POLYTECHNIC 2nd SEMESTER

Fundamentals of Electrical & Electronics Engineering

CHAPTER NOTES

OVERVIEW OF

DIGITAL ELECTRONICS

Boolean Algebra • Logic Gates • Flip-Flops • Counters • TTL ICs

1. Boolean Algebra Laws, Logic Gates (AND/OR/NOT/NAND/NOR/XOR/XNOR), Universal Gates
2. De Morgan's Theorem, SOP/POS, Karnaugh Map Simplification
3. Basic Concept of Flip-Flops as Storage Elements (SR, D, JK, T)
4. Counters — Ripple (Asynchronous) and Synchronous Up/Down Counters
5. Introduction to TTL Digital IC Logic Gates

www.polynoteshub.co.in

1. Boolean Algebra, Logic Gates & Universal Gates

1.1 Introduction to Boolean Algebra

Boolean Algebra is a branch of mathematics developed by George Boole in 1854, which deals with binary variables that can take only two values: 0 (FALSE / LOW) and 1 (TRUE / HIGH). It forms the mathematical foundation for the analysis and design of all digital logic circuits, including computers, microprocessors, and digital communication systems. In digital electronics, Boolean algebra is used to represent, simplify, and implement logic functions using logic gates.

Binary Variables: In Boolean algebra, variables can only be 0 or 1. Operations are performed using three basic operations: AND (·), OR (+), and NOT (complement /overbar). All complex digital circuits are combinations of these three fundamental operations.

1.2 Basic Laws and Rules of Boolean Algebra

Boolean algebra follows a set of fundamental laws and theorems that are used to simplify and manipulate logic expressions. These are grouped below:

(a) Basic Identity Laws

Law	AND Form	OR Form	Explanation
Identity Law	A · 1 = A	A + 0 = A	AND with 1 / OR with 0 leaves variable unchanged
Null / Annulment	A · 0 = 0	A + 1 = 1	AND with 0 always gives 0; OR with 1 always gives 1
Idempotent Law	A · A = A	A + A = A	A variable ANDed or ORed with itself equals itself
Complement Law	A · A' = 0	A + A' = 1	A variable ANDed or ORed with its complement
Double Negation	A'' = A	—	Double complement returns the original variable

(b) Commutative, Associative & Distributive Laws

Law	Boolean Expression	Meaning
Commutative (AND)	A · B = B · A	Order of ANDing does not matter
Commutative (OR)	A + B = B + A	Order of ORing does not matter
Associative (AND)	(A·B)·C = A·(B·C)	Grouping in AND does not affect result
Associative (OR)	(A+B)+C = A+(B+C)	Grouping in OR does not affect result

Distributive 1	A·(B+C) = A·B + A·C	AND distributes over OR
Distributive 2	A+(B·C) = (A+B)·(A+C)	OR distributes over AND

(c) Absorption and Consensus Laws

Law	Expression	Simplified Form
Absorption 1	A + A·B	= A
Absorption 2	A·(A + B)	= A
Redundancy 1	A + A'·B	= A + B
Redundancy 2	A·(A' + B)	= A·B
Consensus	AB + A'C + BC	= AB + A'C (BC is redundant)

1.3 Logic Gates — Symbols, Expressions & Truth Tables

Logic gates are the fundamental building blocks of digital circuits. Each gate performs a specific Boolean operation on one or more binary inputs and produces a binary output. The seven basic logic gates are described below with their Boolean expressions and truth tables.

(a) AND Gate

The AND gate produces an output of 1 only when ALL inputs are 1 simultaneously. If any input is 0, the output is 0. It performs the Boolean multiplication operation.

Y = A · B (also written as Y = AB)

AND Gate Boolean Expression

A	B	Y = A·B
0	0	0
0	1	0
1	0	0
1	1	1

Analogy: Two switches in SERIES — lamp lights only when BOTH switches are closed.

(b) OR Gate

The OR gate produces an output of 1 when ANY one or more of its inputs are 1. The output is 0 only when ALL inputs are 0. It performs the Boolean addition operation.

$$Y = A + B$$

OR Gate Boolean Expression

A	B	Y = A+B
0	0	0
0	1	1
1	0	1
1	1	1

Analogy: Two switches in PARALLEL — lamp lights when ANY switch is closed.

