Digital Logic Design (DI03000111)

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

Course Agenda I

1. Lecture 1.1: Intro to various number systems (Binary, Octal)
2. Lecture 1.2: Intro to various number systems (Decimal, Hexadecimal)
3. Lecture 1.3: Conversion from one number system to another
4. Lecture 1.4: Conversion between fractional numbers and shortcut methods
5. Lecture 1.5: Binary arithmetic operations: addition, subtraction
6. Lecture 1.6: Binary arithmetic operations: multiplication and division

Course Agenda II

7. Lecture 1.7: 1's and 2's Complement of binary number
8. Lecture 1.8: Subtraction using 1's and 2's complement method
9. Lecture 2.1: Digital Logic Gates: AND, OR, NOT
10. Lecture 2.2: Digital Logic Gates: NAND, NOR, EX-OR, EX-NOR
11. Lecture 2.3: Basic Theorems of Boolean Algebra
12. Lecture 2.4: Properties of Boolean Algebra and De Morgan's Theorems

Course Agenda III

13. Lecture 2.5: Implementations of Boolean function using AND-OR-Invert logic gates
14. Lecture 2.6: Universal Gates: NAND & NOR
15. Lecture 2.7: Algebraic Simplification of Boolean Expressions
16. Lecture 2.8: Sum of Product (SOP) Definition and Minterms
17. Lecture 2.9: Simplification using K-map up to 4-variables
18. Lecture 2.10: Don't Care Conditions in Karnaugh Maps
19. Lecture 3.1: Arithmetic Circuits: Half Adder and Full Adder

Course Agenda IV

Course Agenda V

Course Agenda VI

Course Agenda VII

40. Lecture 5.2: Types of Memory: ROM (PROM, EPROM, EEPROM)
41. Lecture 5.3: Overview of Logic Family Definitions: Fan-in, Fan-out, Noise Margin
42. Lecture 5.4: Overview of Logic Family Definitions: Propagation Delay, Power Dissipation, Figure of Merit
43. Lecture 5.5: Comparison of different logic families (TTL, CMOS, ECL) - Part 1
44. Lecture 5.6: Comparison of different logic families (TTL, CMOS, ECL) - Part 2

Course Agenda VIII

45 Lecture 5.7: E-waste Management of Digital ICs/Chips and Course Recap

Lecture 1.1: Intro to various number systems (Binary, Octal)

Complete Lecture Deck – All 45 Lectures

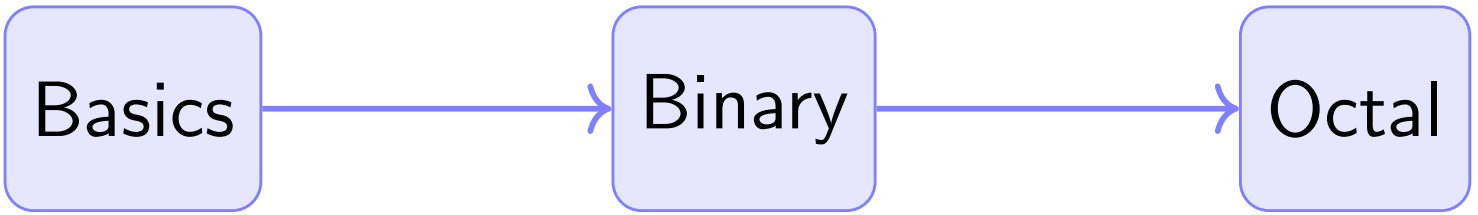
Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

Agenda

- What is a Number System?
- Concept of Base or Radix
- The Binary Number System
- Counting in Binary
- The Octal Number System
- Counting in Octal
- Comparison and Applications



What is a Number System?

- A mathematical way to represent numbers.
- It consists of a set of symbols (digits).
- Used for counting, measuring, and identifying.
- Digital electronics relies on specific systems to store and process data.

Concept of Base (Radix)

- Base or Radix (r) defines the number of unique digits in the system.
- Valid digits range from 0 to (r - 1).
- Determines the positional weight of each digit in a number.
- Example: Decimal has a base of 10, using digits 0 through 9.

$$r = 10 \implies \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

The Binary Number System

- Base / Radix = 2
- Uses only two symbols: 0 and 1 (called bits).
- Directly corresponds to the ON/OFF states of digital logic circuits.
- The fundamental language of computers.

Positional Weights in Binary

- Each position represents a power of 2.
- Moving right to left, weights are: $2^0, 2^1, 2^2, 2^3 \ldots$
- i.e., 1, 2, 4, 8, 16, 32, 64...
- The Rightmost bit is the Least Significant Bit (LSB).
- The Leftmost bit is the Most Significant Bit (MSB).

Weight	2^3	2^2	2^1	2^0
Value	8	4	2	1

Counting in Binary (0 to 7)

- $0 = 000$
- $1 = 001$
- $2 = 010$ (Carry over to the 2s column)
- $3 = 011$
- $4 = 100$ (Carry over to the 4s column)
- $5 = 101$
- $6 = 110$
- $7 = 111$

Decimal	Binary
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

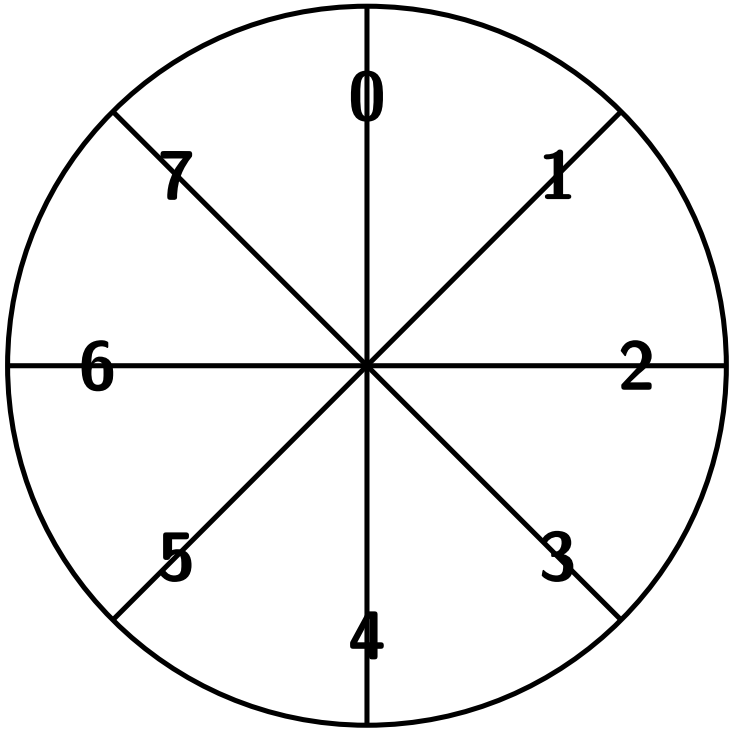
Counting in Binary (8 to 15)

- $8 = 1000$
- $9 = 1001$
- $10 = 1010$
- $11 = 1011$
- $12 = 1100$
- $13 = 1101$
- $14 = 1110$
- $15 = 1111$

Decimal	Binary
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

The Octal Number System

- Base / Radix = 8
- Uses eight symbols: 0, 1, 2, 3, 4, 5, 6, 7.
- Provides a compact way to represent binary numbers.
- Used historically in minicomputers (e.g., PDP-8).



Positional Weights in Octal

- Each position represents a power of 8.
- Moving right to left: $8^0, 8^1, 8^2, 8^3 \ldots$
- i.e., 1, 8, 64, 512...
- Example: $(14)_8$ means $1 \times 8 + 4 \times 1 = 12$ in decimal.

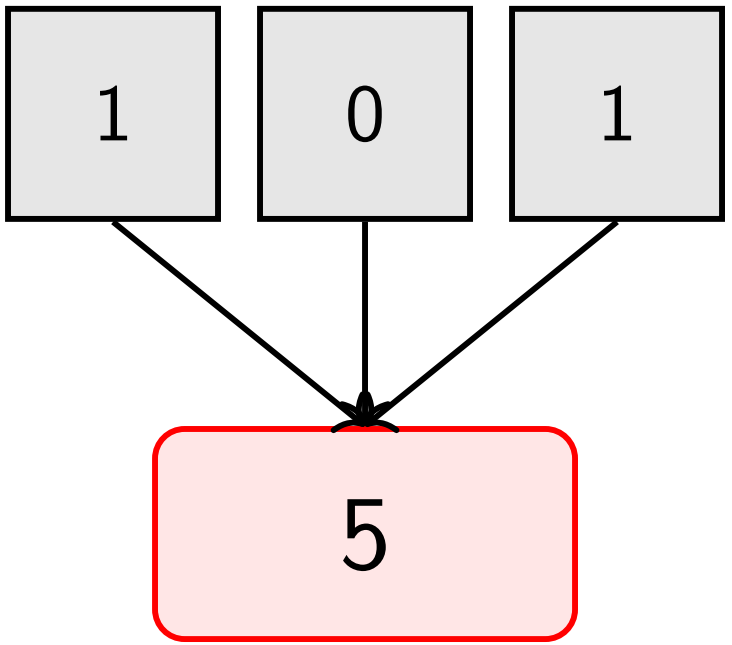
Weight	8^3	8^2	8^1	8^0
Value	512	64	8	1

Counting in Octal

- 0, 1, 2, 3, 4, 5, 6, 7...
- What comes after 7? The answer is 10 (read as one-zero).
- 10 in octal = 8 in decimal.
- Next: 11, 12, 13, 14, 15, 16, 17...
- After 17 comes 20.

Relationship: Binary and Octal

- $2^3 = 8$
- Exactly 3 bits of binary correspond to 1 digit of octal.
- This makes conversion between Binary and Octal very simple.
- Binary: 111 (max 3-bit) = Octal: 7



Summary

- Base (r) defines the number of valid digits (0 to r-1).
- Binary (Base 2) uses 0, 1. It is the language of hardware.
- Positional weights in Binary are powers of 2.
- Octal (Base 8) uses 0-7. It compacts binary data.
- Positional weights in Octal are powers of 8.
- 3 Binary bits map exactly to 1 Octal digit.

System	**Base & Symbols**	**Example**
Binary	Base 2; $\{0, 1\}$	101_2
Octal	Base 8; $\{0, 1, \ldots, 7\}$	74_8

Next Lecture Preview

- Deep dive into the Decimal Number System.
- Introduction to the Hexadecimal (Base 16) System.
- Understanding Alphanumeric digits (A-F).
- Why Hexadecimal is heavily used in modern computing.

Hexadecimal Preview

A B C D E F

Questions?

Lecture 1.2: Intro to various number systems (Decimal, Hexadecimal)

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

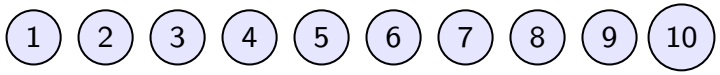
Recap & Agenda

- Recap: Binary and Octal basics
- The Decimal Number System (Formal Review)
- Positional Weights in Decimal
- The Hexadecimal Number System
- Alphanumeric Digits (A-F)
- Counting in Hexadecimal
- Why Hexadecimal?
- Summary & Comparison

✓ Binary/Octal Basics

The Decimal Number System

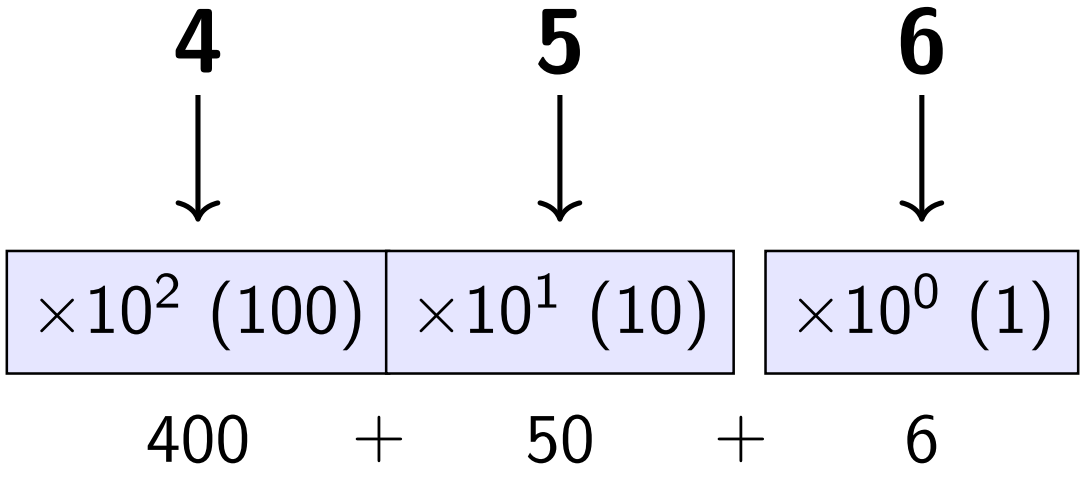
- Base / Radix = 10
- Uses ten symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- The system we use in everyday human life.
- Based on human anatomy (10 fingers).



10 symbols / 10 fingers

Positional Weights in Decimal

- Every digit has a weight based on its position: powers of 10.
- Moving right to left: 10^0 (Units), 10^1 (Tens), 10^2 (Hundreds)...
- Example: 456 in base 10
- $= (4 \times 10^2) + (5 \times 10^1) + (6 \times 10^0)$
- $= (4 \times 100) + (5 \times 10) + (6 \times 1)$
- $= 400 + 50 + 6$



The Hexadecimal Number System

- Base / Radix = 16
- Needs 16 unique symbols.
- Since we only have 10 numeric digits (0-9), we borrow letters.
- Symbols used: 0-9 and A-F.

The 16 Hexadecimal Symbols

0	1	2	3	4	5	6	7
8	9	A	B	C	D	E	F

Hexadecimal Symbols (A-F)

- A = 10, B = 11, C = 12
- D = 13, E = 14, F = 15
- These are single digits in base 16.

Hex Symbol	A	B	C	D	E	F
Decimal Value	10	11	12	13	14	15

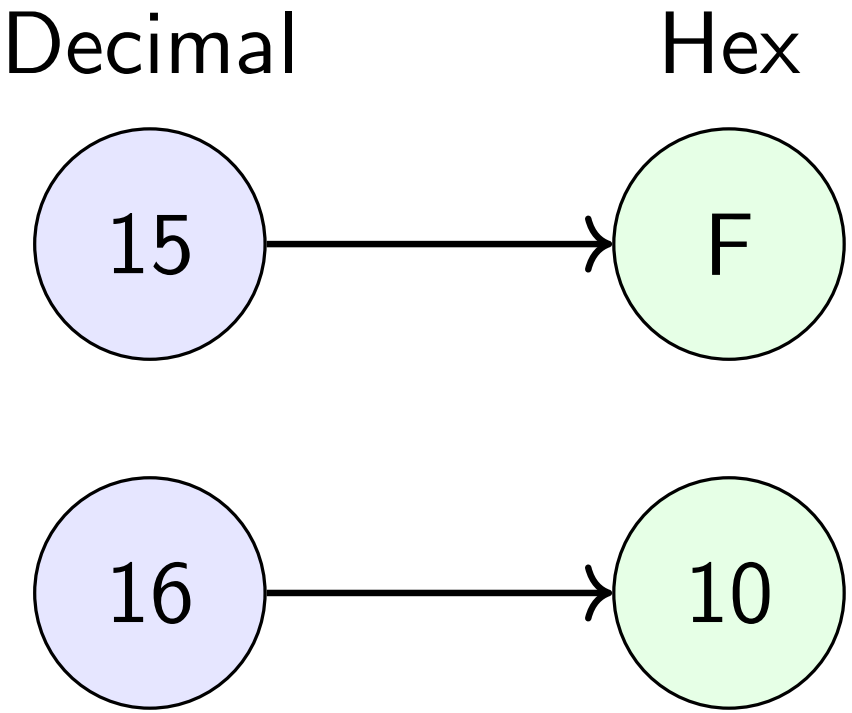
Positional Weights in Hexadecimal

- Positions are powers of 16.
- Right to left: 16^0, 16^1, 16^2...
- i.e., 1, 16, 256, 4096...
- Example: $(1A)_{16}$
- $= (1 \times 16^1) + (A \times 16^0)$
- $= (1 \times 16) + (10 \times 1) = 26$ in decimal.

Position	3	2	1	0
Power	16^3	16^2	16^1	16^0
Weight	4096	256	16	1

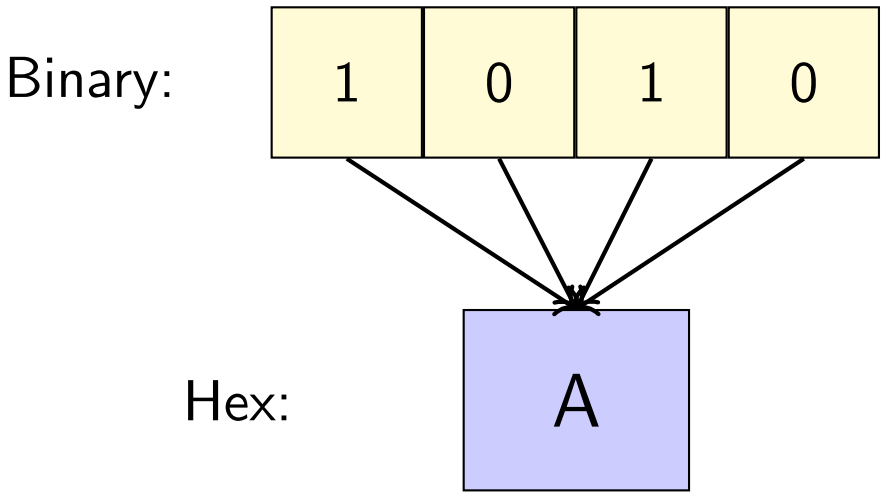
Counting in Hexadecimal

- 0, 1, 2, ..., 9, A, B, C, D, E, F
- What comes after F? We run out of symbols.
- Carry over to the next column: 10 (read as one-zero).
- 10 in hex = 16 in decimal.
- Then 11, 12, ..., 1F
- After 1F comes 20.



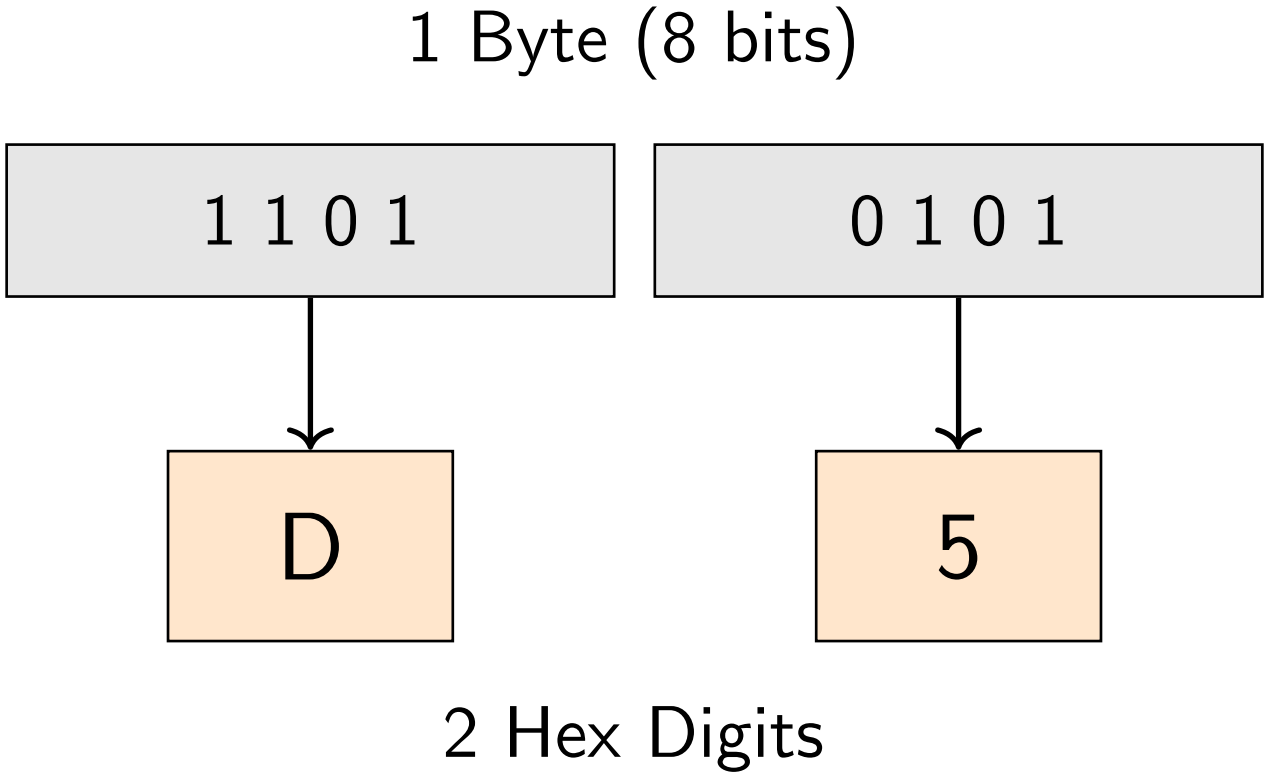
Relationship: Binary and Hexadecimal

- $2^4 = 16$
- Exactly 4 bits of binary correspond to 1 digit of hexadecimal.
- A 'nibble' is 4 bits.
- Binary: 1111 (max 4-bit) = Hex: F (decimal 15)



Why Hexadecimal?

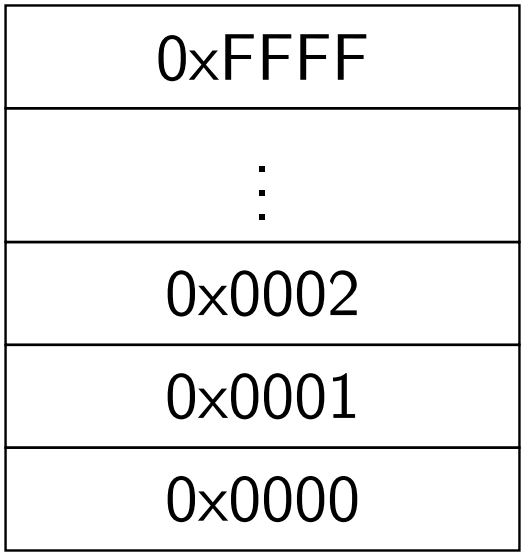
- Computers process data in bytes (8 bits).
- 1 Byte = 2 Hex digits.
- Extremely compact way to write long binary sequences.
- Reduces human error when reading/writing code or memory addresses.



Application: Memory Addressing

- Memory addresses in microprocessors are usually given in Hex.
- Example: Address 0x004FFA
- '0x' is a common prefix indicating Hexadecimal in programming (like C/C++).
- Makes it easy to see bit patterns.

Memory Map



Comparing All Systems

- Decimal: Base 10, Symbols 0-9
- Binary: Base 2, Symbols 0,1 (Hardware)
- Octal: Base 8, Symbols 0-7 (Group of 3 bits)
- Hex: Base 16, Symbols 0-9,A-F (Group of 4 bits)

System	Value of Fifteen
Decimal	15
Binary	1111
Octal	17
Hexadecimal	F

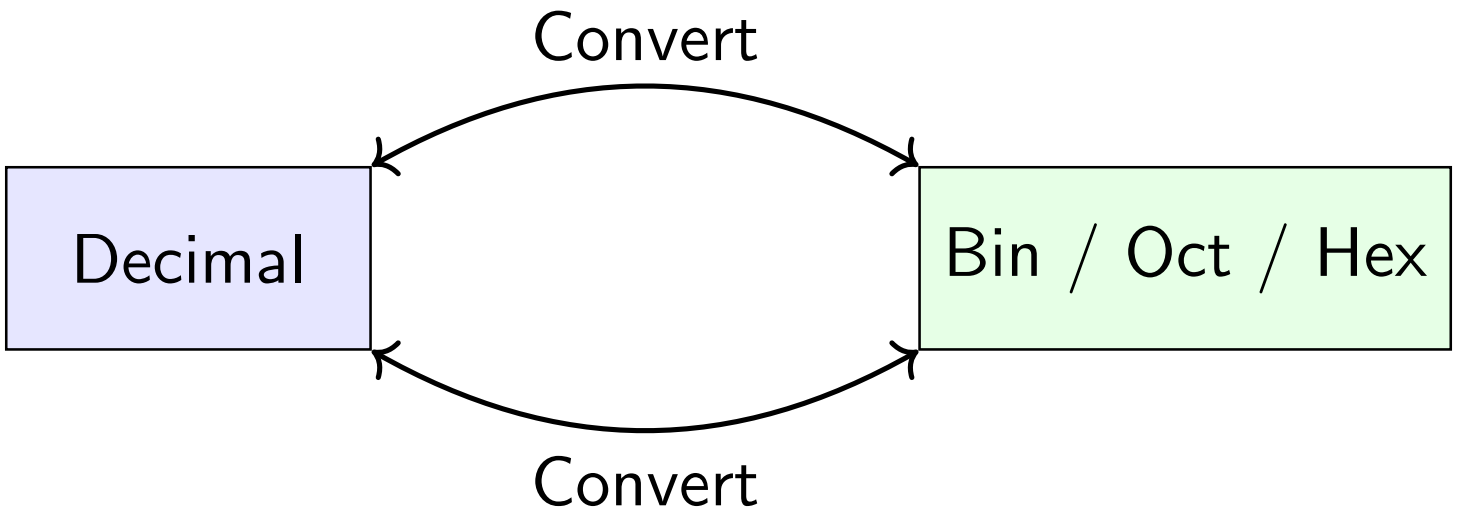
Summary

- Decimal uses powers of 10, perfectly mapping to human counting.
- Hexadecimal uses base 16, employing A-F for values 10-15.
- Positional weights in Hex are powers of 16.
- 4 binary bits map to 1 Hex digit.
- Hex is heavily used to represent memory addresses and raw bytes.

0x

Next Lecture Preview

- Now that we know the systems, how do we convert between them?
- Decimal to Binary / Octal / Hex conversions.
- Binary / Octal / Hex back to Decimal.
- The repeated division and sum-of-weights methods.



Questions?

Lecture 1.3: Conversion from one number system to another

Complete Lecture Deck – All 45 Lectures

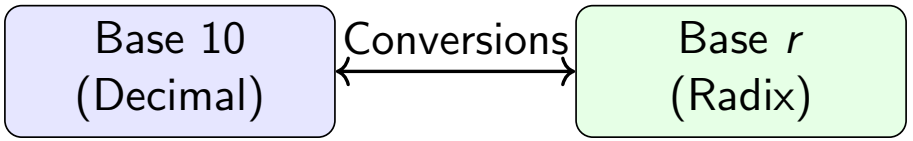
Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

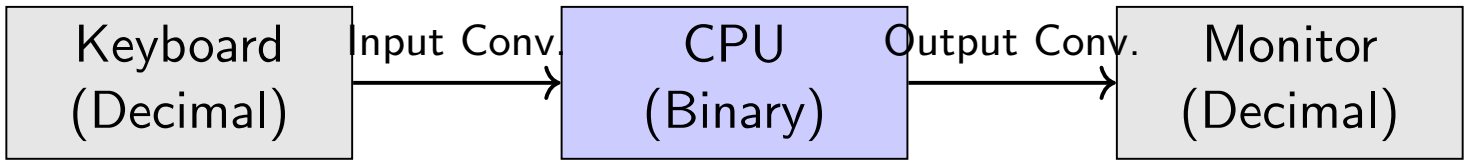
Recap & Agenda

- Why do we need conversions?
- Method 1: Decimal to Any Base (Repeated Division)
- Example: Decimal to Binary / Octal / Hex
- Method 2: Any Base to Decimal (Sum of Weights)
- Example: Binary / Octal / Hex to Decimal
- Practice Problems
- Summary



Need for Conversions

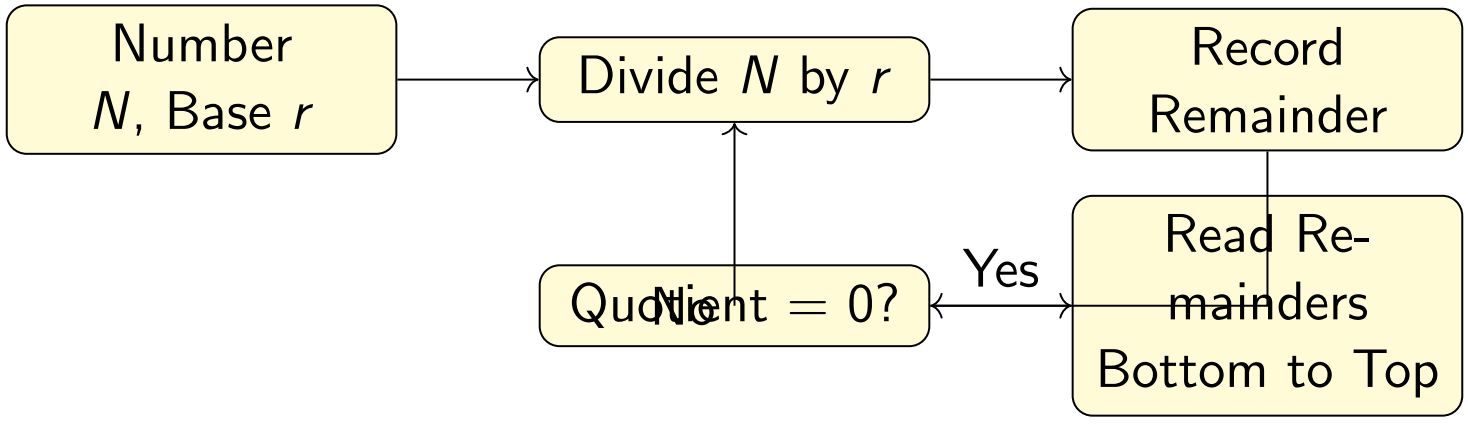
- Humans input data in Decimal.
- Computers process data in Binary.
- Interfacing requires conversion from Decimal to Binary on input.
- Output requires conversion from Binary to Decimal.
- Octal/Hex are used by engineers to read binary easily.



Decimal to Any Base: The Method

Algorithm: Repeated Division by the target Base (r).

1. Divide the decimal number by 'r'.
2. Record the Remainder.
3. Divide the Quotient by 'r' again.
4. Repeat until the Quotient is 0.
5. Read remainders from BOTTOM to TOP.



Example: Decimal to Binary

Convert $(25)_{10}$ to Binary.

Divisor	Dividend	Quotient	Remainder	
2	25	12	1	(LSB)
2	12	6	0	
2	6	3	0	Read UP
2	3	1	1	
2	1	0	1	(MSB)

Answer: 11001

Example: Decimal to Octal

Convert $(156)_{10}$ to Octal.
Divide by target base: 8

Divisor	Dividend	Quotient	Remainder	
8	156	19	4	(LSD)
8	19	2	3	
8	2	0	2	(MSD)

Read UP

$$(156)_{10} = (234)_8$$

Example: Decimal to Hexadecimal

Convert $(254)_{10}$ to Hex.
Divide by target base: 16

Divisor	Dividend	Quotient	Remainder
16	254	15	$14 \rightarrow$ **E**
16	15	0	$15 \rightarrow$ **F**

Read UP

- In Hex: 14 is E, 15 is F.
- Read bottom to top: FE

$$(254)_{10} = (FE)_{16}$$

Any Base to Decimal: The Method

Algorithm: Sum of Positional Weights.

1. Write down the number.
2. Assign weights to each digit based on its position (from right to left: $r^0, r^1, r^2 \ldots$).
3. Multiply each digit by its weight.
4. Sum the results to get the Decimal value.

Sum of Weights Formula

$$N_{10} = d_n \times r^n + \cdots + d_1 \times r^1 + d_0 \times r^0$$

Where d_i are digits and r is the base.

Example: Binary to Decimal

Convert $(1011)_2$ to Decimal.

Digits:	1	0	1	1
Pos:	3	2	1	0
Weights:	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$

Calculation: $(1 \times 8) + (0 \times 4) + (1 \times 2) + (1 \times 1)$

$$= 8 + 0 + 2 + 1 = \mathbf{11}$$

$$(1011)_2 = (11)_{10}$$

Example: Octal to Decimal

Convert $(345)_8$ to Decimal.

