Digital Logic

CS2102

August 23, 2021

Outline of the Course

- Number systems, logic gates and Boolean algebra
- Representation, manipulation and minimization of Boolean Functions
- Design of Combinational Circuits
- Design of Sequential Circuits
- Concept of finite state machines
- Digital Integrated Circuits

Number systems

Systematic way to represent and manipulate numbers

Examples of number systems

- Decimal
- Binary
- Roman
- Sexagesimal

Broad Classification of number systems

- Weighted - decimal, binary etc.
-Non-weighted - Roman, Gray code etc.

Base or radix

- Decimal number system is said to be of base 10, because it uses 10 distinct digits (0, 1,2 ...9) and the coefficients are multiplied by power of 10.
- Binary number system has a base of 2, because it uses two digit (0 and 1) and each coefficient is a_i multiplied by 2^i.
- In general, a number system of base (radix) r uses r distinct digits (0, 1, ... (r-1)) and each digit has a weight of some power of r (say r^k.
  $k \geq 0$ for integer part
  $k <$ for the fractional part
  $D = d_{n-1}d_{n-2}\ldots d_2d_1d_0.d_{-1}d_{-2}\ldots d_m$

Signed and Unsigned Binary Number Representation

- Binary number system is important for designing Digital Circuits.
- Why are binary numbers important?
  - At the low-level, the circuit is implemented using Transistors.
  - A transistor has two stable states, either 'ON' or 'OFF'.
  - As the transistor has two states and can be denoted by two digits (0 or 1) in binary system.
- There are some conventions:
  - Open switch is denoted by 0 and closed switch is denoted by 1.
  - Low voltage is represented by 0 and high voltage is represented by 1.
  - Absence of current is represented by 0 and flow of current is represented by 1.

In this course, we will consider only binary number.
Some conventions:

- Bit $\Rightarrow$ Single binary digit (0 or 1)
- Nibble $\Rightarrow$ Collection of 4 bits
- Byte $\Rightarrow$ Collection of 8 bits
- Word $\Rightarrow$ Collection of 16/32/64 bits

Representing Number in Binary

How many distinct number can be represented using n bits?

- Each bit has two possible states.
- Total number of possible combinations:
  $2 \times 2 \times 2 \ldots \textit{upto } n \textit{ terms} = 2^n$

Number can be signed or unsigned.
An unsigned number has only magnitude, no sign bit.
A signed number has both the magnitude and a sign (+ or -)

Unsigned Binary Number

An n-bit binary number system can have 2^n distinct numbers.
$Minimum = 0$ and $Maximum = 2^n - 1$
For example n = 3, there are 8 possible numbers
000 001 010 011 100 101 110 *and* 111
An n-bit binary integer can denoted as:
$b_{n-1}b_{n-2}\ldots b_2 b_1 b_0$
The equivalent unsigned decimal value is:
$D = b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} \ldots + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0$
Each digit position has a weight that is same power of 2.

Signed Integer Representation

There are three possible approaches to represent singed integers.

- Sign-magnitude representation.
- 1's Complement representation.
- 2's Complement representation.

Sign-Magnitude Representation

- For an n-bit number representation, the most significant bit (MSB) represent sign (0 $\Rightarrow$ positive and 1 $\Rightarrow$ negative).
- The rest of (n -1) bits denote the magnitude of the number.
- Range of numbers: $-(2^{n-1}-1) to +(2^{n-1}-1)$

Sign-magnitude Representation

A problem in sign-magnitude representation:
There are two possible representation of *zero*
+0 = 0000; -0 = 1000 (in 4 bits representation)

Decimal	Sign-Magnitude
+7	0111
+6	0110
+5	0101
+4	0100
+3	0011
+2	0010
+1	0001
+0	0000

Decimal	Sign-Magnitude
-0	1000
-1	1001
-2	1010
-3	1011
-4	1100
-5	1101
-6	1110
-7	1111

1's Complement Representation

Basic Idea

- Positive numbers are represented exactly as in sign-magnitude representation form.
- Negative numbers are represented in 1's complement form
- 1's complement of a number is obtained by complementing every bit of a number. (1 to 0 and 0 to 1).
- MSB will indicate the sign of the number.

Decimal	1's complement
+7	0111
+6	0110
+5	0101
+4	0100
+3	0011
+2	0010
+1	0001
+0	0000

Decimal	1's complement
-0	1111
-1	1110
-2	1101
-3	1100
-4	1011
-5	1010
-6	1001
-7	1000

1's Complement Representation

- In 1's complement representation, the numbers ranges from (maximum $= +(2^{n-1} - 1)$) to (minimum $=$ -$(2^{n-1} - 1)$) in n-bits representation.
- Similar to the sign-magnitude representation, 1's complement representation has two zeros.
- Advantage of the 1's complement representation is that subtraction can be done using addition which leads to saving in circuity.

2's Complement Representation

Basic Idea

- Positive numbers are represented exactly as sign-magnitude form.
- Negative numbers are represented in 2's complement form.
- To get the 2's complement of a number, we have to complement each bit of the number (1 to 0 and 0 to 1), and the add 1 to the resulting complement number.
- MSB will indicate the sign of the number (0 $\Rightarrow$ positive and 1 $\Rightarrow$ negative)

2's Complement Representation

Example of 2's complement representation for n = 4

Decimal	1's complement
+7	0111
+6	0110
+5	0101
+4	0100
+3	0011
+2	0010
+1	0001
+0	0000

Decimal	1's complement
-8	1000
-7	1001
-6	1010
-5	1011
-4	1100
-3	1101
-2	1110
-1	1111

- Range: Maximum = $+(2^{n-1}-1)$ to Minimum = -2^{n-1}
- Advantages of 2's complement representation: (i) Unique representation of zero; (ii) Subtraction can be done using addition which leads to saving circuity.

2's Complement Representation

Additional features of 2's complement representation:

- Weighted number representation with MSB having weight -2^{n-1}.
  $D = -b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} \ldots + b_1 \times 2^1 + b_0 \times 2^0$
  $0101 = 0 + 4 + 0 + 1 = 5$
  $1101 = -8 + 4 + 0 + 1 = -3$
- Shift left by k positions with zero padding multiplies the number by 2^k.
  00001001 = 9 $\Rightarrow$ shift left by 2 positions 00100100 = 36
  11110111= -9 $\Rightarrow$ shift left by 2 positions 11011100 = -36
- Shift right by k positions with sign-bit padding divides the number by 2^k.
  00100100 = 36 $\Rightarrow$ shift right by 2 position 00001001 = 9
- The sign bit can be copied as many times as required at the beginning to extend the size of the number (called sign extension).
  X = 0111 (4 bit number value= 7), sign extend to 8 bit 00000111.

Boolean Algebra

September 3, 2021

Basic Concepts

Boolean Algebra:

- An algebraic system defined on the set $\{0, 1\}$, with two binary operators (AND and OR) and one unary operator NOT. AND operation is also called Logical Product. OR operation and NOT operation are called Logical sum and complement respectively.
- Boolean (switching) variables are two valued variables that can take two distinct values 0 and 1.
- Boolean expression is an expression consisting of Boolean variables, constants and operators.
- Given a Boolean expression, how to prove it? Suppose $x + xy = x$, how do we prove it?
  (i) By verifying the expression for all possible values of the variable. Called truth table verification or perfect induction.
  (ii) By using algebraic manipulation using some rules.

Basic Laws of Boolean Algebra

Basic identities

$x + 1 = 1$

$x + 0 = x$

$x.1 = x$

$x.0 = 0$

Idempotent Law

$x + x = x$

$x.x = x$

- Commutative law

  $x + y = y + x$ $\qquad$ $x.y = y.x$

  Order of variable does not matter.

- Complementation law

  $x + \overline{x} = 1$ $\qquad$ $x.\overline{x} = 0$

- Associative law

  $(x + y + z = x + (y + z)$ $\qquad$ $(x.y).z = x.(y.z)$

- Distributive law

  $x.(y + z) = x.y + x.z$

  $x + (y.z) = (x + y).(x + z)$

Basic Laws of Boolean Algebra

- Absorption law

(i) $x + (x.y) = x$

$x + x.y$

$= x.1 + x.y$

$= x.(1 + y) \Rightarrow$ distribution law

$= x.1$

$= x$

(ii) $x.(x + y) = x\ x.(x + y)$

$= (x + 0).(x + y)$

$= x + 0.y \Rightarrow$ distribution law

$= x + 0$

$= x$

Basic Laws of Boolean Algebra

- Useful law

(i) $x + (\overline{x}.y) = x + y$

$x + (\overline{x}.y)$

$= x.1 + \overline{x}.y$

$= x.(y + \overline{y}) + \overline{x}.y$

$= x.y + x.\overline{y} + \overline{x}.y$

$= x.y + x.y + x.\overline{y} + \overline{x}.y$

$= x.y + x.\overline{y} + x.y + \overline{x}.y$

$= x.(y + \overline{y}) + y.(x + \overline{x})$

$= x.1 + y.1$

$= x + y$

(ii) $x.(\overline{x} + y) = x.y$

Basic Laws of Boolean Algebra

- Consensus Theorem
  (i) $x.y + \overline{x}.z + y.z = x.y + \overline{x}.z$
  (ii) $(x + y).(\overline{x} + z).(y + z) = (x + y).(\overline{x} + z)$
- Involution
  $\overline{(\overline{x})} = x$

These rules may be used for algebraic manipulation of boolean expression.

Boolean Function Minimization

Given a Boolean expression, we can simplify it by using basic laws of Boolean algebra. We can reduce the number of terms in the expression. We can also reduce the number of literals (variables).
Example:

$$x.y + x.y.z + \overline{y}.z$$
$$= x.y.1 + x.y.z + \overline{y}.z$$
$$= x.y(1 + z) + \overline{y}.z$$
$$= x.y.1 + \overline{y}.z$$
$$= x.y + \overline{y}.z$$

Algebraic Manipulation

Principle of Duality

- Most of the rules discussed so far appear in pairs.
- Principle of duality states that a Boolean expression E_2 can be obtained from a given Boolean expression E_1 by interchanging the operations AND and OR and constant 0 and 1. E_1 and E_2 are said to be dual of each other.

Examples:
$x + \overline{x}.y = x + y \Rightarrow$ if this true then,
$x.(\overline{x} + y) = x.y$ is also true (from duality principle)
$x + 1 = 1$ is the dual of $x.0 = 0$

Simplification Examples

Example- 1

$F = x\overline{y} + xy + yz$

$F = x(\overline{y} + y) + yz$

$F = x.1 + yz$

$F = x + yz$

Example - 2

$F = \overline{x}yz + x\overline{y}z + x.y\overline{z} + xyz$

We know that $x + x + x = x$

$F = \overline{x}yz + xyz + x\overline{y}z + xyz + x.y\overline{z} + xyz$

$F = yz(\overline{x} + x) + xz(\overline{y} + y) + xy(\overline{z} + z)$

$F = yz.1 + xz.1 + xy.1$

$F = xy + yz + xz$

De Morgan's Theorem

This is very important law that you can apply for transforming expressions in variety of ways.