(c) NOT Gate (Inverter)

The NOT gate is a single-input gate that inverts or complements the input. If the input is 1, the output is 0, and vice versa. It is also called an inverter.

$$Y = A' \quad (\text{also written as } Y = \bar{A} \text{ or } Y = \neg A)$$

NOT Gate Boolean Expression

A	Y = A'
0	1
1	0

(d) NAND Gate

The NAND gate is a combination of AND followed by NOT. It produces a 0 output only when ALL inputs are 1; otherwise the output is 1. NAND is the complement of AND. It is one of the two Universal Gates — any logic function can be built using only NAND gates.

$$Y = (A \cdot B)' = (AB)'$$

NAND Gate Boolean Expression

A	B	Y = (AB)'
0	0	1
0	1	1
1	0	1
1	1	0

(e) NOR Gate

The NOR gate is a combination of OR followed by NOT. It produces a 1 output only when ALL inputs are 0; otherwise the output is 0. NOR is the complement of OR. It is the second Universal Gate — any logic function can be built using only NOR gates.

$$Y = (A+B)'$$

NOR Gate Boolean Expression

A	B	Y = (A+B)'
0	0	1
0	1	0
1	0	0
1	1	0

(f) Exclusive-OR Gate (Ex-OR / XOR)

The XOR gate produces an output of 1 when the number of 1s at the input is ODD (i.e., inputs are different). For a 2-input gate, the output is 1 when exactly one input is 1. It is widely used in arithmetic circuits (adders) and parity generators.

$$Y = A \oplus B = A'B + AB'$$

Ex-OR Gate Boolean Expression

A	B	Y = A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

(g) Exclusive-NOR Gate (Ex-NOR / XNOR)

The XNOR gate is the complement of XOR. It produces an output of 1 when the inputs are equal (both 0 or both 1). It is also called an equivalence gate, used for comparator circuits and error detection.

$$Y = (A \oplus B)' = AB + A'B'$$

Ex-NOR Gate Boolean Expression

A	B	Y = (A⊕B)'
0	0	1
0	1	0

1	0	0
1	1	1

1.4 Universal Logic Gates

A Universal Gate is a gate that can be used to implement any Boolean function without the need for any other type of gate. Both NAND and NOR are universal gates — they are functionally complete, meaning AND, OR, NOT, and all other gates can be constructed using only NAND (or only NOR) gates. This is of great practical importance because it means a circuit can be built using only one type of gate, simplifying manufacturing and reducing cost.

NAND Gate as Universal Gate

Implementing Basic Gates using only NAND Gates

```
NOT gate:   A ──[NAND]── A'  (both inputs of NAND tied together)

AND gate:   A,B ──[NAND]──[NAND]── AB  (invert the NAND output)

                OR gate:    A ──[NAND(A,A)]──┐
                         B ──[NAND(B,B)]──┤──[NAND]── A+B
                          (invert both inputs, then NAND)
```

NOR Gate as Universal Gate

Implementing Basic Gates using only NOR Gates

```
NOT gate:   A ──[NOR]── A'  (both inputs of NOR tied together)

OR gate:    A,B ──[NOR]──[NOR]── A+B  (invert the NOR output)

                AND gate:   A ──[NOR(A,A)]──┐
                         B ──[NOR(B,B)]──┤──[NOR]── AB
                          (invert both inputs, then NOR)
```

1.5 Gate Summary Table

Gate	Symbol	Expression	Output=1 when	Universal?
AND	D-shape body	Y = AB	ALL inputs = 1	No
OR	Curved body	Y = A+B	ANY input = 1	No
NOT	Triangle + bubble	Y = A'	Input = 0	No
NAND	AND + bubble	Y = (AB)'	NOT all inputs = 1	YES
NOR	OR + bubble	Y = (A+B)'	ALL inputs = 0	YES
XOR	Curved + arc	Y = A⊕B	Inputs are DIFFERENT	No

XNOR	XOR + bubble	Y = (A⊕B)'	Inputs are EQUAL	No

2. De Morgan's Theorem, SOP/POS & Karnaugh Map Simplification

2.1 De Morgan's Theorems

De Morgan's Theorems, formulated by Augustus De Morgan, are two fundamental theorems of Boolean algebra that relate the complement of a product (AND) to a sum of complements (OR), and vice versa. They are extremely useful for converting between NAND/NOR implementations and AND/OR implementations, and for simplifying complex Boolean expressions.

