable AND ARRAY (PAL)**

Fixed OR array

![Diagram](image-url)
Hence, there is no chance of common selection, we will have to regenerate if. There is no question of sharing.

Let us take the same example I required to generate the following product terms:

\[ f_4 = b \bar{c} + b \bar{c} \]

Here, we can not use \( \bar{b} \) and \( b \). Since [3 inputs - 4 output]

But I need 6 product term \( \bar{c} \) for each output \((2, 2, 1, 2)\)
Lecture - 32

- Introduction to Programmable Logic Devices and Programmability

- Programmable read only memory

- PROM Based Design

In last few lecture, we have seen design of
Combinational logic using MSI (Medium Scale Integrated Circuit). Examples are Multiplexers, Decoders, and Give
To some Hardware requirements. In terms of size of IC it is efficient because MSI requires less space
And provides high performance. There is limitation on
This IC, in the sense of the number of gate circuit
This IC in the sense of the number of gate circuit give. Roughly MSI circuit is defined of the Function given. Roughly MSI circuit is defined of the Function given. Roughly MSI circuit is defined.

Having 10-100 gate function on a single IC chip.
Leaving the circuit, if the Circuit is large, it is not easy to map it. More and more number of Hardware
Mapping of the given circuit is difficult.

Second thing, when we have the circuits, large number
of gate function it is nice to have several usage.
I would like to use for different application and
later on I want to change the application, not a very drastic change. Some of the specification may
Change. In that case, I required to redo it. Take
Another IC and perform the required operation.
So, I want to make a different uses with same
hardware, then this concept is known as the
Programmable Logic Devices.

Programmable Logic Devices (PLDs)

Programmability means same hardware can be used for different functionality by changing the design. In
Computer. Hence by changing the program I can
introduce different functionality using the same
hardware. Such as Microprocessors which are Computer
again. In fact, Computer contains CPU, CPU as an
expensive hardware I can use it for different application
by reprogramme it.
Hence, we can apply the concept of reprogrammability using the less complex circuit application also. It provides the option for reuse of Hardware.

There is an Concept of Firmware comes. Computer is an Hardware, Windows is an Software. Some of the change does not takes place very frequently. In today's era the & can use several Software in multi same Hardware.

Firmware is not the programme. The Programmability is required for the reconfiguring it for certain class of operation. This comes in the System.

Firmware defines the usage, the available resources of the Computer. I can have the Hardware, and I can change the part of it. Firmware defines the usage of the Software on which I can run the Applications.

So, I may have the piece of Hardware, in which I can change the part of it, either replace it by another hardware. That particular part contains the particular hardware that programme and replace it with another hardware that contains another programme. I can use the same hardware, in which I can erase one programme and replace another hardware containing another programme. This type of Programmability is known as Firmware.

PROM (Programmable Read only memory)

As we said that, in this course we are interested in Combinational Logic and Sequential Logic. Only the 'PROM' is helpful in this context. Basically, all the type of design, such as Combinational design and Sequential design, whereas Combinational design is used for Sharing. We have the Truth Table. That is need to be implemented. We simplify it using Karnaugh's Map and obtain the minimum Logic Formula.

Instead, I want to implement the truth table by Hardware, which is known as the "Read only memory". ROM is nothing but the Hardware Truth Table.
If one can etch the truth table on hardware, then it is called "ROM". Because, each of the possible combinations of input, there is a corresponding output. So the ROM will have

```
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
</tr>
</tbody>
</table>
```

I can etch all combinations on ROM present on truth table. I can implement it using K-map and logic gates. Instead of it doing & et, take a piece of hardware call the ROM "Read Only Memory", whose having all O/P sets for corresponding input.

Suppose the input combination is 100, and their corresponding output is 1100. It will appear at required position and it is required to be read-off.