For two variables x and y, De Morgan's theorems state that

(i) $\overline{(x + y)} = \overline{x}.\overline{y}$

(ii) $\overline{(x.y)} = \overline{x} + \overline{y}$

Simplification using De Morgan's Theorems

$$F = \overline{(x+y)}.(\overline{x}+\overline{y}) = \overline{x}.\overline{y}.(\overline{x}+\overline{y}) = \overline{x}.\overline{y} + \overline{x}.\overline{y} = \overline{x}.\overline{y} \quad (1)$$

$$\begin{aligned} F &= \overline{(x.\overline{y} + \overline{x}.y)} \\ &= \overline{(x.\overline{y})}.\overline{(\overline{x}.y)} \\ &= (\overline{x} + y).(x + \overline{y}) \\ &= x\overline{x} + \overline{x}.\overline{y} + x.y + y.\overline{y} \\ &= x.y + \overline{x}.\overline{y} \end{aligned}$$

Properties of Boolean Function

September 3, 2021

Minterm and Maxterm

- In a Boolean function, a literal is defined as variable in uncomplemented or complemented form. For example, $x,\ \overline{x},\ y,\ \overline{y}$ etc.
- Consider an n-variable Boolean function $f(x_1, x_2, \ldots, x_n)$.
  Minterm: A product term (AND operation) of all the n literals is called a *minterm*.
  Maxterm: A sum term (OR operation) of all the n literals is called *maxterm*.
- Consider a function $f(x, y, z)$, the minters are $\overline{x}\ \overline{y}\ \overline{z}$, $\overline{x}\ \overline{y}\ z$, $\overline{x}\ y\ \overline{z}$, $\overline{x}\ y\ z$, $x\ \overline{y}\ \overline{z}$, $x\ \overline{y}\ z$, $x\ y\ \overline{z}$, $x\ y\ z$
  Maxterms are: $(x + y + z)$, $(x + y + \overline{z})$, $(x + \overline{y} + z)$, $(x + \overline{y} + \overline{z})$, $(\overline{x} + y + z)$, $(\overline{x} + y + \overline{z})$, $(\overline{x} + \overline{y} + z)$, $(\overline{x} + \overline{y} + \overline{z})$

Properties of Minterms and Maxterms

- A particular minterm assumes value 1 for exactly one combination of variables.
  Consider a function $f(x, y, z) = x\overline{y}z + xy\overline{z} + xyz$
  The first minterm will be 1 when $x = 1, y = 0$ and $z = 1$, the second minter will be 1 when $x = 1, y = 1$ and $z = 0$ and the third minterm will be 1 when $x = y = z = 1$
- A particular maxterm assumes value 0 for exactly one combination of variables.
  For example, the first maxterm of the following function will be 0 when $x = 0, y = 1$ and $z = 0$
  $f(x, y, z) = (x + \overline{y} + z).(\overline{x} + \overline{y} + \overline{z}).(x + y + z)$

Properties of Minterms and Maxterms

For a given Boolean function, and for a given values of input variables:

- All the minterms that have the value 1 are called the true minterms.
- All the minterms that have the value 0 are called the false minterms.
- $f(x, y, z) = \overline{x}y + xyz$, for this function the true minterms are $\overline{x}yz$, $\overline{x}y\overline{z}$ and xyz. As $\overline{x}y = \overline{x}y(z + \overline{z})$
- All the maxterms that have the value 1 are called the true maxterms.
- All the maxterms that have the value 0 are called the false maxterms.

Canonical Form of Representing Function

- A canonical form is unique representation of a function.
- We can obtain two canonical representation directly from the truth table.
  (i) Canonical sum-of-product (disjunction normal form).
  (ii) Canonical product-of-sum (conjunctive normal form.

Canonical Sum-of-Product

- From the truth table, identify all the true miterms - corresponding to rows for which the output of the function is 1.
- Take the sum of all the true minterms.
- Example: Consider following truth table, the canonical sum-of-product form is $s = \overline{x}\ \overline{y}z + \overline{x}y\overline{z} + x\overline{y}\ \overline{z} + xyz$

x	y	z	s
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Canonical Sum-of-Product

We can write the canonical sum-of-product expression in a compact way by noting the decimal equivalent values of the true minterms.
$s = \sum(1, 2, 4, 7)$

Canonical Product-of-Sum

- From the truth table, identify the false minterms. - corresponding to the rows for which the output of the function is 0.
- For each false minterm, form a sum term where a variable will appear in uncomplemented (complemented) form if it has value 0 (1) in the row.
- For example, s can be written as
  $s = (x + y + z)(x + \overline{y} + \overline{z})(\overline{x} + y + \overline{z})(\overline{x} + \overline{y} + z)$
- We can write the canonical product-of-sum expression in compact way by noting the decimal equivalent value of false minterm.
  $s = \prod(0, 3, 5, 6)$

Function Completeness

September 3, 2021

Basic Concept

- In Boolean algebra, the basic operations are NOT, AND and OR.
- Any Boolean expression can be realized using these three operations.
- We say that the set NOT, AND, OR is functionally complete.

NAND is Functionally Complete

- It can be shown that NAND gate is a universal gate, i.e., it is functionally complete.
- How to prove it?
  We can show that we can realize NOT, AND, and OR functions using NAND only.
  NOT: $\overline{x} = \overline{(x.x)}$
  AND: $x.y = \overline{\overline{(x.y)}}$
  OR: $x + y = \overline{(\overline{x}.\ \overline{y})} = \overline{\overline{(x.x)}.\overline{(y.y)}}$

NOR is functionally Complete

- It can be shown that NOR gate is a universal gate.
  $NOR(x+y) = \overline{(x+y)}$
- How to realize basic gates using NOR?
  NOT: $\overline{x} = \overline{(x+x)}$
  AND: $x.y = \overline{(\overline{x} + \overline{y})}$
  OR: $x+y = \overline{\overline{((x+y))}}$

{AND, NOT is Functionally Complete}

- We have already shown that NAND is functionally complete.
- NAND gate can be realized with AND gate followed by NOT gate. Hence, {AND, NOT} is also functionally complete.

{ AND, EXOR, 1} is functionally complete

- We know that $EXOR(A, B) = A\overline{B} + \overline{A}.B$
  The output of a *EXOR* is 1 if the number inputs having values 1 is odd.
- $EXOR(A, 1) = A.\ \overline{1} + \overline{A}.\ 1 = \overline{A}.$ Hence, *EXOR* gate can be used as NOT gate with one input is set to 1.
- We have already shown {AND, NOT} is functionally complete set. Hence, {AND, EXOR, 1} is also functionally complete set.

Two-level AND-OR is equivalent to NAND-NAND

- Any two-level AND-OR realization can be converted to an equivalent NAND-NAND realization by replacing AND and OR gates by NAND gates.
- Why is this possible?
  It is possible because of De Morgan's theorem.
  $AB + CD = \overline{(\overline{(AB)}.\ \overline{(CD)})}$

Two level OR-AND is equivalent to NOR-NOR

- Any two-level OR-AND circuit can be converted to an equivalent NOR-NOR realization by replacing all OR and AND gates by NOR gates.
- Why is this possible?
  This is possible using De Morgan's theorem.
  $(A+B).(C+D) = \overline{(\overline{(A+B)} + \overline{(C+D)})}$

Binary Addition and Subtraction

September 3, 2021

Binary Arithmetic

- The binary number system is widely used in computer system.
- We need to know some essential concepts about binary arithmetic.

Input		x + y		x-y		x.y
x	y	sum	carry	difference	borrow	product
0	0	0	0	0	0	0
0	1	1	0	1	1	0
1	0	1	0	1	0	0
1	1	0	1	0	0	1

Binary Addition

Binary addition is done in the similar way to that of decimal addition. Corresponding bits are added, and if there is a carry that will be added to next left bit.
Examples: (Assume all numbers are positive)

1 1 1 $\Leftarrow$ Carry
0 1 0 1
<u>0 1 1 1</u>
1 1 0 0

1 1 1 1 $\Leftarrow$ Carry
1 0 1 1 0 1
<u>1 0 0 1 1 1</u>
1 0 1 0 1 0 0

Binary Subtraction

- Binary subtraction is performed similar to decimal subtraction.
- We need to understand when the borrow will be generated. We have to use this borrow. we know that if we subtract 1 from 0 in that case borrow will be generated.

Example:

1 0 0 $\Leftarrow$ Borrow

1 0 0 1

$\underline{0\ 1\ 0\ 1}$

0 1 0 0

1 1 0 0 $\Leftarrow$ Borrow

1 0 0 1 0

$\underline{0\ 1\ 1\ 0\ 0}$

0 0 1 1 0

Subtraction using 1's complement

Suppose we want to perform $A - B$ using 1's complement. We have to follow the steps given below.

- Compute the 1's complement of (say $\overline{B}$).
- Perform $R = A + \overline{B}$
  - If a carry is obtained after addition. Add the carry with R (i.e $R = R + 1$). The result is a positive number.
  - Else
    The result is a negative and is in 1's complement form of R

Examples:

Suppose we want to perform $6 - 2$. 1's complement of 2 is 1 1 0 1 (assume 4-bit representation).

$6 = \quad 0\ 1\ 1\ 0$

$\underline{\text{-}2 = 1\ 1\ 0\ 1}$

$\quad\quad 1\ 0\ 0\ 1\ 1$

There is a carry and the carry will be added with 0 0 1 1 (i.e 0 0 1 1 + 0 0 0 1). The result is 0 1 0 0 which is 4 in decimal.

Subtraction using 1's complement

Example:
Suppose we want to perform $3 - 5$. The 1's complement of 5 is 1 0 1 0 (4-bit representation).

$$
\begin{array}{l}
3 = \quad 0\ 0\ 1\ 1 \\
\underline{\text{-}5 = 1\ 0\ 1\ 0} \\
\qquad\quad 1\ 1\ 0\ 1
\end{array}
$$

Since there is no carry, the result is negative. [1 1 0 1] is the 1's complement of [0 0 1 0] that is, the result represents -2 in decimal.

Subtraction using 2's complement

Suppose we want to compute $A - B$ using 2's complement then, we have to follow the steps given below.

- Compute the 2's complement of B (say $\overline{B}$).
- Compute $R = A + \overline{B}$
- If a carry is generated after addition: Ignore the carry and the result is positive.
- Else The result is negative and it is in 2's complement form.

Examples: We want to compute $(6 - 2)$ and $(3 - 5)$
2's complement of 2 is [1 1 1 0] and 2's complement of is [1 0 1 1].
In case of $(6 - 2)$ carry is generated and we have to ignore the carry. The result is positive. For the second case, there is no carry. Hence, the result is negative and it is in 2's complement form ([1 1 1 0] is the 2's complement of 2. so, the result is -2).

6 =	0	1	1	0
-2 =	1	1	1	0
1	0	1	0	0

3 =	0	0	1	1
-5 =	1	0	1	1
	1	1	1	0

BCD and Gray Code

Sekhar Mandal

November 13, 2020

Binary Coded Decimal (BCD) Number Representation

- It is sometimes desirable to manipulate numbers in decimal instead of converting them to binary.
- Decimal to binary and binary to decimal conversion process is complex.
- One popular code to represent decimal is BCD. Each decimal digit is represented by 4-bit binary equivalent. conversion is much easier.

Examples:
Decimal 391 in BCD 0011 1001 0001
Decimal 13.34 in BCD 0001 0011.0011 0100
There are six combinations 1010 1011 1100 1101 1110 1111

Addition of BCD Numbers

When two BCD numbers are added, there may be needed for correction step where 6(0110) will be added to one of the nibble. This correction is required when a nibble is one of the six unused combinations or there is a carry in from the previous nibble.
Examples:
$23 + 46 = 69$

$$\begin{array}{r} 0\ 0\ 1\ 0\ 0\ 0\ 1\ 1 \\ \underline{0\ 1\ 0\ 0\ 0\ 1\ 1\ 0} \\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1 \end{array}$$

Addition of BCD Numbers

Example:

$23 + 48 = 71$

0 0 1 0 0 0 1 1

<u>0 1 0 0 1 0 0 0</u>

0 1 1 0 1 0 1 1 $\Rightarrow$ Here the right most nibble is one of the unused combination. Hence, we have to add [0 1 1 0] with this nibble.

0 1 1 0 1 0 1 1

<u>0 0 0 0 0 1 1 0</u>

0 1 1 1 0 0 0 1

Addition of BCD Numbers

Example:

$28 + 39 = 67$

0 0 1 0 1 0 0 0

<u>0 0 1 1 1 0 0 1</u>

0 1 1 0 0 0 0 1

Here, carry is flowing from one nibble to the other nibble. So, correction is required.

0 1 1 0 0 0 0 1

<u>0 0 0 0 0 1 1 0</u>

0 1 1 0 0 1 1 1

Gray Code

$\Rightarrow$ There are some applications, where if multiple bits are changing between two consecutive number, that may create problem. To elevate the problem, a new code was introduced which is called Gray code.
$\Rightarrow$ Gray code is a type of non-weighted binary code where successive code words differ in only one bit. Any code with this property is called cyclic code.
$\Rightarrow$ Gray code is used in many practical applications that require analog to digital conversion.

- To reduce error in conversion.
- Also, binary to gray and gray to binary conversions are easy.
- Example: to measure the angle of rotation of a wheel.