Theorem 1: $(A \cdot B)' = A' + B'$

The complement of a product = sum of complements

Theorem 2: $(A + B)' = A' \cdot B'$

The complement of a sum = product of complements

Memory Aid: 'Break the bar and change the sign' — when applying De Morgan's theorem, break the overbar over the expression and change AND (·) to OR (+) or OR to AND.

Proof of Theorem 1 by Truth Table

A	B	A·B	(A·B)'	A'	B'	A'+B'
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

The columns (A·B)' and A'+B' are identical for all input combinations, proving Theorem 1.

Extended De Morgan's Theorem

De Morgan's theorems can be extended to any number of variables:

$$(A \cdot B \cdot C \cdot D...)' = A' + B' + C' + D' + ...$$

$$(A+B+C+D...)' = A' \cdot B' \cdot C' \cdot D' \cdot ...$$

2.2 Minterm (SOP) and Maxterm (POS)

Minterm and Sum of Products (SOP)

A Minterm is a product (AND) term in which every variable appears exactly once — either in its true form or complemented form — such that the minterm evaluates to 1 for exactly one combination of input values. For n variables, there are 2^n minterms, numbered m0 to m(2^n-1).

A Sum of Products (SOP) expression is a Boolean expression written as the OR (sum) of one or more product (AND) terms (minterms). It is the standard form obtained directly from the truth table rows where the output is 1.

SOP Example: For F(A,B,C) with output=1 at rows 1,3,5,7: F = A'B'C + A'BC + AB'C + ABC = Σm(1,3,5,7)

Minterm Table for 3 Variables (A, B, C)

Row	A	B	C	Minterm	Symbol
0	0	0	0	A'B'C'	m0
1	0	0	1	A'B'C	m1
2	0	1	0	A'BC'	m2
3	0	1	1	A'BC	m3
4	1	0	0	AB'C'	m4
5	1	0	1	AB'C	m5
6	1	1	0	ABC'	m6
7	1	1	1	ABC	m7

Maxterm and Product of Sums (POS)

A Maxterm is a sum (OR) term in which every variable appears exactly once — such that the maxterm evaluates to 0 for exactly one combination of input values. A Product of Sums (POS) expression is the AND of one or more OR terms (maxterms). It is obtained from the truth table rows where the output is 0.

POS Example: For F with output=0 at rows 0,2,4,6: F = (A+B+C)(A+B'+C)(A'+B+C)(A'+B'+C) = ΠM(0,2,4,6)

Feature	SOP (Sum of Products)	POS (Product of Sums)
Basic term	Minterm (AND term)	Maxterm (OR term)
Term evaluates to	1 for one input combo	0 for one input combo
Expression form	OR of AND terms	AND of OR terms
Derived from	Rows where output = 1	Rows where output = 0
Notation	Σm(...)	ΠM(...)
Gate implementation	AND gates then OR gate	OR gates then AND gate

2.3 Karnaugh Map (K-Map)

Introduction

A Karnaugh Map (K-Map), introduced by Maurice Karnaugh in 1953, is a graphical method for simplifying Boolean expressions. It provides a systematic visual approach to minimising logic functions, eliminating the need for algebraic manipulation. A K-Map is a rectangular grid where each cell represents one minterm of the Boolean function, and adjacent cells differ by exactly one variable (Gray code ordering).

Key Rule: Adjacent cells in a K-Map differ by exactly ONE variable. Groups of 1s (called implicants) must be powers of 2 (1, 2, 4, 8, 16...). The larger the group, the more variables are eliminated.