It is Read only memory. I have obtained the truth table and etched it on hardware. So, ROM is just like hardware truth table. This is the combinational logic. A, B, C is input and W X Y Z is the output. In sequential logic or sequential logic function. I can use three input and corresponding value of O/P is etched. It is fixed and I can not change it. Programmability concept is not coming here. So, for each truth table, I required to use new one. But we want to use same hardware, but for the different applications.
I want to make use of programmability. So I have three functions f(x, y, z); here whatever is the input, there are only 8 combinations of output is possible.

What is the difference, for each combination of input the outputs are different for different systems; different truth table different steering table. Instead of making fixed setted process, we can make it programmable & known as 'PROM'. Introducing the programmability we can write the new value every time at particular position for different applications.

For PROM all the output sets can be replaced by 'Don't Care' condition. So that shows the replaceability.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The process of putting different data by change of system is known as programming. The Programme can be written in any language, such as C-language or VHDL etc. In other words it is a Selectable design.

How it is possible to make the programming? Now here it is possible to make the programming. Now it is a two step process. That I can take hardware, that contains 3 inputs and 4 outputs. The fixed truth table is also known as the look-up table. We can only look up the value, we cannot change the value.

Look up the value, we cannot change the value.

Programmability is required to apply the different function of ROM by same hardware of ROM. This type of ROM is called PROM.
How we can use the concept of Programmability using the different logic function.

**PROM Implementation of Logic Function**

First part of the ROM I need to get all the minterms.

Now programmability comes in the second half too. How can I combine the minterms. For different functionality, the different combination of minterms is required.

For different functionality, different logic and different stuffing the minterms combination remains different.

So, I should be able to select the appropriate minterm and combine them by 'OR' gate. So, the second part I have all the minterms and I required to select the appropriate minterm in order to obtain the required output W, X, Y and Z. That consists of 'OR' gate. The number of 'OR' gate depends upon the output. Number of all the minterms should be easily available to all the 'OR' gate and I must have the flexibility to connect or disconnect the particular minterms. Hence OR-Array is programmable.

The dot '.' represents that connection is retained. Each 'OR' gate contains the 'B' number of input. Now, the concept of programmability comes in the picture.

There is some provision, so that we can remove some of the connection and retain some of the connection and reprogramme it for certain other combination.
Suppose I want to change the connection and I find some errors. There are some mistakes in the connection. I can always redo it. So I should be able to erase it and reprogramme it.

This is known as "EPROM". Erasable programmable read-only memory whose programming can be erased and reprogrammed. We can erase it by fine ultra-violet light or electrically erasable.

So we can have ROM, PROM, EPROM, UV-PROM, EEPROM.

Erasing means erasing whole things like slate. UV rays will erase the whole thing. But EEPROM we can selectively alter the connection and reprogramme and recorrect it. Electrically alterable is also known as the Flash Memory. This is all I have programmable solution for the logic function. This requires the ROM programmer.

We can make ROM content table.

<table>
<thead>
<tr>
<th>Address</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C D</td>
<td>W X Y Z</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>1 0 1 1</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>1 0 0 0</td>
</tr>
</tbody>
</table>

This is done by a simple programmer, written in Hexadecimal.
Contents of this lecture

- Design using Multiplexers and Flip-Flops
- Design using Promes and Flip-Flops

We are discussing the MSI and LS1 circuits and implementation of combinational logic using these circuits. We talked about the multiplexers, Decoders and Programmable logic devices, the multiplexers, Deciders and Programmable logic devices.

The multiplexers, Decoders and Programmable Logic devices are Sequential in nature, which is used in circuits. Sequential circuits are Sequential in nature, which is used in sequential circuits.

When we design Circuit or Subsystem or System, it is a Sequential Circuit. Of course, Sequential circuits also contain some part of combinational logic, and Sequential circuits knows the previous output.

Implementation of Sequential circuits using MSI and LS1

The Sequential circuits also contain some part of combinational logic. The combinational part can be implemented using MSI and LS1 logics. Flip-Flops can be implemented for the Sequential part of the circuit.