Digits:	3	4	5
Pos:	2	1	0
Weights:	$8^2 = 64$	$8^1 = 8$	$8^0 = 1$

Calculation: $(3 \times 64) + (4 \times 8) + (5 \times 1)$

$$= 192 + 32 + 5 = \mathbf{229}$$

$$(345)_8 = (229)_{10}$$

Example: Hexadecimal to Decimal

Convert $(2A)_{16}$ to Decimal.
Substitute A with 10.

Digits:	2	$A \rightarrow 10$
Pos:	1	0
Weights:	$16^1 = 16$	$16^0 = 1$

Calculation: $(2 \times 16) + (10 \times 1)$

$$= 32 + 10 = \mathbf{42}$$

$$(2A)_{16} = (42)_{10}$$

Summary

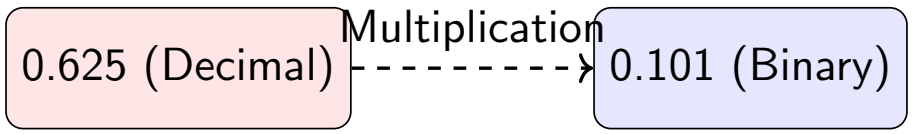
Decimal → Any Base	**Any Base → Decimal**
Use **Repeated Division** by target base.	Use **Sum of Weights**.
Read remainders from bottom (MSB) to top (LSB).	Multiply each digit by $\text{base}^{\text{position}}$ and sum.

Hexadecimal Letters

A=10, B=11, C=12, D=13, E=14, F=15

Next Lecture Preview

- Converting fractional numbers (e.g., 12.625).
- Method for Decimal Fractions $\rightarrow$ Any Base (Repeated Multiplication).
- Direct conversion between Binary, Octal, and Hexadecimal.
- Grouping bits to bypass Decimal entirely.



Questions?

Lecture 1.4: Conversion between fractional numbers and shortcut methods

Complete Lecture Deck – All 45 Lectures

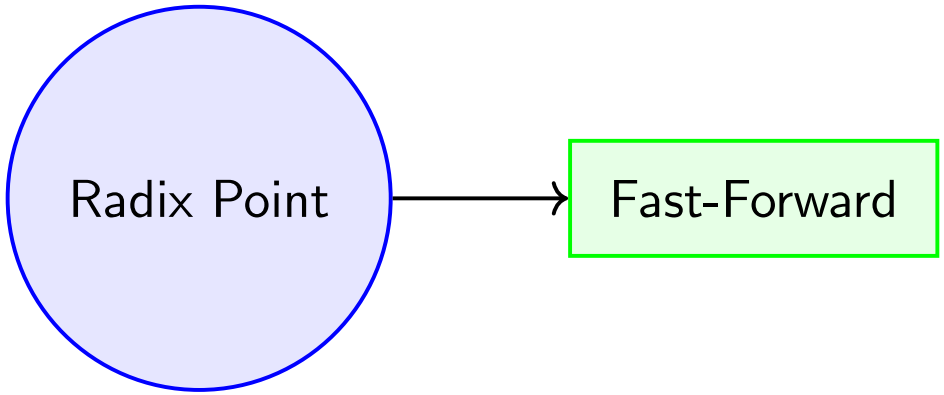
Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

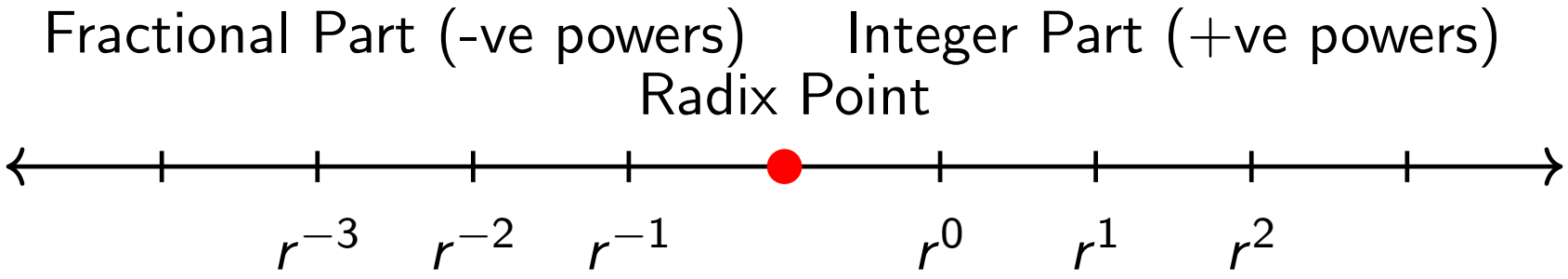
Recap & Agenda

- Fractional Numbers in Different Bases
- Decimal Fraction to Any Base (Repeated Multiplication)
- Any Base Fraction to Decimal (Negative Weights)
- Direct Conversion: Binary $\leftrightarrow$ Octal
- Direct Conversion: Binary $\leftrightarrow$ Hexadecimal
- Direct Conversion: Octal $\leftrightarrow$ Hexadecimal
- Summary



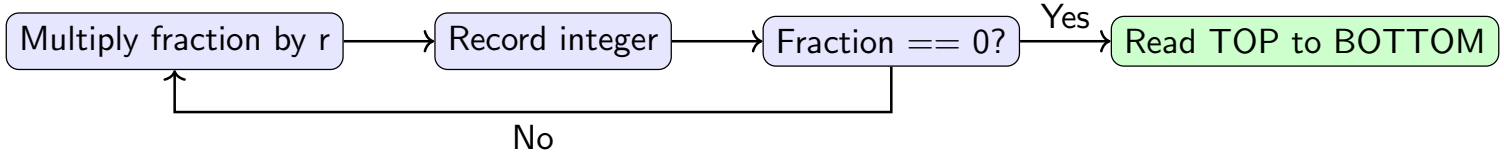
Understanding Fractional Weights

- The Radix Point separates integer and fractional parts.
- Positions to the right of the point have negative powers.
- Decimal: 10^{-1} (0.1), 10^{-2} (0.01)...
- Binary: 2^{-1} (0.5), 2^{-2} (0.25), 2^{-3} (0.125)...
- Octal: 8^{-1}, 8^{-2}...
- Hex: 16^{-1}...



Decimal Fraction to Any Base

- Algorithm: Repeated Multiplication by target base.
- 1. Multiply only the fractional part by base 'r'.
- 2. Record the integer part of the result (this is your digit).
- 3. Take the new fractional part and multiply again.
- 4. Repeat until the fraction is 0, or to desired precision.
- 5. Read digits from TOP to BOTTOM.



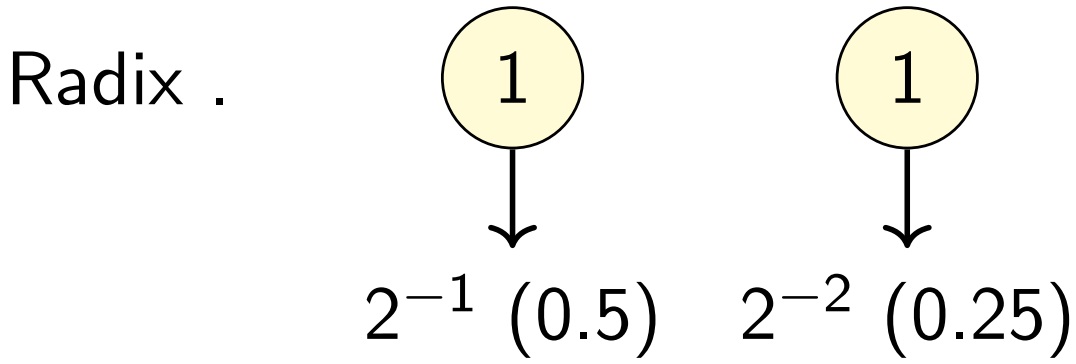
Example: Decimal to Binary Fraction

- Convert $(0.625)_{10}$ to Binary.
- Fraction is 0, stop. Read top to bottom: .101
- $(0.625)_{10} = (0.101)_2$

Fraction	$\times$ **Base**	**Result**	**Integer Recorded**	**New Fraction**
0.625	$\times$ 2	1.250	**1** (Top)	0.250
0.250	$\times$ 2	0.500	**0**	0.500
0.500	$\times$ 2	1.000	**1** (Bottom)	0.000

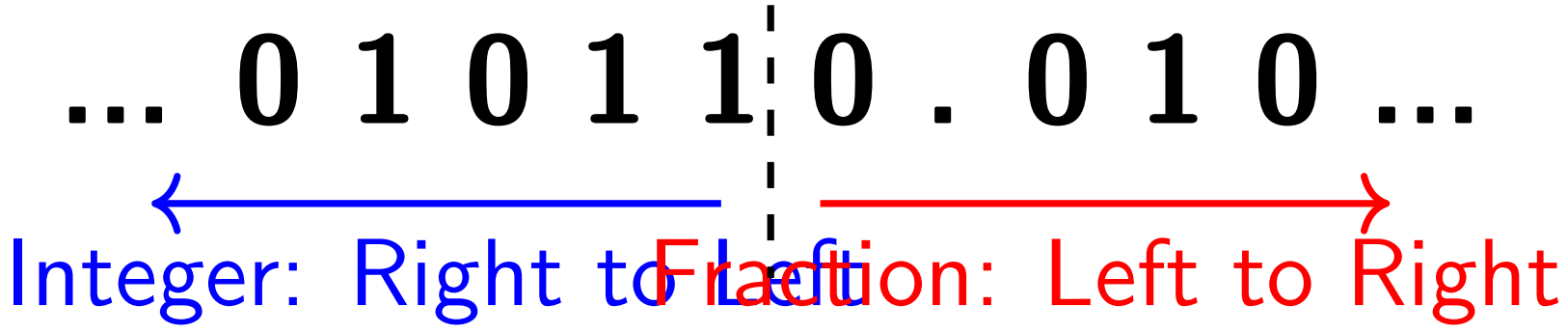
Example: Any Base Fraction to Decimal

- Algorithm: Sum of Weights (with negative powers).
- Convert $(0.11)_2$ to Decimal.
- Positions: $-1, -2$
- Weights: $2^{-1}(0.5), 2^{-2}(0.25)$
- Calculation: $(1 \times 0.5) + (1 \times 0.25) = 0.5 + 0.25 = 0.75$
- $(0.11)_2 = (0.75)_{10}$



Direct Conversion: Binary to Octal

- Because $8 = 2^3$, 3 binary bits = 1 octal digit.
- Method: Group bits into threes, starting from the radix point.
- For integers: Group right to left. Add leading zeros if needed.
- For fractions: Group left to right. Add trailing zeros if needed.
- Convert each 3-bit group to its octal equivalent (0-7).



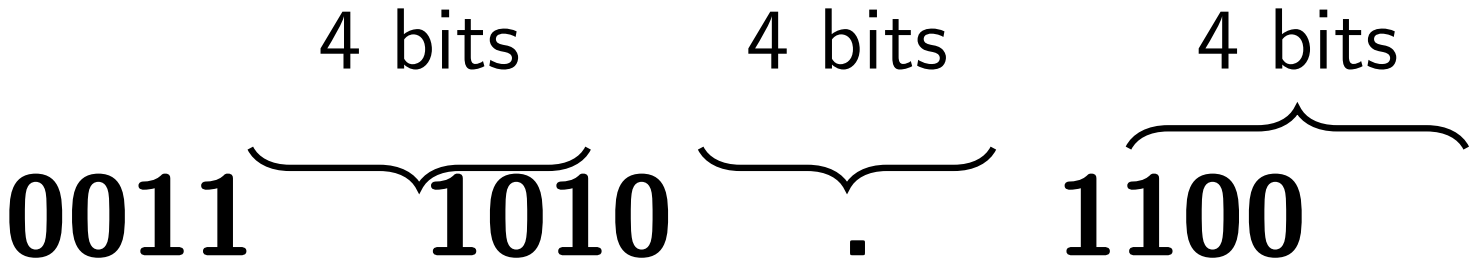
Example: Binary to Octal

- Convert $(10110.01)_2$ to Octal.
- Integer part (right to left): 10 110 $\rightarrow$ pad $\rightarrow$ 010 110 $\rightarrow$ 2 6
- Fraction part (left to right): .01 $\rightarrow$ pad $\rightarrow$ .010 $\rightarrow$ 2
- Result: $(26.2)_8$

$$\begin{array}{ccccc} 010 & & 110 & \cdot & 010 \\ \downarrow & & \downarrow & & \downarrow \\ 2 & & 6 & . & 2 \end{array}$$

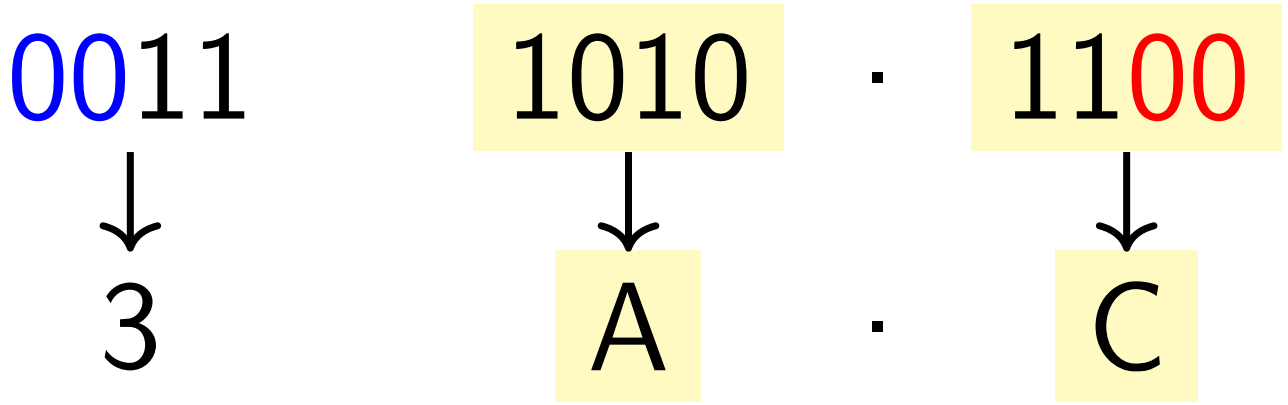
Direct Conversion: Binary to Hexadecimal

- Because $16 = 2^4$, 4 binary bits $=$ 1 hex digit.
- Method: Group bits into fours, starting from the radix point.
- Same padding rules: integers pad left, fractions pad right.
- Convert each 4-bit group to its hex equivalent (0-F).



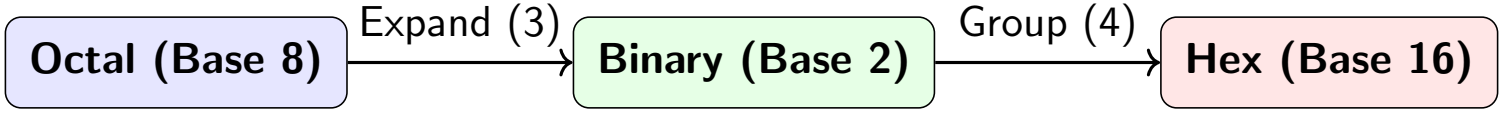
Example: Binary to Hexadecimal

- Convert $(111010.11)_2$ to Hex.
- Integer: 11 1010 $\rightarrow$ pad $\rightarrow$ 0011 1010
- 0011 = 3, 1010 = A
- Fraction: .11 $\rightarrow$ pad $\rightarrow$ .1100
- 1100 = C
- Result: $(3A.C)_{16}$



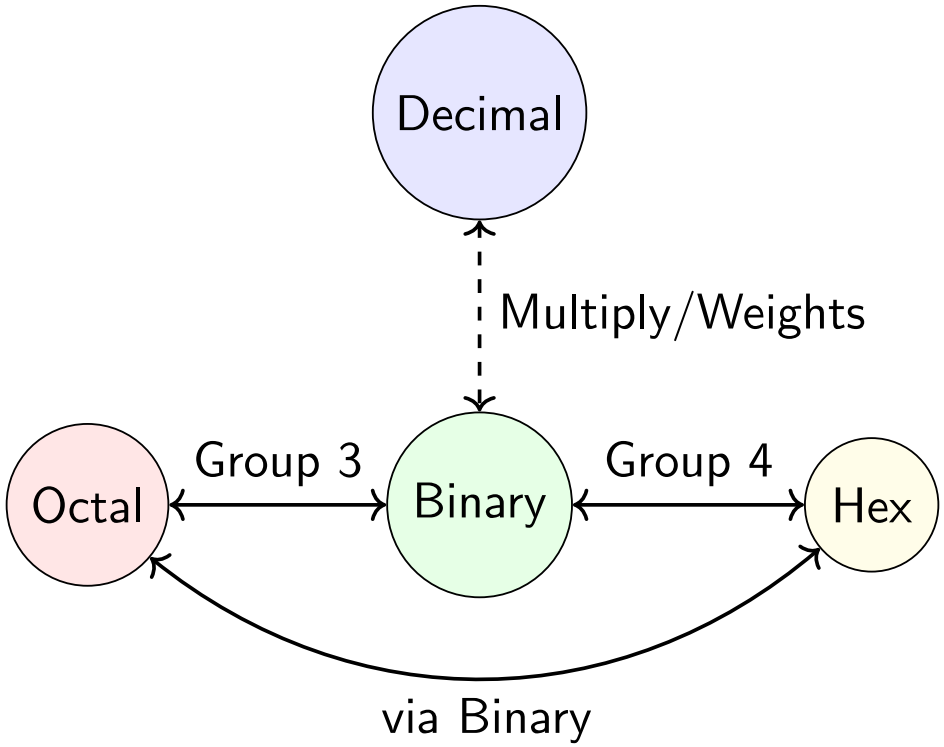
Direct Conversion: Octal to Hexadecimal

- We cannot go directly from Octal to Hex easily.
- Intermediate Step: Convert Octal to Binary first.
- Then group the Binary into fours to get Hex.
- Example: $(74)_8 \rightarrow$ Hex
- Octal to Binary: $7 = 111$, $4 = 100 \rightarrow 111100$
- Binary to Hex: $0011\ 1100 \rightarrow 3\ C \rightarrow (3C)_{16}$



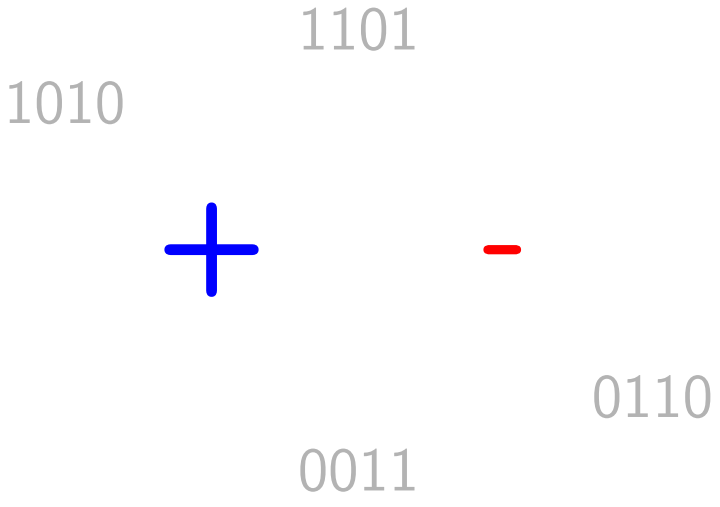
Summary of All Methods

- Fraction Dec to Base: Multiply, read Top to Bottom.
- Fraction Base to Dec: Sum of negative weights.
- Bin to Oct/Hex: Group bits (3 for Oct, 4 for Hex) from radix point.
- Oct/Hex to Bin: Expand digits into 3/4 bits.
- Oct $\leftrightarrow$ Hex: Use Binary as a middle-man.



Next Lecture Preview

- Binary Arithmetic Operations.
- How do computers add and subtract at the hardware level?
- Rules for Binary Addition.
- Rules for Binary Subtraction (Borrowing).
- We will start building the logic for ALU (Arithmetic Logic Unit).



Questions?

Lecture 1.5: Binary arithmetic operations: addition, subtraction

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

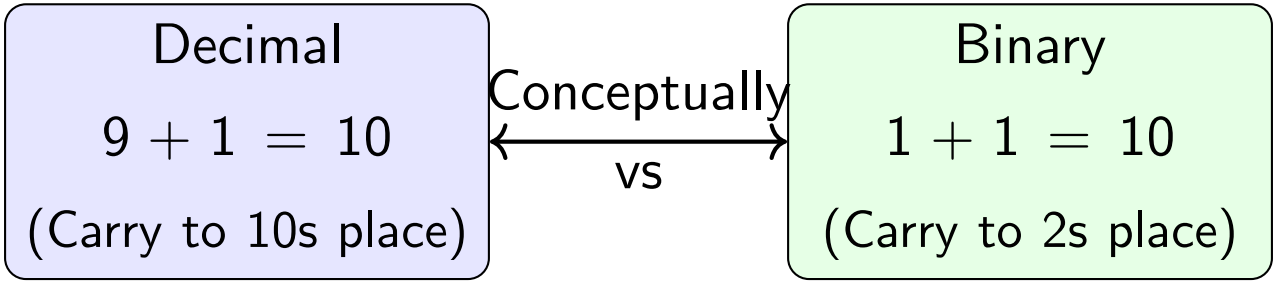
Table of Contents

Recap & Agenda

- ⊕ Introduction to Binary Arithmetic
- ⊕ Rules of Binary Addition
- ⊕ Addition Examples & Carries
- ⊖ Rules of Binary Subtraction
- ⊖ Understanding the Borrow
- ⊖ Subtraction Examples
- ± Summary

Intro to Binary Arithmetic

- Arithmetic operations in digital systems are performed in binary.
- Exactly the same concepts as decimal arithmetic.
- The difference: we only have two digits (0 and 1).
- Because the base is smaller, carries and borrows happen much more frequently.



Rules of Binary Addition

- $0 + 0 = 0$
- $0 + 1 = 1$
- $1 + 0 = 1$
- $1 + 1 = 0$, with a carry of 1
- $1 + 1 + 1 = 1$, with a carry of 1 (when adding a carry from previous bit)

A	+	B		=	Sum	Carry
0	+	0		=	0	0
0	+	1		=	1	0
1	+	0		=	1	0
1	+	1		=	0	1
1	+	1	+ 1 (prev)	=	1	1

Example: Simple Binary Addition

- Add: $(1010)_2 + (0101)_2$

$$\begin{array}{cccccc} & & 1 & 0 & 1 & 0 \\ + & & 0 & 1 & 0 & 1 \\ \hline & & 1 & 1 & 1 & 1 \end{array}$$

Check: $10 + 5 = 15$. Binary 1111 is 15.

Example: Addition with Carries

- Add: $(1011)_2 + (1101)_2$

Carries:	1	1	1	1		
		1	0	1	1	$(11)_{10}$
+		1	1	0	1	$(13)_{10}$
1	1	0	0	0		$(24)_{10}$

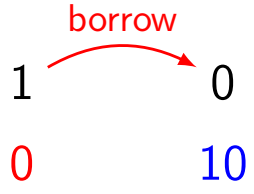
Rules of Binary Subtraction

- 0 - 0 = 0
- 1 - 0 = 1
- 1 - 1 = 0
- 0 - 1 = 1, with a borrow of 1 from the next higher significant bit.

A	-	B	=	Diff	Borrow
0	-	0	=	0	0
1	-	0	=	1	0
1	-	1	=	0	0
0	-	1	=	1	1

Understanding the 'Borrow'

- When you borrow in decimal, you borrow 10.
- When you borrow in binary, you borrow a power of 2.
- Borrowing from the next column turns the current '0' into '10' (which is 2 in decimal).
- So, $10_2 - 1_2 = 1_2$ (in binary terms: $2 - 1 = 1$).



The borrowed '1' acts as two '1's in the lower column.

Example: Simple Subtraction

- Subtract: $(1101)_2 - (1001)_2$

$$\begin{array}{rcccc} & 1 & 1 & 0 & 1 \\ - & 1 & 0 & 0 & 1 \\ \hline & 0 & 1 & 0 & 0 \end{array}$$

Check: $13 - 9 = 4$.

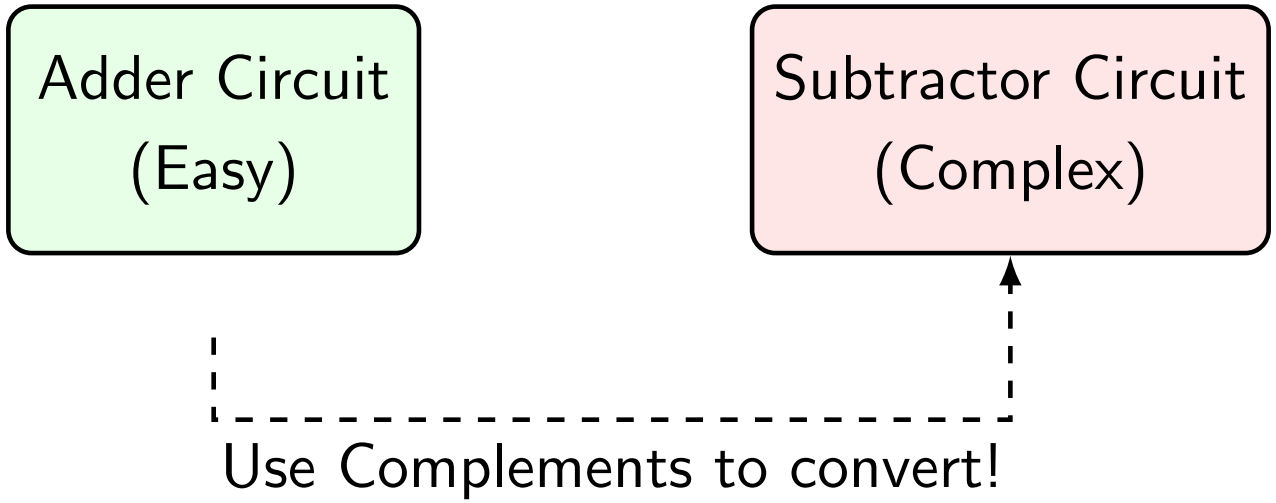
Example: Subtraction with Borrow

- Subtract: $(1010)_2 - (0111)_2$

Borrows:	0	1	10	10	
	1	0	1	0	$(10)_{10}$
-	0	1	1	1	$(7)_{10}$
	0	0	1	1	$(3)_{10}$

Hardware Implications

- Addition is implemented using logic gates in circuits called 'Adders'.
- Direct subtraction requires complex 'borrow' logic circuits.
- Is there a way to do subtraction using ONLY adder circuits?
- Yes, using the method of 'Complements', which we will learn soon.



Summary

- Binary addition follows four simple rules.
- $1 + 1 = 0$ with a carry of 1.
- Binary subtraction relies on borrowing.
- 0 - $1 = 1$ with a borrow of 1.
- A borrowed '1' acts as '10' (decimal 2) in the current column.

Addition Carry

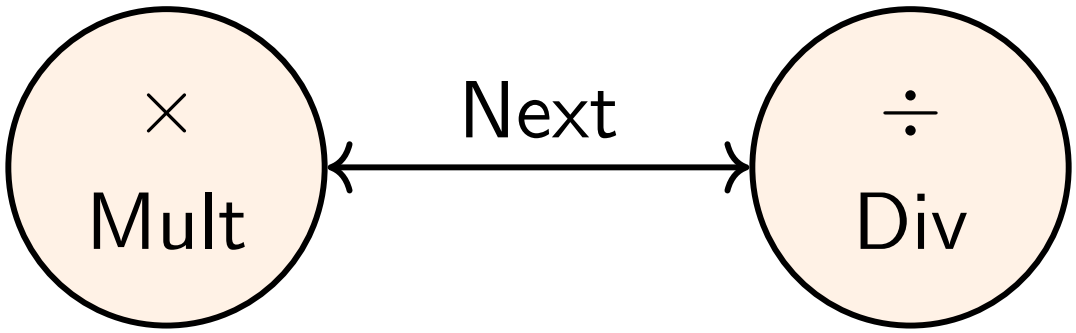
$1 + 1 = 10_2$

Subtraction Borrow

$10_2 - 1 = 1$

Next Lecture Preview

- Binary Multiplication.
- Binary Division.
- Using shift operations for multiplication.
- Completing the basic arithmetic operations.



Questions?

Lecture 1.6: Binary arithmetic operations: multiplication and division

Complete Lecture Deck – All 45 Lectures

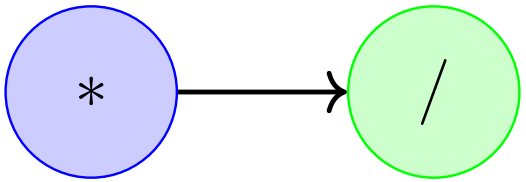
Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

Recap & Agenda

- Rules of Binary Multiplication
- The Shift-and-Add Method
- Example: Binary Multiplication
- Rules of Binary Division
- Long Division in Binary
- Example: Binary Division
- Summary



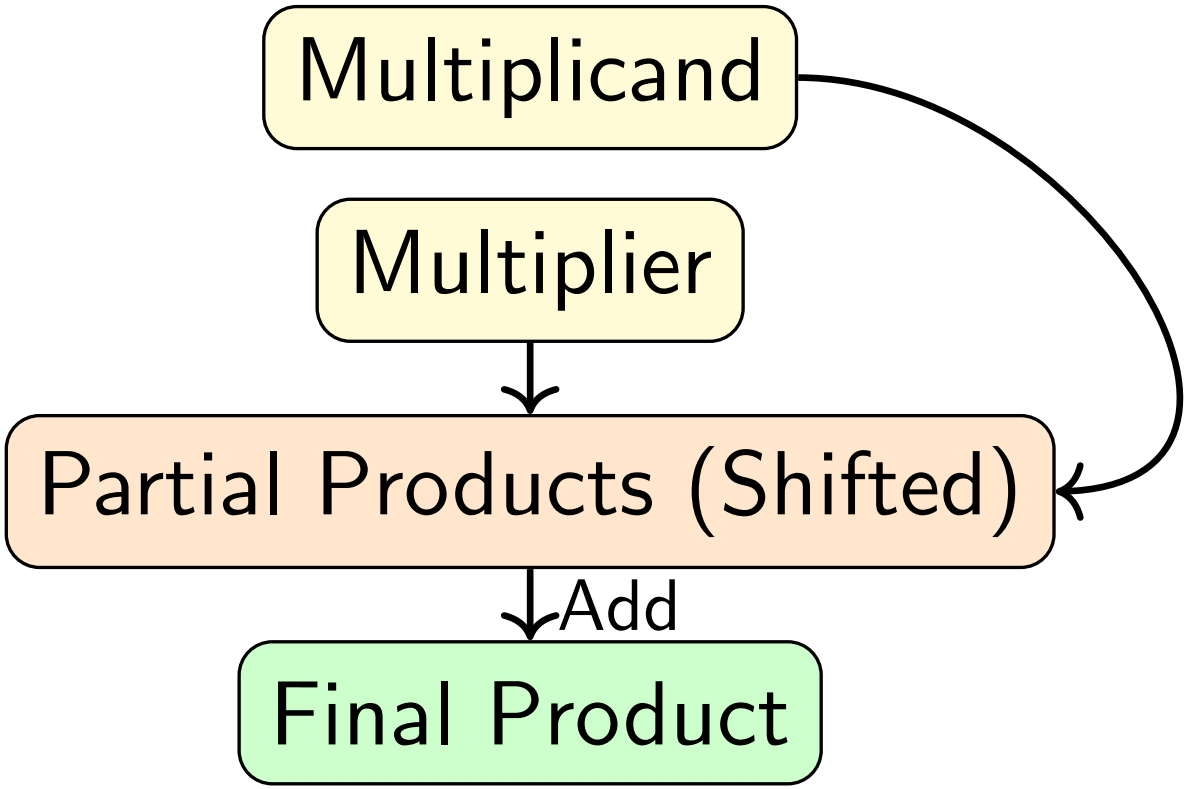
Rules of Binary Multiplication

- Very simple, exactly like the logical AND operation.
- $0 \times 0 = 0$
- $0 \times 1 = 0$
- $1 \times 0 = 0$
- $1 \times 1 = 1$
- No carries generated during the multiplication step itself.

$\times$	**0**	**1**
0	0	0
1	0	1

The Multiplication Process

- Uses the 'Shift and Add' method.
- For each bit in the multiplier:
- If bit is 1, copy the multiplicand (shifted appropriately).
- If bit is 0, write all 0s (shifted appropriately).
- Add all the partial products together.