2-Variable K-Map (4 cells)

2-Variable K-Map Layout

```
          B=0    B=1
A=0   |  m0   |  m1   |
A=1   |  m2   |  m3   |
```

3-Variable K-Map (8 cells)

3-Variable K-Map Layout (Gray Code column ordering)

```
          BC=00  BC=01  BC=11  BC=10
A=0   |   m0   |   m1   |   m3   |   m2   |
A=1   |   m4   |   m5   |   m7   |   m6   |

Note: Columns follow Gray code: 00→01→11→10 (not binary order!)
```

4-Variable K-Map (16 cells)

4-Variable K-Map Layout

```
           CD=00  CD=01  CD=11  CD=10
AB=00  |   m0   |   m1   |   m3   |   m2   |
AB=01  |   m4   |   m5   |   m7   |   m6   |
AB=11  |  m12   |  m13   |  m15   |  m14   |
AB=10  |   m8   |   m9   |  m11   |  m10   |

K-Map wraps around: top↔bottom and left↔right edges are adjacent!
```

Rules for K-Map Simplification

- Enter 1s in the K-Map cells corresponding to minterms where the output function is 1
- Group adjacent 1s into rectangles — groups must contain 1, 2, 4, 8, or 16 cells (powers of 2)
- Groups must be as large as possible — larger groups → fewer variables in the simplified term

- The K-Map wraps around — cells on opposite edges are adjacent
- Every 1 in the K-Map must be covered by at least one group (Essential Prime Implicants first)
- A cell can be included in more than one group if it results in larger groups
- Do not include cells with 0s in the groups
- For each group, identify variables that remain constant — those form the simplified AND term

Worked Example — 3-Variable K-Map

Problem: Simplify F(A,B,C) = Σm(0,1,2,3,4,5) [Output is 1 for minterms 0,1,2,3,4,5]

K-Map Solution

```
                    BC=00  BC=01  BC=11  BC=10
 A=0  |   1   |   1   |   1   |   1   |  ← Group 1: row A=0 (4 cells)
 A=1  |   1   |   1   |   0   |   0   |  ← Group 2: col BC=00,01, both
                     rows (4 cells)

        Group 1 (A=0, all BC): A=0 is constant → term = A'
        Group 2 (B=0, A=0&1):  B=0 is constant → term = B'
                        Final: F = A' + B'
```

The original expression Σm(0,1,2,3,4,5) which would have six minterms simplifies beautifully to just F = A' + B' using the K-Map method — a two-term, two-literal expression implementable with just one NOT gate each and one OR gate.

2.4 Simplification by Boolean Algebra — Worked Example

Problem: Simplify: F = ABC + ABC' + A'BC + A'B'C

Step	Expression	Boolean Law Used
Start	F = ABC + ABC' + A'BC + A'B'C	—
Group 1st two	= AB(C + C') + A'BC + A'B'C	Distributive: AB(C+C')
Simplify C+C'	= AB(1) + A'BC + A'B'C	Complement Law: C+C'=1
Simplify	= AB + A'BC + A'B'C	Identity: AB·1=AB
Group last two	= AB + A'C(B + B')	Distributive
Simplify B+B'	= AB + A'C(1)	Complement Law
Final Result	F = AB + A'C	Simplified (2 terms)

3. Basic Concept of Flip-Flops as Storage Elements

3.1 Introduction to Flip-Flops

A Flip-Flop (FF) is a bistable multivibrator — a sequential logic circuit that has two stable states (0 and 1) and can store one bit of binary information. Unlike combinational logic circuits (where output depends only on current inputs), flip-flops have memory: the output depends on both the current input AND the previous state. Flip-flops are the fundamental storage elements in digital systems, forming the basis of registers, counters, shift registers, and memory devices.

Key Property: A flip-flop can remain in its current state indefinitely until triggered by a clock signal. This memory property makes it the basic unit of digital storage — one flip-flop stores exactly 1 bit of information.

Flip-flops are clocked (synchronous) devices — their state changes occur only at the edge (rising edge or falling edge) of a clock pulse, ensuring orderly and synchronized operation in digital systems. The four most common types are the SR, D, JK, and T flip-flops.