Gray Code

- Gray code also called self-reflecting code. Suppose we have the Gray code representation for m-bit.
- To obtain the Gray code representation for (m + 1)-bit, we write down two m-bit representation one below the other with second one being the mirror image of the first one.
- We then add 0 at the beginning of every code in the first group and 1 at the beginning of every code in the second group.

Binary to Gray code conversion

- Let $g_{n-1}g_{n-2}\ldots g_2g_1g_0$ and $b_{n-1}b_{n-2}\ldots b_2b_1b_0$ denote n-bit Gray code and its equivalent binary representation respectively.
  $g_i = b_i \oplus b_{i+1}$ for $0 \leq i \leq n-2$ and
  $g_{n-1} = b_{n-1}$
- $0 \oplus 0 = 0 \quad 0 \oplus 1 = 1 \quad 1 \oplus 0 = 1 \quad 1 \oplus 1 = 0$

Decimal	binary representation	Gray code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Gray code to binary code conversion

Start with the MSB and proceed to the LSB and set

- $b_i = g_i$, if number of 1's preceding g_i is even.
- $b_i = \overline{g_i}$, if number of 1's preceding g_i is odd.

$g_2 g_1 g_0$	$b_2 b_1 b_0$
000	000
001	001
011	010
010	011
110	100
111	101
101	110
100	111

Simplification of Boolean Functions using Karnaugh Map

Sekhar Mandal

August 24, 2020

Karnaugh Map

- What is Karnaugh map?
  A Karnaugh map may be considered as a pictorial representation of a truth table.
  It provides straightforward procedure for minimizing Boolean functions.
- For an n-variable function, there are 2^n cells in the map (one for each minterm).
- Adjacent 2 cells differ in only one variable.
- Adjacent $2^2 = 4$ cells differ in 2 variables.
- Adjacent 2^m cells differ in m variables.

Karnaugh Map

- We try to group 2^m cells corresponding true minterms. Try to make cubs (groups) bigger, ensure all the minterms are covered.
- How does you label the Karnaugh map such that its property is ensured?
  (i) such that two adjacent differ in the value of one variable.
  (ii) help in combing cells into cubs.
  $ABC + A\,\overline{B}C = AC$
  $A\overline{B}\,\overline{C} + A\,\overline{B}C + AB\,\overline{C} + ABC = A$
  Drawback

- Since it is pictorial approach, it is difficult to visualize function with more than 5 or 6 variables.
- We shall discuss examples with upto 4 variables.

Basic Approach

1. Fill up the cells of K-map with true minterms of the function.
2. Group the true minterms of the function into cubs such that
   - The size of the cubes are maximized.
   - Every true minterms is covered by at least one cube.
3. Write down the minimized sum-of-product expression for the function by creating one product term out of every cube that has been selected.

Three Variable Karnaugh Map

Examples:

Four Variable Karnaugh Map

The basic concept of labelling the variables remains the same:

- Adjacent cells differ in value of a single variable.
- Top-bottom and left-right cells are also adjacent.
- The four corner cells are considered adjacent to each other.

Examples:

Handling Don't Care Inputs

- There exists functions for which some of the inputs are treated as don't cares, and the corresponding output values do not matter.
  - Such inputs never appear. For example, in case of BCD number miterms 1010 to 1111 never come in the inputs.
- The don't care minterms are labelled as'X' in the K-map.
- When creating the cubes, we can include cells marked 'X' along with those marked as '1' to make larger cube.
- But it is not necessary to cover all the cells marked by 'X'.

Examples:

Some Definitions

- **Implicant:** Given a function f of n variables, a minterm p is an implicant of f if and only if, for all combinations of the n variables for which $p = 1$, f is also 1.
  Consider the the function $f = A\overline{B}C + A\overline{C} + \overline{B}\ \overline{C}$
  The term $A\overline{B}C$ is an implicant because when A = 1, B = 0 and C = 1, the term is 1 as well as the function is also 1.
- **Prime implicant:** An implicant is said to be prime implicant if after removing any literal from it, the resulting product is no longer an implicant..
- With respect to K-map, it is a cube that is not completely covered by another implicant represent a larger cube.

Consider the function$f = \overline{A}B + AC + \overline{B}\ \overline{C}$
$\overline{A}B$ is an prime implicant. A B C (010 and 0 1 1), for these combinations of the inputs, $\overline{A}B = 1$. If we remove B from $\overline{A}B$, the resulting term $\overline{A}$ is not an implicant. Because 0 0 0 and 0 0 1 for this two combinations of the inputs the function is not equal to 1.

Essential Prime Implicant:

A prime implicant of a function is said to be essential prime implicant if it covers at least one minterm of the function that is not cover by any other prime implicant.

Some Results:

- Every irredundant sum-of-product expression equivalent to a function F is a union of prime implicant of F.
- The set of all essential prime implicants must be present in any irredundant sum-of-product expression.
- Any prime implicant covered by the some of the essential prime implicants must not be present in an irredundant sum-of-product expression.

Simplification of Boolean Functions using Tabulation Method

Sekhar Mandal

September 7, 2020

Motivation

- The K-map method of minimization is convenient when the number of variables less than six.
- The map method is a trial-and-error procedure that depends on the human ability to recognize certain pattern.
- It is also difficult to automate the method.
- For large functions, a more systematic procedure is required and for that Quine-MeCluskey method can be used.
- The Quine-MeCluskey method is easy to automate.

Basic Concept

- The basic concept remain the same.
- Repeated application of the theorem $A\overline{X} + AX = A$ to all adjacent pair term. This will produce the set of all prime implicants at the end.
- This is a two steps process. First generate all the prime implicants then, select the prime implicants that will cover the all miniterms of the function.
- Example:
  $F(A, B, C, D) = \overline{A}\ \overline{B}\ \overline{C}D + \overline{A}\ \overline{B}\ \overline{C}\ \overline{D} + A\ \overline{B}\ \overline{C}\ \overline{D} + A\ \overline{B}\ \overline{C}D$
  $F(A, B, C, D) = \overline{A}\ \overline{B}\ \overline{C} + A\ \overline{B}\ \overline{C}$
  $F(A, B, C, D) = \overline{B}\ \overline{C} \Rightarrow$ Prime implicant.
- We can go combing any pair of product terms that differ in the value of single literal.

Basic Concept

1. Two k-variable terms can be combined into single (k -1) variable term, if and only if they differ in only one literal.
2. We use binary representation of the minterms for convenience.
3. Two minterms can be combined if their binary representation differ in only one position.
4. We use the symbol '-' to indicate the absence of a literal.

Identification of all Prime Implicant (step -1)

- Arrange all the minterms in groups based on the number of 1's. The number of 1's in a term is called in index.
- Compare every term of a group (index i) with the each term of another group having index (i +1).
  - Merge the two term where possible using the rule $AX + A\overline{X} = A$.
  - Place a check mark to the each term that has been combined with at least one term.
- Now compare the terms that are generated as the output of the previous iteration in the same fashion, generate a new term by combining two terms that differ by only one position.
- The process continues until no further combination are possible.
- The remaining unchecked terms are the prime implicants of the function.

Example

Consider the following Boolean function
$F(w, x, y, z) = \sum(0, 1, 2, 8, 10, 11, 14, 15)$

	w	x	y	z	
0	0	0	0	0	√
1	0	0	0	1	√
2	0	0	1	0	√
8	1	0	0	0	√
10	1	0	1	0	√
11	1	0	1	1	√
14	1	1	1	0	√
15	1	1	1	1	√

	w	x	y	z	
0,1	0	0	0	-	
0,2	0	0	-	0	√
0,8	-	0	0	0	√
2,10	-	0	1	0	√
8,10	1	0	-	0	√
10,11	1	0	1	-	√
10,14	1	-	1	0	√
11,15	1	-	1	1	√
14,15	1	1	1	-	√

Example Cont.

	w	x	y	z
0,2,8,10	-	0	-	0
0,8,2,10	-	0	-	0
10,11,14,15	1	-	1	-
10,14,11,15	1	-	1	-

The set of prime implicants are $\{\overline{w}\ \overline{x}\ \overline{y},\ \overline{x}\ \overline{z},\ w\ y\}$

Prime Implicant Chart

How to select the smallest set of prime implicants that cover all the minters of the function.
For this purpose, We shall use prime implicant chart.

- It is a tabular data structure, which pictorially depicts the covering relationship between prime implicants and minterms.
- Useful to select the minimum set of prime implicants.
- Minterms are listed along columns, while prime implicants listed along rows.
- A 'X' is entered in the table if corresponding prime implicant covers corresponding minterm.
- If a column has a single 'X', the prime implicant corresponding to the row in which the 'X' appears is an essential prime implicant.

Selection of essential Prim implicants

- A check mark is placed in the chart to the essential prime implicants to indicate that they have selected.
- Next check each column whose minterm is covered by the essential prime implicants.
- If the essential prime implicants do not cover all the minterms of the function, select the prime implicants that will cover the uncovered minterms.

Example of Prime Implicant Chart

Consider the following Boolean function:
$F(w, x, y, z) = \sum(1, 4, 6, 7, 8, 9, 10, 11, 15)$ and the prime implicates are
$\overline{x}\ \overline{y}\ z$, $\overline{w}\ x\ \overline{z}$, $\overline{w}\ x\ y$, $x\ y\ z$, $w\ y\ z$, and $w\ \overline{x}$.
We have to create a prime implicant chart (table)

		1	4	6	7	8	9	10	11	15
$\surd\ \overline{x}\ \overline{y}\ z$	1,9	X					X			
$\surd\ \overline{w}\ x\ \overline{z}$	4,6		X	X						
$\overline{w}\ x\ y$	6,7			X	X					
$x\ y\ z$	7,15				X					X
$w\ y\ z$	11,15								X	X
$\surd\ w\ \overline{x}$	8,9,10,11					X	X	X	X	
		$\surd$	$\surd$	$\surd$		$\surd$	$\surd$	$\surd$	$\surd$	

Latch and Flip-Flop

October 5, 2020

Definition of Sequential Circuit:

- The outputs of a sequential circuit depend not only on the present inputs, but also the past history of the circuit. The circuit memorizes its present sate.
- Latches and flip-flops are the basic building block of storage devices.
- There are two types of sequential circuits, synchronous and asynchronous.
- Basic idea behind storing bit.

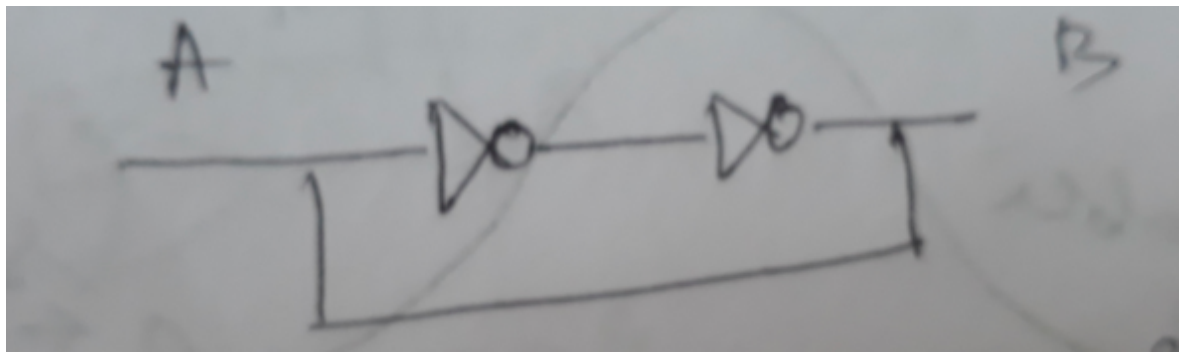


Figure: Cascaded Inverters with feedback.