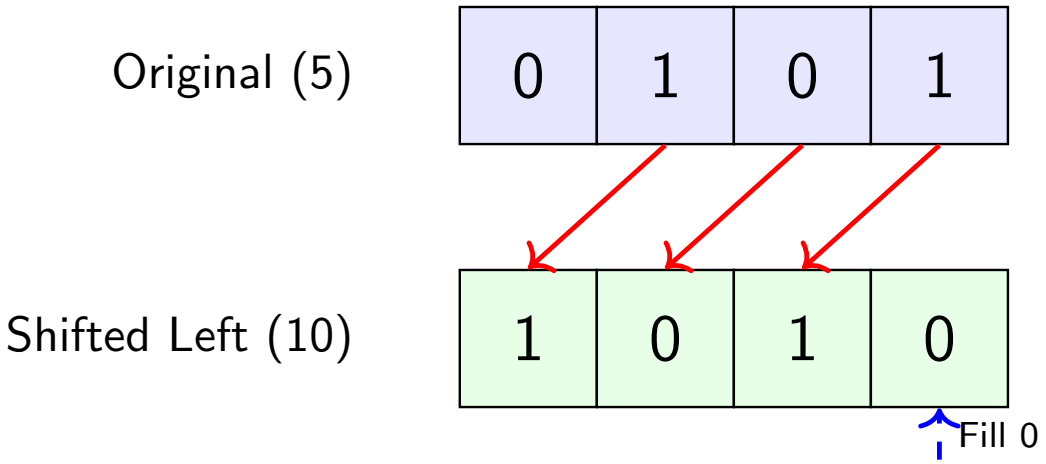
Example: Binary Multiplication

- Multiply: $(1101)_2$ by $(101)_2$

1101	(13) Multiplicand
$\times$ 101	(5) Multiplier
1101	(partial for 1)
0000 $\leftarrow$	(partial for 0, shifted)
1101 $\leftarrow\leftarrow$	(partial for 1, shifted)
1000001	(65) Final Product

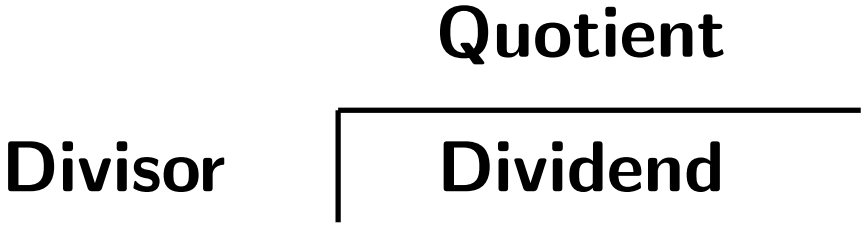
Hardware Note: Shift Registers

- In a CPU, multiplication is often done using a Shift Register and an Adder.
- Multiplying by 2 is just a left shift: $0101(5) \ll 1 = 1010(10)$.
- Dividing by 2 is a right shift: $1100(12) \gg 1 = 0110(6)$.



Rules of Binary Division

- Process similar to decimal long division.
- Two outcomes when comparing divisor to dividend portion:
- 1. Divisor fits: write 1 in quotient, subtract.
- 2. Divisor doesn't fit: write 0 in quotient, bring down next bit.



Example: Binary Division

- Divide: $(1111)_2$ by $(11)_2$
- Divisor=11, Dividend=1111

	101	(Quotient)
11	| 1111	
	-11	
	001	
	000	
	011	
	-11	
	0	(Remainder)

- Result: Quotient = 101, Remainder = 0. (Check: $15/3 = 5$)

Another Division Example

- Divide: $(10110)_2$ by $(10)_2$ (Check: $22/2 = 11$)

	1011	(Quotient)
10	10110	
	-10	
	001	
	000	
	011	
	-10	
	010	
	-10	
	0	(Remainder)

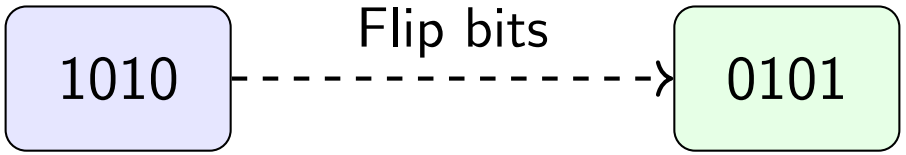
Summary

- Multiplication rules: $1 \times 1 = 1$, all else is 0.
- Multiplication uses Shift-and-Add method.
- Left shift = multiply by 2.
- Division uses standard long division.
- At each step, the divisor goes in either 0 or 1 times.
- Right shift = divide by 2.

Operation	**Equivalent Shift**
Multiply by 2	Shift Left ($\ll 1$)
Divide by 2	Shift Right ($\gg 1$)

Next Lecture Preview

- 1's and 2's Complement of a binary number.
- What are complements?
- How do they help hardware designers?
- Representing negative numbers in digital systems.



Questions?

Lecture 1.7: 1's and 2's Complement of binary number

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

Recap & Agenda

- Why do we need Complements?
- What is a Complement?
- Finding the 1's Complement
- Finding the 2's Complement
- Shortcut for 2's Complement
- Representing Signed Numbers
- Summary

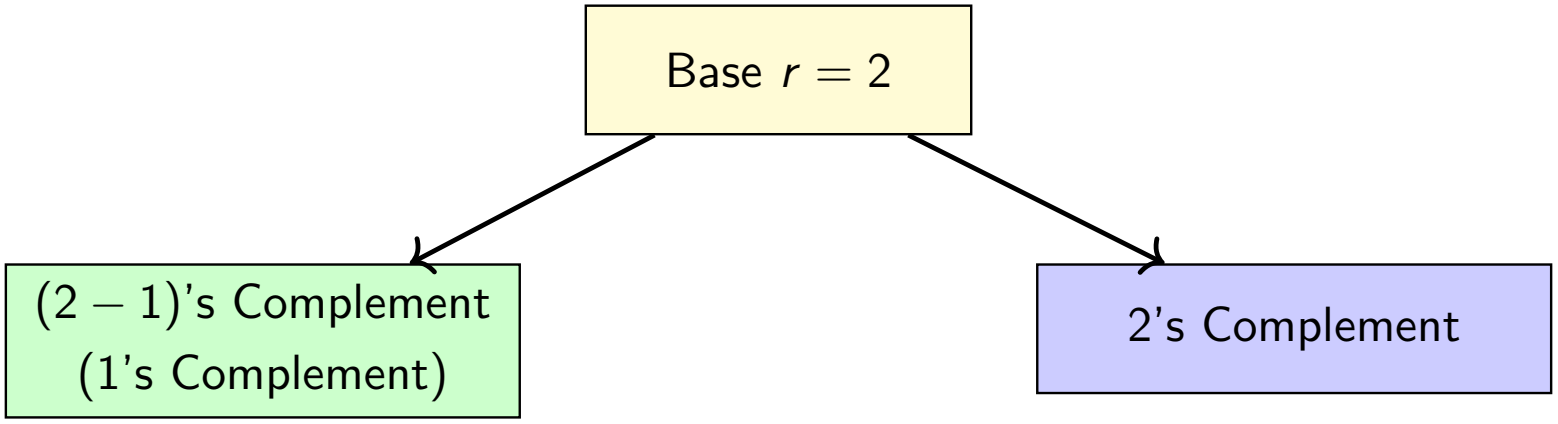
2's Complement

Why Complements?

- Hardware Limitation: Circuits can only easily represent 0 and 1.
- There is no 'minus sign' (-) in hardware logic.
- How do we tell the computer a number is negative (-5)?
- How do we build a simpler ALU that doesn't need complex borrow logic for subtraction?
- Solution: Complement systems.

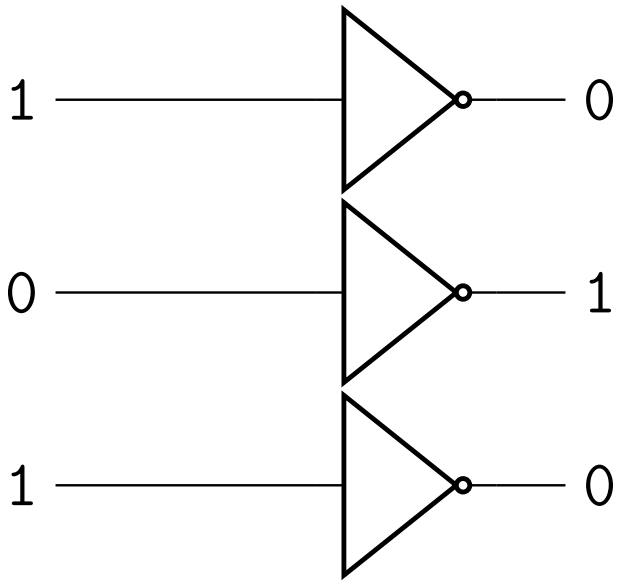
What is a Complement?

- Every base 'r' system has two types of complements:
- 1. The Radix Complement (r's complement).
- 2. The Diminished Radix Complement ((r-1)'s complement).
- For Base 2, these are the 2's complement and 1's complement.



Finding the 1's Complement

- Rule: Simply invert every bit.
- Change every 0 to 1, and every 1 to 0.
- Hardware equivalent: Passing the bits through NOT gates (Inverters).



Example: 1's Complement

- Find the 1's complement of (1011001).

Original:	1	0	1	1	0	0	1
	↓	↓	↓	↓	↓	↓	↓
Invert:	0	1	0	0	1	1	0

Result: (0100110)

Finding the 2's Complement

- Rule: Find the 1's complement, then add 1 to the LSB.
- Formula: 2's Comp = 1's Comp + 1
- This is the standard, most widely used representation for negative integers in computing.

2's Complement = (NOT N) + 1

Example: 2's Complement

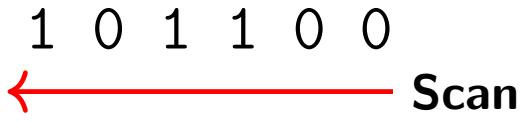
- Find the 2's complement of (101100).
- Step 1: 1's complement (invert) $\rightarrow$ 010011
- Step 2: Add 1 to LSB

$$\begin{array}{cccccccc} & & & & & {\scriptstyle 1} & & {\scriptstyle 1} \\ & 0 & 1 & 0 & 0 & 1 & 1 & \\ + & & & & & & 1 & \\ \hline & 0 & 1 & 0 & 1 & 0 & 0 & \end{array}$$

Result: (010100)

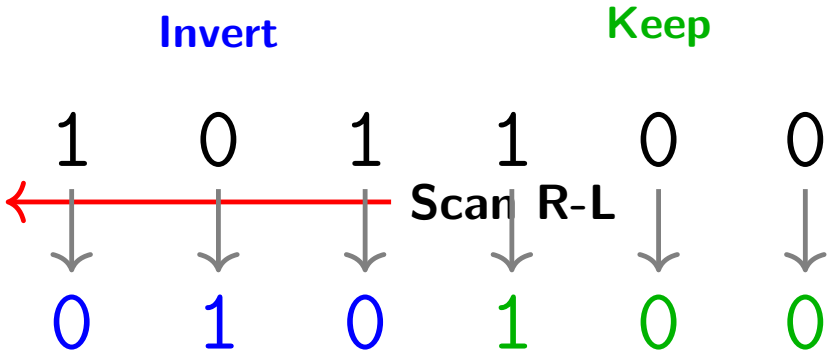
Shortcut for 2's Complement

- Want to find 2's complement instantly?
- 1. Scan the number from Right to Left (LSB to MSB).
- 2. Copy all bits exactly as they are up to and including the FIRST '1'.
- 3. Invert all remaining bits to the left.
- Let's try (101100).



Shortcut Example

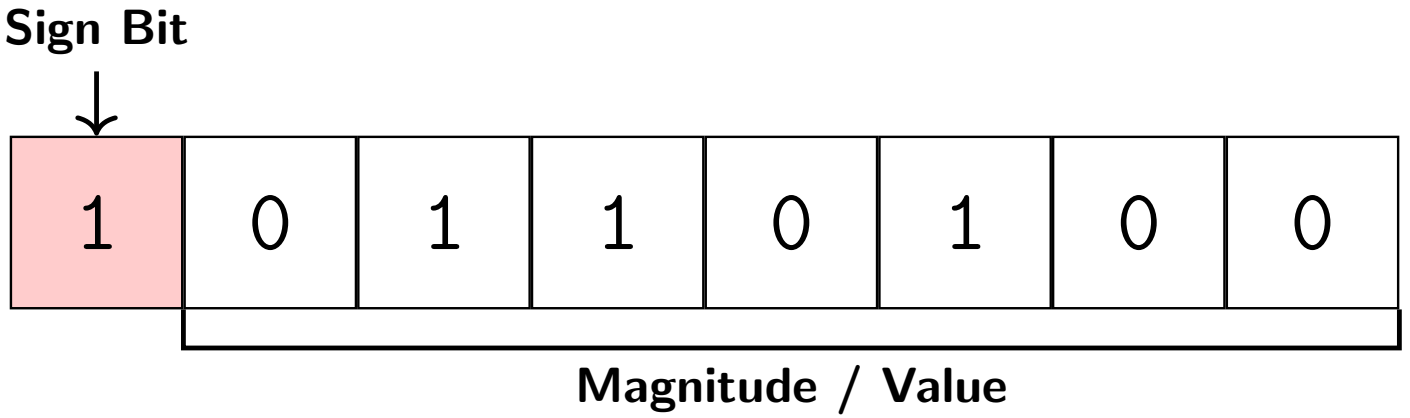
- —[text width=2.5cm, align=right]— Original: 1 0 1 1 0 0
- Scan R-L



Same result as before!

Representing Signed Numbers

- In a signed binary system, the MSB is the Sign Bit.
- 0 means Positive (+), 1 means Negative (-).
- Positive numbers are stored in normal binary.
- Negative numbers are stored in 2's complement form.



Summary

- 1's complement = Invert all bits (0→1, 1→0).
- 2's complement = 1's complement + 1.
- Shortcut: Keep bits from right up to first 1, invert the rest.
- MSB is used as a sign bit: 0 for +, 1 for -.
- Negative numbers are typically stored as 2's complements.

Key Takeaways:

- 1's comp: Simple bit inversion.
- 2's comp: Add 1 to inverted bits.
- Essential for negative numbers.

Next Lecture Preview

- Subtraction using complements.
- How to do M - N by actually doing M + (-N).
- Step-by-step rules for 1's complement subtraction.
- Step-by-step rules for 2's complement subtraction.

M - N $\Rightarrow$ M + (2's comp of N)

Questions?

Lecture 1.8: Subtraction using 1's and 2's complement method

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

Recap & Agenda

- The Principle: Subtraction as Addition
- Rules: 1's Complement Subtraction
- Example: M ¿ N and M ¡ N (1's comp)
- Rules: 2's Complement Subtraction
- Example: M ¿ N and M ¡ N (2's comp)
- Why Hardware Prefers 2's Complement
- Unit 1 Summary

Subtraction as Addition

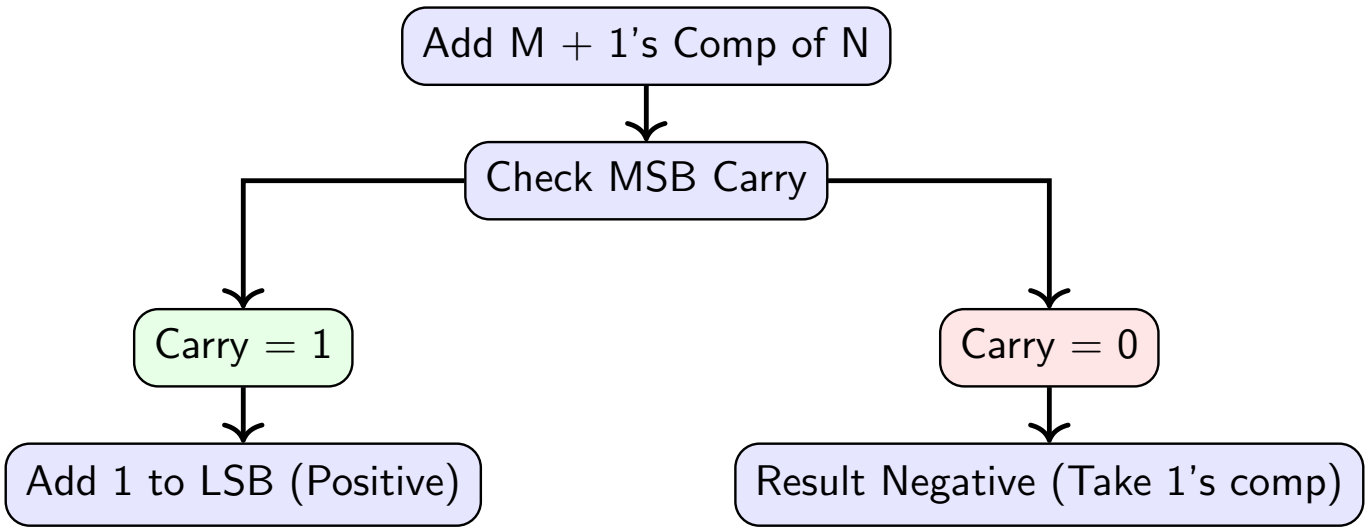
- Algebraically, M - N is the same as M + (-N).
- We know that complements represent negative numbers.
- Therefore, to subtract N from M:
  1. Find the complement of N.
  2. Add it to M.
- No 'borrowing' required! Only adder circuits needed.

Core Principle

$$M - N = M + (-N) = M + (\text{Complement of } N)$$

Rules: 1's Complement Subtraction

- To perform M - N:
  1. Ensure both numbers have the same number of bits (pad with 0s).
  2. Find the 1's complement of N (invert bits).
  3. Add M and the 1's complement of N.
  4. Check the carry out of the MSB:
     - If Carry = 1 (End-around carry): Add it to the LSB. Answer is positive.
     - If Carry = 0: Answer is negative. Take 1's complement of the result to find magnitude.



Example: 1's Comp (M ¿ N)

- Subtract (1011) - (0101) [11 - 5 = 6]
- M = 1011, N = 0101 → 1's comp of N = 1010

$$
\begin{array}{rr}
 & 1\ 0\ 1\ 1 \\
 & +\ 1\ 0\ 1\ 0 \\
\hline
\text{Carry:} & 1\ 0\ 1\ 0\ 1 \\
 & \quad +\ \ 1 \quad \text{End-around carry} \\
\hline
 & 0\ 1\ 1\ 0
\end{array}
$$

- End-around carry: Add 1 to LSB → 0101 + 1 = 0110 (+6).

Example: 1's Comp (M ¡ N)

- Subtract (0101) - (1011) [5 - 11 = -6]
- M = 0101, N = 1011 $\rightarrow$ 1's comp of N = 0100

$$\begin{array}{r} 0\ 1\ 0\ 1 \\ +\ 0\ 1\ 0\ 0 \\ \hline \text{Carry: } 0\ 0\ 1\ 0\ 0\ 1 \end{array}$$

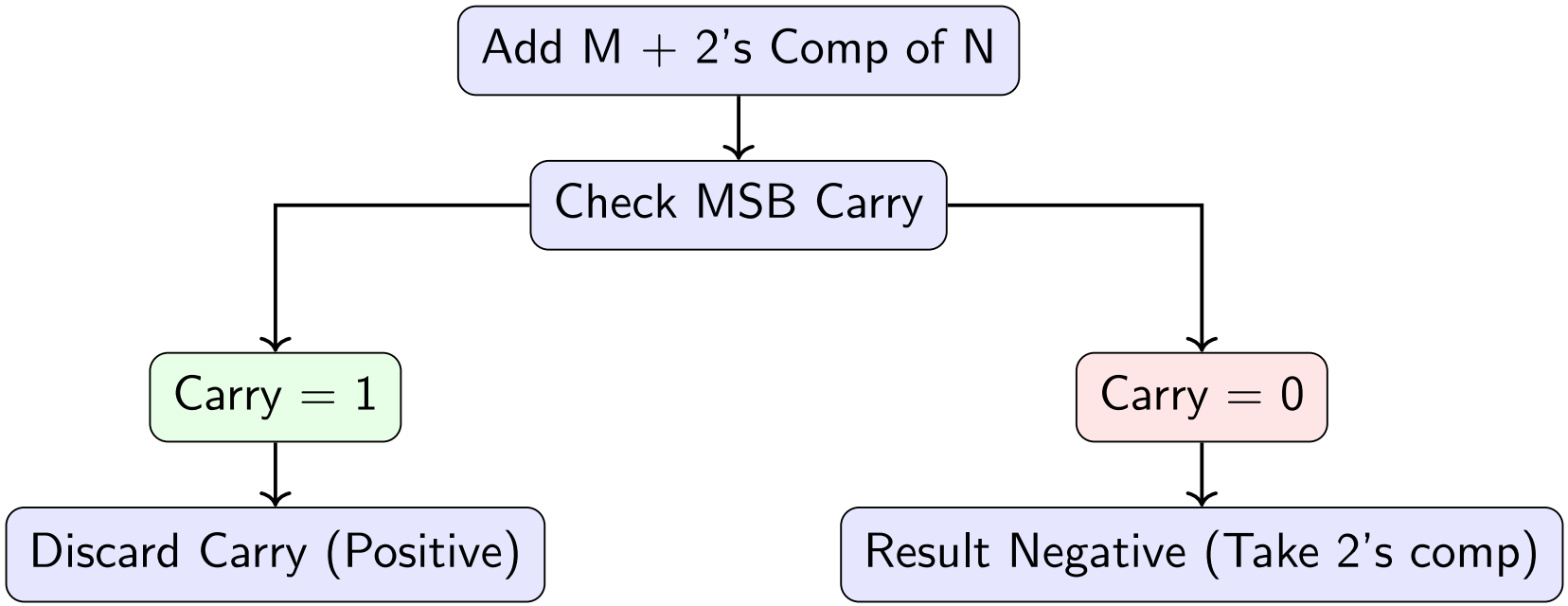
No Carry. Result is Negative.

$\downarrow$

1's comp of 1001 $\rightarrow$ 0110 (-6)

Rules: 2's Complement Subtraction

- To perform M - N:
  1. Ensure equal bit lengths.
  2. Find the 2's complement of N.
  3. Add M and the 2's complement of N.
  4. Check the carry out of the MSB:
     - If Carry = 1: DISCARD IT. Answer is positive.
     - If Carry = 0: Answer is negative. Take 2's complement of the result to find magnitude.



Example: 2's Comp (M ¿ N)

- Subtract (1010) - (0011) [10 - 3 = 7]
- M = 1010, N = 0011 $\rightarrow$ 2's comp of N = 1101

$$\begin{array}{rr} & 1\ 0\ 1\ 0 \\ + & 1\ 1\ 0\ 1 \\ \hline \text{Carry: } & \cancel{1}\ 0\ 1\ 1\ 1 \end{array}$$

Discard Carry $\rightarrow +7$

Example: 2's Comp (M ¡ N)

- Subtract (0011) - (1010) [3 - 10 = -7]
- M = 0011, N = 1010 $\rightarrow$ 2's comp of N = 0110

$$\begin{array}{r} 0\ 0\ 1\ 1 \\ +\ 0\ 1\ 1\ 0 \\ \hline \text{Carry: } 0\ 0\ 1\ 0\ 0\ 1 \end{array}$$

No Carry. Result is Negative.

$\downarrow$

2's comp of 1001 $\rightarrow$ 0111 (-7)

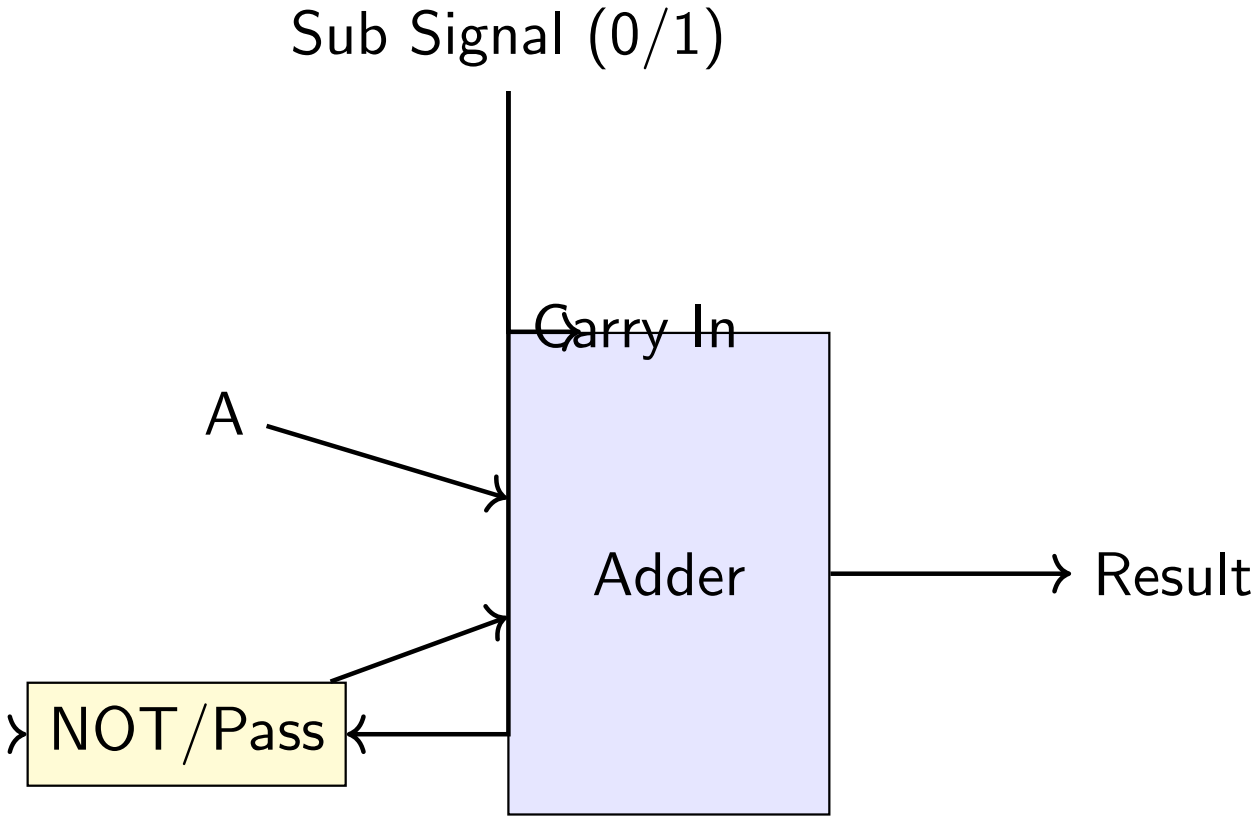
Why Hardware Prefers 2's Complement

- **No End-Around Carry:**
  - 1's comp requires a second addition step if there is a carry.
  - 2's comp does it in one pass.
- **Single Zero Representation:**
  - 1's comp has +0 (0000) and -0 (1111).
  - 2's comp only has one zero (0000).
- **Result:** Simpler, faster logic circuits.

Feature	**1's Complement**	**2's Complement**
Zero Representation	Two (+0, -0)	One (0)
End-around Carry	Yes (slower)	No (discard, faster)
Hardware	More complex	Simpler

The Adder/Subtractor Circuit

- With 2's complement, the ALU only needs an Adder circuit.
- To Add (A + B): pass B directly to Adder.
- To Subtract (A - B): pass B through NOT gates, and set the Adder's 'Carry In' bit to 1.
- (NOT B) + 1 is exactly the 2's complement of B!

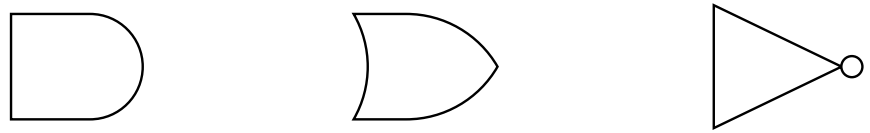


Summary of Unit 1

- We learned Binary, Octal, Hex and how to convert between them.
- We covered binary arithmetic: $+$, -, *, $/$.
- We learned about Complements and signed numbers.
- Subtraction is performed using 2's complement addition.
- This forms the mathematical foundation of digital logic.

Next Unit Preview

- Unit 2: Logic Gates and Boolean Algebra
- Introduction to AND, OR, NOT gates.
- Universal Gates (NAND, NOR).
- Boolean theorems and simplifying logic expressions.



Questions?

Lecture 2.1: Digital Logic Gates: AND, OR, NOT

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

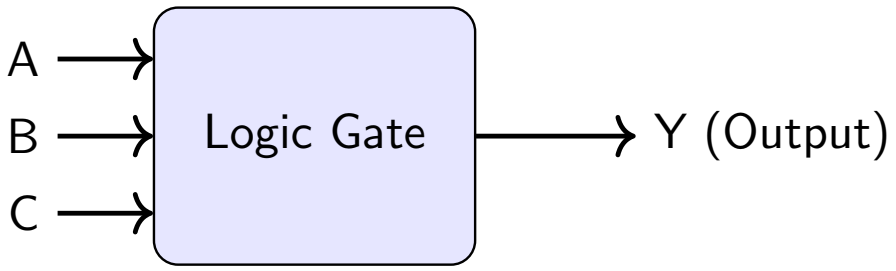
Lecturer, GP Palanpur

Table of Contents

Lecture 2.1: Digital Logic Gates: AND, OR, NOT

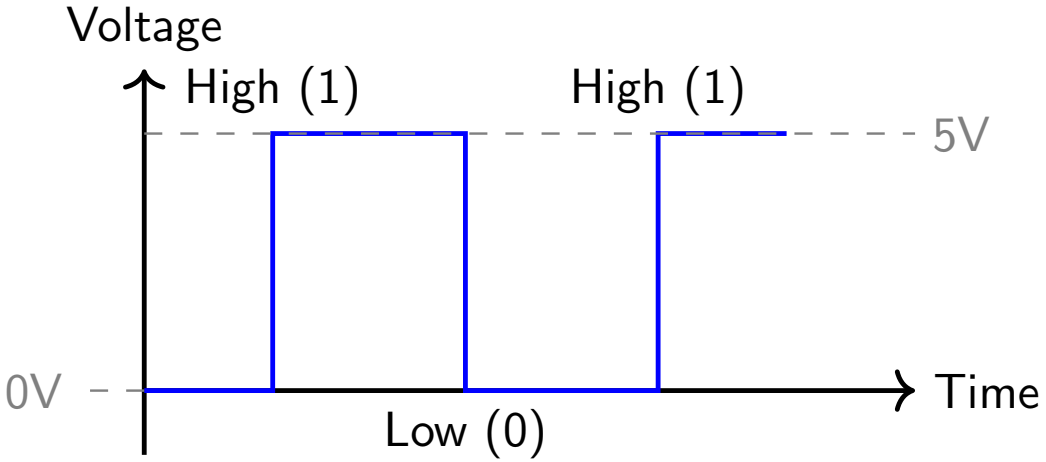
Introduction to Logic Gates

- Logic gates are the basic building blocks of any digital system.
- They are electronic circuits having one or more than one input and only one output.
- The relationship between the input and the output is based on a certain logic.
- Gates process signals which represent true or false (1 or 0).



Digital Signals and Logic Levels

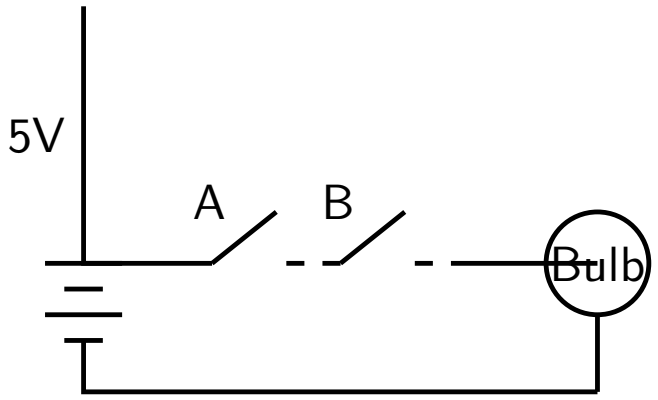
- Digital systems use binary numbers: 0 and 1.
- Logic 1 is typically represented by a high voltage (e.g., 5V).
- Logic 0 is typically represented by a low voltage (e.g., 0V).
- Truth tables describe how a logic circuit's output depends on the logic levels present at the circuit's inputs.