3.2 SR Flip-Flop (Set-Reset Flip-Flop)

Construction and Working

The SR flip-flop is the simplest flip-flop, having two inputs — S (Set) and R (Reset) — and two complementary outputs Q and Q'. It can be built using two cross-coupled NAND gates or two cross-coupled NOR gates. The Set input drives the output Q to 1 (set state), and the Reset input drives Q to 0 (reset state). When both inputs are 0, the flip-flop retains its previous state (memory function).

S	R	Q(next)	Q'(next)	State
0	0	Q(prev)	Q'(prev)	No Change (Hold/Memory)
0	1	0	1	RESET (Q=0)
1	0	1	0	SET (Q=1)
1	1	X	X	INVALID (Forbidden state)

Forbidden State: In an SR flip-flop, S=1 and R=1 simultaneously is a forbidden condition because it drives both Q and Q' to 0 (for NOR-based) or both to 1 (for NAND-based), violating the requirement that Q and Q' must be complements of each other.

3.3 D Flip-Flop (Data / Delay Flip-Flop)

Construction and Working

The D flip-flop overcomes the forbidden state problem of the SR flip-flop by having only a single data input D. Internally, R is always the complement of S (R = S' = D'), so S=R=1 can never occur. On the active clock edge, the output Q takes the value of the D input at that moment. The output remains unchanged until the next clock edge — hence it is also called a Delay flip-flop (the input is delayed by one clock cycle before appearing at the output).

D	CLK Edge	Q(next)	Operation
0	↑ (Rising)	0	Output resets to 0
1	↑ (Rising)	1	Output sets to 1
X	No edge	Q(prev)	No change — memory

Q(next) = D

D Flip-Flop characteristic equation — output simply follows input D at each clock edge

- Applications: Data registers, shift registers, data latches, pipeline stages, memory cells

3.4 JK Flip-Flop

Construction and Working

The JK flip-flop is the most versatile and commonly used flip-flop. It resolves the forbidden state problem of the SR flip-flop by making the J=K=1 condition a valid toggle operation. The J input acts like S (Set) and the K input acts like R (Reset). When both J=1 and K=1 simultaneously, the output toggles (changes to the opposite state) on each clock edge.

J	K	Q(next)	Operation
0	0	Q(prev)	No Change — Hold/Memory
0	1	0	RESET — Q goes to 0
1	0	1	SET — Q goes to 1
1	1	Q'(prev)	TOGGLE — Q changes state

Q(next) = J·Q' + K'·Q

JK Flip-Flop characteristic equation

- The JK flip-flop can function as SR (by connecting J=S, K=R), D (by connecting K=J'), or T flip-flop (by connecting J=K=1)
- Applications: Counters, shift registers, frequency dividers

3.5 T Flip-Flop (Toggle Flip-Flop)

Construction and Working

The T flip-flop has a single input T (Toggle). It is obtained from a JK flip-flop by connecting J and K inputs together. When T=0, the output remains unchanged (hold). When T=1, the output toggles (changes to the opposite state) on each active clock edge. Since a T flip-flop divides the clock frequency by 2 (output changes every 2 clock cycles), it is the fundamental building block of binary counters.

T	CLK Edge	Q(next)	Operation
0	↑	Q(prev)	No Change — Hold
1	↑	Q'(prev)	TOGGLE — Q flips

$$Q(next) = T \oplus Q = T \cdot Q' + T' \cdot Q$$

T Flip-Flop characteristic equation

- Each T flip-flop with T=1 permanently acts as a divide-by-2 frequency divider
- Applications: Binary counters, frequency dividers, clock dividers

3.6 Comparison of All Flip-Flop Types

Feature	SR FF	D FF	JK FF	T FF
Inputs	S, R	D	J, K	T
Forbidden state	S=R=1	None	None	None
Characteristic eq.	—	Q=D	Q=JQ'+K'Q	Q=T⊕Q
Hold condition	S=R=0	No clock edge	J=K=0	T=0
Toggle condition	N/A	N/A	J=K=1	T=1
Complexity	Simple	Simple	Complex	Simple
Main use	Basic latch	Registers	Counters	Counters

4. Counters — Ripple (Asynchronous) and Synchronous Up/Down Counters

4.1 Introduction to Counters

A counter is a sequential logic circuit made up of flip-flops that counts the number of input clock pulses and progresses through a predefined sequence of binary states. Counters are among the most widely used digital circuits, found in clocks, timers, frequency meters, event counters, and address generators in memory systems.