Hence, it is an Integrated effect. Secondly, we will also see Later on how Sequential circuits such as 4-bit counters can also be used for implementation of practical circuits.

Let us take an example of State Graph.

A Sample State Diagram
Steering logic jumps from state to state from 000 to 100. Let us use D-flip-flop to make the logic implementation.

State table

<table>
<thead>
<tr>
<th>Present State</th>
<th>Input</th>
<th>Next State</th>
<th>O/P</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>G</td>
<td>C</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>x</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>x</td>
</tr>
</tbody>
</table>

Of course A+, b+, c+ will become D-Flip-Flop input.

One method is to simply fill in each one K-map and implement it using logic gates. Now I want to use multiplexers.

Still it is combinational logic. Still we require D-flip-flop. As we know that the multiplexer is a combinational building block. Combinational logic only controls the state transition of the flip-flops. The circuit is going to look like this.
We have designed the State diagram implementation using Multiplexers and D-Skip-Steps. We have used MSI.

Now, I want to use PROM instead of MUX's.

**PROM Based Implementation**

We are going to design a system which consists of 4 input and 4 output. Size (2^4 x 4).

Basically, input in PROM is called the address.
Now let us required to programme the PROM according to the given state table.

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>CONTENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
</tr>
</tbody>
</table>

Similarly we can implement it on PLA.

---

![PLA Diagram]

But we don't know, how many minimum one of minimum can take place.
Lecture 035

→ Design using Programmable Logic Array.
→ Design using Programmable array logic.

We are discussing the implementation of sequential circuits using the 
74161 and 74162 circuits. In case of 
Programmable logic array, the identification of common
functions can be beneficial for the Hardware. We can 
discussed some array for the implementation of Programmable
logic array and a Programmable array logic.

PLA Implementation of Steering Gates

Truth table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

P₀ = A + B + C
P₁ = A⋅B
P₂ = A⋅C
P₃ = B⋅C
P₄ = A⋅B⋅C

Dₐ = A + B + C
D₀ = A⋅B + C
D₀ = A⋅B + C
D₂ = A⋅B + C
D₂ = A⋅B + C
D₃ = A⋅B + C
D₃ = A⋅B + C

We can obtain the formula, implementing the transition
Table in State Table.

Size of the Programmable logic array is 4 input and
4 outputs and 4 product term.
This is the PLA solution.

Note fast PAL solution + 4 inputs, 4 outputs,
Each output OR gate requires 2 inputs
1 Comb. + 3R Outputs.
Lecture - 86

Design using MSI Counters

Design using MSI Shift registers

We have studied how to design combinational logic circuits using MSI and LSJ blocks, such as multiplexers. Besides Programmable logic devices, especially PROM, PLA, etc., we take another example such as 4-bit counters or shift registers. It comes under the category of MSI which is sequential. It is like reducing the number of gates equivalent. So, like reducing the number of gates and make the circuit compact. So, whenever we have sequential circuits such as flip-flops, then their individual IC's are connected together. Will all the problem of interconnectivity and space. All these exist even in sequential circuits. So, in that context, we require even in sequential circuits to minimize the IC's, no of interconnectivity, even for the sequential building blocks. To implement certain logic, we require some circuits such as Counters, Shift registers. A shift register is nothing but the branch of flip-flops. Finally, we can have circuits both some combinational part, which contains some of the IC's and sequential part with some of IC's and combination of these two result in a very compact circuit.

We are talking about the mapping of given function, obtained from its truth table. How to implement the given K-map on the multiplexer's based solution. Now, in this lesson, we will see how to implement given truth table using counters and shift registers.