Lecture 2.1: Digital Logic Gates: AND, OR, NOT

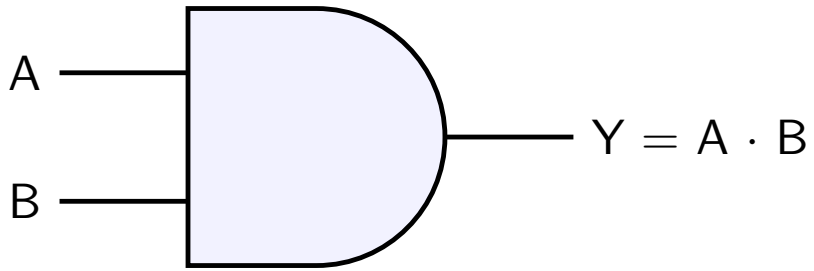
The AND Gate: Concept

- The AND gate performs logical multiplication.
- It can have two or more inputs, but only one output.
- The output is HIGH (1) only when ALL inputs are HIGH (1).
- If any input is LOW (0), the output is LOW (0).



The AND Gate: Symbol and Expression

- Standard IEEE/ANSI symbol for a 2-input AND gate features a flat input side and a curved output side.
- Boolean expression: $Y = A \cdot B$ or simply $Y = AB$
- Pronounced as 'Y equals A AND B'.



The AND Gate: Truth Table

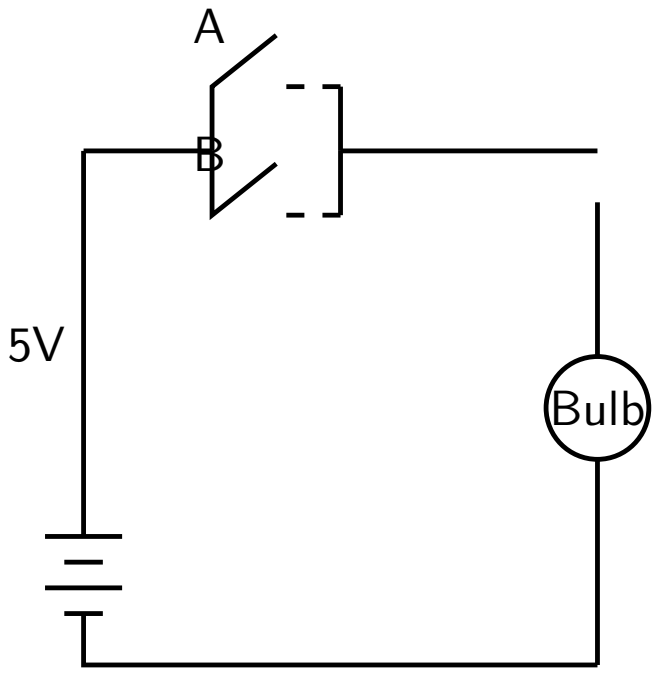
- A truth table lists all possible combinations of input values and the corresponding output.
- For 2 inputs, there are $2^2 = 4$ combinations.

Inputs		**Output**
A	**B**	$\mathbf{Y = A \cdot B}$
0	0	0
0	1	0
1	0	0
1	1	1

Lecture 2.1: Digital Logic Gates: AND, OR, NOT

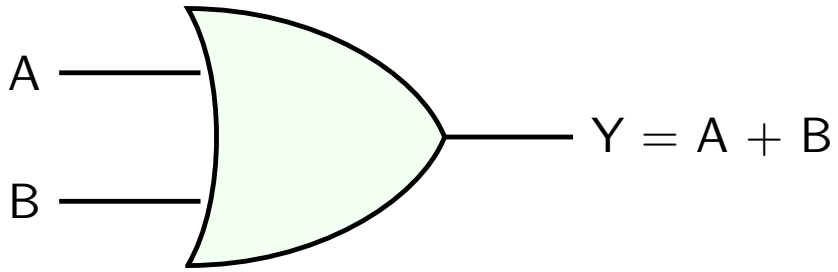
The OR Gate: Concept

- The OR gate performs logical addition.
- It can have two or more inputs and one output.
- The output is HIGH (1) if AT LEAST ONE input is HIGH (1).
- The output is LOW (0) only when ALL inputs are LOW (0).



The OR Gate: Symbol and Expression

- Standard symbol features a curved input side and a pointed output side.
- Boolean expression: $Y = A + B$
- Pronounced as 'Y equals A OR B'.



The OR Gate: Truth Table

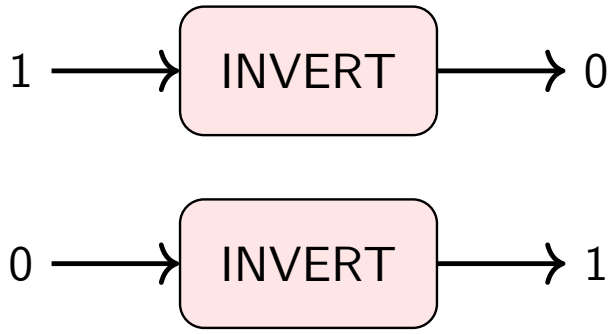
- For a 2-input OR gate:
- Inputs: 00 $\rightarrow$ Output: 0
- Inputs: 01 $\rightarrow$ Output: 1
- Inputs: 10 $\rightarrow$ Output: 1
- Inputs: 11 $\rightarrow$ Output: 1

Inputs		**Output**
A	**B**	**Y = A + B**
0	0	0
0	1	1
1	0	1
1	1	1

Lecture 2.1: Digital Logic Gates: AND, OR, NOT

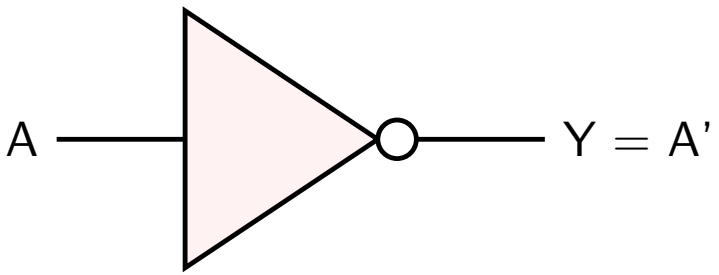
The NOT Gate (Inverter)

- The NOT gate performs logical inversion or complementation.
- It is unique because it has exactly ONE input and ONE output.
- If the input is HIGH (1), the output is LOW (0).
- If the input is LOW (0), the output is HIGH (1).



NOT Gate: Symbol and Truth Table

- Symbol: A triangle pointing in the direction of signal flow, with a small circle (bubble) at the output.
- Boolean expression: Y = A' or $Y = \bar{A}$
- Truth Table has only 2 rows: Input 0 $\rightarrow$ Output 1, Input 1 $\rightarrow$ Output 0.

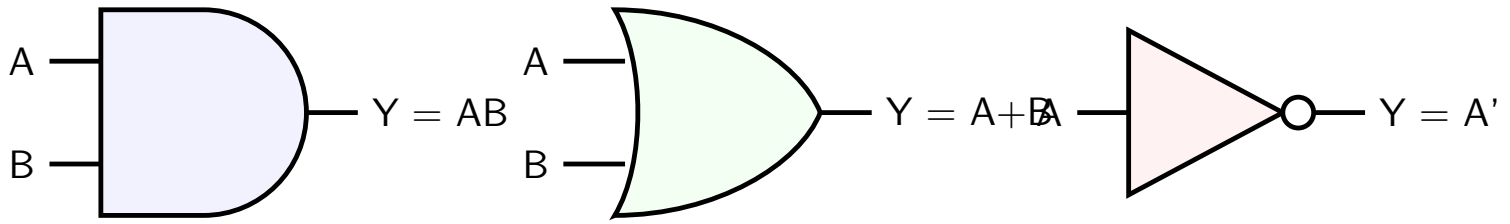


Input A	**Output Y = A'**
0	1
1	0

Lecture 2.1: Digital Logic Gates: AND, OR, NOT

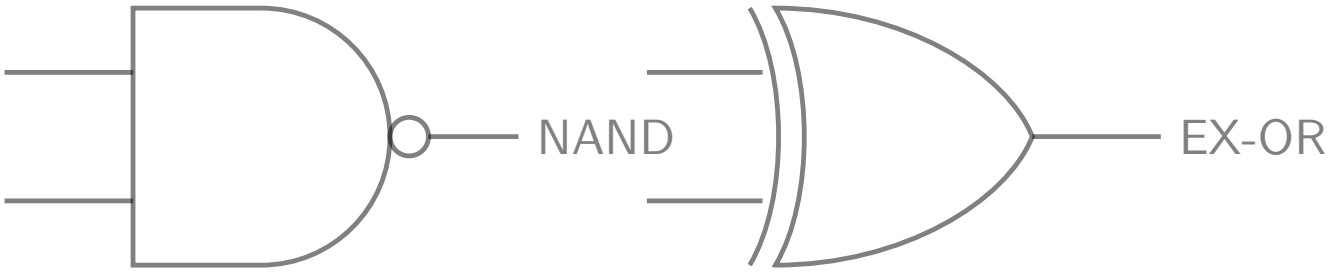
Summary

- AND Gate: Output is 1 only when all inputs are 1 (Multiplication, $Y = A \cdot B$).
- OR Gate: Output is 1 when any input is 1 (Addition, $Y = A + B$).
- NOT Gate: Output is the exact opposite of the input (Inversion, $Y = A'$).
- Truth tables are systematic ways to describe gate behavior.



Next Lecture Preview

- Next time, we will explore Derived Gates: NAND, NOR, EX-OR, and EX-NOR.
- We'll see how combining the basic gates leads to new, incredibly useful operations.
- Please review the truth tables for AND, OR, and NOT before the next session.



Questions?

Lecture 2.2: Digital Logic Gates: NAND, NOR, EX-OR, EX-NOR

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

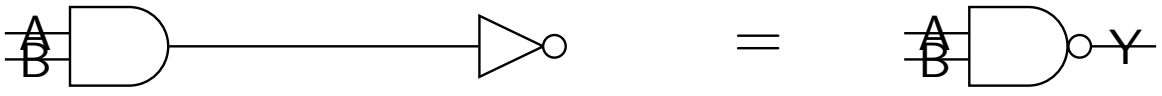
Recap & Agenda

- Recap of AND, OR, NOT
- Introduction to Derived Gates
- The NAND Gate
- The NOR Gate
- The Exclusive-OR (EX-OR) Gate
- The Exclusive-NOR (EX-NOR) Gate
- Summary and Q&A

Introduction to Derived Gates

Introduction to Derived Gates

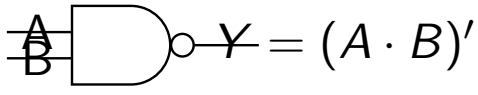
- Derived gates are created by combining basic logic gates (AND, OR, NOT).
- NAND is essentially an AND gate followed by a NOT gate.
- NOR is essentially an OR gate followed by a NOT gate.
- EX-OR and EX-NOR perform specialized parity and equality checking operations.



The NAND Gate

The NAND Gate: Concept & Symbol

- NAND stands for 'NOT AND'.
- It outputs LOW (0) ONLY when all its inputs are HIGH (1).
- For all other combinations, the output is HIGH (1).
- Symbol: An AND gate with a small inversion bubble at the output.



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

The NAND Gate: Truth Table

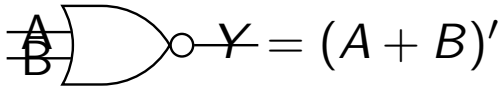
- Let's look at the 2-input NAND truth table:

Input A	**Input B**	**AND Output (A · B)**	**NAND Output (A · B)'**
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

The NOR Gate

The NOR Gate: Concept & Symbol

- NOR stands for 'NOT OR'.
- It outputs HIGH (1) ONLY when all its inputs are LOW (0).
- If any input is HIGH (1), the output is LOW (0).
- Symbol: An OR gate with a small inversion bubble at the output.



The NOR Gate: Truth Table

- Let's look at the 2-input NOR truth table:

Input A	Input B	OR Output (A + B)	NOR Output (A + B)'
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

The Exclusive-OR (EX-OR) Gate

The Exclusive-OR (EX-OR) Gate: Concept

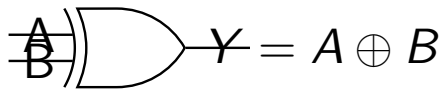
- Also known as XOR gate.
- Outputs HIGH (1) when the inputs are DIFFERENT (one is 0, the other is 1).
- Outputs LOW (0) when the inputs are THE SAME (both 0s or both 1s).
- It acts as an 'inequality detector'.

Odd number of 1s $\rightarrow$ Output 1

Even number of 1s
(or zero) $\rightarrow$ Output 0

EX-OR Gate: Symbol & Truth Table

- Symbol: Like an OR gate, but with a double curved line at the input.
- Expression: $Y = A \oplus B$ (A circle around a plus sign).

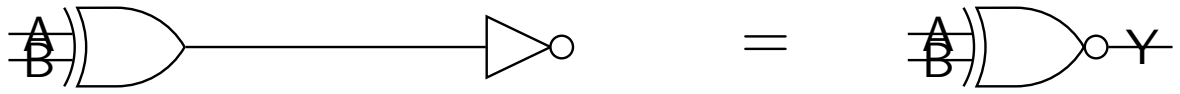


A	B	Output Y
0	0	0
0	1	1
1	0	1
1	1	0

The Exclusive-NOR (EX-NOR) Gate

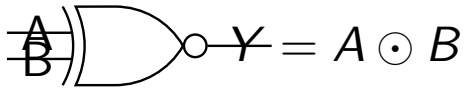
The Exclusive-NOR (EX-NOR) Gate: Concept

- Also known as XNOR gate.
- It is the logical inverse of the EX-OR gate.
- Outputs HIGH (1) when the inputs are THE SAME.
- Outputs LOW (0) when the inputs are DIFFERENT.
- It acts as an 'equality detector'.



EX-NOR Gate: Symbol & Truth Table

- Symbol: An EX-OR symbol with an inversion bubble at the output.
- Expression: $Y = A \odot B$ or $Y = (A \oplus B)'$.

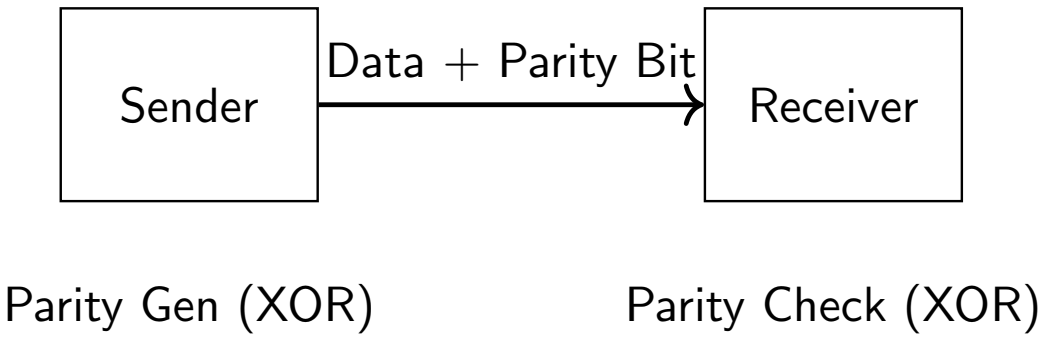


A	B	Output Y
0	0	1
0	1	0
1	0	0
1	1	1

Applications & Summary

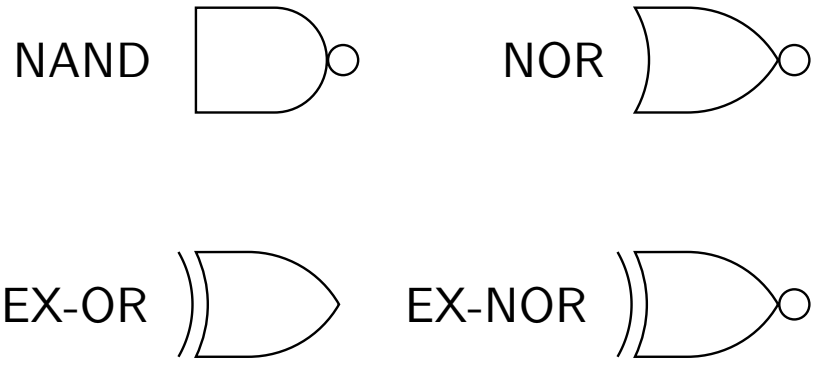
Applications of EX-OR / EX-NOR

- Parity Generation and Checking: Used in digital data transmission to detect errors.
- Binary Adders: EX-OR is the core of half adders and full adders.
- Comparators: EX-NOR is used to build digital comparators to check if two binary words are equal.



Summary

- NAND: Inverse of AND. Output 0 only if all inputs are 1.
- NOR: Inverse of OR. Output 1 only if all inputs are 0.
- EX-OR: Inequality detector. Output 1 if inputs are different.
- EX-NOR: Equality detector. Output 1 if inputs are the same.



Next Lecture Preview

- Next, we will shift from hardware logic gates to the mathematics that governs them.
- We'll introduce Boolean Algebra.
- We'll cover the basic theorems that allow us to simplify complex digital circuits.

$$A \cdot B = B \cdot A$$
$$A + A \cdot B = A$$
$$(A + B)' = A' \cdot B'$$

Questions?

Lecture 2.3: Basic Theorems of Boolean Algebra

Complete Lecture Deck – All 45 Lectures

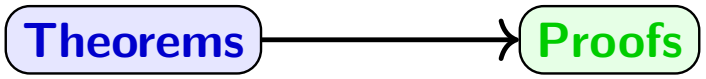
Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

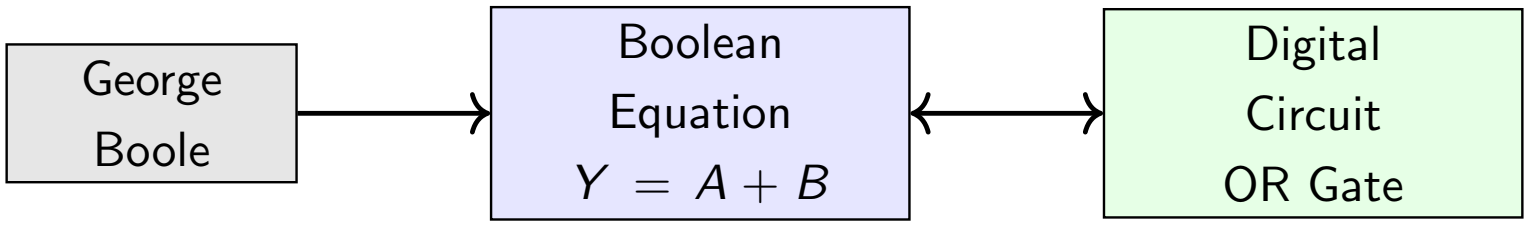
Recap & Agenda

- What is Boolean Algebra?
- Boolean Variables and Operations
- Single Variable Theorems (Identity, Null, Idempotent, Complement)
- Proving Theorems with Truth Tables
- Summary and Q&A



What is Boolean Algebra?

- Developed by George Boole in 1854.
- It is the mathematics of digital systems.
- Unlike regular algebra where variables can represent any number, Boolean variables can only take TWO values: 0 or 1.
- Used to analyze and simplify digital logic circuits.



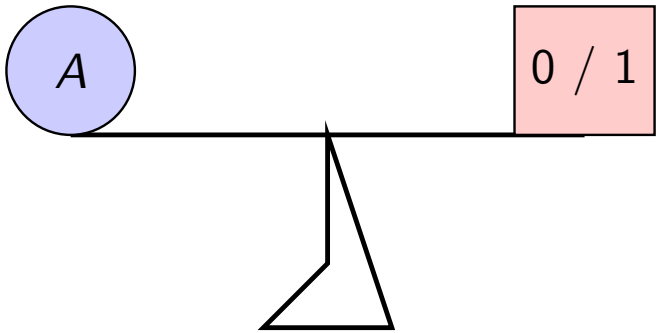
Boolean Variables and Operations

- Variables: Usually denoted by uppercase letters like A, B, C, X, Y.
- Complement (NOT): A', $\bar{A}$, or $\sim A$. Reverses the value.
- Addition (OR): $A + B$. Evaluates to 1 if A or B is 1.
- Multiplication (AND): $A \cdot B$. Evaluates to 1 only if both are 1.

Operation	**Symbol**	**Logic Gate**	**Meaning**
Addition	$+$	OR	logical sum
Multiplication	$\cdot$	AND	logical product
Complement	$'$ or $^-$	NOT	inversion

Basic Postulates

- Postulates are the fundamental truths we accept without proof.
- If A is a Boolean variable, it can only be:
- $A = 0$ or $A = 1$
- Inverse Postulate: If $A = 0$, then $A' = 1$. If $A = 1$, then $A' = 0$.



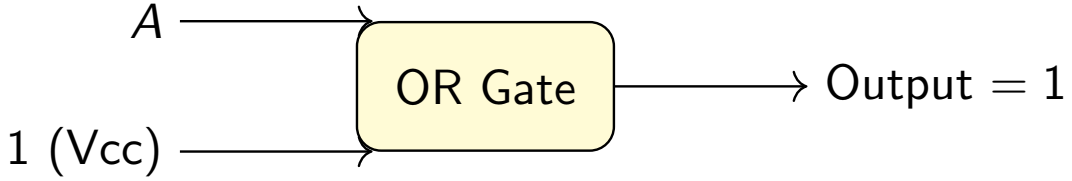
Properties of 0 and 1 (Identity Laws)

- ORing with 0 has no effect: $A + 0 = A$
- ANDing with 1 has no effect: $A \cdot 1 = A$
- Proof for $A + 0 = A$:
  - If $A = 0$: $0 + 0 = 0$ (Matches A)
  - If $A = 1$: $1 + 0 = 1$ (Matches A)

A	**0**	**A + 0**
0	0	0
1	0	1

Properties of 0 and 1 (Null / Dominance Laws)

- ORing with 1 forces the output to 1: $A + 1 = 1$
- ANDing with 0 forces the output to 0: $A \cdot 0 = 0$
- Proof for $A + 1 = 1$:
  - If $A = 0$: $0 + 1 = 1$
  - If $A = 1$: $1 + 1 = 1$



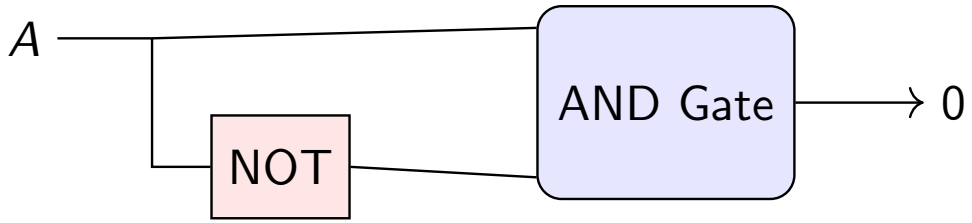
Idempotent Laws

- A variable operating on itself yields the variable.
- OR form: $A + A = A$
- AND form: $A \cdot A = A$
- This means redundancy can be eliminated in Boolean algebra.

A	**A + A**	**A · A**
0	0	0
1	1	1

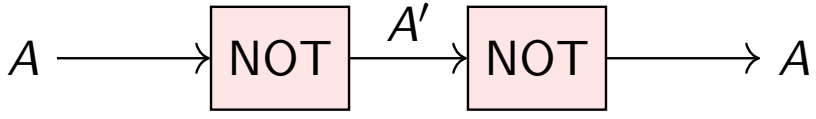
Complement / Inverse Laws

- A variable ORed with its complement is always 1: $A + A' = 1$
- A variable ANDed with its complement is always 0: $A \cdot A' = 0$
- Why? Because one of them is GUARANTEED to be 1, and the other is GUARANTEED to be 0.



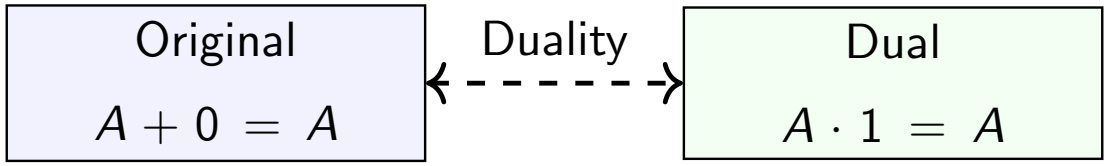
Involution Law (Double Inversion)

- Inverting a variable twice returns it to its original state.
- $(A')' = A$
- Or A double bar equals A.
- Logic equivalent: Two NOT gates in series cancel each other out.



Principle of Duality

- A fascinating property of Boolean algebra.
- You can derive a valid Boolean expression from another valid one by:
  1. Changing all OR ($+$) to AND ($\cdot$)
  2. Changing all AND ($\cdot$) to OR ($+$)
  3. Changing all 1s to 0s, and 0s to 1s.



Summary of Single-Variable Theorems

- Identity: $A + 0 = A$, $A \cdot 1 = A$
- Null: $A + 1 = 1$, $A \cdot 0 = 0$
- Idempotent: $A + A = A$, $A \cdot A = A$
- Complement: $A + A' = 1$, $A \cdot A' = 0$
- Involution: $A'' = A$

Law	**OR Form**	**AND Form (Dual)**
Identity	$A + 0 = A$	$A \cdot 1 = A$
Null	$A + 1 = 1$	$A \cdot 0 = 0$
Idempotent	$A + A = A$	$A \cdot A = A$
Complement	$A + A' = 1$	$A \cdot A' = 0$
Involution		$(A')' = A$

Next Lecture Preview

- In the next lecture, we will expand to Multi-Variable properties of Boolean algebra.
- We'll cover Commutative, Associative, and Distributive laws.
- We will also introduce the extremely important De Morgan's Theorems.

De Morgan's Theorem

$$(A + B)' = A' \cdot B'$$

Questions?

Lecture 2.4: Properties of Boolean Algebra and De Morgan's Theorems

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

Agenda

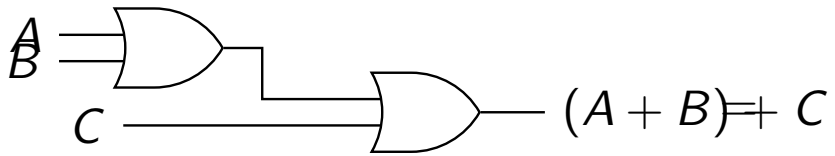
- Commutative and Associative Laws
- Distributive Laws (The standard and the special Boolean version)
- Absorption Law
- **De Morgan's First Theorem**
- **De Morgan's Second Theorem**
- Summary and Q&A

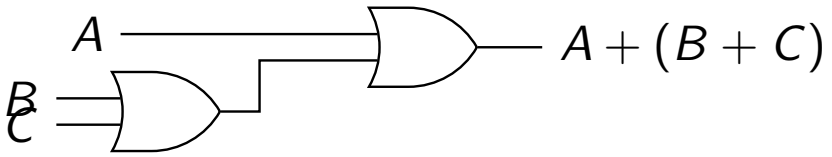
Commutative Law

- The order in which variables are ORed or ANDed does not matter.
- OR form: $A + B = B + A$
- AND form: $A \cdot B = B \cdot A$
- Hardware meaning: Swapping the inputs of a logic gate doesn't change the output.

Associative Law

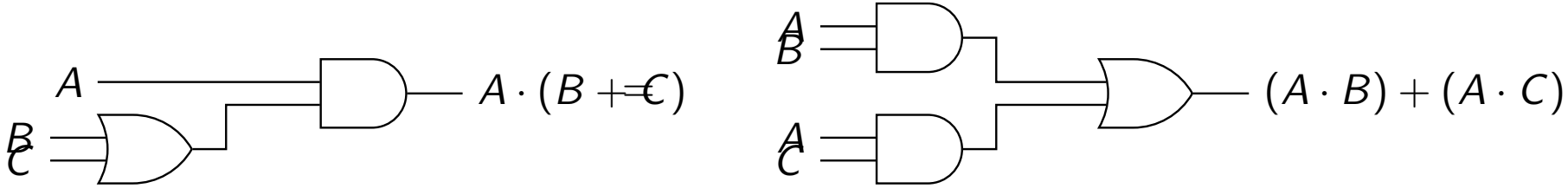
- When performing the same operation on multiple variables, the grouping doesn't matter.
- OR form: $A + (B + C) = (A + B) + C$
- AND form: $A \cdot (B \cdot C) = (A \cdot B) \cdot C$
- Hardware meaning: Cascading gates in different orders yields the same result.





Distributive Law (Standard)

- Multiplying across an addition.
- Equation: $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$
- This is identical to regular algebra.
- Hardware meaning: An AND gate feeding from an OR gate can be expanded to two AND gates feeding an OR gate.



Distributive Law (The Boolean Special)

- Adding across a multiplication.
- Equation: $A + (B \cdot C) = (A + B) \cdot (A + C)$
- This does NOT exist in regular algebra!

Proof by ExpansionProof

$$\begin{aligned}
(A+B)(A+C) &= AA + AC + AB + BC \\
&= A + A(C+B) + BC \quad (\text{since } AA = A) \\
&= A(1 + C + B) + BC \quad (\text{factoring out } A) \\
&= A(1) + BC \quad (\text{since } 1 + X = 1) \\
&= A + BC
\end{aligned}$$

Absorption Law

- Used to 'absorb' redundant variables.
- Law 1: $A + A \cdot B = A$
- Law 2: $A \cdot (A + B) = A$
- Proof of Law 1: $A + AB = A(1 + B)$. Since $1 + B = 1$, it becomes $A(1) = A$.

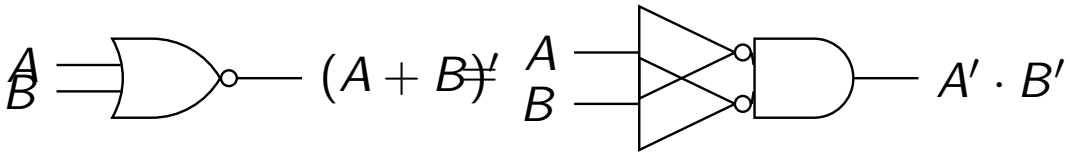
A	B	$A \cdot B$	$A + A \cdot B$
0	0	0	**0**
0	1	0	**0**
1	0	0	**1**
1	1	1	**1**

Introduction to De Morgan's Theorems

- Formulated by Augustus De Morgan.
- Crucial for simplifying expressions containing inverted groups.
- They provide a mathematical way to convert between AND and OR operations.
- Mnemonics: 'Break the line, change the sign'.

De Morgan's First Theorem

- The complement of a sum is equal to the product of the complements.
- Equation: $(A + B)' = A' \cdot B'$
- NOR gate is equivalent to a Bubbled-AND gate.
- 'Break the line over the PLUS, change the PLUS to a DOT'.



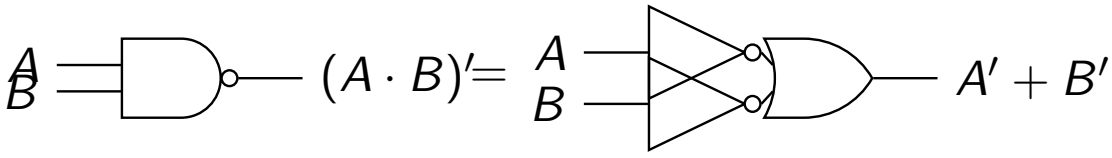
Proof of De Morgan's First Theorem

Let's build the truth table:

A	B	$A+B$	$(\mathbf{A+B})'$	A'	B'	$\mathbf{A'\cdot B'}$
0	0	0	**1**	1	1	**1**
0	1	1	**0**	1	0	**0**
1	0	1	**0**	0	1	**0**
1	1	1	**0**	0	0	**0**

De Morgan's Second Theorem

- The complement of a product is equal to the sum of the complements.
- Equation: $(A \cdot B)' = A' + B'$
- NAND gate is equivalent to a Bubbled-OR gate.
- 'Break the line over the DOT, change the DOT to a PLUS'.