Design of Latches

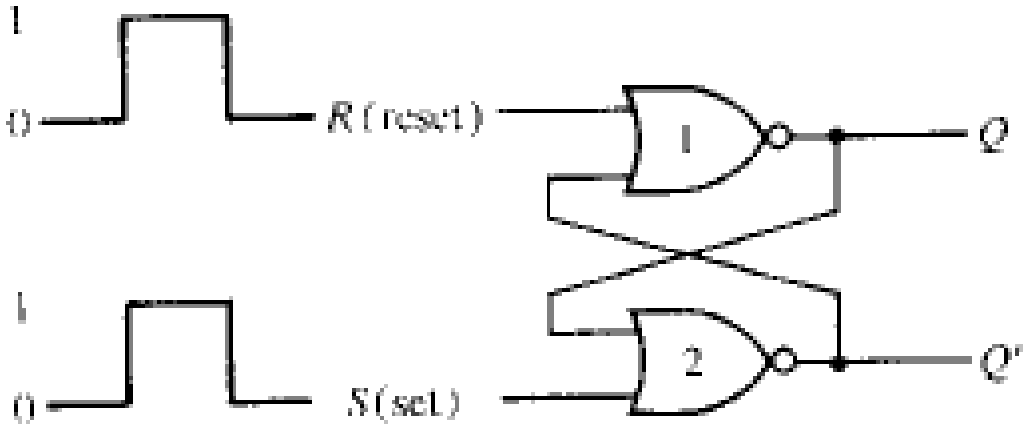
- A latch is a temporary storage device that has two stable states, 0 and 1.
- A flip-flop is a special kind of latch where a clock signal triggers the change in the stored value.
- Various types of Latches/flip-flops are:

  S-R (set-reset) type
  D (delay) type
  J-K type
  T (toggle) type

S-R Latch

1. It has a pair of cross-coupled NOR or NAND gates.
2. It has two inputs (S and R) and two outputs (Q and $\overline{Q}$).
3. The output can be set to 0 or 1 by applying suitable values on S and R inputs.

- The output of a NOR gate is 0 if any of the input is 1 and the output is 1 when only all the inputs are 0.



(a) Logic diagram

S	R	Q	Q'	
1	0	1	0	
0	0	1	0	(after S = 1, R = 0)
0	1	0	1	
0	0	0	1	(after S = 0, R = 1)
1	1	0	0	

(b) Truth table

Figure: S-R latch, (a) Logic diagram, (b) Truth table

S-R Latch

- Latch has two useful states. When $Q = 1$ and $\overline{Q} = 0 \Rightarrow$ set state. When $Q = 0$ and $\overline{Q} = 1 \Rightarrow$ clear state.
- When $S = 1$ and $R = 1$, $Q = 0$ and $\overline{Q} = 0$ which violates the definition of latch. Hence, $S = 1$ and $R = 1 \Rightarrow$ invalid state.
- $S = 0$ and $R = 0$ is a condition, which suppose to cause any change in the output of the S-R latch.
- First, we apply $S = 1$ and $R = 1$ then, $Q = 0$ and $\overline{Q} = 0$. Now apply $S = 0$ and $R = 0$ which makes the output to $Q = 1$ and $\overline{Q} = 1$. If the speed of the two gates are same then, output will oscillates.
- But in reality, two gates never have the same speed, one will be slightly slower than other. In that case we can not predict the output. The phenomenon is known as race condition.

S-R Latch

Race Condition

- A scenario where the final output depends on the relative speed of gates.
- If we apply $S = 1$ and $R = 1$, and then $S = 0$ and $R = 0$, the outputs will to either $Q = 0$, $\overline{Q} = 1$ or $Q = 1$, $\overline{Q} = 0$, depending upon the relative speeds of the two gates.

S-R Flip-flop

The S-R latch can be modified by providing an additional control input that determines when the state of the circuit is to be changed.

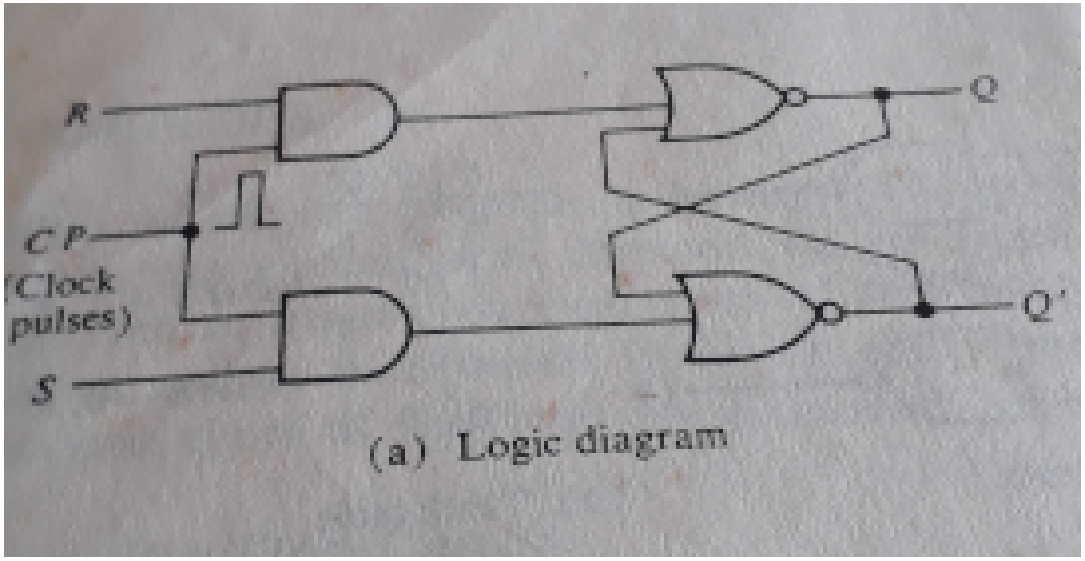


Figure: Logic diagram of S-R Flip-flop

- When the clock plus goes to 1, information from the S and R inputs is allowed to reach to the input of the basic latch.

S-R Flip-flop

The characteristic Table of the S-R flip-flop is given below:

Q(t)	S	R	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	indeterminate
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	indeterminate

- The set state is reached when $S = 1$, $R = 0$ and $CP = 1$.
- The attend the clear state, we have to apply $S = 0$, $R = 1$ and $CP = 1$.
- The characteristic equation of the S-R flip-flop is $Q(t+1) = S + \overline{R}Q(t)$

D Flip-Flop

- The logic diagram of D flip-flop is show below which is a modification of S-R flip-flop.

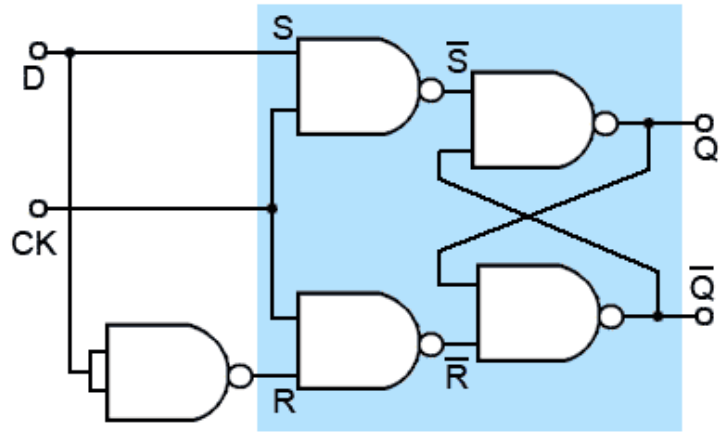


Figure: Logic diagram of D Flip-flop

- Right most two NAND gates form the basic S-R latch and next two NAND gates modify it into a clocked S-R flip-flop.
- The D input is applied directly to S input and its compliment is applied to the R input.

D Flip-Flop

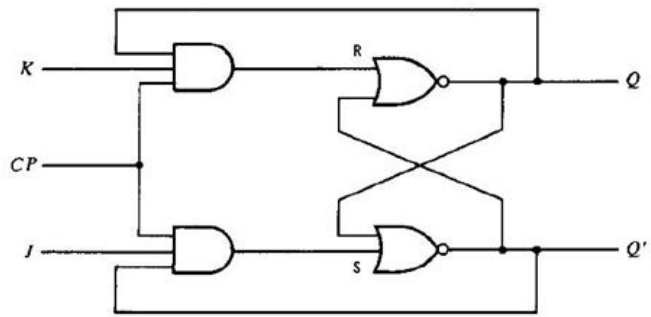
- As long as clock plus input is at 0, $\overline{S} = 0$ and $\overline{R} = 0$, regardless the value of the other inputs. This conforms that the two inputs of the basic S-R latch remain at 0 initially.
- The D input is sampled during the occurrence of a clock plus.
- When D = 1 and clock plus is also 1 then, $\overline{S} = 0$, $\overline{R} = 1$ and the flip-flop goes to set state.
- When D = 0 and clock plus is also 1 then, $\overline{S} = 1$, $\overline{R} = 0$ and the flip-flop goes to clear state.
- The characteristic table of D flip-flop is given below

Q(t)	D	Q(t+1)
0	0	0
0	1	1
1	0	0
1	1	1

- The Characteristic equation is $Q(t+1) = D$

J-K Flip-Flop

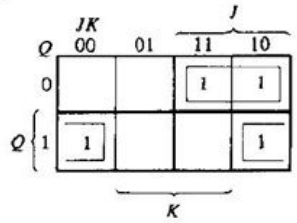
- J-K flip-flop is a refinement of the S-R flip-flop. The indeterminate state of the S-R flip-flop is defined in J-K flip-flop.
- When J = K = 1, the flip-flop switches to its complement state, that is if Q = 1, it goes to Q = 0, and vice versa.
- The Logic diagram and characteristic table of J-K flip-flop are shown below:



(a) Logic diagram

Q	J	K	Q(t + 1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

(b) Characteristic table



$Q(t+1) = JQ' + K'Q$

(c) Characteristic equation

J-K Flip-Flop

- Race in condition: Due to the feedback in J-K flip-flop, a CP signal which remains a 1 (while $J = K = 1$) after the outputs have been complement once will repeated and a continuous transition of the outputs.
- To avoid the undesirable operation, the clock pulse duration must be shorter than the propagation delay of J-K flip-flop.
- The restriction on clock pulse width can be eliminated with a master-slave or edge-triggered flip-flop.

T Flip-Flop

- The T flip-flop is a single-input version of J-K flip-flop. The T input is obtained from a JK type if both inputs are tied together.
- The T flip-flop goes its complement state when $T = 1$ and $CP = 1$.
- The logic diagram of the T flip-flop is shown below.

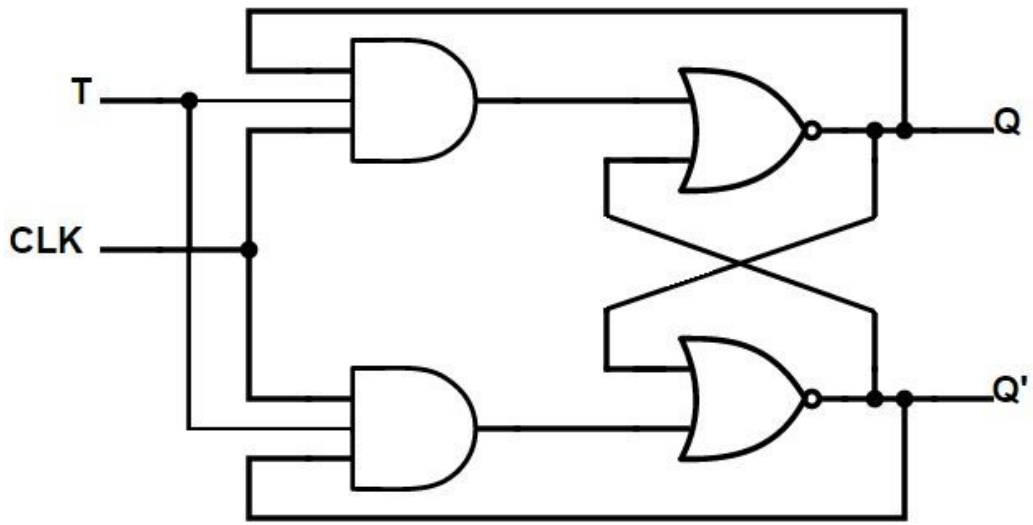


Figure: Logic diagram of T Flip-flop

Triggering of Flip-Flop

- The state of a flip-flop is changed by a momentary change in the input signal. This momentary change is called a trigger.
- Clocked flip-flops are triggered by pulses. A pulse starts from an initial value 0, goes to 1, and after a short time, returns to its initial 0 value.
- A flip-flop changes its state at rising edge of the clock pulse then, it is called positive edge trigger flip-flop. If the a flip-flop changes its state at falling edge of the clock pulse then, it is called negative edge trigger flip-flop.