An arbitrary sequence counter using a 4-bit Counter.

```
0 - 8 - 12 - 4 - 6 - 7 - 11 - 3
```

```
10 - 15 - 14 - 5
```
Use the counters in the following specifications

**4 Bit Counters**

This counter will be negative edge triggered clock.
It contains the load features for parallel in and parallel out. It contains the 4 inputs \( D_A, D_B, D_C \) and \( D_D \) and also contains the 4 output \( Q_A, Q_B, Q_C \) and \( Q_D \).
A particular input combination can be applied causing the load terminal. The values of \( D_A, D_B, D_C \) and \( D_D \) can be stored into the counters. CLR feature clears the counters making them zero. Count feature counts up and down. With every clock pulse, it will go by one down. The count up/down signal is decided by UP/DOWN terminal.
Hence, using this equipment, I can count up, count down, I can load, as well as, I can clear the data.

Let us try to prepare some table known as transition table:

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State</th>
<th>CLR</th>
<th>LOAD</th>
<th>CNT</th>
<th>UP/DOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C D</td>
<td>A' B' C' D'</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Transition Table

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State</th>
<th>CLR</th>
<th>LOAD</th>
<th>CNT</th>
<th>UP/DOWN</th>
<th>DA</th>
<th>DB</th>
<th>DC</th>
<th>DS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABCD</td>
<td>ABCD</td>
<td>1</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>0000</td>
<td>1000</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>0001</td>
<td>0001</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>0010</td>
<td>0010</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>0011</td>
<td>0011</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>0100</td>
<td>0100</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>0101</td>
<td>0101</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>0110</td>
<td>0110</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>0111</td>
<td>0111</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>1000</td>
<td>1000</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>1001</td>
<td>1001</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>1010</td>
<td>1010</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>1011</td>
<td>1011</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>1100</td>
<td>1100</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>1101</td>
<td>1101</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>1110</td>
<td>1110</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>1111</td>
<td>1111</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
</tbody>
</table>

We can go for ROM solution because I can have the PROM solution.

PROM solution for combination logic -
Size = 2^4 x 8 bits

<table>
<thead>
<tr>
<th>Address</th>
<th>CONTENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>C8</td>
</tr>
<tr>
<td>1</td>
<td>00</td>
</tr>
<tr>
<td>2</td>
<td>C4</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>3F</td>
<td>CA</td>
</tr>
</tbody>
</table>
Implementation of Arbitrary Counter with PROM

Gate Based Solution (Traditional Solution)

\[ \text{UP/DOWN} = CNT = 1 \] (Case 1)

\[ \text{CLR} = C + A + B + D \]

\[ \text{LOAD} = \left( \frac{C + \delta + D}{\delta + D} \right) \]
System Design

Traffic Light Signal Controller

We studied about different types of circuit design such as small scale integrated circuit, medium scale integrated circuit, and large scale integrated circuits. In case of combinational logic, starting from the gate level implementation, we also saw how to implement the circuit using multiplexers, and programmable logic devices. We talked about sequential circuits, flip-flops, counters and registers to implement the various combinational logic circuits. We can take various examples such as code converters or pattern recognition. Let us do.

But all these things come in the category of Subsystem. In the digital system design, most of the applications today involves the digital system design, e.g., traffic light controller, controller to control the movement of lifts, or controllers of chemical in chemical plants and controlling the levels of various fluids. It could be a automated electronics. It could be simple home appliances, or washing machine, remote control of T.V. We can have more complex device such as flight control, in aircrafts or flight simulation of a rocket launch. It could be a simple example of computer or calculators. Whatever the system the complexity is to be formidable.

So, when a system is given with clear specification, if it ideally characterizes the input signals and output signal acquire, we must be able to predict the State graph and if we are once able to create the state graph. Of course it may contain the large numbers of state. And we use some techniques like CPLD and FPGA. But conceptually it requires some preferable techniques. Now let us take some example, how it is important to understand the design requirement, e.g., input requirements and output requirement and how it can be made easily and how to build all these things.
Before choosing the component to required to identify the system input and output requirements clearly. This approach is called top-down design approach.