Proof of De Morgan's Second Theorem

Let's build the truth table:

A	B	$A \cdot B$	$(\mathbf{A} \cdot \mathbf{B})'$	A'	B'	$\mathbf{A}' + \mathbf{B}'$
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**

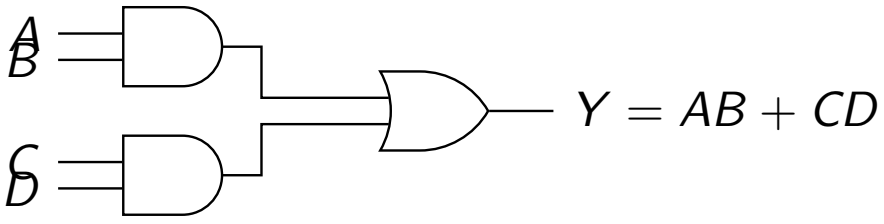
Summary

Cheat SheetLaws

- $A + B = B + A \quad ; \quad AB = BA$
- $A + (B + C) = (A + B) + C \quad ; \quad A(BC) = (AB)C$
- $A(B + C) = AB + AC \quad ; \quad A + BC = (A + B)(A + C)$
- $A + AB = A \quad ; \quad A(A + B) = A$
- $(A + B)' = A'B' \quad ; \quad (AB)' = A' + B'$

Next Lecture Preview

- In the next lecture, we will look at physical Implementations of Boolean functions.
- We'll introduce AND-OR-Invert (AOI) logic gates.
- We'll practice translating our Boolean equations directly into hardware circuit diagrams.



Questions?

Lecture 2.5: Implementations of Boolean function using AND-OR-Invert logic gates

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

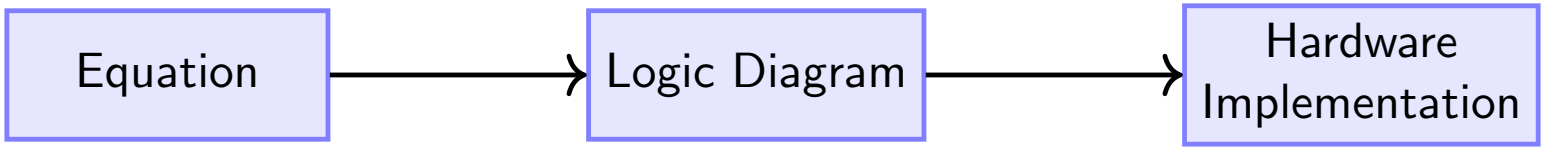
Lecturer, GP Palanpur

Table of Contents

Agenda

Agenda

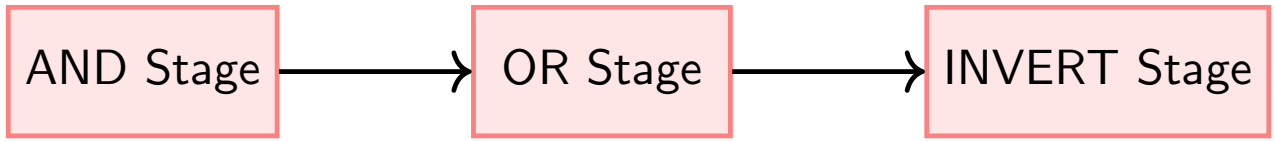
- What is AOI Logic?
- Building Blocks of AOI
- Translating Equations to Circuits
- Implementation Examples
- Extracting Equations from Circuits
- Summary and Q&A



What is AOI Logic?

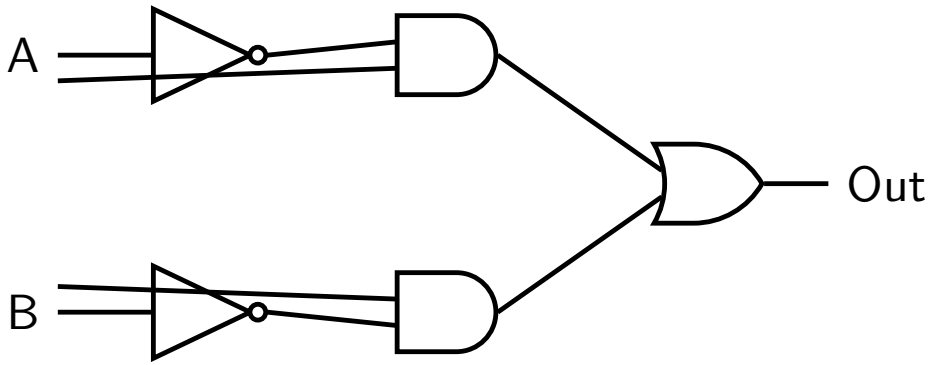
What is AND-OR-Invert (AOI) Logic?

- AOI logic uses a specific combination of gates to implement Boolean functions.
- It consists of one or more AND gates followed by an OR gate (or a NOR gate for the 'Invert' part).
- It maps perfectly to 'Sum of Products' (SOP) boolean expressions.
- Available as integrated circuit (IC) packages because of its efficiency.



The Three Stages of AOI

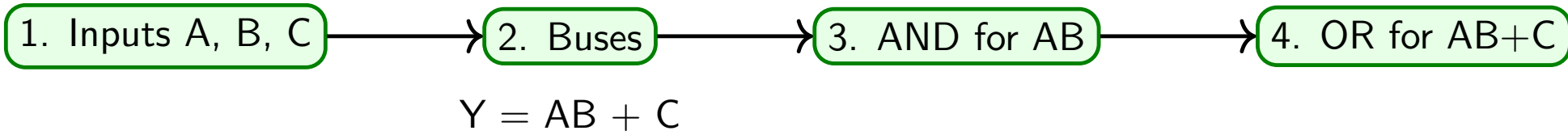
- 1. Inverter Stage (Optional but common): NOT gates to provide complemented inputs (e.g., A').
- 2. AND Stage: Multiple AND gates to generate product terms (e.g., AB, CD).
- 3. OR/NOR Stage: A single OR/NOR gate to sum the product terms.
- Structure: inputs $\rightarrow$ [NOTs] $\rightarrow$ [ANDs] $\rightarrow$ [OR/NOR] $\rightarrow$ output



Translating Equations to Circuits

Translating Equations to Circuits: Rule of Thumb

- Step 1: Identify all input variables and their complements.
- Step 2: Draw a vertical line for each variable and its complement (input buses).
- Step 3: Place an AND gate for every 'product' term in the equation.
- Step 4: Connect the AND gate outputs to a final OR gate (Sum stage).



Implementation Examples

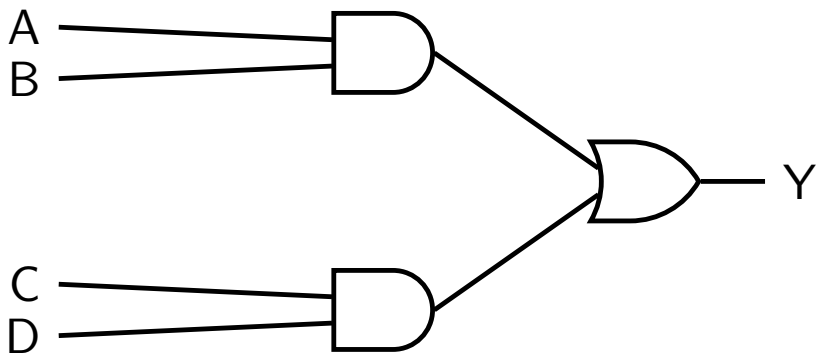
Example 1: Basic Implementation

- Function: $Y = A \cdot B + C \cdot D$
- Analysis:
- - 4 input variables: A, B, C, D.
- - No complements needed.
- - Two product terms (A·B) and (C·D), requiring two AND gates.
- - One sum operation, requiring one OR gate.

$$Y = A \cdot B + C \cdot D$$

Example 1: The Logic Diagram

- Drawing the circuit for $Y = A \cdot B + C \cdot D$:
- 1. Top AND gate receives inputs A and B.
- 2. Bottom AND gate receives inputs C and D.
- 3. Both AND gate outputs feed into a 2-input OR gate.
- 4. The output of the OR gate is Y.



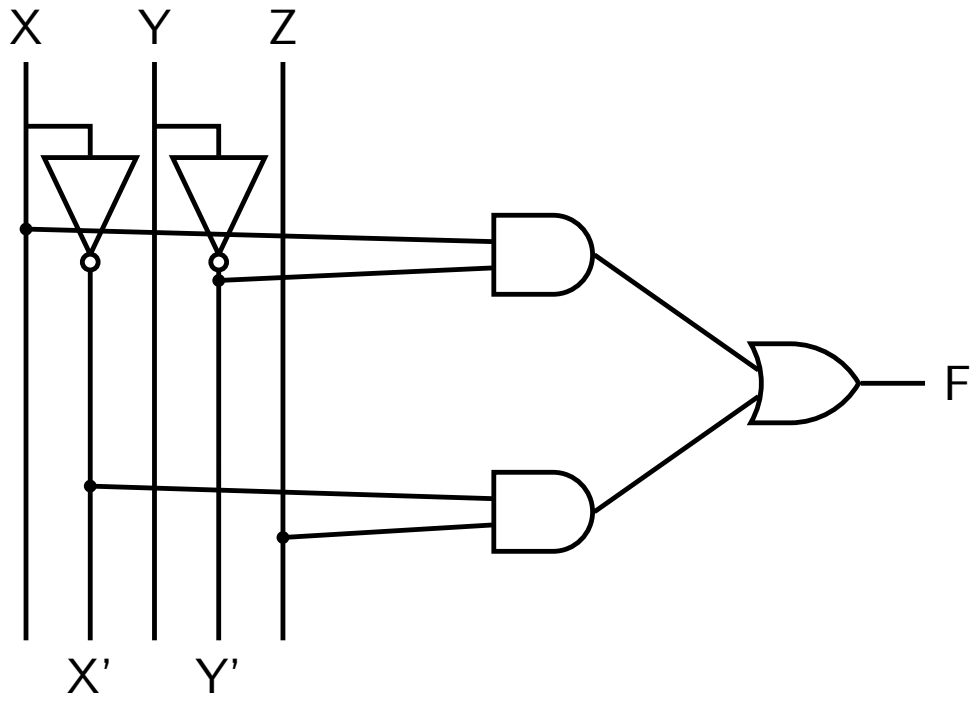
Example 2: Inverted Inputs

- Function: F = X·Y' + X'·Z
- Analysis:
- - Variables: X, Y, Z.
- - Need complements: Y' and X'. Add NOT gates to Y and X inputs.
- - Two AND gates for the product terms.
- - One OR gate for the sum.

$$F = X \cdot Y' + X' \cdot Z$$

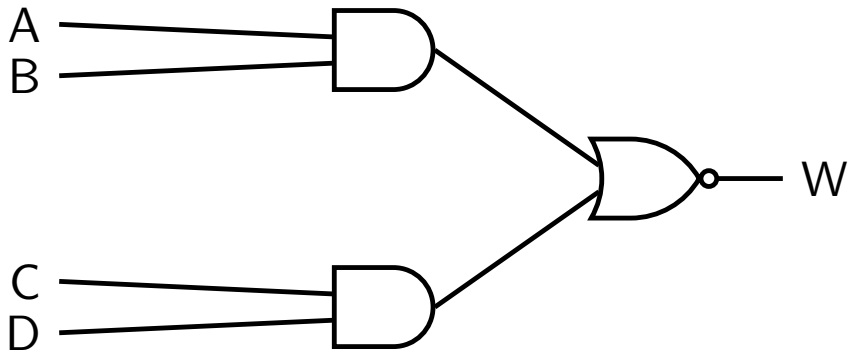
Example 2: The Logic Diagram

- Drawing the circuit for F = X·Y' + X'·Z:
- 1. Pass Y through a NOT gate. Pass X through a NOT gate.
- 2. First AND gate takes X and Y'.
- 3. Second AND gate takes X' and Z.
- 4. Both outputs feed into the OR gate to produce F.



Adding the 'Invert' (NOR stage)

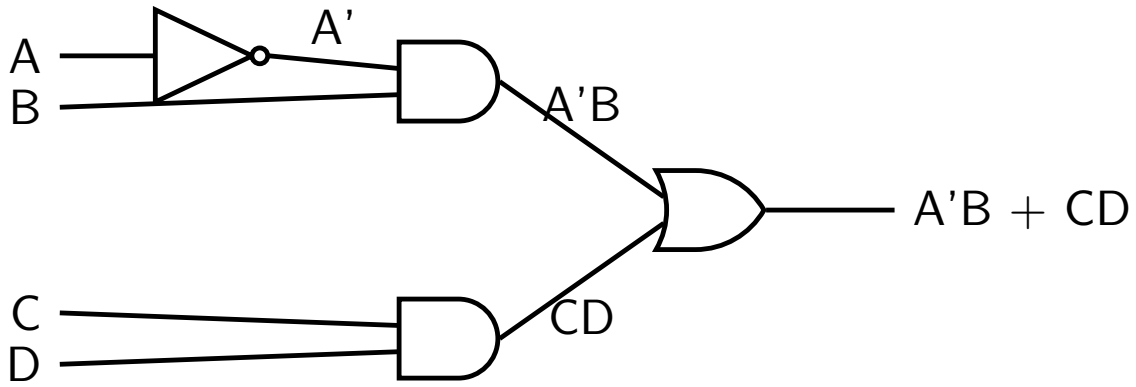
- If the entire expression is complemented, e.g., W = (A·B + C·D)'
- We use an AND-OR-INVERT circuit.
- We implement the core sum of products as usual.
- We replace the final OR gate with a NOR gate.



Extracting Equations from Circuits

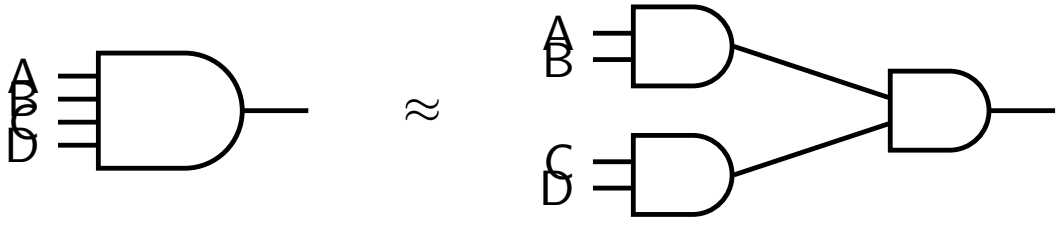
Extracting Equations from Circuits

- Reverse process: Given a circuit, find the expression.
- Step 1: Label the output of every NOT gate.
- Step 2: Write the boolean product at the output of every AND gate.
- Step 3: Combine these products at the final OR/NOR gate.
- Practice: Trace the signals left-to-right.



Hardware Considerations (Fan-in)

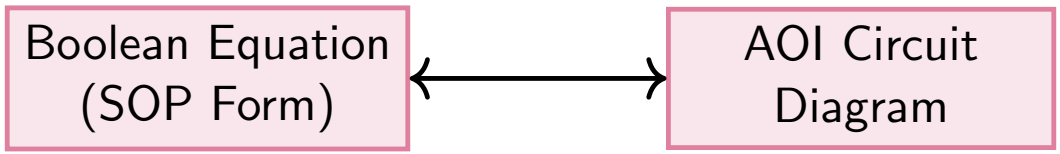
- In theory, an AND gate can have infinite inputs.
- In reality, physical gates have a limit, called 'Fan-in'.
- If an equation requires a 5-input AND gate but you only have 2-input gates, you must cascade them.
- Cascading increases delay. AOI ICs are designed to minimize this.



Summary

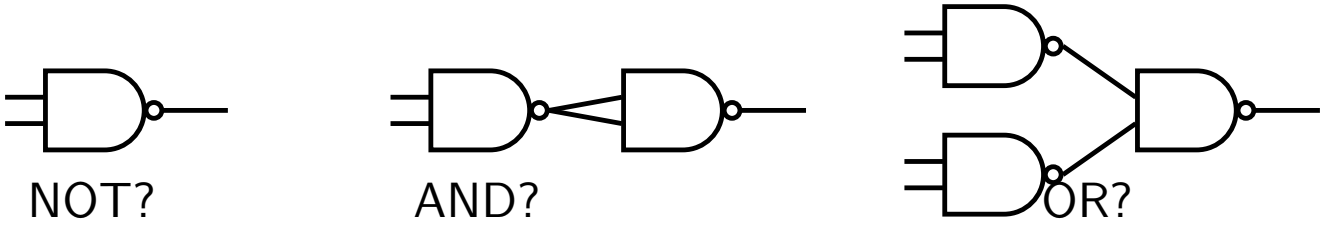
Summary

- AOI logic follows a strict NOT $\rightarrow$ AND $\rightarrow$ OR/NOR structure.
- It directly implements Sum of Products (SOP) expressions.
- Translating involves counting product terms (for ANDs) and summing them (with an OR).
- Input buses keep complex circuit diagrams organized.



Next Lecture Preview

- Next, we will look at Universal Gates.
- We'll learn why NAND and NOR are considered 'Universal'.
- We will see how to build ANY digital circuit using ONLY NAND gates or ONLY NOR gates.



Questions?

Lecture 2.6: Universal Gates: NAND & NOR

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

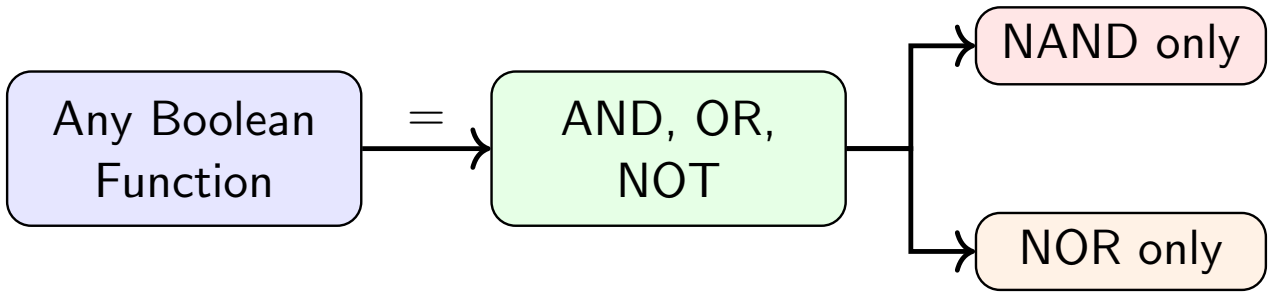
Table of Contents

Agenda

- Concept of a Universal Gate
- Why use Universal Gates?
- NAND as a Universal Gate (Building NOT, AND, OR)
- NOR as a Universal Gate (Building NOT, OR, AND)
- Practical Circuit Conversion
- Summary and Q&A

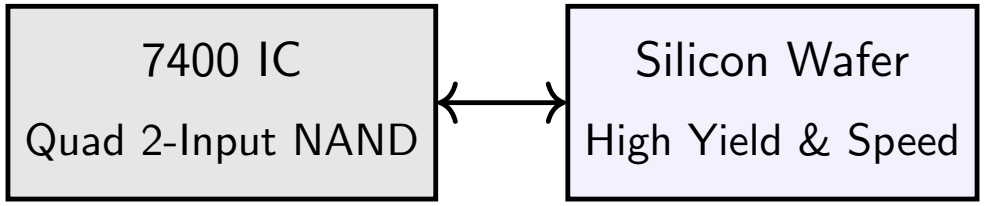
What is a Universal Gate?

- A universal gate is a logic gate that can implement any Boolean function without the need to use any other type of gate.
- There are exactly two universal gates: NAND and NOR.
- Since any Boolean function can be written using AND, OR, and NOT, if a gate can mimic these three, it is universal.



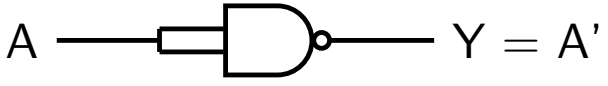
Why Use Universal Gates?

- **Manufacturing Economy:** Cheaper to fabricate an IC with thousands of identical gates than one with a mix of gates.
- **Inventory Management:** Only stock one type of IC (e.g., 7400 Quad NAND) to build any prototype.
- **Technological Advantage:** NAND and NOR gates are inherently faster and require fewer transistors to build at the silicon level than AND/OR gates.



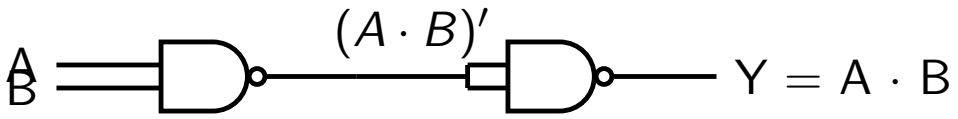
NAND as Universal: Making a NOT Gate

- **Goal:** Create Y = A' using a NAND gate.
- **Implementation:** Tie all inputs of the NAND gate together.
- **Boolean Algebra Proof:**
  - NAND equation: $Y = (A \cdot B)'$
  - If $A = B$, then $Y = (A \cdot A)'$
  - By Idempotent Law, $A \cdot A = A$, so $Y = A'$



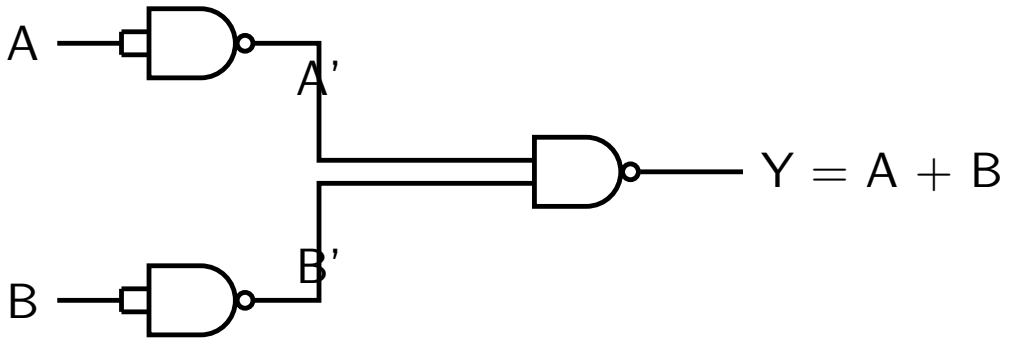
NAND as Universal: Making an AND Gate

- **Goal:** Create $Y = A \cdot B$ using NAND gates.
- A NAND gate inherently does AND, but inverted.
- **Implementation:** Use one NAND gate, and invert its output using a second NAND gate (wired as NOT).
- **Proof:** $Y = ((A \cdot B)')' = A \cdot B$ (Involution law).



NAND as Universal: Making an OR Gate

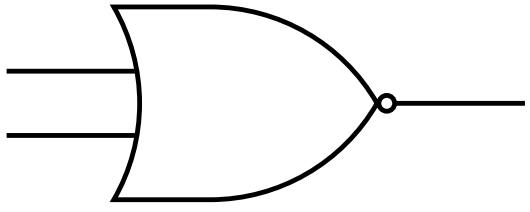
- **Goal:** Create Y = A + B using NAND gates.
- **Hint:** Use De Morgan's Theorem! $(A + B) = (A' \cdot B')'$
- **Implementation:** Invert both inputs A and B using NAND gates, then feed them into a third NAND gate.
- Requires 3 NAND gates total.



NOR as a Universal Gate

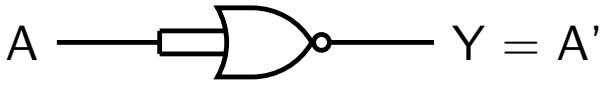
- The logic for NOR is the exact dual of the logic for NAND.
- NOR stands for NOT-OR.
- We must prove we can build NOT, OR, and AND using only NOR gates.

The Versatile NOR Gate



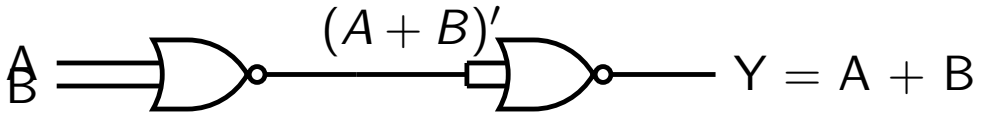
NOR as Universal: Making a NOT Gate

- **Goal:** Create Y = A' using a NOR gate.
- **Implementation:** Tie all inputs of the NOR gate together.
- **Boolean Algebra Proof:**
  - NOR equation: $Y = (A + B)'$
  - If $A = B$, then $Y = (A + A)'$
  - By Idempotent Law, $A + A = A$, so $Y = A'$



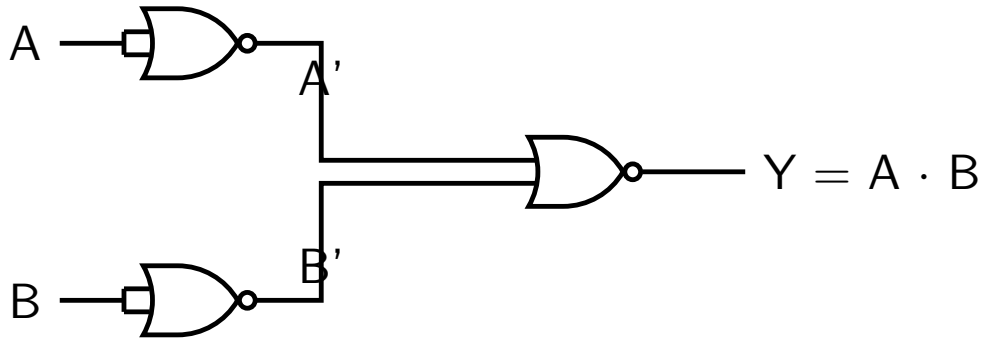
NOR as Universal: Making an OR Gate

- **Goal:** Create Y = A + B using NOR gates.
- A NOR gate inherently does OR, but inverted.
- **Implementation:** Use one NOR gate, and invert its output using a second NOR gate (wired as NOT).
- **Proof:** $Y = ((A + B)')' = A + B$.



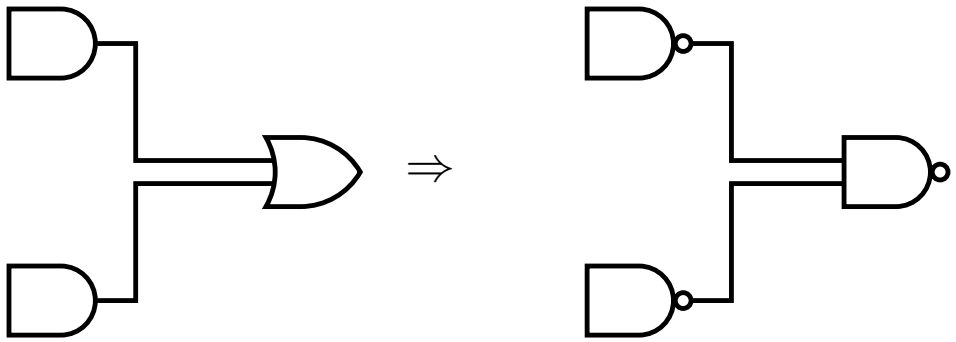
NOR as Universal: Making an AND Gate

- **Goal:** Create $Y = A \cdot B$ using NOR gates.
- Use De Morgan's: $(A \cdot B) = (A' + B')'$
- **Implementation:** Invert both inputs A and B using NOR gates, then feed them into a third NOR gate.
- Requires 3 NOR gates total.



Practical Circuit Conversion

- How to convert any AOI circuit to all-NAND logic:
- 1. Draw the standard AND-OR circuit.
- 2. Add bubbles (inverters) at the outputs of all AND gates.
- 3. Add bubbles at the inputs of all OR gates.
- 4. Notice that an OR gate with bubbled inputs IS a NAND gate.
- 5. The circuit is now entirely NAND gates!



Summary

- Universal gates can implement any logic function alone.
- NAND and NOR are the only universal gates.
- NOT is formed by tying inputs together.
- AND/OR is formed by double inversion or De Morgan's equivalencies.
- They are cheaper, faster, and easier to manufacture.

Logic Gate	**NANDs Required**	**NORs Required**
NOT	1	1
AND	2	3
OR	3	2

Next Lecture Preview

- In the next lecture, we tackle Algebraic Simplification of Boolean expressions.
- We will use the theorems we learned to reduce complex equations into their simplest forms.
- Simplification means fewer gates, lower cost, and faster circuits.

$$Y = A \cdot B + A \cdot B' + A' \cdot B \xrightarrow{\text{Simplify}} Y = A + B$$

Questions?

Lecture 2.7: Algebraic Simplification of Boolean Expressions

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

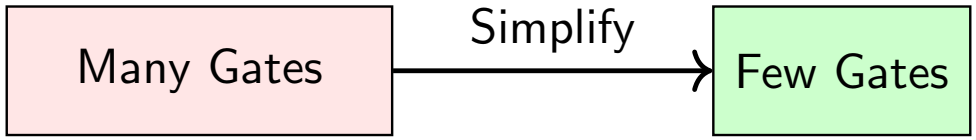
Agenda

- Why Simplify?
- Review of Key Boolean Rules
- Step-by-Step Simplification Method
- **Simplification Examples** (Core of the lecture)
- Using De Morgan's in Simplification
- Limitations of the Algebraic Method
- Summary and Q&A

Lecture 2.7: Algebraic Simplification of Boolean Expressions

Why Simplify Boolean Expressions?

- Direct Translation: Complex equations = Many gates. Simple equations = Few gates.
- Cost: Fewer gates mean cheaper hardware (fewer ICs, smaller PCB footprint).
- Power: Fewer gates consume less electrical power.
- Speed: Fewer gates in a sequence means less propagation delay, making the circuit faster.
- Reliability: Fewer components mean fewer things that can break.



Lecture 2.7: Algebraic Simplification of Boolean Expressions

The Core Toolkit (Review)

To simplify, you must recognize these patterns:

Idempotent: $A + A = A$
Complement: $A + A' = 1, A \cdot A' = 0$
Absorption: $A + AB = A, A(A + B) = A$
Factoring: $AB + AC = A(B + C)$

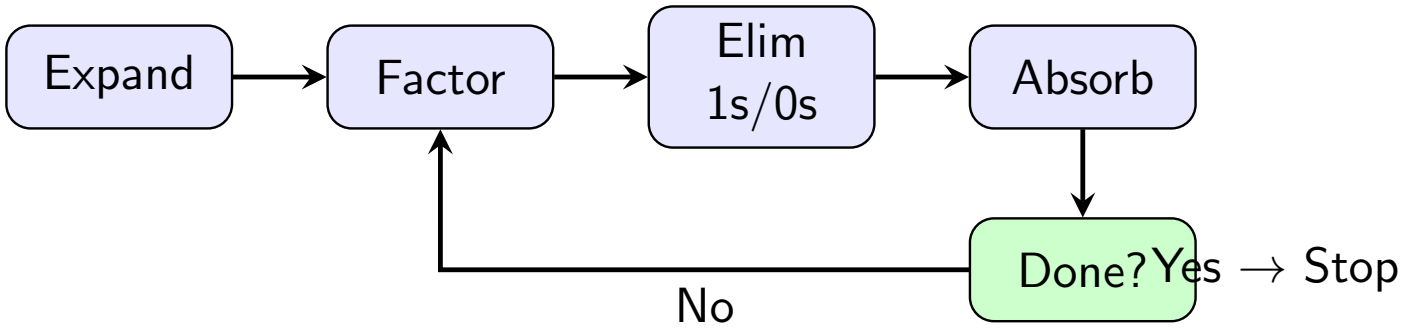
Special Distributive: $A + A'B = A + B$

Lecture 2.7: Algebraic Simplification of Boolean Expressions

Simplification Method

There is no fixed algorithm for algebraic simplification, but follow these tips:

- 1. Remove parentheses by multiplying out (distributing).
- 2. Look for common terms to factor out.
- 3. Look for terms like $(A + A')$ which evaluate to 1.
- 4. Look for absorption opportunities $(A + AB)$.
- 5. Repeat until no more terms can be reduced.



Lecture 2.7: Algebraic Simplification of Boolean Expressions

Simplification Example 1

Simplify: $Y = AB + AB' + A'C$

$Y = AB + AB' + A'C$	Given
$Y = A(B + B') + A'C$	Step 1: Factor out A
$Y = A(1) + A'C$	Step 2: Complement Law $(B + B' = 1)$
$Y = A + A'C$	Step 3: Identity Law
$Y = A + C$	Step 4: Special Distributive $(X + X'Y = X + Y)$

Final Answer: $Y = A + C$

Simplification Example 2

Simplify: $Y = A'B'C + A'BC + AB'$

$Y = A'B'C + A'BC + AB'$	Given
$Y = A'C(B' + B) + AB'$	Step 1: Factor out $A'C$
$Y = A'C(1) + AB'$	Step 2: Complement Law $(B' + B = 1)$
$Y = A'C + AB'$	Identity Law

Can we simplify further? No common terms. We stop here.