Edge Trigger D Flip-Flop

- In this type of flip-flop, the output transitions occur at a specific level of the clock pulse. When the pulse input level exceeds this threshold level, the inputs are locked out and flip-flop is therefore unresponsive to further changes in inputs until the clock pulse returns to 0 and another pulse occurs.
- The logic diagram of a D-type positive edge-trigger is shown below.

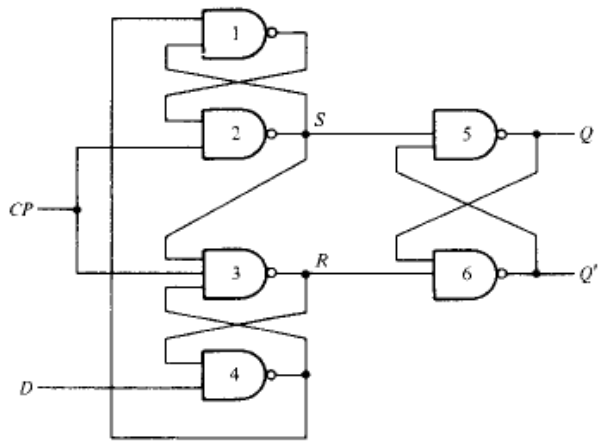


Figure: Logic diagram of D type positive-edge triggered Flip-flop

Edge Trigger D Flip-Flop

- The circuit consists of three basic latches. NAND gates 1 and 2 make up one basic latch and gates 3 and 4 form another latch. The third latch consisting of gates 5 and 6 provides the outputs of the circuit.
- The input S and R of the third latch must be maintained at logic -1 for the outputs to remain in their steady-state values.
- When S = 0 and R = 1, the output goes to set state with Q = 1.
- When S = 1 and R = 0, the output goes to clear state with Q = 0.
- The inputs S and R are determined from the states of the other two basic latches.

Edge Trigger D Flip-Flop

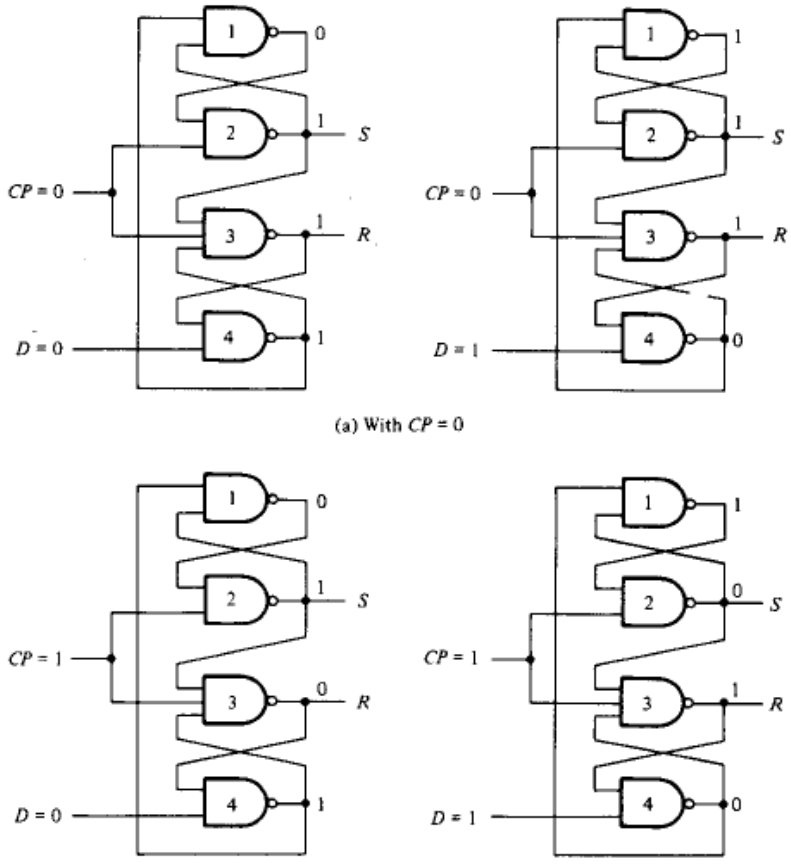


Figure: Operation of the D type positive-edge triggered Flip-flop

Edge Trigger D Flip-Flop

- When CP = 0 and D = 0 or D = 1, in either case outputs of gates 2 and 3 are equal to 1, thus making S = R = 1, which is the condition for a steady-state output.
- When D = 0, the output of gate 4 is at logic 1, which causes the output of gate 1 equal to 0.
- When D = 1, the output of gate 4 is at logic 0, which causes the output of gate 1 equal to 1.
- There is a definite time, called *setup time*, in which the D input must be maintained at a constant value prior to the application of the pulse.
- The *setup time* is equal to the propagation delay to through gates 4 and 1 since a change in D causes a change in the outputs of these two gates.

Edge Trigger D Flip-Flop

- Assume that D does not change during the *setup time* and CP becomes 1. If D = 0 and CP =1, then S remains in at 1 but R changes from 1 to 0. This causes output Q goes to 0.
- Now if while CP = 1, there is a change in the D input, the output of gate 4 will remain at 1 (even if D goes to 1).
- There is definite time, called the *hold time*, that the D input must not change after application of clock pulse. The *hold time* is equal to the propagation delay of gate 3, since it must ensure that R becomes 0 in order to maintain the output of gate 4 at 1, regardless the value of D.
- If D = 1 and CP = 1, then S = 0 and R = 1, which causes the output Q = 1.

Design of Synchronous Sequential Circuits

October 12, 2020

Introduction

- In case of a sequential circuit, the outputs depend not only on the present inputs but also on the internal state of the circuit.
- The internal state also changes with time.
- The number of states of a circuit is finite, and hence, a sequential circuit is also called a Finite State Machine (FSM).
- Most of the practical circuits are sequential in nature.
- In case of combinational circuit, truth table, Boolean expressions can be used to represent the behaviour of the circuit. Similarly, a FSM can be represented either in the form of state table or state diagram.

Examples

- A circuit to detect 3 or more consecutive 1's in a serial bit stream.
- The bits are applied serially in synchronism with clock.
- The output will become 1 when it detects 3 or more consecutive 1's in the stream.

Input	0 1 1 0 1 1 1 1 0 1 0 1 1 1
output	0 0 0 0 0 0 1 1 0 0 0 0 0 1

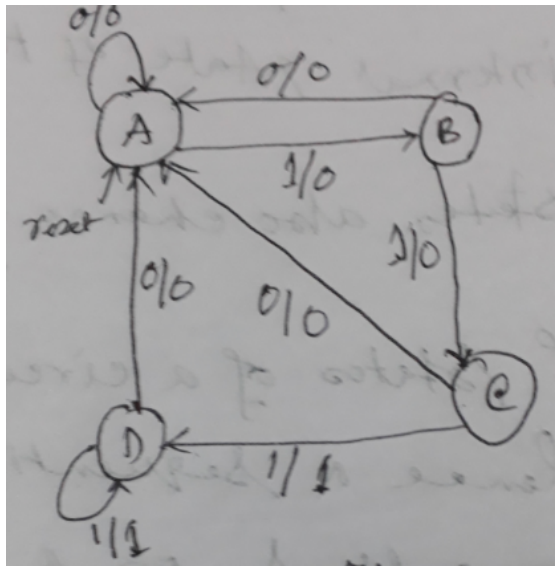


Figure: State Diagram of the proposed circuit

Examples

Table: State Table of the Proposed Circuit

	I/P = 0		I/P = 1	
PS	NS	O/P	NS	O/P
A	A	0	B	0
B	A	0	C	0
C	A	0	D	1
D	A	0	D	1

Mealy and Moore FSM

- Formal definition of FSM: A deterministic FSM can be mathematically defined as a 6 tuples: $(\Sigma, \Gamma, S, s_0, \delta, \omega)$
- $\Sigma \Rightarrow$ Set of input combinations. $\Gamma \Rightarrow$ Set of output combinations. $S \Rightarrow$ A finite set of states. $s_0 \subset S \Rightarrow$ is the initial state. $\delta \Rightarrow$ is the state transition function (it gives the next state). $\omega \Rightarrow$ is the output function.
- $\delta : S \times \Sigma$ - Present state and present inputs determine the next state (NS).
- $\omega : S \times \Sigma \rightarrow \Gamma$ For Mealy machine (output depends on PS + Inputs)
- For Moore machine: $\omega : S \rightarrow \Gamma$ (output depends on the present state).
- Pictorial Depiction.

Design of Synchronous Sequential Circuits

- From a description of the problem, obtain a state table or state diagram.
- Check whether the table contains any redundant states; if so remove them.
- Select a state assignment and determine the type of Flip-flop.
- Derive transition and output tables.
- Derive the excitation table and obtain excitation and output functions.
- Minimize the function and obtain the circuit diagram.

Construction of State diagram/Table from Specification

- A state diagram specifies the different states of a FSM, the condition under which state changes occur, and the corresponding output.
- States are denoted as circles, and labelled with unique symbols.
- Transitions are represented as directed arrows between pair of states.
- Each transition is labelled by α/β, where α denotes input combination and β denotes output combination.
- A state table is an alternative way of representing state diagram.
- For each value of present state, it specifies the next state and output for every input combination.

Example -1

Design a serial parity detector

- A continuous stream of bits is fed to a circuit in synchronism with clock. The circuit will generating a bit stream as output, where a 0 will indicate even number of 1's seen so far and a 1 will indicate odd numbers of 1's seen so far.
- The state diagram of the aforesaid problem is given below:

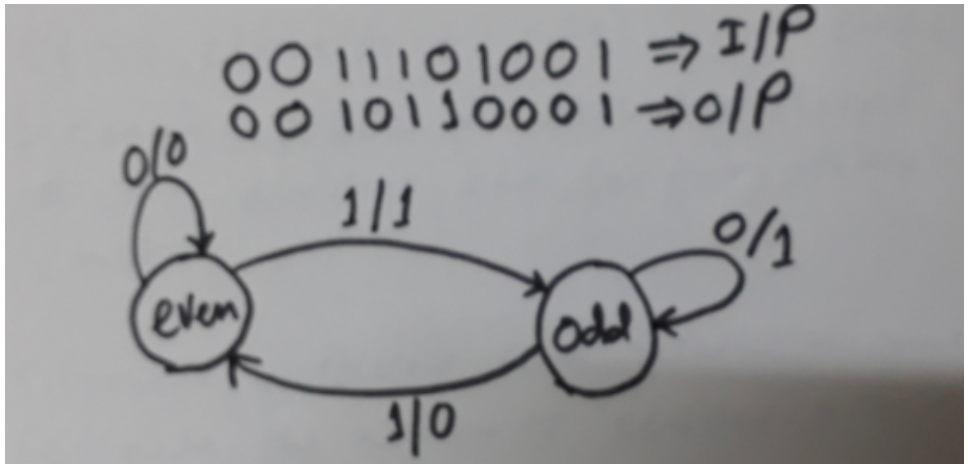


Figure: State Diagram of the proposed circuit

Table: State Table of the Proposed Circuit

	I/P = 0		I/P = 1	
PS	NS	O/P	NS	O/P
even	even	0	odd	1
odd	odd	1	even	0

Example-2

Design a sequence detector

- A circuit accepts a serial bit stream 'X' as input and produce a serial bit stream 'Z' as output.
- Whenever the bit pattern '0110' appears in the input stream, its output $Z = 1$ and at all other times, $Z = 0$.
- Overlapping occurrences of the patterns are also detected.
- This is an example of Mealy Machine.
- The state diagram of the proposed circuit is given below:

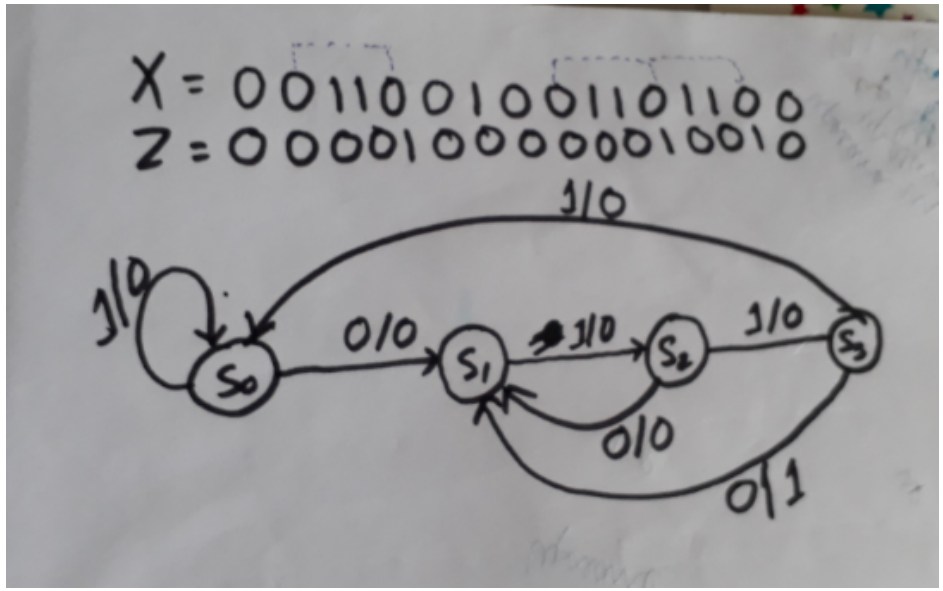


Figure: State Diagram of the proposed circuit

State Table Minimization

- Basic Concept:
  - Identify equivalent states and merge them into a single state.
  - Can lead to more compact representation.
- State Equivalence: Two states S_i and S_j are said to be equivalent if and only if for every single input sequence X, the outputs are the same and the next states are equivalent.
  $\omega(S_i, X) = \omega(S_j, X)$ and $\delta(S_i, X) \equiv \delta(S_j, X)$
  Where $\omega(S_i, X)$ is the output for the present state S_i and input X and $\delta(S_i, X)$ is the corresponding next state.

Steps for Minimization using Implication Table

1. Construct an Implication table that contains a square for every pair of states.
2. Compare each pair of rows in the state table, if the output associated with state S_i and S_j are different, place 'X' in square S_i - S_j to indicate S_i and S_j are not equivalent. If the outputs are same, place the implied pair in the square S_i - S_j. (if the next states of S_i and S_j are m and n respectively for some input X, then m - n is an implied pair.
3. Go through the table square by square if square S_i - S_j contains the implied pair m -n and square m - n contains a 'X' then, S_i and S_j are not equivalent and place 'X' in square S_i - S_j.
4. If any 'X' is added in step 3, repeat step 3 until no more 'X' can be added.
5. For each square S_i - S_j that does not contain 'X', the states S_i and S_j are equivalent.

Example

Consider the following state table:

Table: An example state table

	I/P = 0		I/P = 1	
PS	NS	O/P	NS	O/P
a	a	0	b	0
b	c	0	d	0
c	a	0	d	0
d	e	0	f	1
e	a	0	f	1
f	g	0	f	1
g	a	0	f	1

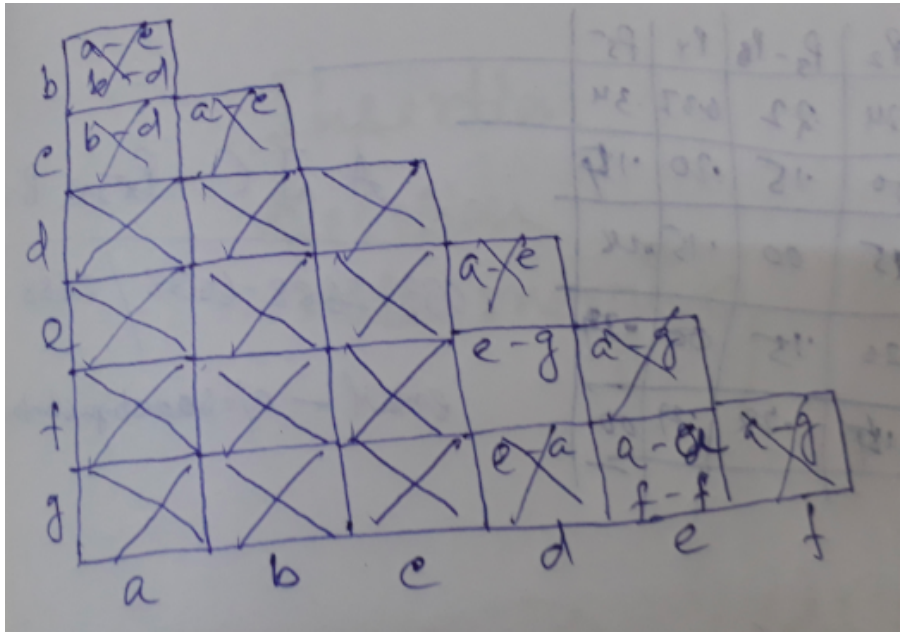


Figure: Implication table for the above mention state table

Serial Adder Design

- A serial adder has two external inputs and one external outputs.
- The state table and state diagram are shown below:

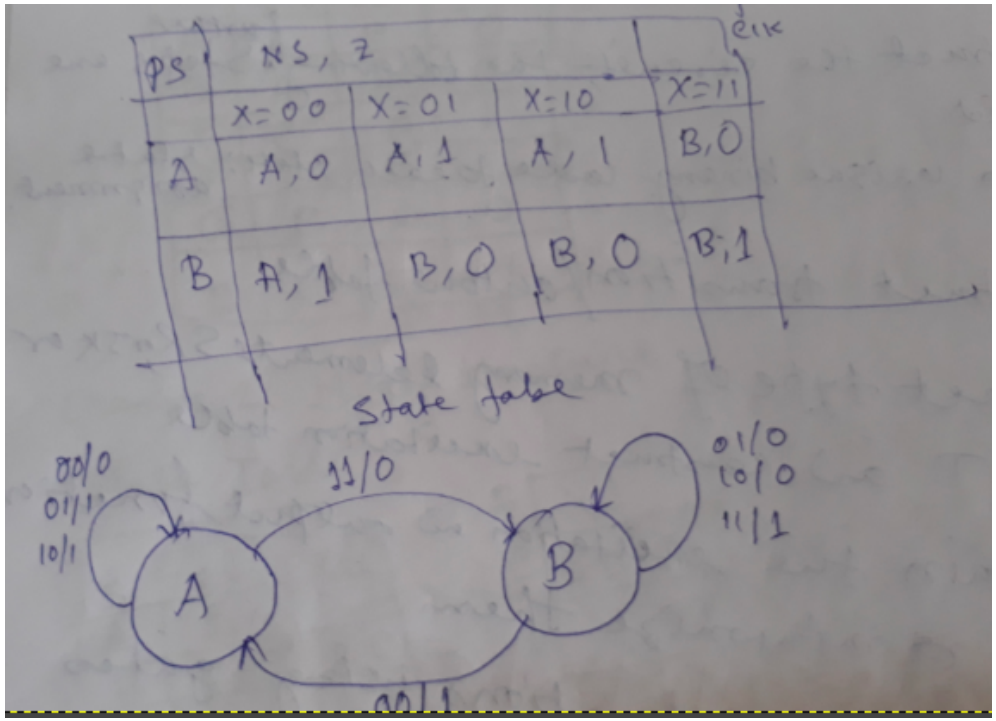


Figure: State table and state diagram of serial adder.

State Assignment and Transition/output Table

- There are two states and so one bit is sufficient. Suppose A = 0 and B = 1. Note that this not unique.
- Transition/output Table is shown below:

Table: Transition/output Table of a Serial Adder.

	NS				Output (Z)			
PS	00	01	10	11	00	01	10	11
0	0	0	0	1	0	1	1	0
1	0	1	1	1	1	0	0	1

- Selection of memory element: Suppose we use D flip-flop

Excitation and Output Table

- The excitation/output table is show below:

Table: Transition/output Table of a Serial Adder.

PS(y)	X_1	X_2	NS	D	Z
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	1	0
1	1	0	1	1	0
1	1	1	1	1	1

- The expression for $Z = X_1 \oplus X_2 \oplus y$ and $D = X_1X_2 + X_1y + X_2y$

Example 2: Design of a Sequence Detector

- A circuit accepts a serial bit stream 'X' as input and produce a serial bit stream 'Z' as output.
- Whenever the bit pattern '0110' appears in the input stream, its output $Z = 1$ and at all other times, $Z = 0$.
- Overlapping occurrences of the patterns are also detected.
- This is an example of Mealy Machine.
- The state diagram of the proposed circuit is given below:

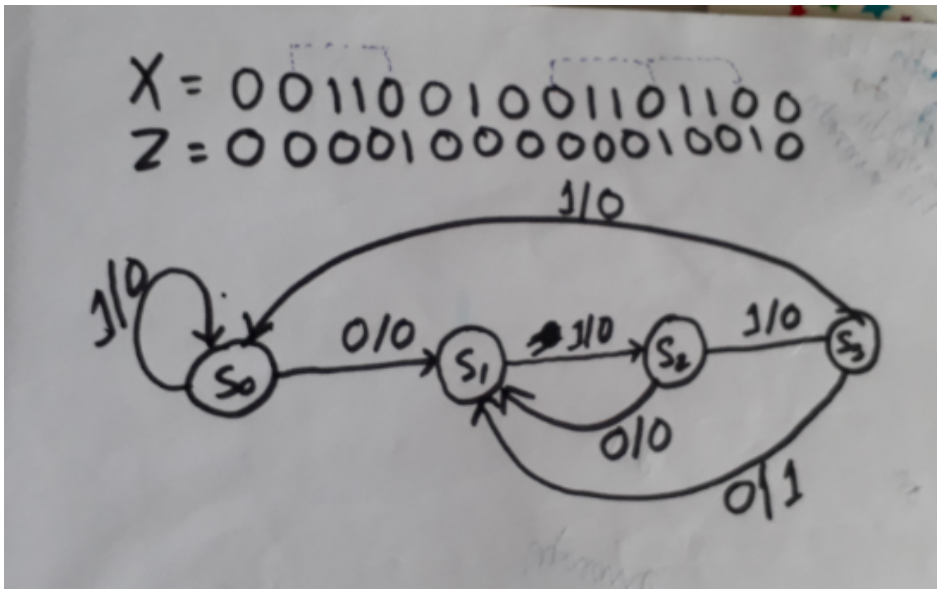


Figure: State Diagram of the proposed circuit

Example 2: Design of a Sequence Detector

- The state table of the sequence detector is as follows:

PS	NS, Z	
	$X = 0$	$X = 1$
S_0	S_1, 0	S_0, 0
S_1	S_1, 0	S_2, 0
S_2	S_1, 0	S_3, 0
S_3	S_1, 1	S_0, 0

- The states are assigned as $S_0 = 00$, $S_1 = 01$, $S_2 = 10$, and $S_3 = 11$. The transition/output table is shown below:

PS	NS		Z	
	$X = 0$	$X = 1$	$X = 0$	$X = 1$
00	01	00	0	0
01	01	10	0	0
10	01	11	0	0
11	01	00	1	0

Example 2: Design of a Sequence Detector

- Suppose we use T flip-flop as memory element, then the excitation table is as follows:

X	y_1	y_2	Y_1	Y_2	T_1	T_2	Z
0	0	0	0	1	0	1	0
0	0	1	0	1	0	0	0
0	1	0	0	1	1	1	0
0	1	1	0	1	1	0	1
1	0	0	0	0	0	0	0
1	0	1	1	0	1	1	0
1	1	0	1	1	0	1	0
1	1	1	0	0	1	1	0

The expressions for T_1, T_2 and Z are as follows:
$T_1 = \overline{X}y_1 + Xy_2$, $T_2 = \overline{X}\overline{y_2} + Xy_2 + y_1y_2$ and $Z = \overline{X}y_1y_2$

Design of Registers

November 30, 2020

Introduction

- A register consists of a group of flip-flops with a common clock input, used for storing binary information (data).
- Depending on the configuration, there can be several different variations.
  - Parallel-in-parallel-out (PIPO)
  - Serial-in-serial-out (SISO)
  - Parallel-in-serial-out (PISO)
  - Serial-in-parallel-out (SIPO)
- If a register supports either serial-in or serial-out or both modes, then the register is called a shift register.