Take a example of computer or calculator design. We can take computer as building block approaches, arithmetic logic unit, memory unit, input & output unit. There are functionally attachment in each one of these. Such as ALU, it can consists of arithmetic and logic building blocks. It can be gate or multiplexer oriented. So, when we go from top to down and down, we will be able to break a system into Subsystem, Subsystem to a functional unit and functional unit can be available in the form of MSI, LSI or SSI. Then we try to analyze the sequence of events using State graph.

Let us take a first example of traffic light controller. It provides the path to the vehicles in some controlled manner. At last we required to control the set timing. We can have the timing based on requirements.

Let us take some example

**Traffic Light Controllers**

Let us take an example of Main Street and Side Street.

![Diagram of traffic light controller](attachment:image.png)
So, we have mainly 6 stages. Now the problem is how a specification can be translated. The Six Stage gives two ideas about drawing the state machine. Inputs are the

1. TY (Yellow time),

2. TMG (Main Street Green time),

3. TSG (Side green time). So, we have 4 lights,

each of 3 combinations. Circuit are in various states,

4. moves from state to state depending on the conditions, that

5. becomes state graph, after that we can implement it.

So, the problem translation, from logic description to

drawing the state graph is the most critical thing.

This is called the Algorithm or procedure to solve a particular problem.

Sequences of events can be represented as follow (Pseudocode)

1 \rightarrow M1G, M2G, MTR, SR
   \text{After TMG}
2 \rightarrow M1G, M2Y, MTR, SR
   \text{After TY}
3 \rightarrow M1G, M2R, MTG, SR
   \text{After TSG}
4 \rightarrow M1Y, M2R, MTY, SR
   \text{After TY}
5 \rightarrow M1R, M2R, MTR, SG
   \text{After TSG}
6 \rightarrow M1R, M2R, MTR, SY
   \text{After TY Go to 1}
Pseudo code is an English description of a particular problem, but not in a particular system. It is also called an algorithm.

Let us draw the State graph. (Moore graph). Moore State graph defines state as an output. In this particular case, states define the outputs.

Hence, there will be a system clock and a counter.

Hence, state graph 6 states 3 variables
4 input 3 state variable
12 outputs

Combination logic

12 light output
Lecture 38

We are discussing the traffic light design system. In order to design a particular system, we require to follow some particular design processes, starting from specification to the implementation. Understanding the problem clearly and turning the condition in terms of signal and hardware is the most important factor.

So, we have six states in this particular state graph.

So, signals are:

\[ M_1 \Rightarrow L \rightarrow R, R, Y, G \]
\[ M_2 \Rightarrow R \rightarrow L, R \]
\[ T \Rightarrow V, T \]
\[ G \Rightarrow A \]
\[ T_{MG} \rightarrow t_{MG} \]
\[ T_{TG} \rightarrow t_{TG} \]
\[ T_{SG} \rightarrow t_{SG} \]
\[ T_{G} \rightarrow T \]

When we are monitoring the time, there must be some hardware for duration. Hence, there must be hardware timer.

Depending upon the number of clock periods, we can define the time duration for different light signals.
Some clock can be used for the flip-flop, then only we can reflect the State Change on the times. Fortunately, these are non-overlapping intervals. Hence we can set the discrete clock pulses for different signals. Start times act as the count enable. GT is also an output.

Let us define the States:

- $S_0 \rightarrow 0000$
- $S_1 \rightarrow 0001$
- $S_2 \rightarrow 0010$
- $S_3 \rightarrow 0011$
- $S_4 \rightarrow 0100$
- $S_5 \rightarrow 0101$
- $S_6 \rightarrow 0110$

<table>
<thead>
<tr>
<th>Present State</th>
<th>Input</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