Basic Building Block: Counters are built using T flip-flops or JK flip-flops (with J=K=1, which gives toggle behaviour). An n-bit counter uses n flip-flops and can count from 0 to $2^n - 1$ (modulo-2^n counter).

Counters are broadly classified into two categories based on their clocking method:

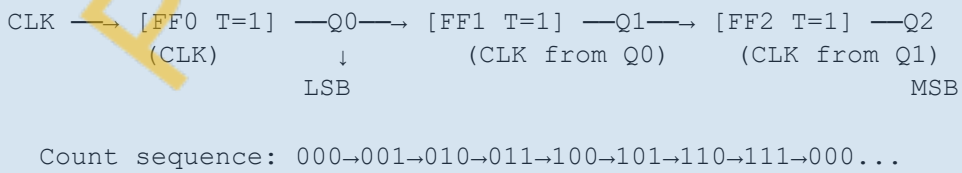
- Asynchronous (Ripple) Counter: Not all flip-flops are clocked simultaneously. The output of each FF clocks the next one — the clock signal 'ripples' through the chain.
- Synchronous Counter: All flip-flops receive the same clock signal simultaneously. State changes occur at the same instant.

4.2 Ripple Counter (Asynchronous Counter)

Construction and Working

In a ripple counter, the flip-flops are connected in series (cascade). The clock pulse is applied only to the first flip-flop (LSB). The output Q of each flip-flop serves as the clock input for the next flip-flop in the chain. All flip-flops are configured as T flip-flops with T=1 (or JK flip-flops with J=K=1), so each one toggles on the falling edge (or rising edge, depending on design) of its clock input.

3-Bit Ripple Up Counter — Block Diagram



Count sequence: 000→001→010→011→100→101→110→111→000...
Total states: $2^3 = 8$ (modulo-8 counter)

Truth Table — 3-Bit Ripple Up Counter

Clock Pulse	Q2 (MSB)	Q1	Q0 (LSB)	Decimal Count
0 (Initial)	0	0	0	0
1	0	0	1	1

2	0	1	0	2
3	0	1	1	3
4	1	0	0	4
5	1	0	1	5
6	1	1	0	6
7	1	1	1	7
8	0	0	0	0 (Reset — recycled)

Advantages and Disadvantages of Ripple Counter

Advantages	Disadvantages
Simple construction — minimal hardware	Propagation delay accumulates through the chain (ripple effect)
Low power consumption	Speed is limited by the total delay of all flip-flops in series
Easy to expand to more bits	Intermediate glitches/spikes can occur in the output
Fewer connections needed	Not suitable for high-speed applications

4.3 Ripple Down Counter

A ripple down counter counts in the reverse direction — from (2^n-1) down to 0. It is implemented by connecting the Q' (complement) output of each flip-flop as the clock input for the next flip-flop, instead of Q. For a 3-bit down counter, the sequence is: 111→110→101→100→011→010→001→000→111 (recycles).

4.4 Synchronous Counter

Construction and Working

In a synchronous counter, all flip-flops receive the same clock pulse simultaneously. This eliminates the ripple delay problem of asynchronous counters. The J and K inputs of each flip-flop are determined by combinational logic that computes the next state based on the current state. Since all flip-flops change state at the same clock edge, the output is glitch-free and the counter can operate at much higher speeds.