Simplification Example 3 (Using De Morgan's)

Simplify: $Y = ((AB)' + C)'$

$Y = \overline{\overline{(AB)} + C}$	Given
$Y = \overline{\overline{(AB)}} \cdot \overline{C}$	Step 1: Break bar, change $+$ to $\cdot$
$Y = (AB) \cdot \overline{C}$	Step 2: Involution Law
$Y = ABC'$	Final Answer

Simplification Example 4

Simplify: $Y = ABC + AB'C + A'BC + A'B'C$

$Y = ABC + AB'C + A'BC + A'B'C$	Given
$Y = AC(B + B') + A'C(B + B')$	Step 1: Group and factor
$Y = AC + A'C$	Step 2: $B + B' = 1$
$Y = C(A + A')$	Step 3: Factor C
$Y = C(1)$	Step 4: $A + A' = 1$
$Y = C$	Final Answer

Simplification Example 5 (Consensus Theorem)

The Consensus Theorem: $AB + A'C + BC = AB + A'C$
Notice the term 'BC' is redundant.

Consensus Theorem Color Code:
$AB + A'C + BC = AB + A'C$

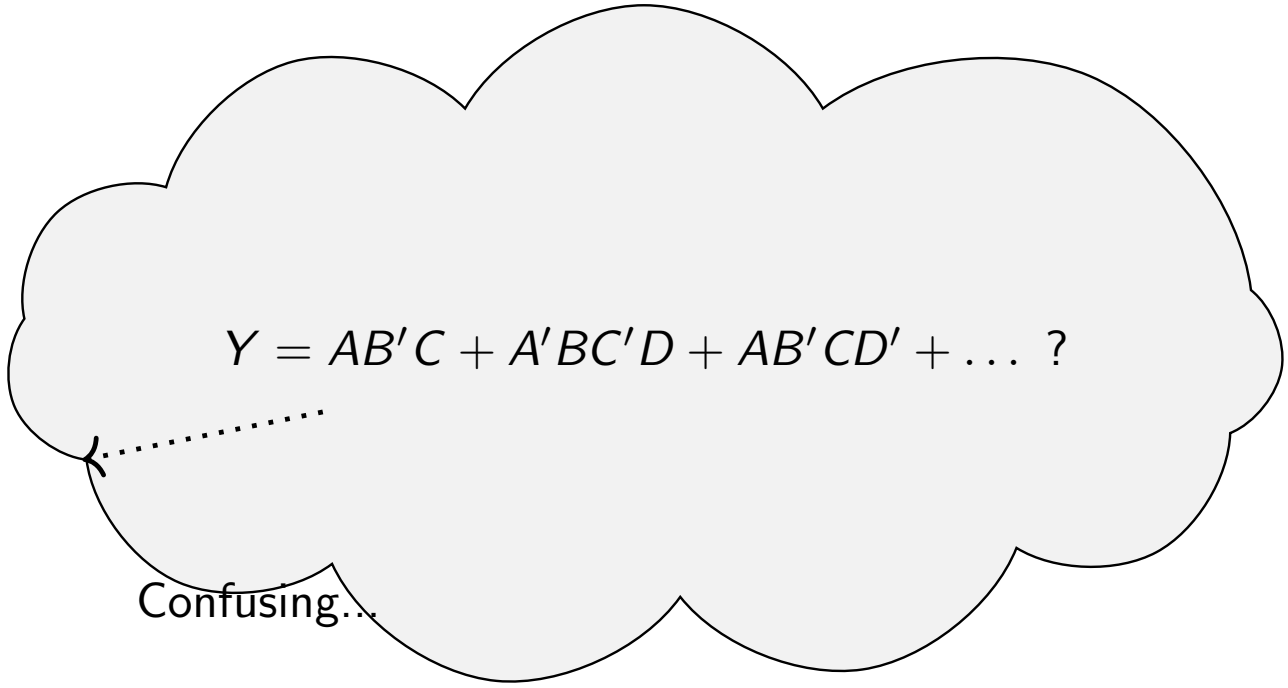
Simplify: $Y = XY + X'Z + YZ + XYZ$

- Step 1: Absorb XYZ into XY
  $(XY + XYZ = XY(1 + Z) = XY)$.
- $Y = XY + X'Z + YZ$
- Step 2: Apply Consensus Theorem.
- Final Answer: $Y = XY + X'Z$

Lecture 2.7: Algebraic Simplification of Boolean Expressions

Limitations of Algebraic Simplification

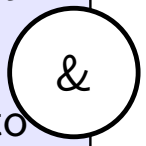
- **Lack of Specific Rules:** There is no strict step-by-step algorithm. It requires intuition and trial-and-error.
- **Uncertainty:** It is difficult to know if you have reached the ABSOLUTE simplest form.
- **Complexity:** Becomes extremely tedious and error-prone for equations with 4, 5, or more variables.



Lecture 2.7: Algebraic Simplification of Boolean Expressions

Summary

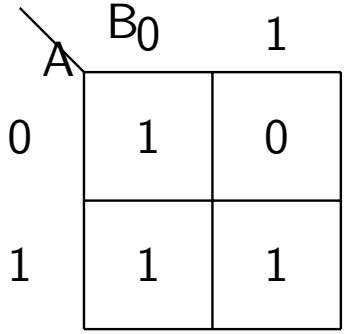
- Simplification reduces cost, size, and power while increasing speed.
- It relies heavily on Factoring, Complement Law, and Absorption.
- Always look for terms that can combine to equal 1 (e.g., $A + A'$).
- Algebraic methods require intuition and lack certainty of achieving the minimal form.



Lecture 2.7: Algebraic Simplification of Boolean Expressions

Next Lecture Preview

- Because of the limitations of Algebra, we need a better tool.
- Next time, we will define standard forms of expressions: Sum of Product (SOP).
- This is the foundation for the visual simplification method called the Karnaugh Map (K-map).



Questions?

Lecture 2.8: Sum of Product (SOP) Definition and Minterms

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

Agenda

- Standard Forms of Boolean Expressions
- What is a Product Term?
- Sum of Products (SOP) defined
- Concept of Minterms
- Standard (Canonical) SOP form
- Converting Truth Tables to SOP
- Summary and Q&A

Lecture 2.8: Sum of Product (SOP) Definition and Minterms

Standard Forms of Expressions

- To simplify logic systematically, expressions must be in a standard format.
- There are two main standard formats:
  1. Sum of Products (SOP)
  2. Product of Sums (POS)
- We will focus primarily on SOP, as it maps directly to AND-OR-Invert hardware and is the most common.

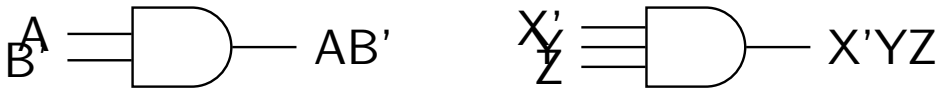
SOP
$AB + CD$

POS
$(A + B)(C + D)$

Lecture 2.8: Sum of Product (SOP) Definition and Minterms

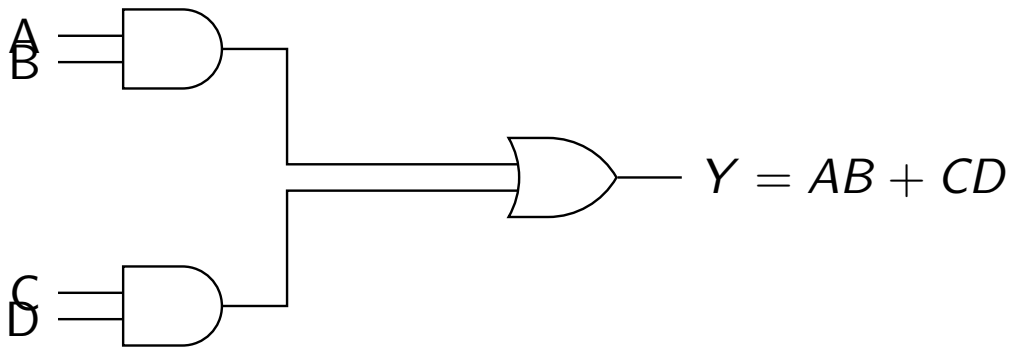
What is a Product Term?

- A product term is a single variable, or the logical AND (multiplication) of several variables.
- The variables can be in true form (A) or complemented form (A').
- Examples of product terms: A, AB', X'YZ, A'B'C'D.
- In hardware, a product term is generated by an AND gate.



Sum of Products (SOP) Defined

- An SOP expression is formed when two or more product terms are logically added (ORed) together.
- Example 1: $Y = AB + CD$
- Example 2: $F = X'Y + XYZ' + X'Z$
- It consists of an AND stage feeding into a single final OR stage.
- Notice there are no parenthesis in a pure SOP expression.



Lecture 2.8: Sum of Product (SOP) Definition and Minterms

Concept of Minterms

- A Minterm is a special type of product term.
- It is a product term that contains EVERY variable in the function, either complemented or uncomplemented.
- For a 2-variable system (A, B), minterms are: A'B', A'B, AB', AB.
- For an n-variable system, there are exactly 2^n possible minterms.

A	B	Minterm
0	0	A'B'
0	1	A'B
1	0	AB'
1	1	AB

Minterm Notation

- We use shorthand notation: a lowercase 'm' with a subscript.
- The subscript is the decimal equivalent of the binary pattern.
- Rule for minterms: Uncomplemented variable = 1, Complemented = 0.
- Example (3 variables: A,B,C): $A'B'C' \rightarrow 000 \rightarrow m_0$
- Example: $AB'C \rightarrow 101 \rightarrow m_5$
- Example: $ABC \rightarrow 111 \rightarrow m_7$

A	B	C	Term	Notation
0	0	0	A'B'C'	m_0
...	...	...	...	...
1	1	1	ABC	m_7

Standard (Canonical) SOP Form

- A standard SOP expression is an SOP expression where EVERY term is a minterm.
- Non-standard: $Y = A + BC$ (A is missing B,C. BC is missing A).
- Standard: $Y = A'BC + AB'C + ABC$
- Also written using summation shorthand: $Y = \Sigma m(3, 5, 7)$

$$Y = \underbrace{A'BC}_{m_3} + \underbrace{AB'C}_{m_5} + \underbrace{ABC}_{m_7} = \Sigma m(3, 5, 7)$$

Lecture 2.8: Sum of Product (SOP) Definition and Minterms

Converting Truth Tables to SOP

- The most powerful feature of standard SOP is how easy it is to derive from a truth table.
- Step 1: Look ONLY at the rows where the Output is 1 (HIGH).
- Step 2: For each of those rows, write the corresponding minterm.
- Step 3: OR (add) all those minterms together.
- Ignore the rows where the output is 0.

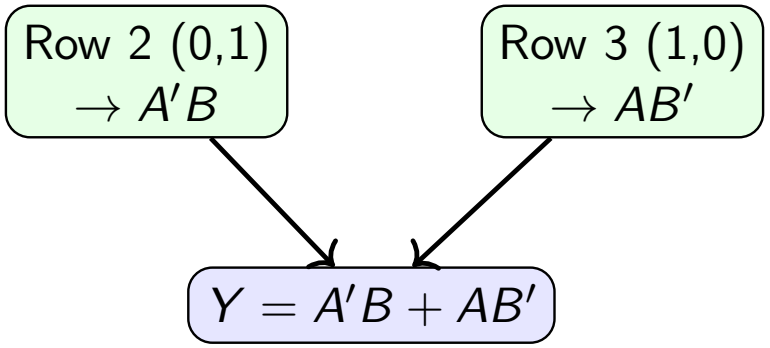
Example: Truth Table to SOP (Part 1)

- Given Truth Table (A,B — Y):
- 0 0 | 0
- 0 1 | 1 $\rightarrow (A = 0, B = 1)$
- 1 0 | 1 $\rightarrow (A = 1, B = 0)$
- 1 1 | 0

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

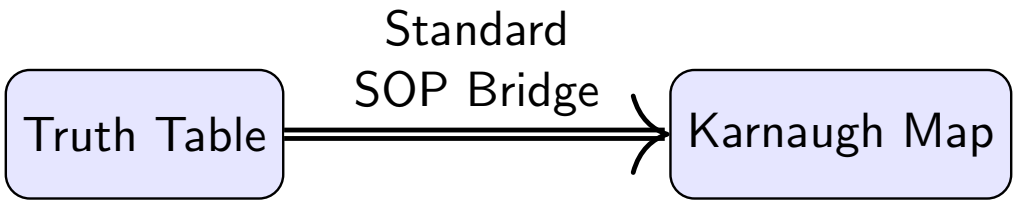
Example: Truth Table to SOP (Part 2)

- Row 2 (0, 1) output is 1. Minterm is A'B (m_1).
- Row 3 (1, 0) output is 1. Minterm is AB' (m_2).
- Combine them with an OR ($+$):
- $Y = A'B + AB'$
- Or in shorthand: $Y = \Sigma m(1, 2)$.



Why Standard SOP is Important

- It acts as a bridge between the behavioral specification (Truth Table) and the simplification tool (K-map).
- Every unique truth table has exactly one unique Standard SOP expression.
- Non-standard expressions can be converted to standard by multiplying missing variables with (X+X').



Lecture 2.8: Sum of Product (SOP) Definition and Minterms

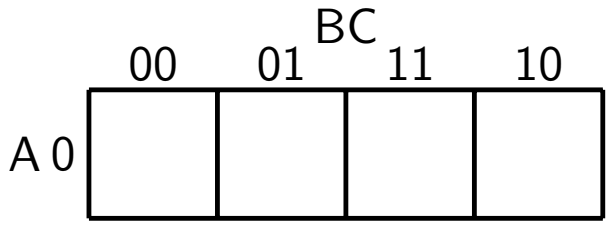
Summary

- SOP implies a 2-level AND-OR structure.
- Minterms contain all variables for a specific truth table row.
- Minterms where the variable is 0 get a prime ('), 1s do not.
- To get SOP from a truth table, sum the minterms where output is 1.
- Shorthand uses the $\Sigma m()$ notation.

Symbol	**Binary (A,B)**	**Boolean**
m_0	00	A'B'
m_1	01	A'B
m_2	10	AB'
m_3	11	AB

Next Lecture Preview

- Next, we introduce the Karnaugh Map (K-map).
- The K-map is a graphical tool used to simplify standard SOP expressions without using algebra.
- We will learn how to map up to 4 variables to find the most minimal circuit design effortlessly.



Questions?

Lecture 2.9: Simplification using K-map up to 4-variables

Complete Lecture Deck – All 45 Lectures

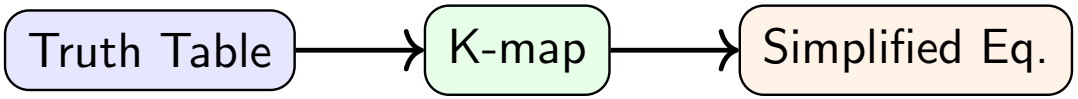
Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

Agenda

- Introduction to K-maps
- Layout and Gray Code
- 2, 3, and 4-Variable K-map Structures
- Plotting Minterms
- Grouping Rules (1, 2, 4, 8, 16)
- Extracting the Minimized Expression
- Examples
- Summary and Q&A



What is a Karnaugh Map?

- Invented by Maurice Karnaugh in 1953.
- It is a visual representation of a truth table.
- Instead of a linear list, the truth table is folded into a 2D grid.
- Cells are arranged so that adjacent cells differ by exactly ONE variable (Gray code).
- This arrangement automatically identifies terms for the Consensus and Adjacency theorems (A+A'=1).

Truth Table

0	0	0
0	1	1
1	0	1
1	1	0

$\Rightarrow$

2D K-map

0	1
1	0

The Importance of Gray Code

- Standard binary sequence: 00, 01, 10, 11 (2 bits change between 01 and 10).
- Gray code sequence: 00, 01, 11, 10.
- In Gray code, ONLY ONE bit changes from any step to the next.
- K-map rows and columns MUST be labeled in Gray code order.
- This ensures spatial adjacency equals logical adjacency.

Standard Binary		**Gray Code**	
Sequence	Bits Changed	Sequence	Bits Changed
00	-	00	-
01	1	01	1
10	2 (0→1, 1→0)	11	1 (0→1)
11	1	10	1 (1→0)

2-Variable K-map

- Variables: A, B. Total cells: $2^2 = 4$.
- Grid is 2x2. Rows labeled A (0, 1), Columns labeled B (0, 1).
- Cells represent minterms m_0, m_1, m_2, m_3.

A \ B	0	1
0	m_0	m_1
1	m_2	m_3

3-Variable K-map

- Variables: A, B, C. Total cells: $2^3 = 8$.
- Grid is 2x4. Rows labeled A (0, 1), Columns labeled BC (00, 01, 11, 10) $\rightarrow$ Notice the Gray code!
- Minterm order across a row: m_0, m_1, m_3, m_2.

A \ BC	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

4-Variable K-map

- Variables: A, B, C, D. Total cells: $2^4 = 16$.
- Grid is 4x4. Rows: AB (00, 01, 11, 10). Cols: CD (00, 01, 11, 10).
- Minterm placement involves skips on both rows and columns.

AB \ CD	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

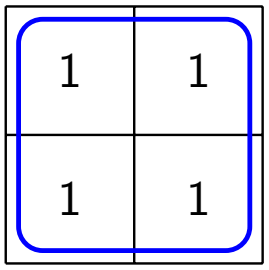
Plotting SOP on a K-map

- Look at the given equation (e.g., $Y = \Sigma m(1, 3, 5, 7)$).
- Place a '1' in the cells corresponding to those minterm numbers.
- Place a '0' (or leave blank) in all other cells.

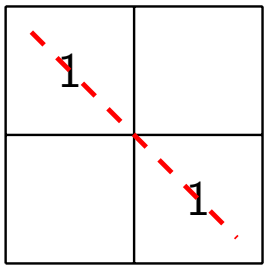
A \ BC	00	01	11	10
0	0 (0)	**1** (1)	**1** (3)	0 (2)
1	0 (4)	**1** (5)	**1** (7)	0 (6)

Rules for Grouping 1s

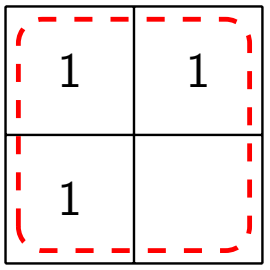
- Goal: Draw ovals around groups of 1s to simplify the expression.
- Groups must contain 1, 2, 4, 8, or 16 cells (Powers of 2).
- Groups can only be horizontal or vertical (No diagonals!).
- Groups should be as LARGE as possible.
- Every 1 must be in at least one group. Groups can overlap.



Correct (2x2)



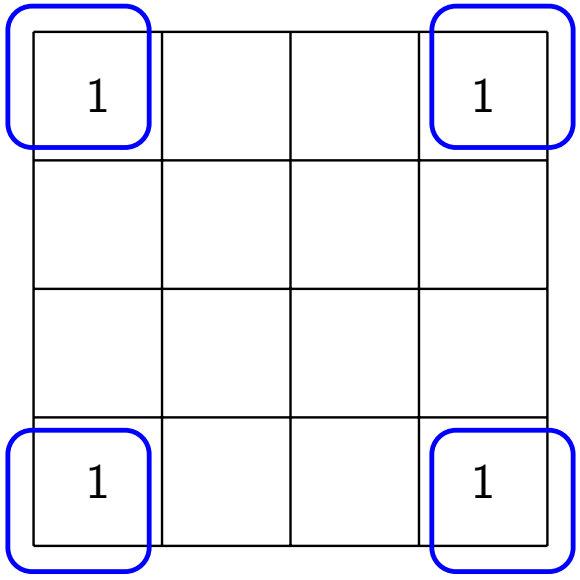
Wrong (Diagonal)



Wrong (Group of 3)

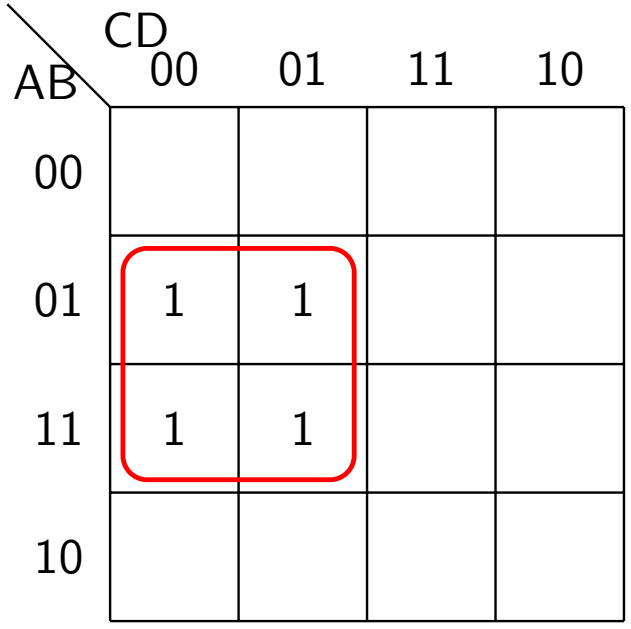
Map Rolling (Wrap-around)

- The K-map wraps around like a cylinder or torus.
- The left edge is adjacent to the right edge.
- The top edge is adjacent to the bottom edge.
- Four corners of a 4-variable map can form a single group of 4!



Extracting the Expression

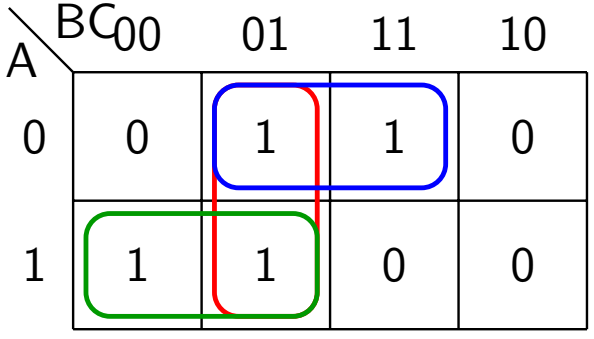
- For each group you created:
- Look at the variables spanning that group.
- If a variable changes value (0 to 1, or 1 to 0) within the group, it is ELIMINATED.
- If a variable stays constant (all 0s or all 1s), it remains in the term.
- 0 = complemented (A'), 1 = true (A).
- OR (+) the terms for each group together.



AB: 01 and 11 → A changes, B=1
CD: 00 and 01 → D changes, C=0
Term = B C'

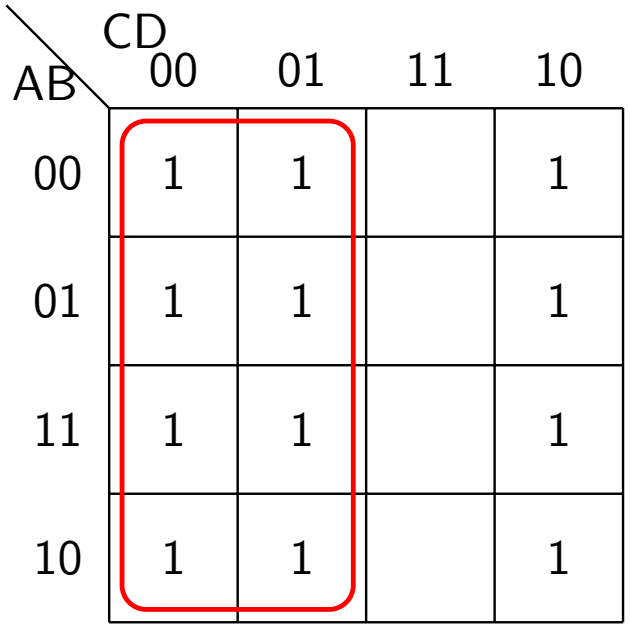
Example: 3-Variable K-map

- Simplify: $Y = \Sigma m(1, 3, 4, 5)$
- Grp 1: Cells 1, 5 (Vert col). A changes. B=0, C=1. $\rightarrow$ B'C.
- Grp 2: Cells 1, 3 (Horiz row). A=0. B changes. C=1. $\rightarrow$ A'C.
- Grp 3: Cells 4, 5 (Horiz row). A=1. B=0. C changes. $\rightarrow$ AB'.
- Final: $Y = B'C + A'C + AB'$



Example: 4-Variable K-map

- Simplify: $Y = \Sigma m(0, 1, 2, 4, 5, 6, 8, 9, 10, 12, 13, 14)$
- Plot the 1s. Notice a massive block.
- Grouping the left 2 columns (8 cells) gives C'.
- A group of 8 eliminates 3 variables!



8 cells $\rightarrow C'$

Summary

- K-maps use Gray code to arrange cells so physical adjacency = logical adjacency.
- Plot 1s based on minterms.
- Group in blocks of 1, 2, 4, 8, or 16.
- Maps wrap around at the edges.
- Variables that change within a group are eliminated.

Gray Code	Valid Groups	Wrap-around
00, 01, 11, 10	2^n	Edges touch

Next Lecture Preview

- What happens if there are certain input combinations that will NEVER occur in reality?
- Next lecture, we introduce the 'Don't Care' condition in K-maps.
- We will learn how to use these wildcards to achieve even greater circuit simplification.

1	X
0	X

Questions?

Lecture 2.10: Don't Care Conditions in Karnaugh Maps

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

Agenda

Agenda

- What is a 'Don't Care' Condition?
- Why do they exist?
- Representing Don't Cares
- Rules for Grouping with Don't Cares
- Example 1: Simplification
- Example 2: BCD to 7-Segment concept
- Summary and Q&A

What is a 'Don't Care' Condition?

What is a 'Don't Care' Condition?

- In some logic circuits, certain combinations of inputs will NEVER occur.
- Because they never occur, we literally 'don't care' what the output is for those combinations.
- The output could be 0, or it could be 1. It won't affect the system because the state is impossible.
- We treat these undefined outputs as wildcards.

A	B	C	Output
0	0	0	0
0	0	1	1
0	1	0	X (Impossible)

Origin of Don't Cares (Example)

Origin of Don't Cares (Example)

- Binary Coded Decimal (BCD).
- BCD uses 4 bits to represent decimal digits 0 through 9.
- Valid inputs: 0000 (0) up to 1001 (9).
- Invalid inputs: 1010 (10) up to 1111 (15).
- If a circuit is designed ONLY to accept BCD inputs, combinations 10 through 15 will never happen.
- Outputs for rows 10-15 are 'Don't Cares'.

Representing Don't Cares

Representing Don't Cares

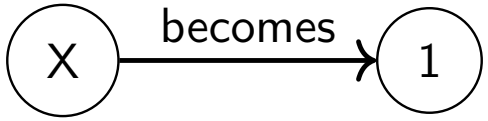
- In a Truth Table or K-map, Don't Cares are usually represented by:
  - An 'X'
  - A 'd'
  - A 'Φ' (phi)
- In equations, they are grouped separately:
- $Y = \Sigma m(1, 3, 5) + \Sigma d(2, 4, 6)$

A \ BC	00	01	11	10
0	0	1	X	0
1	X	1	0	X

Using Don't Cares in K-maps

Using Don't Cares in K-maps

- Since we 'don't care' if the output is 0 or 1, we can choose which one it is!
- Rule: You can treat an 'X' as a '1' IF AND ONLY IF it helps you make a LARGER group of 1s.
- Rule: You treat an 'X' as a '0' if it does not help you.
- Result: Larger groups = Simpler equations = Less hardware.



Rule for Grouping with Don't Cares

Rule for Grouping with Don't Cares

- 1. You MUST group all actual '1's.
- 2. You MAY group 'X's if they expand a group (e.g., from 2 to 4, or 4 to 8).
- 3. You MUST NOT group 'X's by themselves. If an X doesn't help group a 1, leave it out (it becomes a 0).
- There is no requirement to cover all X's.

Example: Solving with Don't Cares

Example: Solving with Don't Cares (Step 1)

- Given: $Y = \Sigma m(1, 3, 7) + d(0, 5)$
- Plot the K-map:
- Place '1' in cells 1, 3, 7.
- Place 'X' in cells 0, 5.
- Place '0' in remaining cells.

A \ BC	00	01	11	10
0	X 0	1 1	1 3	0 2
1	0 4	X 5	1 7	0 6

Example: Solving with Don't Cares (Step 2)

- Group the map:
- We have 1s at 1, 3, 7. Xs at 0, 5.
- Without Xs: We would group (1,3) and (3,7). Result: A'C + BC.
- WITH Xs: We can group 1, 3, 5, 7 together!
- Cell 5 is an X. Cell 1, 3, 7 are 1s.
- By treating the X at 5 as a 1, we can form a group of FOUR: (1, 3, 5, 7).

A \ BC	00	01	11	10
0	X	1	1	0
1	0	X	1	0

Example: Solving with Don't Cares (Step 3)

- Group (1, 3, 5, 7) spans rows A=0 and A=1 (A is eliminated).
- It spans columns BC=01, 11 (B changes, C is constant at 1).
- The entire group reduces to just: C.
- What about the X at 0? We don't need it. Leave it as 0.
- Final simplified equation: $Y = C$.

Comparison: With vs. Without Don't Cares

Comparison: With vs. Without Don't Cares

- Without Don't Cares: $Y = A'C + BC$
  - Requires two AND gates, one OR gate, one NOT gate.
- With Don't Cares: $Y = C$
  - Requires ZERO gates. Just a wire connected directly to input C.
- Conclusion: Utilizing Don't Cares aggressively minimizes circuits.

Without Don't Cares:
$Y = A'C + BC$
Hardware: High

With Don't Cares:
$Y = C$
Hardware: None (Wire)

Summary

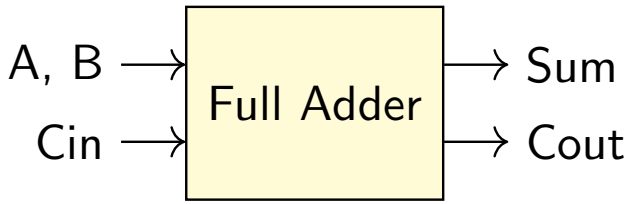
Summary

- Don't Cares (X) represent input combinations that will never occur.
- We can treat X as 0 or 1, whichever yields a simpler equation.
- Use Xs to make groups of 1s larger.
- Do not group Xs if they don't help cover a 1.
- This is a key technique for minimizing real-world circuits like BCD logic.

Next Unit Preview

Next Unit Preview

- Congratulations on completing Unit 2!
- In Unit 3, we will move from abstract logic to Combinational Logic Circuits.
- We will use everything we've learned to design Adders, Subtractors, Multiplexers, and Decoders.
- The real engineering starts now.



Questions?

Lecture 3.1: Arithmetic Circuits: Half Adder and Full Adder

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

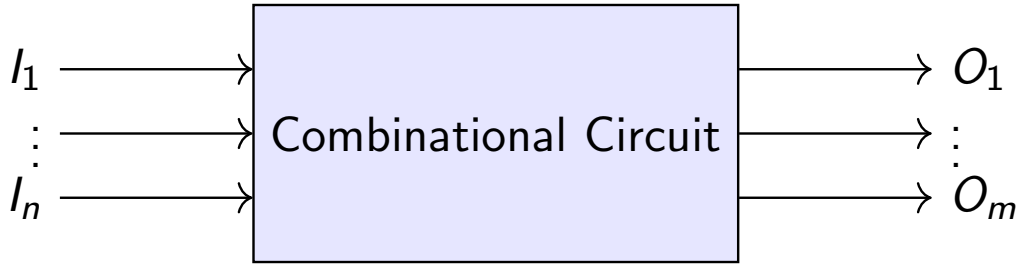
Table of Contents

Agenda

- Introduction to Combinational Logic
- Concept of Binary Addition
- The Half Adder: Truth Table and Boolean Expressions
- Half Adder Logic Circuit
- The Full Adder: Handling Carry-in
- Full Adder: Truth Table and K-Map Simplification
- Full Adder Logic Circuit

Combinational Logic Overview

- Outputs are determined strictly by the present combination of inputs.
- No memory elements or feedback loops.
- Can perform specific functions like addition, subtraction, multiplexing.
- Design flow: Problem Statement $\rightarrow$ Truth Table $\rightarrow$ K-Map $\rightarrow$ Logic Circuit.