GT, M1G, M1Y, M1R
### Transition Table for the Particular PROM Implementation

<table>
<thead>
<tr>
<th>Current State</th>
<th>Input</th>
<th>New State</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC</td>
<td></td>
<td>ABC</td>
</tr>
<tr>
<td>000</td>
<td></td>
<td>000</td>
</tr>
<tr>
<td>001</td>
<td>Tmg</td>
<td>001</td>
</tr>
<tr>
<td>010</td>
<td>ty</td>
<td>010</td>
</tr>
<tr>
<td>011</td>
<td>tto</td>
<td>011</td>
</tr>
<tr>
<td>100</td>
<td>ty</td>
<td>100</td>
</tr>
<tr>
<td>101</td>
<td>tsg</td>
<td>101</td>
</tr>
<tr>
<td>110</td>
<td>ty</td>
<td>110</td>
</tr>
<tr>
<td>111</td>
<td></td>
<td>111</td>
</tr>
</tbody>
</table>

**PROM + Inputs + 4**

**Outputs + 16**

27 X 16 bits, each with 3 D-flip-flop

Let us perform the **mux based** implementations.
\[ \text{Partitioning the Problem as Functional Units and the Controller.} \]

A 1's Counter Functional Block

In last lecture we saw the design of traffic light controller. The example helped us to understand the problems with clear specifications, identified the input-output signal required. Then draw an appropriate state graph which satisfies all the requirements and once you have procedure to have an ideal state graph then we have procedure to implement it in hardware level. Here let's try to do one more example.

Basically, there are two types of design. One is Control aspect. One is Control, such as in the traffic light control, the aspect. The control as a most important thing, there is no hardware involved. Control is most important thing, there is no hardware involved except lights and may be a timer. So, we can incorporate the signals required for those into the state graph. And we may have to design according to the availability of the components. Such as we are buying to do a arithmetic logic unit to a computer, we do not go to design arithmetic without a ALU. All the 4 bit ALU we can buy. Then control this ALU using an algorithm.

There is a sequential process in order to understand the given problem. So, controller controls the functional block in accordance with the specifications of the design. But in order to that do that first of all it is very urgent to know that, what are the signals available. From these functional unit, and signals available from the functional unit as output. Then we can incorporate our state graph state of the machine in which it exist based on the sequence it go through. On the other hand, we also have extra input and output that define the state. So, what should be the signals that decide the addition, subtraction and multiplication signals for 4 bit of ALU. What should be the signal given to it for subtraction possible. Similarly, the output of the functional unit like the operation completed. Suppose we are doing the addition with the carry. If the carry is present
Proceed and if the Carry is not available then Step 1 of for example. Hence we required to introduce the input signal to the State graph. Carry in now acts the input signal to State graph. Hence input of the State graph can be the output of A.L.O. Input. Output of State graph can be input of A.L.O. So we design a particular. For complex circuit we required to introduce 4 bit A.L.O. On 2. We may have the memory unit.

In many practical circuit we required to separate out the functional units and control units. This is an approach of top-down design. We must have the overview what we want to do. The basic unit need to be logic gates. It can be like full adder. The function to logic gates. It can be 4 bit full adder. The function of logic gate concept is the important aspect of unit and control unit. Concept is the important aspect of unit and control unit. Concept is the important aspect of unit and control unit. Concept is the important aspect. Sometimes when designing a particular digital system sometimes, when designing becomes complex. Such as 210 in a computer, it controllers where many features need to be control.

Top-down design

It work in two level. In 1st level we define the problem completely. Overview the problem. Give complete set of input output signals. Overview the functional unit and control unit availability and how are they interconnected. Interconnected unit availability and how are they interconnected? In terms of the signals and then draw the State graph. Hence, this is the top-down design sequence on higher level.