3-Bit Synchronous Up Counter — Connection Logic

```
FF0 (Q0):  J0 = K0 = 1          (always toggles on every clock)
      FF1 (Q1):  J1 = K1 = Q0         (toggles when Q0=1)
 FF2 (Q2):  J2 = K2 = Q0·Q1     (toggles when Q0=1 AND Q1=1)

              CLK ──→ [FF0] ──→ [FF1] ──→ [FF2]
               └──────────────────────────────┘ (same CLK to all)
```

```
All flip-flops change state at the SAME clock edge — no ripple!
```

Synchronous vs Ripple Counter — Comparison

Feature	Ripple (Asynchronous) Counter	Synchronous Counter
Clock	Only 1st FF clocked; rest ripple	All FFs clocked simultaneously
Speed	Slow (delays add up)	Fast (parallel operation)
Glitches	Present (ripple effect)	Absent (all change together)
Complexity	Simple — fewer connections	Complex — needs combinational logic
Power	Low	Slightly higher
Application	Low-speed: clocks, timers	High-speed: processors, high-freq counters
Example IC	7490, 7493	74163, 74191

4.5 Up/Down Counter

An Up/Down counter is a counter that can count both upward (incrementing) and downward (decrementing) depending on a mode control signal (U/D or M). When the Up/Down control input is 1 (Up mode), the counter counts in the ascending binary sequence. When it is 0 (Down mode), the counter counts in descending order.

- Up mode (U/D = 1): 0→1→2→3→4→5→6→7→0... (modulo-8 example)
- Down mode (U/D = 0): 7→6→5→4→3→2→1→0→7...
- The mode selection is implemented using multiplexers or additional logic on the J/K inputs
- Synchronous up/down counters: IC 74191 (4-bit), IC 74193 (4-bit with separate up/down clocks)

5. Introduction to Digital IC Logic Gates — TTL

5.1 Digital IC Logic Families

Digital integrated circuits (ICs) implementing logic gates are manufactured in several logic families, each with its own technology, transistor type, and performance characteristics. A logic family is a group of ICs that share the same technology and are compatible in terms of supply voltage, input/output voltage levels, and interface signals. The two major logic IC families are:

- TTL (Transistor-Transistor Logic): Uses bipolar junction transistors (BJTs). The dominant logic family from the 1960s to 1990s.
- CMOS (Complementary Metal Oxide Semiconductor): Uses MOSFETs. Dominant today due to ultra-low power consumption.

Syllabus Focus: This section covers TTL logic only, as specified in the FEEE Polytechnic 2nd Semester syllabus.

5.2 TTL (Transistor-Transistor Logic)

Introduction

TTL (Transistor-Transistor Logic) is a logic family in which bipolar junction transistors (BJTs) are used for both the input stage and the output stage of each logic gate. Introduced by Texas Instruments in 1964 as the 7400 series, TTL became the most widely used logic family for several decades. TTL ICs are still used today in educational laboratories, industrial equipment, and legacy systems.

Standard TTL: The 74xx series (commercial grade: 0°C to +70°C) and 54xx series (military grade: −55°C to +125°C). Both series have identical pin configurations and are interchangeable electrically.

TTL Circuit Structure (NAND Gate Example)

A standard TTL NAND gate consists of four stages: the Multi-Emitter Input Transistor stage, the Phase Splitter stage, the Totem-Pole Output stage, and the feedback clamp diodes for noise suppression.

TTL NAND Gate — Internal Structure Overview

```
          Stage 1: Multi-Emitter Transistor (Q1)
                  ├── Emitter 1 ── Input A
└── Emitter 2 ── Input B          (one emitter per input)

              Stage 2: Phase Splitter (Q2)
  Drives both the output transistors in opposite phase

          Stage 3: Totem-Pole Output (Q3 and Q4)
Q3 (upper) ── ON when output is HIGH → pulls output to VCC
```

```
Q4 (lower) ── ON when output is LOW  → pulls output to GND

VCC = +5V    Logic 0 = 0 to 0.8V    Logic 1 = 2.0V to 5.0V
```