Basic PIPO Register

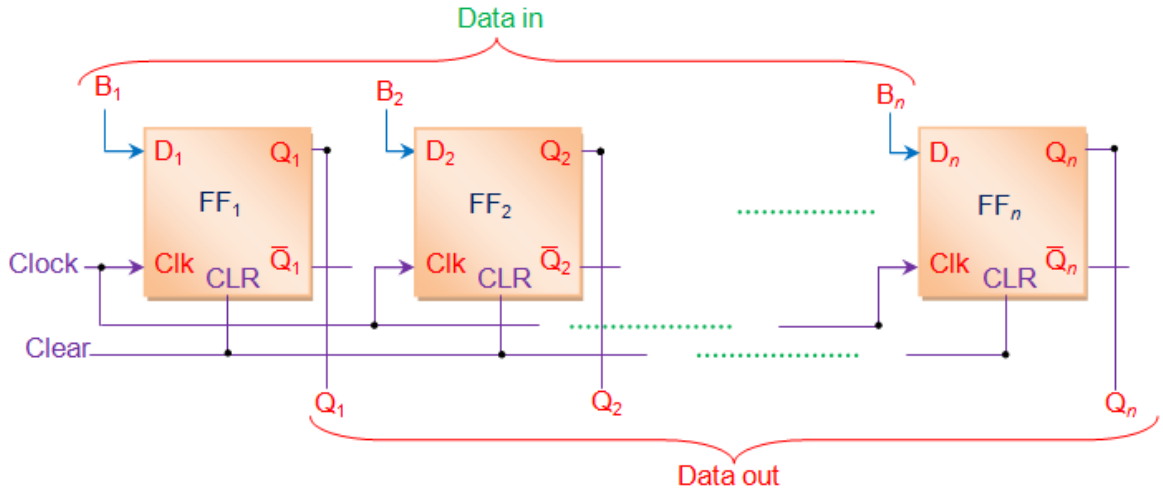


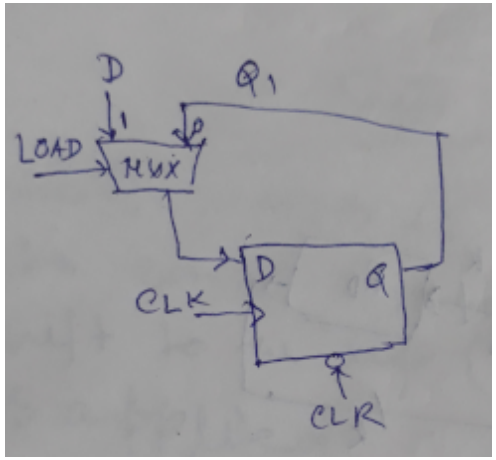
Figure 1 *n*-bit Parallel-In Parallel-Out Register

Figure: n-bit parallel-in-parallel-out register.

- Parallel inputs are fed to the D-inputs of the flip-flops.
- When clear $= 1$, the outputs of all flip-flops are at zero.
- Clock input is fed in parallel to all flip-flops.
- When the active clock edge comes (positive or negative edge), the available data $(B_1, B_2, \ldots B_n)$ in the inputs is stored in the register and also available at its outputs.

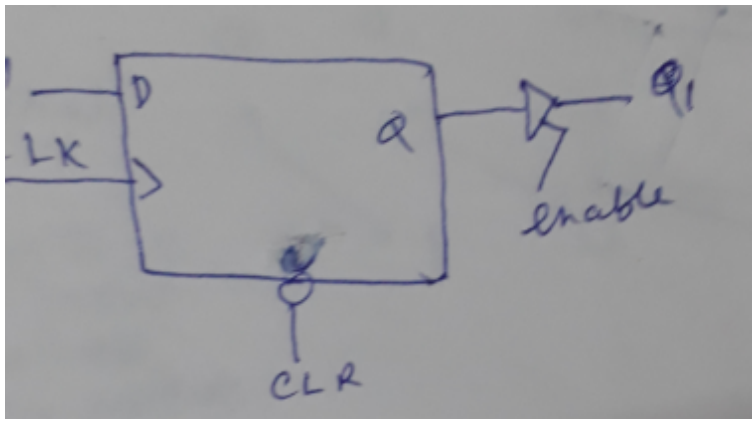
Addition of LOAD Signal with Basic PIPO

- In practice, the clock is coming continuously and there is a separate signal LOAD that specifies when the register is to be loaded with new data.
- There are two possible solutions:
  (a) Use a gated clock: Not a good solution, as gating the clock with another signal can cause timing problem. Load has to come before clock is arrived.
  (b) Separate out clock and LOAD using multiplexer circuit.
  - Better and recommended solution.
  - Each flip-flop in the register gets replaced by



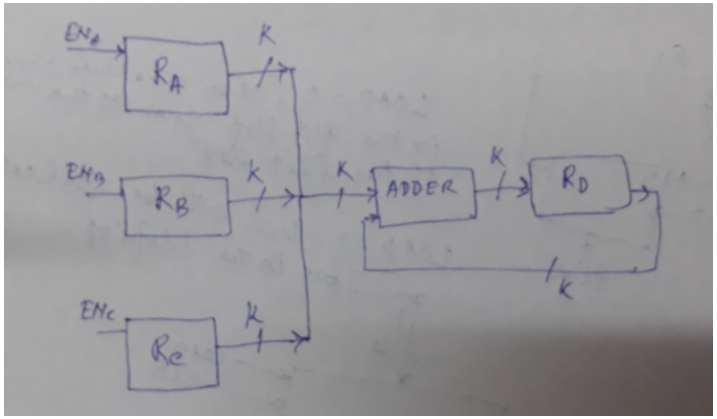
Addition of Enable input with the PIPO register

- Many registers have an enable input that can be used to force the output lines into high impedance state, if required.
- We need tri-state buffer with every flip-flop output. When enable $= 1$, $Q_1 = Q$ and enable $= 0$, the output is at high impedance state.



- Why enable input is required?
  To resolve bus conflicts in bus-based architectures.

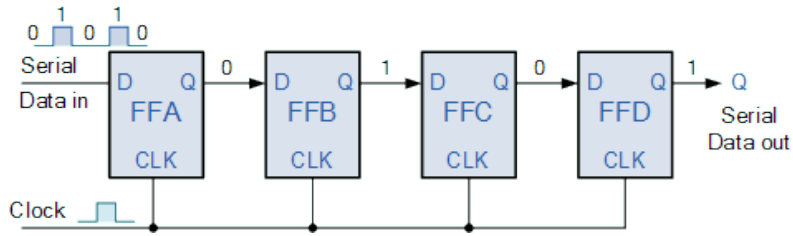
An Example Scenario



- R_A, R_B, R_C, and R_D are k-bit registers.
- Exactly one of the three enable inputs must be activated at any given time.
- Clock is fed in parallel to all the registers.
- If $EN_A = 1$, then $R_D = R_D + R_A$
- If $EN_B = 1$, then $R_D = R_D + R_B$
- If $EN_C = 1$, then $R_D = R_D + R_C$

Shift Register

- A shift register is register in which the binary data can be stored and the data can be shifted to the left (or right) when a shift signal is applied.
- It can be constructed by connecting D, SR or JK filp-flops in cascade.
- A 4-bit shift register is shown below.



- The first FF receives data from external world. The output of the 1^{st} FF is connected to the input of the 2^{nd} FF and so on. We get the serial output from the 4^{th}

Shift Register

- A sample timing diagram is shown below assuming that the initial state is $Q_1 Q_2 Q_3 Q_4 = 0101$

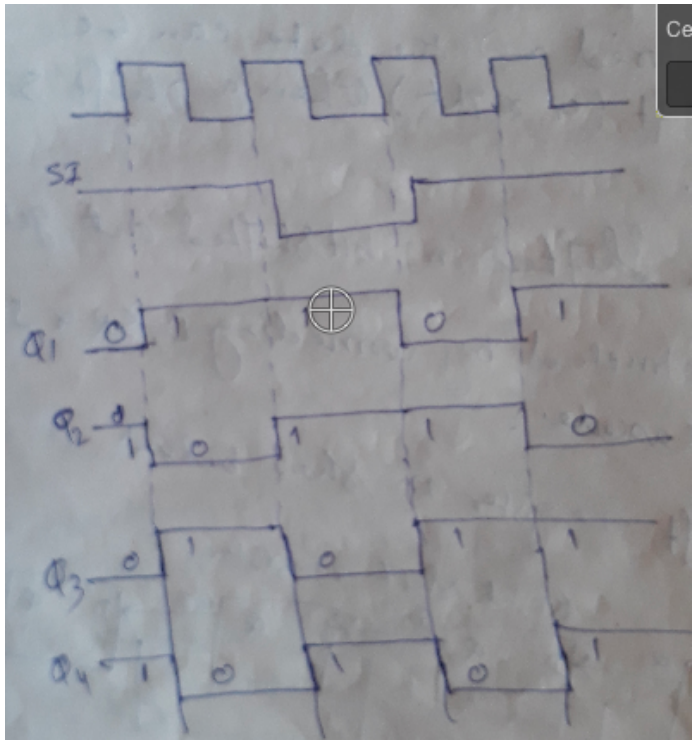


Figure: Timing Diagram

Different Types of Shift Registers

Depending upon the ways the various storage devices are connected, there can different types of shift-register:

(a) Ring counter

(b) Twisted ring or Johnson counter

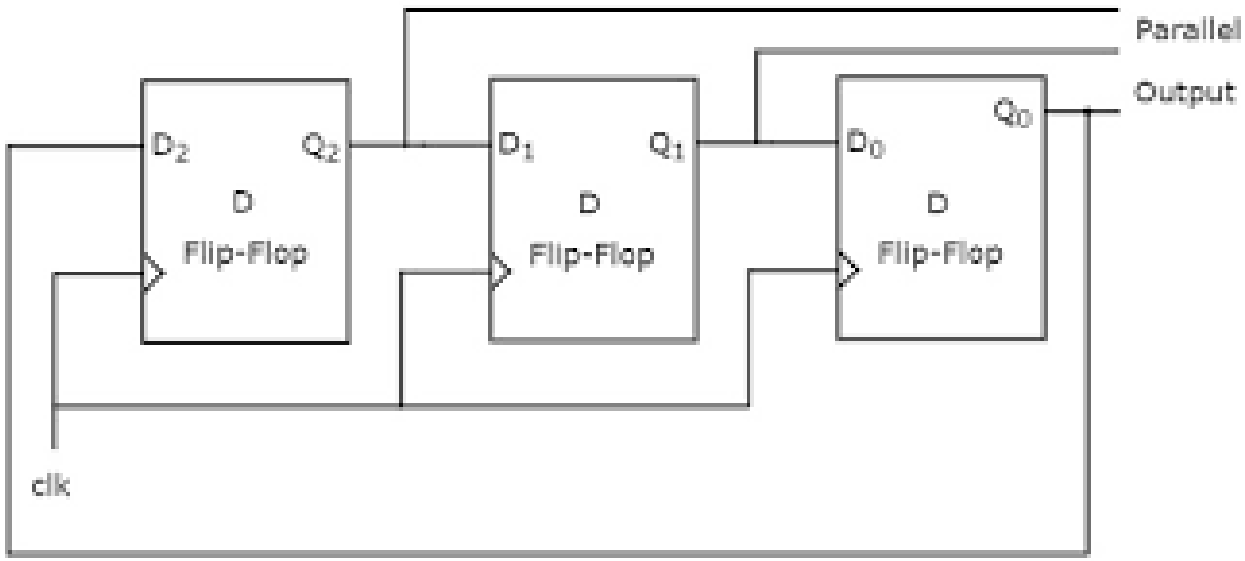
(c) Bidirectional shift register

(d) Universal shift register

(e) Liner feedback shift register

Ring Counter

- This is obtained from SISO shift register by connecting the Q output of the last FF to the D input of the first FF.
- Typically a ring counter is initially with a single 1 and all FFs remain 0's
- This can generate multi-phase clock.
- For a k-bit ring counter, the contents of the register gets repeated after k clocks



Timing Diagram of a 3-bit Ring Counter

A 3 bit ring counter assumes $Q_1 Q_2 Q_3 = 100$ the initial state.