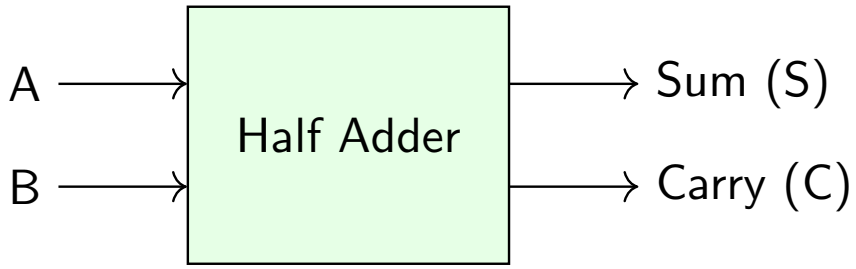
Binary Addition Basics

- $0 + 0 = 0$, Carry $= 0$
- $0 + 1 = 1$, Carry $= 0$
- $1 + 0 = 1$, Carry $= 0$
- $1 + 1 = 0$, Carry $= 1$
- When adding two 1-bit numbers, we generate two outputs: Sum and Carry.

Bit A	Bit B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

The Half Adder

- A combinational circuit that adds two 1-bit binary numbers (A and B).
- It has two outputs: Sum (S) and Carry (C).
- Limitation: It cannot accept a carry from a previous lower-order bit addition.



Half Adder: Truth Table

- Inputs: A, B
- Outputs: Sum, Carry
- A=0, B=0 $\rightarrow$ Sum=0, Carry=0
- A=0, B=1 $\rightarrow$ Sum=1, Carry=0
- A=1, B=0 $\rightarrow$ Sum=1, Carry=0
- A=1, B=1 $\rightarrow$ Sum=0, Carry=1

A	**B**	**Sum (S)**	**Carry (C)**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

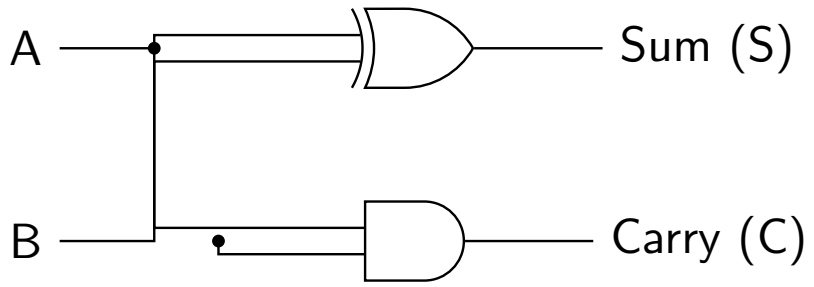
Half Adder: Boolean Expressions

- From Truth Table:
- Sum (S) $= A'B + AB' = A \oplus B$ (XOR operation)
- Carry (C) $= AB$ (AND operation)
- We can implement a Half Adder using just one XOR gate and one AND gate.

$$S = \sum m(1,2) = A \oplus B$$
$$C = \sum m(3) = AB$$

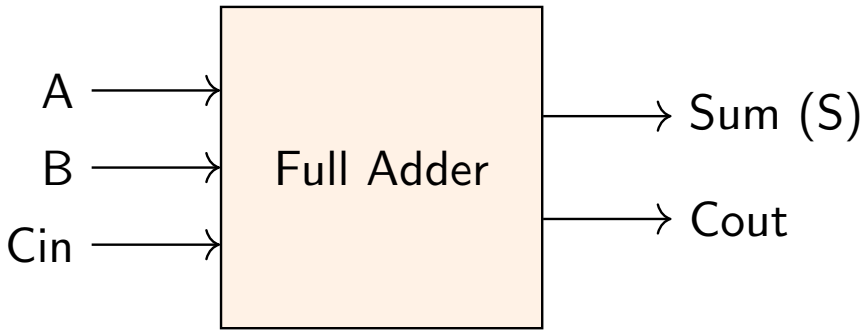
Half Adder: Logic Circuit

- Input A connected to XOR and AND gates.
- Input B connected to XOR and AND gates.
- XOR output is the Sum.
- AND output is the Carry.



The Full Adder: Concept

- To add multi-bit numbers, we must account for carry-in from previous bit additions.
- A Full Adder adds three 1-bit numbers: A, B, and Carry-in (Cin).
- Outputs: Sum (S) and Carry-out (Cout).



Full Adder: Truth Table

- Inputs: A, B, Cin (8 combinations)
- For A=0, B=1, Cin=1: Sum=0, Cout=1 (since 1+1 = 2)
- For A=1, B=1, Cin=1: Sum=1, Cout=1 (since 1+1+1 = 3)
- Sum is 1 when odd number of inputs are 1.
- Cout is 1 when two or more inputs are 1.

A	**B**	**Cin**	**Sum (S)**	**Cout**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Full Adder: K-Map Simplification (Sum)

- Plotting the Sum output on a 3-variable K-Map.
- Minterms for Sum: 1, 2, 4, 7.
- No adjacent cells to group. It forms a checkerboard pattern.
- Expression simplifies to: $S = A \oplus B \oplus Cin$

A\BCin	00	01	11	10
0	0	1	0	1
1	1	0	1	0

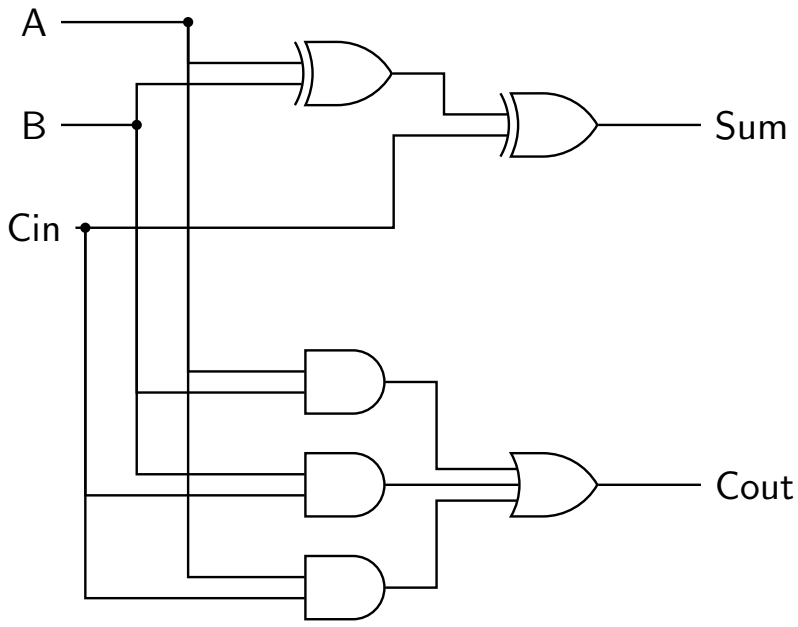
Full Adder: K-Map Simplification (Cout)

- Plotting the Cout output on a 3-variable K-Map.
- Minterms for Cout: 3, 5, 6, 7.
- We can form three groups of two (pairs).
- Expression simplifies to: $Cout = AB + BCin + ACin$

A\BCin	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Full Adder: Logic Circuit

- Sum circuit: Two cascaded 2-input XOR gates (or one 3-input XOR gate).
- Cout circuit: Three AND gates and one OR gate to implement $AB + BCin + ACin$.
- Alternatively, a Full Adder can be built using two Half Adders and an OR gate.



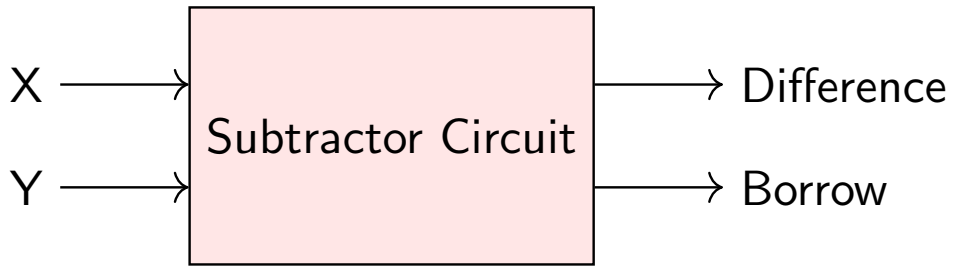
Summary

- Combinational circuits have outputs dependent only on present inputs.
- Half Adder adds two bits: $S = A \oplus B$, $C = AB$.
- Full Adder adds three bits (including Cin): $S = A \oplus B \oplus Cin$, $Cout = AB + BCin + ACin$.
- Full Adders are the building blocks for multi-bit binary addition.

Circuit	**Inputs**	**Outputs**
Half Adder	A, B	Sum, Carry
Full Adder	A, B, Cin	Sum, Cout

Next Lecture Preview

- Arithmetic Circuits: Half Subtractor and Full Subtractor.
- We will learn how to perform binary subtraction using digital logic.
- We will derive truth tables and circuits using the same methodology.



Questions?

Lecture 3.2: Arithmetic Circuits: Half Subtractor and Full Subtractor

Complete Lecture Deck – All 45 Lectures

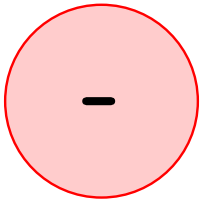
Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

Agenda

- Recap of Binary Addition
- Rules of Binary Subtraction
- The Half Subtractor: Truth Table and Equations
- Half Subtractor Logic Circuit
- The Full Subtractor: Handling Borrow-in
- Full Subtractor: K-Map Simplification
- Full Subtractor Logic Circuit



Rules of Binary Subtraction

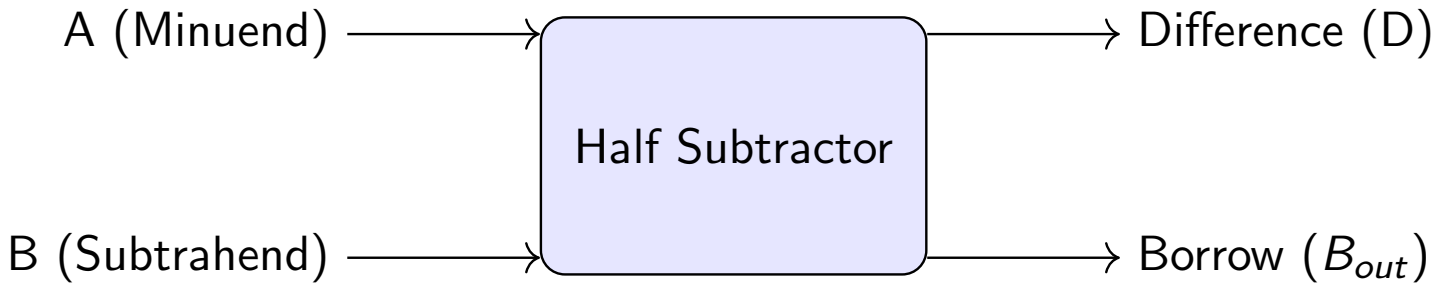
- Outputs of subtraction: Difference (D) and Borrow (B).

A (Minuend)	B (Subtrahend)	Difference (D)	Borrow (B)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

- **Note:** For 0 - 1, we must borrow 1 from the next higher significant bit.

The Half Subtractor

- A combinational circuit that subtracts two 1-bit binary numbers (Minuend A and Subtrahend B).
- Outputs: Difference (D) and Borrow-out (Bout).
- Limitation: Cannot accept a borrow-in from a previous lower-order bit subtraction.



Half Subtractor: Truth Table

- Inputs: A (Minuend), B (Subtrahend)

A	**B**	**D**	B_{out}
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Half Subtractor: Boolean Expressions

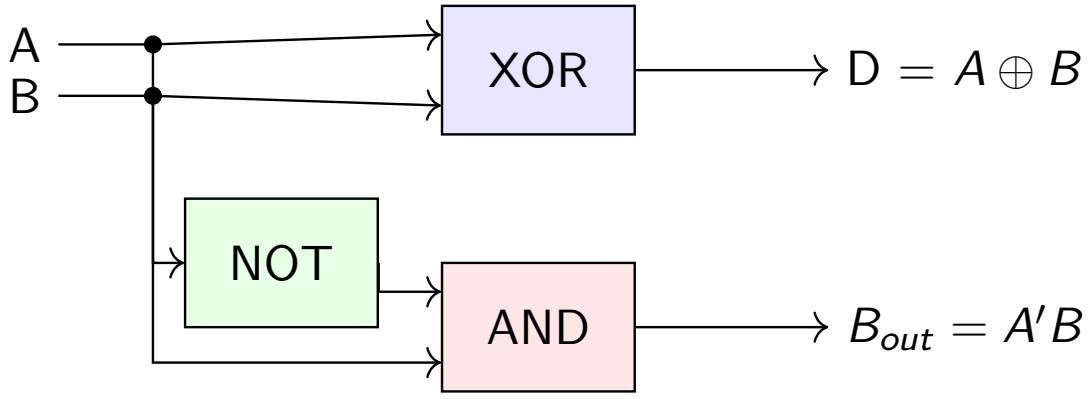
- From Truth Table:
- Difference (D) $= A'B + AB' = A \oplus B$
- Borrow (B_{out}) $= A'B$
- Notice that Difference uses the exact same XOR gate as the Sum in a Half Adder.

Key Equations

$$D = A \oplus B$$
$$B_{out} = A'B$$

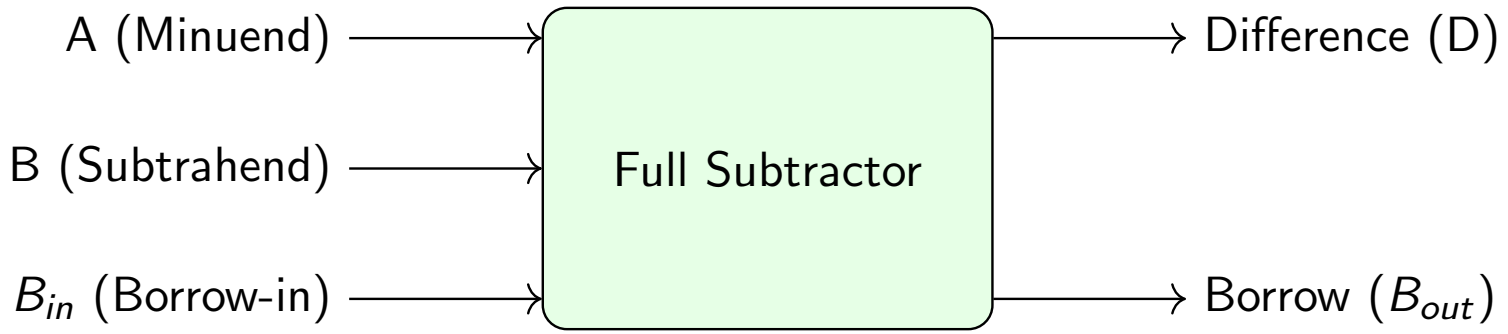
Half Subtractor: Logic Circuit

- Difference circuit: 2-input XOR gate.
- Borrow circuit: NOT gate (to invert A) connected to a 2-input AND gate along with B.
- Requires one XOR, one NOT, and one AND gate.



The Full Subtractor

- To subtract multi-bit numbers, we must account for borrows from lower-order bits.
- A Full Subtractor performs subtraction on three bits: A (Minuend), B (Subtrahend), and B_{in} (Borrow-in).
- Outputs: Difference (D) and Borrow-out (B_{out}).



Full Subtractor: Truth Table Formulation

- Inputs: A, B, B_{in} (8 combinations)
- Example 1: A=0, B=0, B_{in}=1. Compute 0 - 0 - 1. D=1, B_{out}=1.
- Example 2: A=1, B=1, B_{in}=1. Compute 1 - 1 - 1. D=1, B_{out}=1.

A	**B**	B_{in}	**D**	B_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Full Subtractor: Difference (D) K-Map

- Minterms for Difference: 1, 2, 4, 7.
- Plotting these on a 3-variable K-map yields a checkerboard pattern.
- Expression simplifies to: $\mathrm{D} = A \oplus B \oplus B_{in}$.
- Just like the Full Adder, Difference is the XOR of all three inputs.

$A \backslash BB_{in}$	00	01	11	10
0	0	1	0	1
1	1	0	1	0

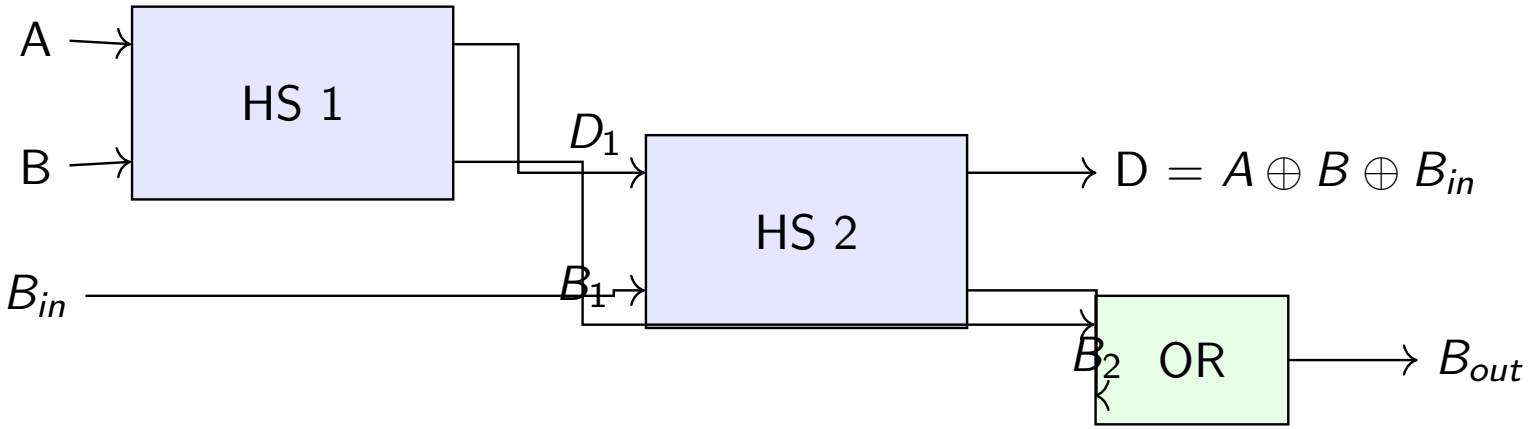
Full Subtractor: Borrow-out (B_{out}) K-Map

- Minterms for Borrow-out: 1, 2, 3, 7.
- Plotting these yields three overlapping pairs.
- Expression simplifies to: $B_{out} = A'B + A'B_{in} + BB_{in}$.

$A \backslash BB_{in}$	00	01	11	10
0	0	1	1	1
1	0	0	1	0

Full Subtractor: Logic Circuit

- Difference: Two 2-input XOR gates cascaded.
- Borrow-out: NOT gates, AND gates, and an OR gate.
- Alternatively, can be built using two Half Subtractors and an OR gate.



Comparing Adders and Subtractors

- Sum and Difference equations are IDENTICAL (XOR operations).
- Carry vs Borrow equations differ only by NOT gates on the first input.
- This similarity allows us to build unified Adder/Subtractor circuits.

Full Adder	**Full Subtractor**
$Sum = A \oplus B \oplus C_{in}$	$Diff = A \oplus B \oplus B_{in}$
$C_{out} = AB + AC_{in} + BC_{in}$	$B_{out} = A'B + A'B_{in} + BB_{in}$

Summary

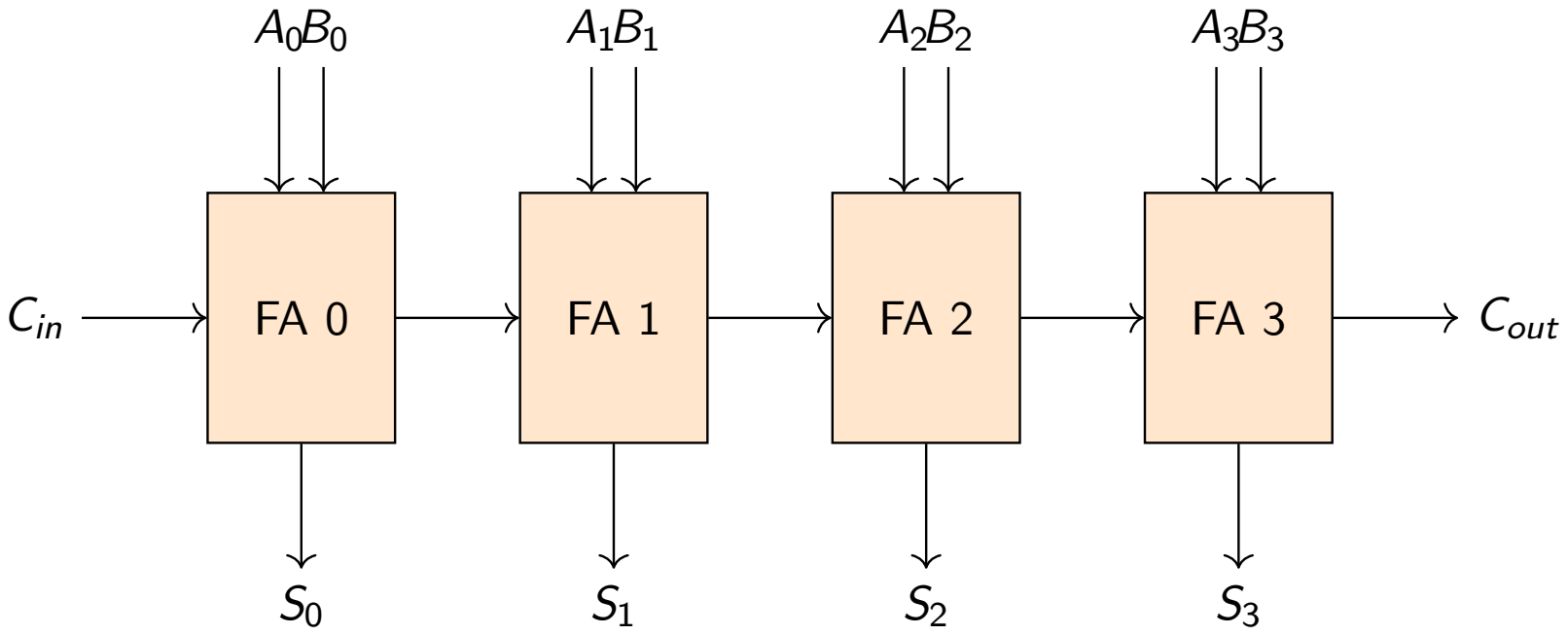
- Half Subtractor subtracts two bits but cannot handle borrow-in.
- Difference uses XOR; Borrow uses AND with an inverted input.
- Full Subtractor subtracts three bits, fully supporting borrow chaining.
- Difference $= A \oplus B \oplus B_{in}$; $B_{out} = A'B + A'B_{in} + BB_{in}$.

Key Equations

$$Diff = A \oplus B \oplus B_{in}$$
$$B_{out} = A'B + A'B_{in} + BB_{in}$$

Next Lecture Preview

- Block Diagram of Parallel Adder.
- We will learn how to connect multiple Full Adders together.
- We will see how to add 4-bit numbers (like 1010 + 0110) in hardware.



Questions?

Lecture 3.3: Parallel Adder Using Half and Full Adder Blocks

Complete Lecture Deck – All 45 Lectures

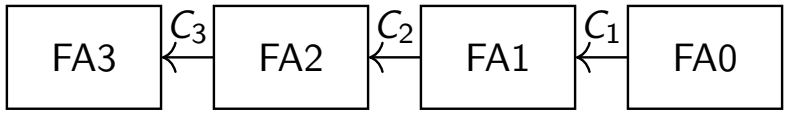
Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

Agenda

- Need for Multi-bit Addition
- Concept of Parallel Processing in Logic
- Block Diagram of a 4-bit Parallel Adder
- The Role of the Half Adder (LSB)
- Cascading Carry Bits (Ripple Carry)
- Example: Adding two 4-bit numbers
- Propagation Delay in Ripple Carry Adders



Lecture 3.3: Parallel Adder Using Half and Full Adder Blocks

Need for Multi-bit Addition

- A single Full Adder only adds 1-bit numbers.
- Real-world computers use 8-bit, 16-bit, 32-bit, or 64-bit numbers.
- To add two 4-bit numbers (e.g., $A_3A_2A_1A_0 + B_3B_2B_1B_0$), we need multiple circuits.
- The components must operate together to produce a 4-bit sum and carry-out.

1-Bit Addition

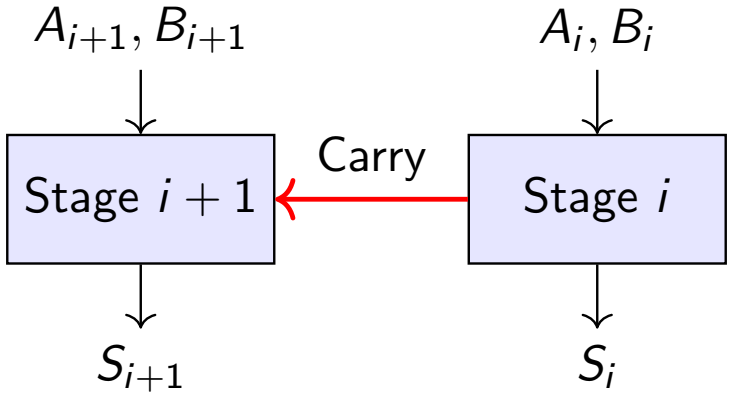
$$\begin{array}{r} A_0 \\ +\ B_0 \\ \hline S_0 \end{array}$$

4-Bit Addition

$$\begin{array}{r} A_3\ A_2\ A_1\ A_0 \\ +\ B_3\ B_2\ B_1\ B_0 \\ \hline S_3\ S_2\ S_1\ S_0 \end{array}$$

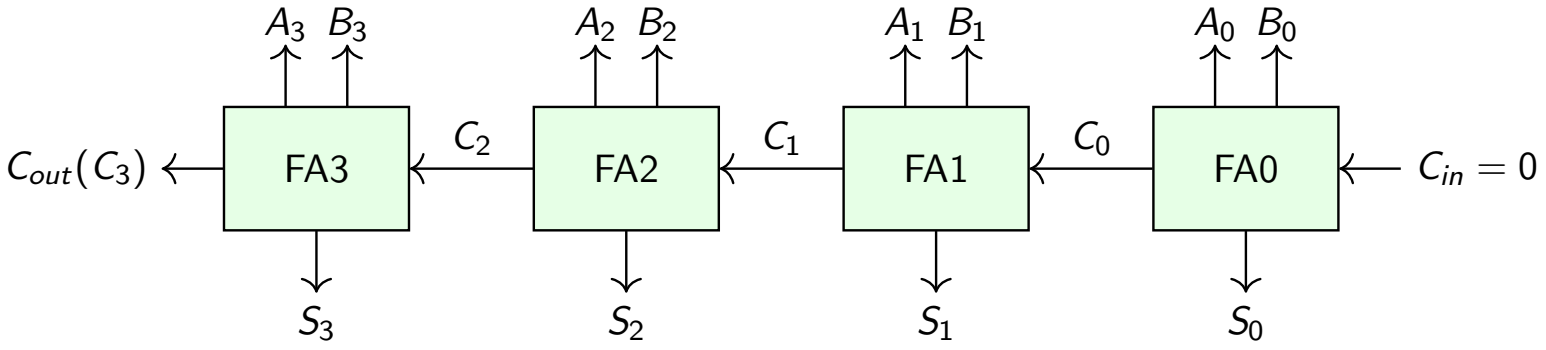
Concept of the Parallel Adder

- A parallel adder is a digital circuit that adds multi-bit numbers simultaneously.
- It consists of Full Adders connected in a cascade.
- The Carry-out (C_{out}) of one adder is connected to the Carry-in (C_{in}) of the next higher-order adder.
- Also known as a 'Ripple Carry Adder'.



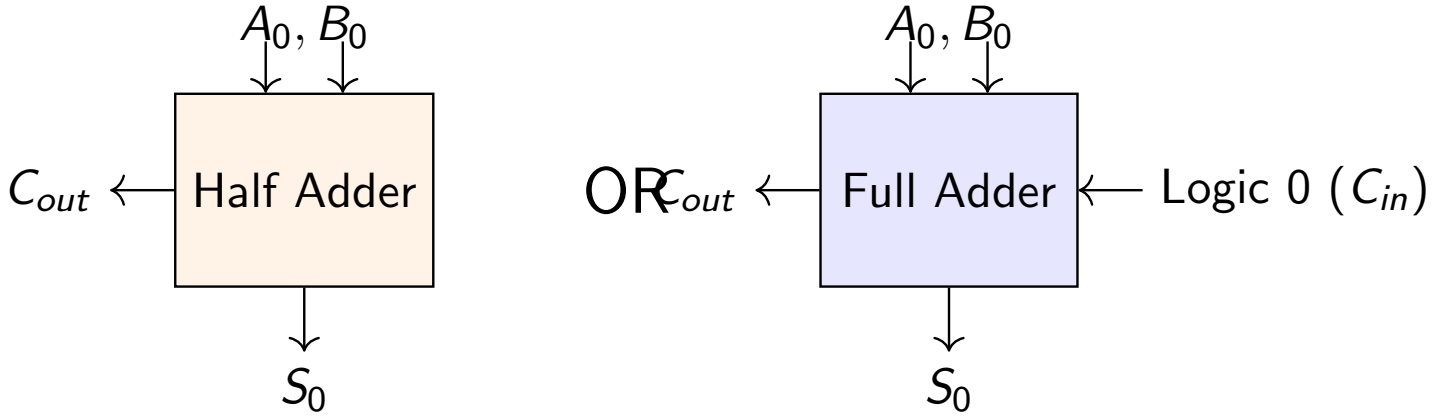
The 4-Bit Parallel Adder Architecture

- Let Inputs be $A = A_3A_2A_1A_0$ and $B = B_3B_2B_1B_0$.
- Stage 0 (LSB): Adds A_0, B_0. Produces S_0 and C_0.
- Stage 1: Adds A_1, B_1, C_0. Produces S_1 and C_1.
- Stage 2: Adds A_2, B_2, C_1. Produces S_2 and C_2.
- Stage 3 (MSB): Adds A_3, B_3, C_2. Produces S_3 and final C_3.



Role of the Half Adder at LSB

- The very first stage (adding A_0 and B_0) has no previous carry-in.
- Therefore, we can use a Half Adder for the LSB stage.
- Alternatively, we can use a Full Adder and hardwire its C_{in} to Logic 0.
- Using a Full Adder with $C_{in} = 0$ is standard practice in generic ICs to maintain uniformity.



Example: Adding 1010 and 0011

- $A = 1010$ (Decimal 10), $B = 0011$ (Decimal 3)
- Stage 0: $0 + 1 + (0) = 1$ ($S_0 = 1, C_0 = 0$)
- Stage 1: $1 + 1 + (0) = 0$ ($S_1 = 0, C_1 = 1$)
- Stage 2: $0 + 0 + (1) = 1$ ($S_2 = 1, C_2 = 0$)
- Stage 3: $1 + 0 + (0) = 1$ ($S_3 = 1, C_3 = 0$)
- Final Result: $C_3S_3S_2S_1S_0 = 0\ 1101$ (Decimal 13)

	C_3	C_2	C_1	C_0	
Carry	(0)	**0**	**1**	**0**	(0)
A		1	0	1	0
B	+	0	0	1	1
Sum	**0**	**1**	**1**	**0**	**1**

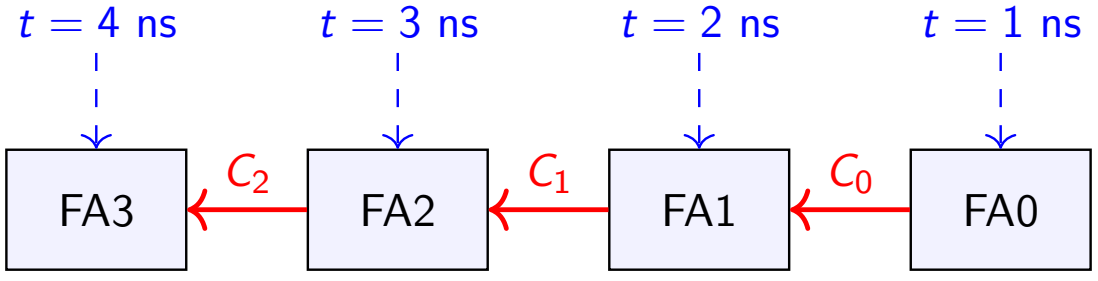
Example: Generating a Final Carry

- $A = 1111$ (15), $B = 0001$ (1)
- $S_0 : 1 + 1 + 0 \rightarrow S_0 = 0, C_0 = 1$
- $S_1 : 1 + 0 + 1 \rightarrow S_1 = 0, C_1 = 1$
- $S_2 : 1 + 0 + 1 \rightarrow S_2 = 0, C_2 = 1$
- $S_3 : 1 + 0 + 1 \rightarrow S_3 = 0, C_3 = 1$
- Final Result: 1 0000 (Decimal 16). The 5th bit is the final carry-out.