5.3 TTL Electrical Characteristics

Parameter	Symbol	TTL Value	Meaning
Supply Voltage	VCC	+5 V (±0.25 V)	Operating power supply
Logic HIGH Output	VOH	≥ 2.4 V (typ 3.4 V)	Output voltage for logic 1
Logic LOW Output	VOL	≤ 0.4 V (typ 0.2 V)	Output voltage for logic 0
Logic HIGH Input	VIH	≥ 2.0 V	Minimum input voltage for logic 1
Logic LOW Input	VIL	≤ 0.8 V	Maximum input voltage for logic 0
Noise Margin HIGH	NMH	0.4 V (min)	VOH − VIH = 2.4 − 2.0
Noise Margin LOW	NML	0.4 V (min)	VIL − VOL = 0.8 − 0.4
Propagation Delay	tpd	5–10 ns (typical)	Time for output to respond to input change
Fan-Out	—	10 (standard TTL)	Number of TTL inputs one output can drive
Power per Gate	PD	~10 mW per gate	Static power dissipation

5.4 TTL Sub-Families

Over the years, the basic TTL family has evolved into several sub-families offering improved speed, lower power, or both. All sub-families are pin-compatible with the standard 7400 series.

Sub-Family	Prefix	Propagation Delay	Power/Gate	Key Feature
Standard TTL	74xx	10 ns	10 mW	Original family — reference
Low-Power TTL	74Lxx	33 ns	1 mW	Lower power, slower speed
High-Speed TTL	74Hxx	6 ns	22 mW	Faster, higher power
Schottky TTL	74Sxx	3 ns	19 mW	Schottky diodes prevent saturation
Low-Power Schottky	74LSxx	9.5 ns	2 mW	Best balance: low power + good speed
Advanced Schottky	74ASxx	1.7 ns	8 mW	Very fast, moderate power
Adv. Low-Power S.	74ALSxx	4 ns	1 mW	Low power and fast — popular in 1980s–90s

Fast TTL	74Fxx	3 ns	4 mW	High speed bipolar

5.5 Common TTL IC Numbers and Functions

IC Number	Function	Gate Type & Count
7400	Quad 2-input NAND	4 NAND gates, 2 inputs each
7402	Quad 2-input NOR	4 NOR gates, 2 inputs each
7404	Hex Inverter (NOT)	6 NOT gates (single input each)
7408	Quad 2-input AND	4 AND gates, 2 inputs each
7432	Quad 2-input OR	4 OR gates, 2 inputs each
7486	Quad 2-input XOR	4 XOR gates, 2 inputs each
7474	Dual D Flip-Flop	2 positive edge-triggered D flip-flops
7476	Dual JK Flip-Flop	2 JK flip-flops with preset and clear
7490	Decade Counter	BCD (0–9) ripple counter
7493	4-bit Binary Counter	4-bit ripple up counter
74163	4-bit Sync Counter	Synchronous presettable 4-bit binary counter
7447	BCD to 7-Segment	BCD decoder/driver for 7-segment display

5.6 TTL Logic Levels and Noise Margins

Noise margin is a measure of the ability of a logic circuit to tolerate noise (unwanted voltage fluctuations) without misinterpreting a logic level. TTL has separate noise margins for HIGH and LOW levels:

```
NMH = VOH(min) − VIH(min) = 2.4 − 2.0 = 0.4 V    (HIGH noise margin)
```

```
NML = VIL(max) − VOL(max) = 0.8 − 0.4 = 0.4 V    (LOW noise margin)
```

This means a TTL circuit can tolerate up to 0.4 V of noise on the signal lines before a logic error occurs. In practice, proper decoupling capacitors (0.1 µF between VCC and GND, one per IC) are used to suppress power supply noise.

5.7 Advantages and Disadvantages of TTL

Advantages of TTL	Disadvantages of TTL

Well established, standardised, widely available	Higher power consumption (~10 mW/gate) vs CMOS
Fast switching speed (ns range)	Fixed supply voltage of +5 V (strict tolerance)
Good drive capability (Fan-out = 10)	Not as compact as CMOS for VLSI
Wide range of functions available in 74xx series	Inputs left floating act as logic 1 (potential hazard)
High noise immunity in industrial environments	Generates more heat — needs power management in dense boards

End of Chapter: Overview of Digital Electronics
FEEE | Polytechnic 2nd Semester | www.polynoteshub.co.in