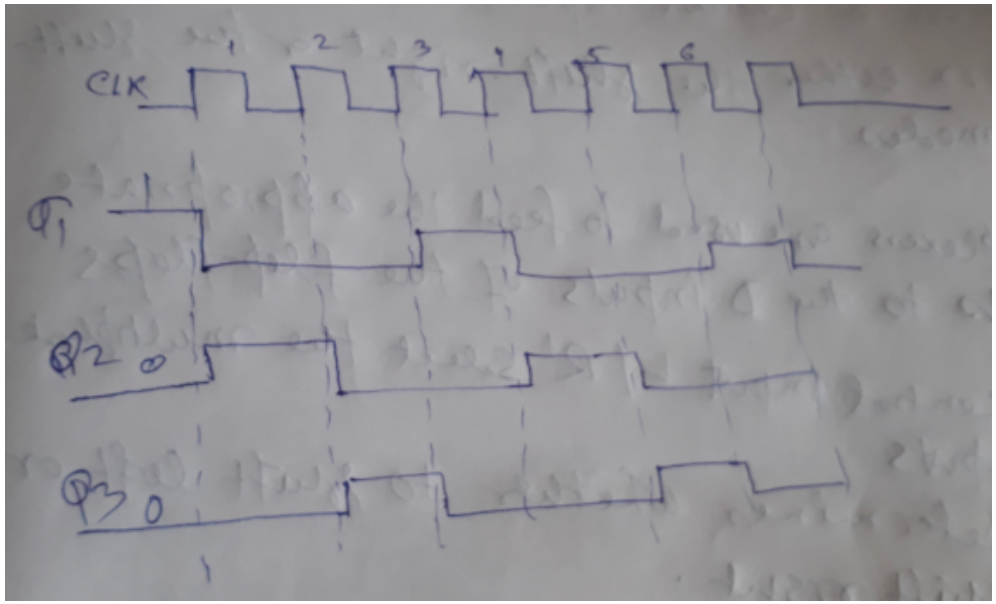
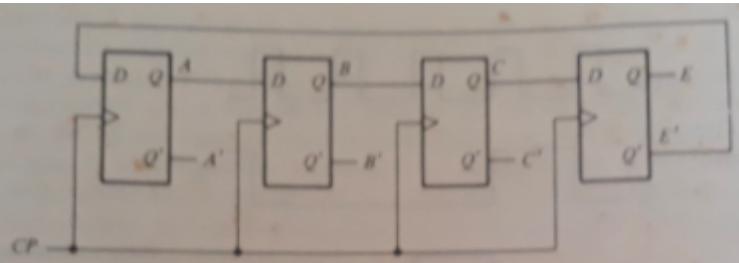


Figure: Timing diagram of a 3 bit ring counter.

Johnson Counter

1. This is obtained from a SISO shift register by connecting the $\overline{Q}$ output of the last flip-flop to the D input of the first flip-flop.
2. The Johnson counter can be initialized to the all 0 state.
3. For a k-bit Johnson counter, contents of the register gets repeated after 2k clocks.

Johnson Counter



(a) 4-stage switch-tail ring counter

Sequence number	Flip-flop outputs A	B	C	E	AND gate required for output
1	0	0	0	0	$A'E'$
2	1	0	0	0	AB'
3	1	1	0	0	BC'
4	1	1	1	0	CE'
5	1	1	1	1	AE
6	0	1	1	1	$A'B$
7	0	0	1	1	$B'C$
8	0	0	0	1	$C'E$

(b) Count sequence and required decoding.

Bidirectional Shift register

- Base on the control signal, the shift register works in either the shift-right mode or shift-left mode.
- Multiplexers are used to feed the appropriate signals to the D inputs of the flip-flops.
- A control signal $L/\overline{R}$ select the multiplexer inputs. Determine whether to shift left or shift right.
  $L/\overline{R} = 1 \Rightarrow$ left shift
  $L/\overline{R} = 0 \Rightarrow$ right shift
- $L/\overline{R} = 0$, then 0 input of the MUX will be selected. SI will come to D_1, Q_1 will come to D_2 and so on.
- $L/\overline{R} = 1$, SI will come to D_4, Q_4 will come to D_3 and so on.

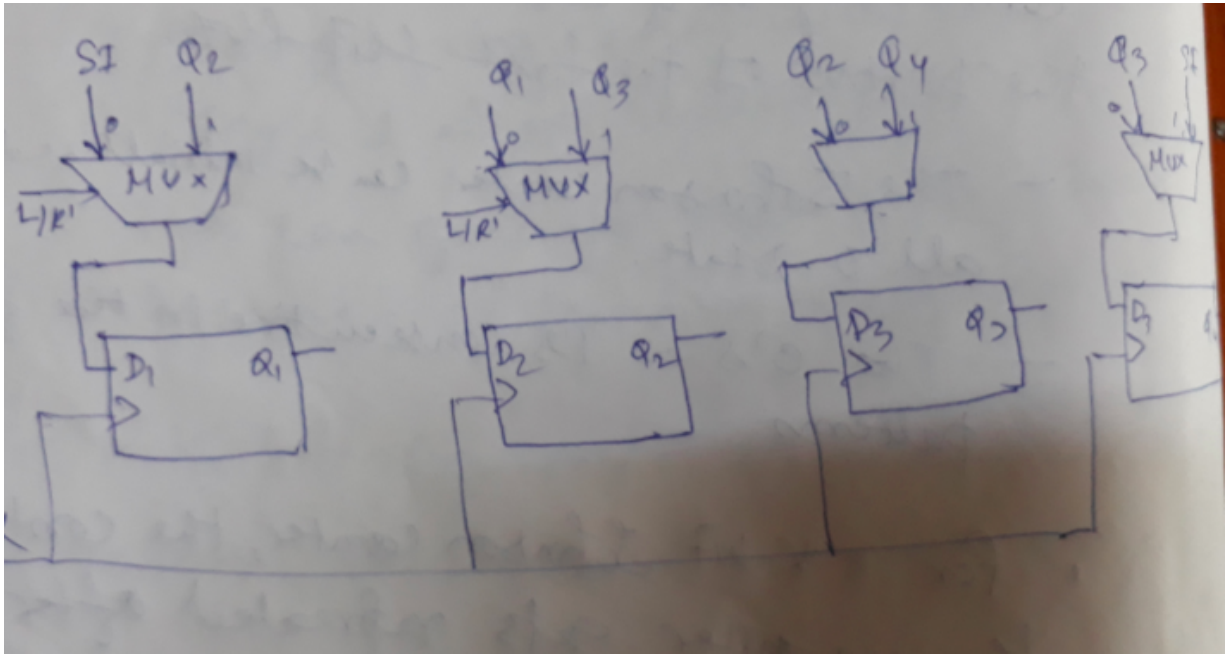


Figure: Bidirectional Shift Register.

Universal Shift Register

A Universal shift register is a bidirectional shift register, whose input can be either in serial or in parallel and whose output also be either in serial or in parallel.

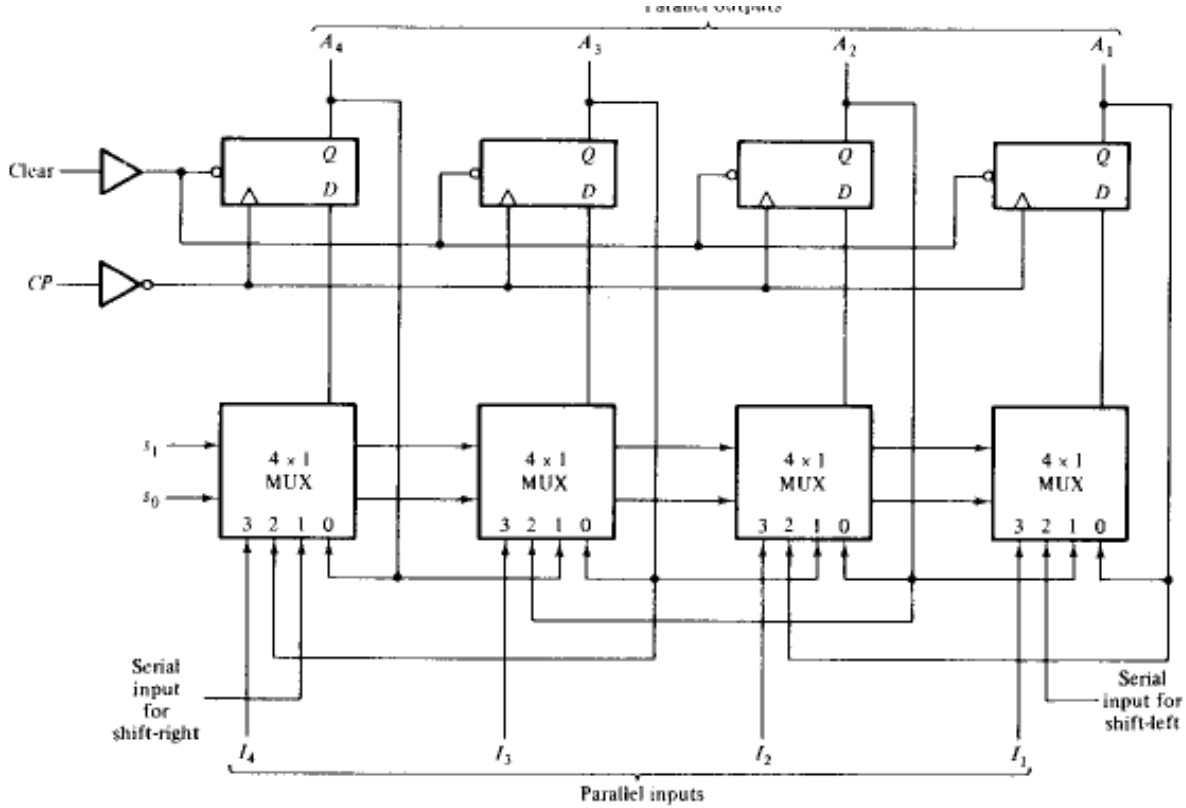


Figure: Universal Shift Register.

Table: Function Table for the Universal Register.

Mode control		Operation
s_1	s_0	
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel loading

Linear feedback shift register (LFSR)

- It is basically a SISO right shift register where the serial input is generated as a linear combination of some of the FF outputs.

Table: Transition sequences of the LFSR.

CLK$\rightarrow$	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Q_1	0	1	1	1	1	0	1	0	1	1	0	0	1	0
Q_2	0	0	1	1	1	1	0	1	0	1	1	0	0	1
Q_3	0	0	0	1	1	1	1	0	1	0	1	1	0	0
Q_4	1	0	0	0	1	1	1	1	0	1	0	1	1	0

- The circuit is used to generate random patterns.
- The LFSR is typically initialized to 1 0 0 0.

Excitation and Output Table

- The circuit diagram of LFSR is given below:

bit states except for (0000).

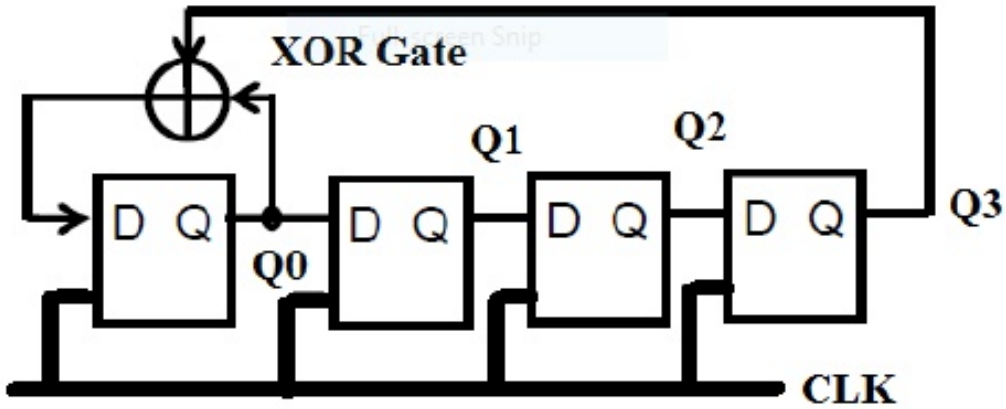


Figure: Linear Feedback Shift Register.