- Control unit
- Functional units

It can be trivial or complex unit. But it is possible to partition the system in terms of control unit and functional unit.
If it is trivial we required to partition, but if it is non-trivial we required to partition it. Suppose I have a multiplier that includes the shift and add algorithm. Before add it we required to shift it. So, we required feedback from the functional unit to the control unit. Before knowing the previous state output, we can not perform the next step. Hence, I required the status signal from the functional unit to be back to the control unit. These signals are called status signal. Several status signals are things to do. According to the status signal, the controller gives the command to do the next operation. The functional unit only perform the operation according to given command, but the functional unit requires data in order to perform particular operation. In addition to this functional unit requires data input and data output. Likewise controller unit requires data input and data output. Sometimes we required to restart some unit as well. Some operation which are not related to functional operation of functional unit, the control unit may give the command that operation is completed. It can have the external outputs. This is the generalized model of the top-down design system. But there is no regiment in the partition. Some time lines can be inside the control unit. It is a soft partition and we can move the thing in between them. We have studied many functional units separately. Control unit we have studied in state diagram. If functional unit does not perform any specific operation then it can be the part of control unit. But the reverse is not true.
For a particular problem first of all we should know the functionality of particular operation. The input and output requirement for each functionality.

Suppose we require the control signal that provides shift and add from the controller, then we draw the state graph for multiplier. I will say, shift and add. So, shifting is not a problem, i.e., we can provide the control signal to the shift and adder, and it will shift but if the control signal is not given it will not shift. But add a four-bit adder, so there any control signal called adder. So, how can I use the signal, sometime, it is a problem. How can I use control the adder, and use it when I want to add. In that combination circuit, the combinational circuit delay remains in nano-second. As soon as input A and B given it should calculate the result. It should not wait for separate control signal. Hence 4-bit adder can fit into registers and output register can be control. I can load it when I want to add it.

Suppose I have a 4-bit full adder.

Because we can not give the command signal on 4-bit full adder hence we can use 5-bit shift register and we can provide the command signal to the load terminal.
Many times we required to perform some operation on manipulation. We required to find out number of ones which is called as parity detector. If it is odd, called odd parity and if it is even it is called as even parity. Suppose, before paritygenerates we required to generate the number of '1's. Suppose I have a bit sequence and I want to count the number of ones in that particular bit sequence. This will become the part of the bigger unit. In local unit if it is necessary to count the total number of '1's in a particular register.

To Count the number of '1's in the data stored in Register A and store the result in Register B.

In this case, what are the function unit, of course Register A and Register B. Then I required to do a state graph and how to load the thing. How can I do it and how to load the thing. How can I do it and the result came, then how can I do it and store the data.

Once the Control unit state graph, then we can do it and implement it using multiplexer. Which makes PAL.

1st part is to decide the function unit.

The size of register basically depends upon the number of data bit given in particular data sequence. Hence I require the parallel in and parallel out that can able to hold this parallel in and parallel out that can able to hold the 'n' bit data defined by the customer. And what should be the size of this register. For 'n' bit number the size should be \((\log_2 n + 1)\).

Since 'n' is register, that can able to store the data, hence it must have the loading feature. Now we required to store the data in a generate algorithm. How can I find the data in a particular bit stream. Look at the LSB, if it is one add to the shift register and shift the data and when the flag is over, then I can do the controlling algorithm.
Lecture - 40

We have to load the data inside one register and count the no. of 1's in particular sequence of data and put the value of N, counting the no. of 1's in another register and stop the operation. This is a simple system we have talked about the simple functional unit requirement. Of course we need two registers, register A and register B. Partitioning in function unit and control unit. The Data goes in function unit and output comes from the functional unit but the functional units are controlled by the Command signal. The certain category of output of function unit is given back to the control unit in the form of the Status Signal. Which helps the Control unit to jump from block to block. In addition, control unit can have external input and external output. External input basically contains things like Start, Reset and Stop like that.

We also called the functional unit as architectural problem. The Group of Functional unit is called the architecture. So, the architecture consist of two registers. The architecture is controlled by control unit. The Partition of Functional unit and control unit is required. Architecture is the unit of any system that can able to process the data. But the controller unit is one that can able to control the operation.

In this problem, register 'A' load the data, we have taken the 'N' bit register. It must have the Parallel in and Parallel out. And then we want to load the data and ready to load the data. For which I required the Load signal.
In order to control the system, we require a clock signal. Then for algorithm, we are looking for LSB (Least Significant Bit). We see that whether it is 1 or zero. If '1' comes then go and add '0' to the result of the register. Then shift the data right. Till exhaust shift register, then shift the data right. All the bit we are going on shifting the LSB. But all the bit we are required every time we are looking for LSB bit. So, we are required every time we are looking for LSB bit. Shift register, which are having the feature of shifting a shift register, which are having the feature of Shift register. The components are supposed to have that component should be easily available.

Register 'B' is one unit which stores the value of the No. of 1's maximum number of 1's in a bit of sequence will be 'm' and it has to be \( |\text{Reg}_{m+1}| \). This will be a counter, which will be initially clear. And the clock will go, but it will not count, it will count only when count enable signal will work. Now, clock only, when count enable signal will work. Now, clock only, when count enable signal will work. The clock 'C' is common to all the. So, we have decided the architecture of the functional blocks. So, we are required to draw the State graph of the control unit. It is a very simple State graph.

\[ S_0 \rightarrow \text{Read State} \]
\[ S \rightarrow \text{Start Signal} \]
\[ A \rightarrow \text{Content of register} \]
\[ 13'0' \]
\[ q_0 \rightarrow \text{LSB} \]

**Diagram:**

- Initial state: \( S_0 \)
- Transition: \( q_0 = 1 \) to \( S_1 \)
- Transition: \( q_0 = 0 \) to \( S_0 \)
- Transition: \( S_1 \) to \( S_2 \) with \( q_0 = 1 \)
- Transition: \( S_2 \) to \( S_3 \) with \( q_0 = 0 \)
- Transition: \( S_3 \) to \( S_4 \) with \( q_0 = 1 \)
- Transition: \( S_4 \) to \( S_0 \) with \( q_0 = 0 \)

\( q_{t+1} = L \)

\( q_0(\Delta t) \)
When Start Signal S comes, loading the Register A) and start counting and putting the value in register B. Before reaching N=5, we required to Clear the reg resistance B, and load register A.

For this 8-State Machine

Solution: Count, Shift-right (A)? Control unit.

Load A, CLR-B

Input: S, A, φo

Let us draw the State Table

<table>
<thead>
<tr>
<th>Present State</th>
<th>Input Condition</th>
<th>Next State</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>X Y Z</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0</td>
<td>0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>1</td>
<td>1 1 0</td>
<td>1 1 0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>1</td>
<td>1 1 0</td>
<td>1 1 0</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>0 1 0</td>
<td>1</td>
<td>1 1 0</td>
<td>1 1 0</td>
</tr>
<tr>
<td>0 1 1</td>
<td>0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>0 1 1</td>
<td>1</td>
<td>1 1 0</td>
<td>1 1 0</td>
</tr>
<tr>
<td>1 0 0</td>
<td>0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1</td>
<td>1 1 0</td>
<td>1 1 0</td>
</tr>
<tr>
<td>1 0 1</td>
<td>0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>1 0 1</td>
<td>1</td>
<td>1 1 0</td>
<td>1 1 0</td>
</tr>
<tr>
<td>1 1 0</td>
<td>0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1</td>
<td>1 1 0</td>
<td>1 1 0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1</td>
<td>1 1 0</td>
<td>1 1 0</td>
</tr>
</tbody>
</table>

So we can implement it using mux.
LOAD = X'YZS
CLE =
SHIFT N = X'Y'Z
CB = XY'Z
With this example, let's have seen the system design. We talked about the top-down design. We have separate the functional unit which is called the architectural unit and control unit. Get the combinational functional unit from the component, which are available to use. We have to very careful about the input/output requirements. Then go to the State graph of the control unit. Just go through the algorithm systematically and draw the State graph. Functional units are already available, and we should know the signals which are available. Of course, the data input and data output are available.