	C_3	C_2	C_1	C_0	
Carry	(1)	**1**	**1**	**1**	(0)
A		1	1	1	1
B	+	0	0	0	1
Sum	**1**	**0**	**0**	**0**	**0**

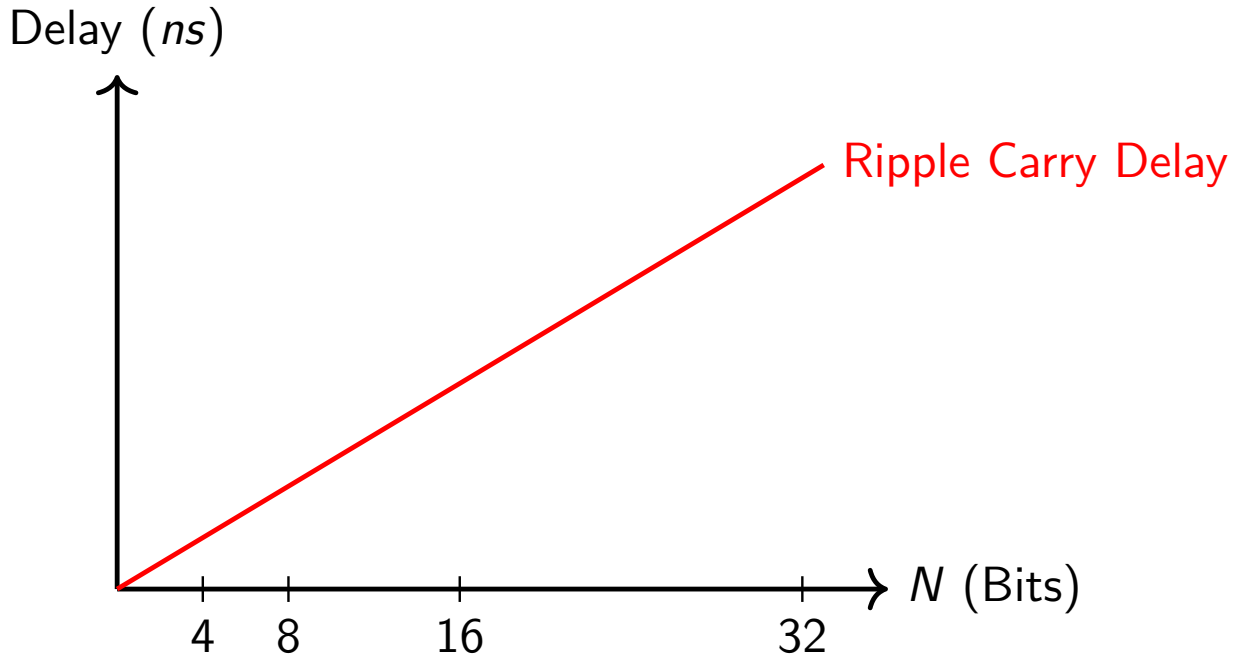
The Ripple Carry Effect

- Full Adder 1 cannot compute its final Sum and Carry until it receives the Carry from Full Adder 0.
- Full Adder 2 must wait for Full Adder 1, and so on.
- The carry signal 'ripples' through the adders like a wave.
- This waiting creates a sequential delay.



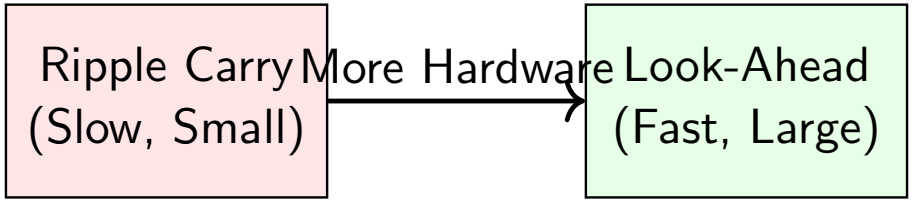
Propagation Delay

- Every logic gate has a small internal delay (e.g., 2 nanoseconds).
- In a Ripple Carry Adder, the critical path is the carry chain.
- Total Delay $\approx N \times$ (Delay of one Full Adder), where N is the number of bits.
- For a 32-bit ripple carry adder, the delay becomes significant and slows down the CPU.



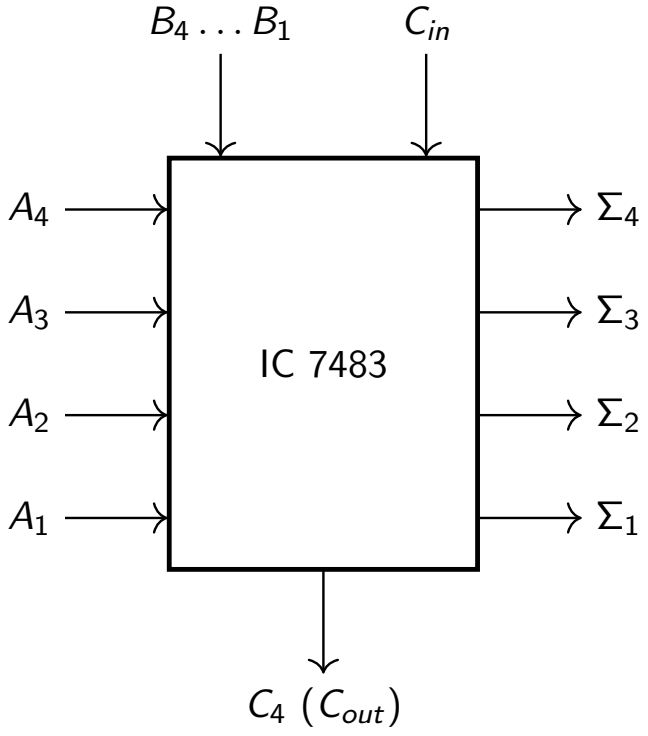
Solving the Delay: Look-Ahead Carry Adder (Preview)

- To solve the ripple delay, engineers designed 'Carry Look-Ahead Adders'.
- These circuits calculate the carries in advance based on the A and B inputs.
- Trade-off: Extremely fast, but requires significantly more hardware and logic gates.
- (Detailed design of Look-Ahead is typically an advanced topic).



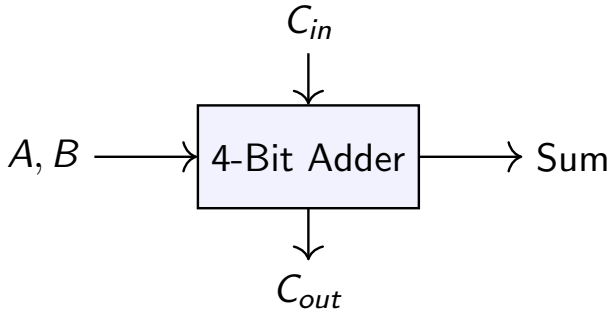
IC 7483: 4-Bit Binary Adder

- The 7483 is a standard TTL Integrated Circuit that contains a 4-bit parallel adder.
- Inputs: A (4 bits), B (4 bits), C_{in}.
- Outputs: Sum (4 bits), C_{out}.
- It uses internal look-ahead circuitry for speed.



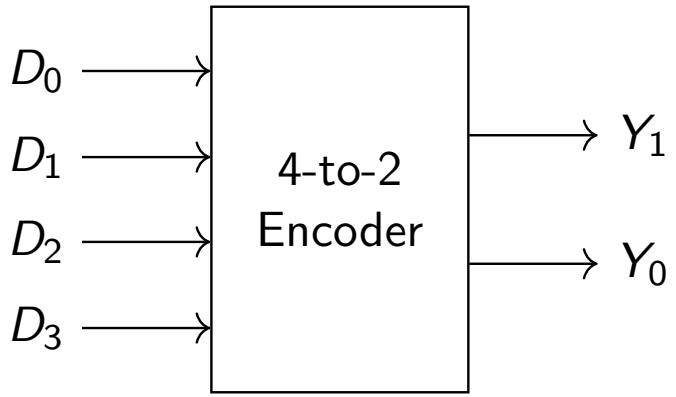
Summary

- Parallel adders add multi-bit numbers simultaneously.
- Constructed by cascading Full Adders, with the LSB capable of being a Half Adder.
- In a Ripple Carry Adder, the carry propagates from LSB to MSB.
- Propagation delay limits the speed of ripple carry adders in large bit-width systems.



Next Lecture Preview

- Topic: Encoders (4:2, 8:3).
- We shift from arithmetic circuits to data routing and conversion circuits.
- We will learn how to compress multiple input lines into fewer binary output lines.



Questions?

Lecture 3.4: Encoders (4:2 and 8:3)

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

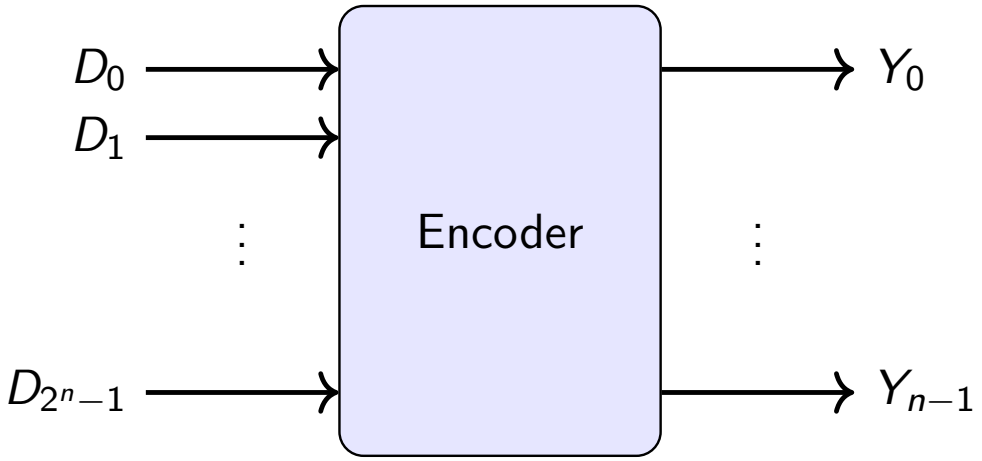
Lecture 3.4: Encoders (4:2 and 8:3)

Agenda

- What is an Encoder?
- Applications of Encoders
- 4-to-2 Line Encoder: Truth Table and Logic
- Limitation of Basic Encoders
- 8-to-3 Line Encoder (Octal to Binary)
- Truth Table and Logic Circuit for 8:3 Encoder
- Introduction to Priority Encoders

What is an Encoder?

- A combinational circuit that performs the inverse operation of a decoder.
- Has up to 2^n input lines and exactly n output lines.
- It 'encodes' the active input line into a binary code at the output.
- Assumption: Only one input line is active (high) at any given time.



2^n inputs, n outputs

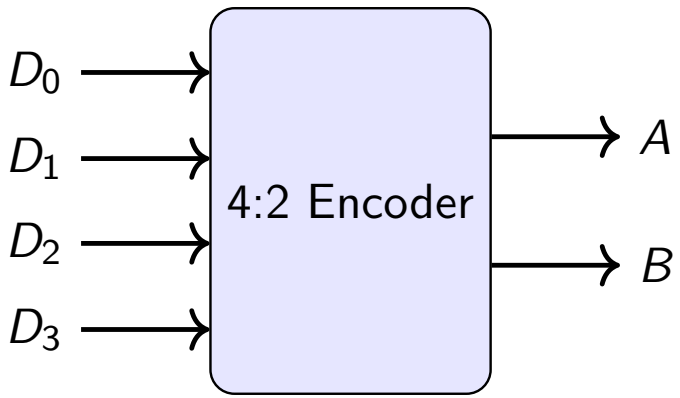
Applications of Encoders

- **Keyboard Encoders:** Converting a key press into an ASCII or binary code.
- **Analog-to-Digital Converters (ADCs):** Translating discrete voltage levels into binary.
- **Interrupt Requests:** Handling multiple peripheral interrupts in a microprocessor.
- **Data Compression:** Reducing the number of data lines required to transmit information.

Lecture 3.4: Encoders (4:2 and 8:3)

4-to-2 Line Encoder

- Has 4 inputs (D_0, D_1, D_2, D_3) and 2 outputs (A, B).
- It outputs a 2-bit binary number corresponding to the active input.
- For example, if D_2 is active (1), the output AB will be 10 (binary for 2).



4-to-2 Encoder: Truth Table

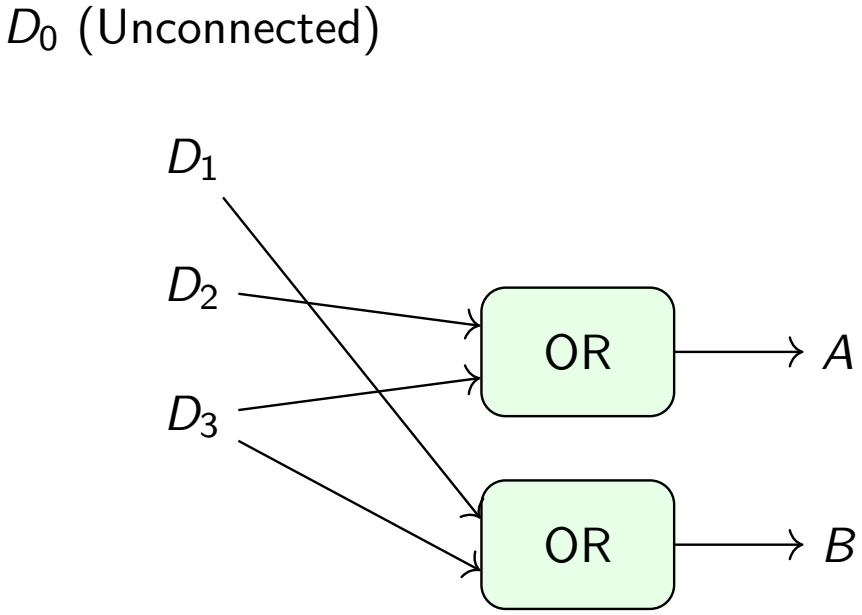
Inputs				Outputs	
D_0	D_1	D_2	D_3	A	B
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

4-to-2 Encoder: Logic Expressions

- Look at the Truth Table for output A: A is 1 when D_2 is 1 OR D_3 is 1.
- Therefore, $A = D_2 + D_3$
- Look at output B: B is 1 when D_1 is 1 OR D_3 is 1.
- Therefore, $B = D_1 + D_3$

4-to-2 Encoder: Logic Circuit

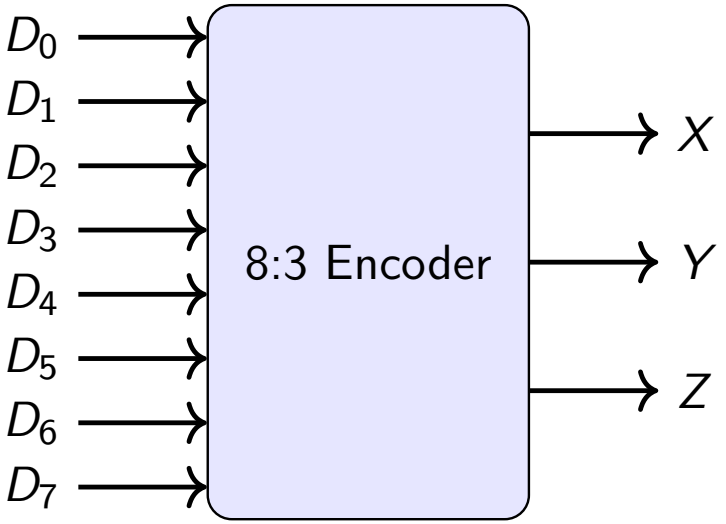
- Implemented using two 2-input OR gates.
- OR gate 1: Inputs $D_2, D_3 \rightarrow$ Output A.
- OR gate 2: Inputs $D_1, D_3 \rightarrow$ Output B.
- Note: D_0 is not connected to any logic gate!



Lecture 3.4: Encoders (4:2 and 8:3)

8-to-3 Line Encoder (Octal to Binary)

- Has 8 inputs (D_0 through D_7) and 3 outputs (X, Y, Z).
- Converts an octal input representation into a 3-bit binary code.
- If D_5 is active, the output XYZ will be 101.



8-to-3 Encoder: Truth Table

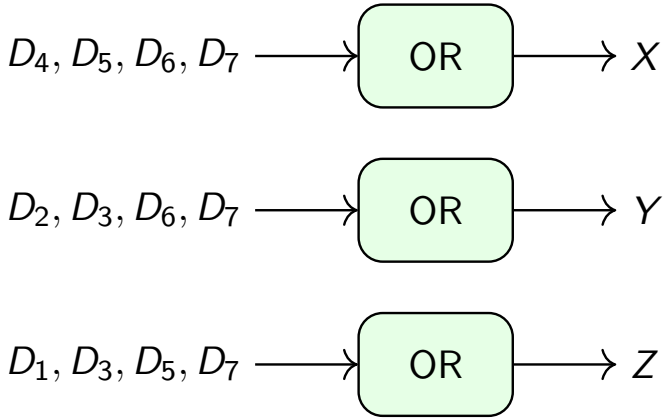
Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	X	Y	Z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

8-to-3 Encoder: Logic Expressions

- Output X (MSB) is 1 for inputs 4, 5, 6, 7.
- $X = D_4 + D_5 + D_6 + D_7$
- Output Y is 1 for inputs 2, 3, 6, 7.
- $Y = D_2 + D_3 + D_6 + D_7$
- Output Z (LSB) is 1 for odd inputs.
- $Z = D_1 + D_3 + D_5 + D_7$

8-to-3 Encoder: Logic Circuit

- Implemented using three 4-input OR gates.
- Just like the 4:2 encoder, input D_0 is not physically connected.
- Very fast, low-complexity circuit.



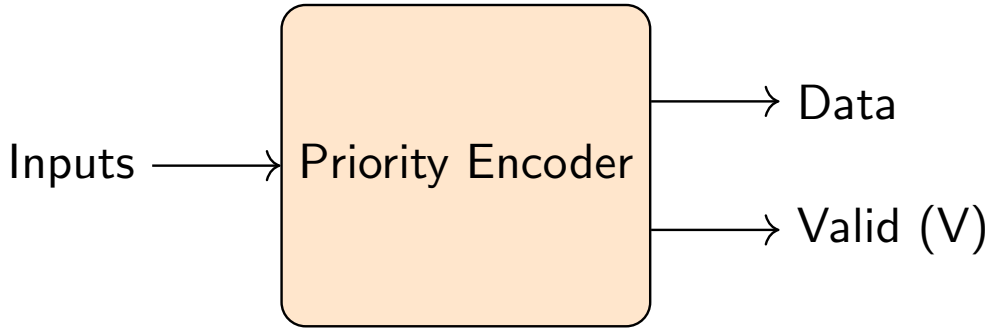
Lecture 3.4: Encoders (4:2 and 8:3)

Limitation: The Ambiguity Problem

- What happens if all inputs are 0? The output is 000.
- What happens if D_0 is 1? The output is also 000.
- The circuit cannot distinguish between 'D_0 is active' and 'No inputs are active'.
- What happens if D_2 AND D_4 are pressed simultaneously? The output is corrupted.

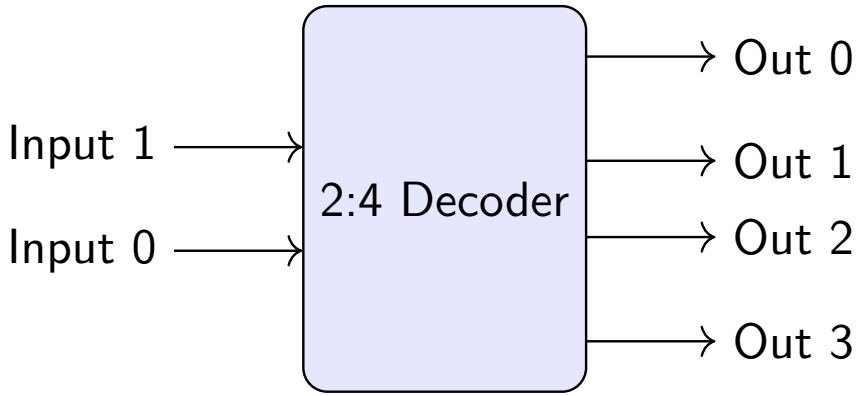
Summary & Priority Encoders

- Encoders compress 2^n inputs into n outputs.
- 4:2 encoder uses two OR gates; 8:3 uses three OR gates.
- To solve the ambiguity problem, 'Priority Encoders' are used.
- Priority Encoders assign a rank to inputs and have a 'Valid' bit output to indicate if any button is pressed.



Next Lecture Preview

- Topic: Decoders (2:4, 3:8).
- We will study the exact opposite of the encoder.
- Decoders expand n inputs into 2^n outputs.
- Used heavily in memory addressing.



Questions?

Lecture 3.5: Decoders (2:4 and 3:8)

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

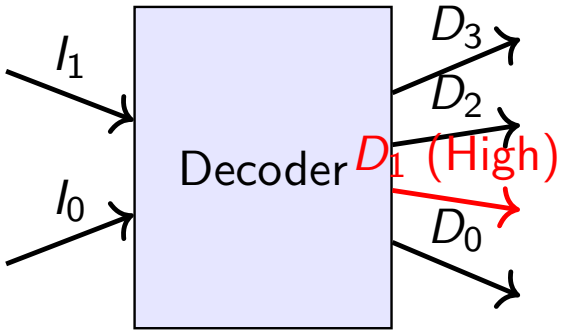
Lecture 3.5: Decoders (2:4 and 3:8)

Agenda

- What is a Decoder?
- Applications of Decoders
- 2-to-4 Line Decoder: Truth Table and Logic
- The Enable (E) Input
- 3-to-8 Line Decoder (Binary to Octal)
- Truth Table and Logic Circuit for 3:8 Decoder
- Building Large Decoders from Smaller Ones

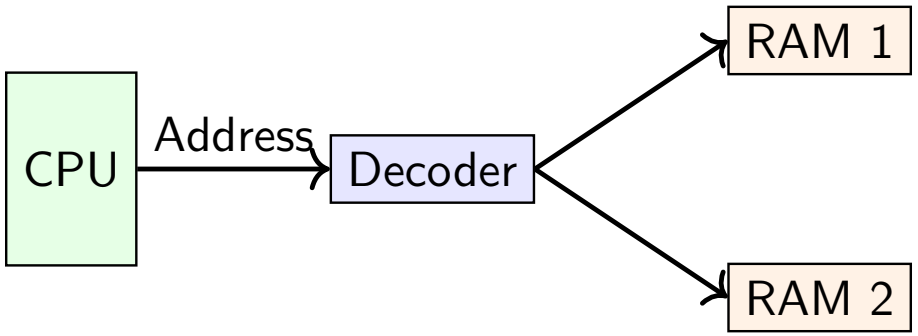
What is a Decoder?

- A combinational circuit that converts n input lines into a maximum of 2^n output lines.
- It 'decodes' binary information.
- Based on the binary input, exactly ONE output line goes high (active-high).
- All other output lines remain low.



Applications of Decoders

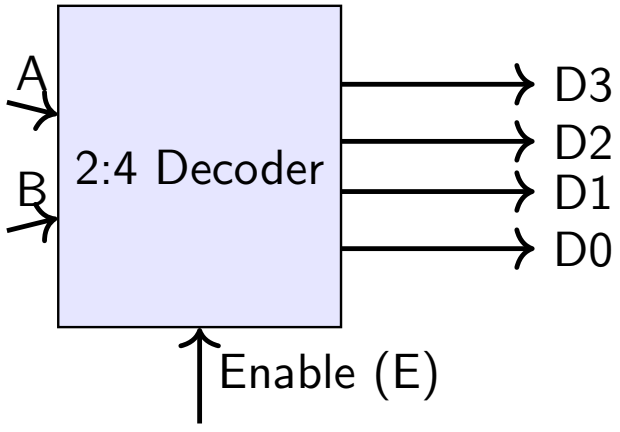
- **Memory Addressing:** Selecting a RAM chip based on an address bus.
- **Instruction Decoding:** Translating instructions into control signals.
- **Data Demultiplexing:** (when combined with an Enable pin).
- **Seven-Segment Displays:** (e.g., BCD to 7-segment).



Lecture 3.5: Decoders (2:4 and 3:8)

2-to-4 Line Decoder

- Has 2 inputs (A, B) and 4 outputs (D0, D1, D2, D3).
- Also features an Enable (E) input.
- If Enable is 0, all outputs are 0 regardless of inputs.
- If Enable is 1, the circuit decodes the input A,B.



2-to-4 Decoder: Truth Table

E	A	B	D0	D1	D2	D3
0	X	X	0	0	0	0
1	0	0	**1**	0	0	0
1	0	1	0	**1**	0	0
1	1	0	0	0	**1**	0
1	1	1	0	0	0	**1**

2-to-4 Decoder: Logic Expressions

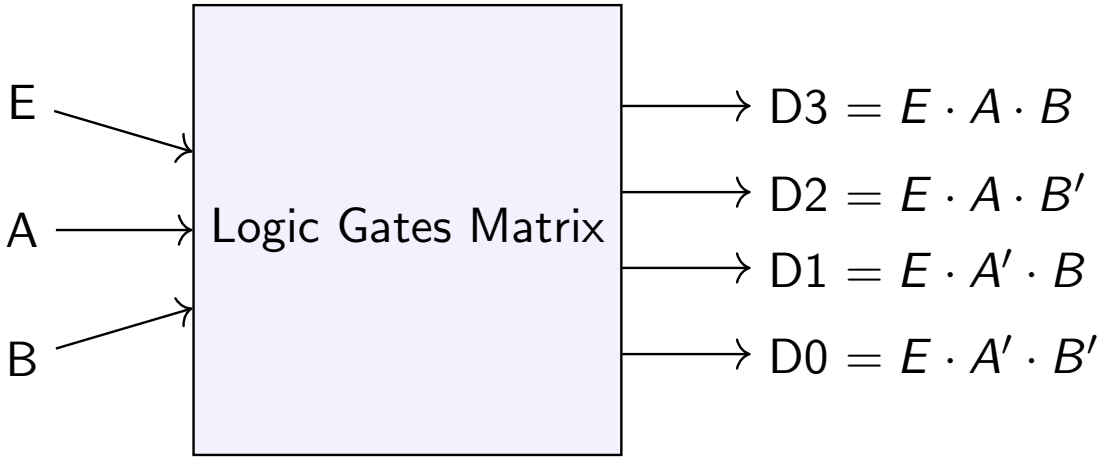
- Each output corresponds to a specific minterm of the inputs.

Logic Equations

$$D_0 = E \cdot A' \cdot B'$$
$$D_1 = E \cdot A' \cdot B$$
$$D_2 = E \cdot A \cdot B'$$
$$D_3 = E \cdot A \cdot B$$

- We use AND gates to implement this logic.

2-to-4 Decoder: Logic Circuit



Active-Low vs Active-High

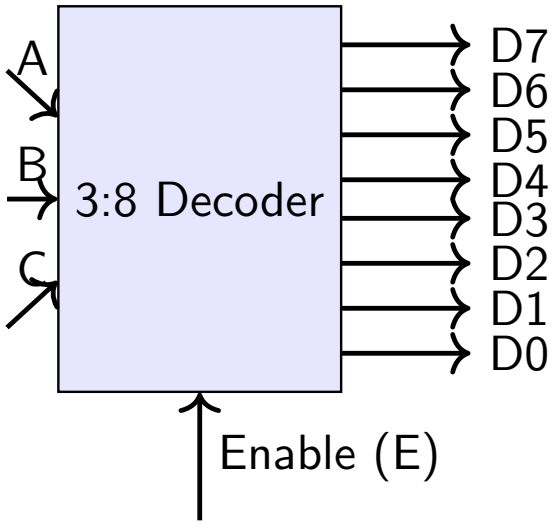
- We designed an **Active-High** decoder (selected output is 1, rest are 0).
- Commercial ICs (like the 74138) are often **Active-Low**.
- In Active-Low, selected output is 0, others are 1 (using NAND gates).

Type (Input=00)	**D0**	**D1**	**D2**	**D3**
Active-High	**1**	0	0	0
Active-Low	**0**	1	1	1

Lecture 3.5: Decoders (2:4 and 3:8)

3-to-8 Line Decoder (Binary to Octal)

- Has 3 inputs (A, B, C) and 8 outputs (D0 to D7).
- Often called a Binary to Octal decoder.
- Converts a 3-bit binary input into one of eight active lines.



3-to-8 Decoder: Truth Table Summary

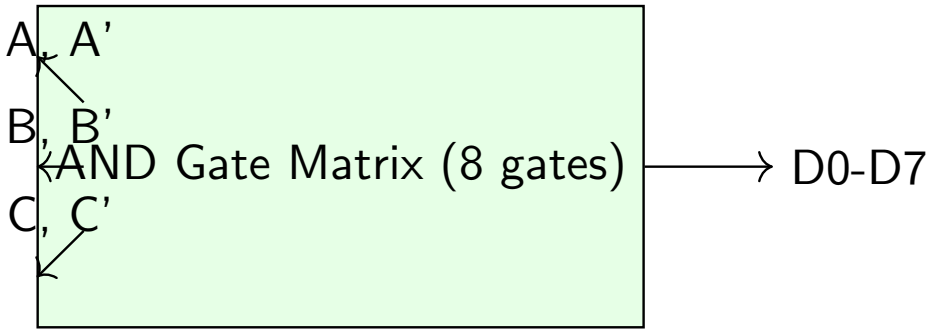
- If Enable = 1:

A	B	C	Active Output
0	0	0	D0 is High
0	1	1	D3 is High
1	1	1	D7 is High

- The logic uses eight 4-input AND gates (A, B, C, and E).

3-to-8 Decoder: Logic Circuit

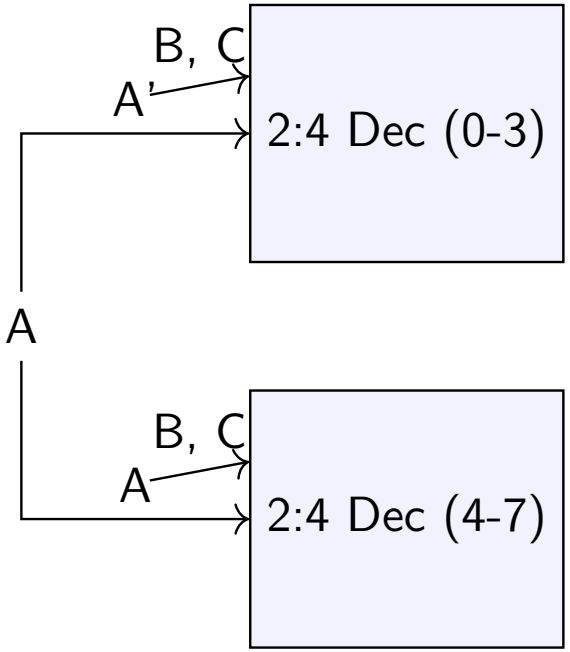
- Requires three NOT gates to generate A', B', C'.
- Requires eight AND gates for the outputs D0-D7.
- Matrix ensures exactly one AND gate receives all 1s for any input.



Lecture 3.5: Decoders (2:4 and 3:8)

Expanding Decoders

- Build a 3-to-8 decoder using two 2-to-4 decoders.
- MSB of the input (A) is connected to Enable pins.
- A' enables the first decoder (000-011).
- A enables the second decoder (100-111).



Lecture 3.5: Decoders (2:4 and 3:8)

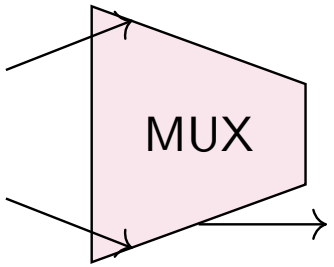
Summary

- Decoders expand n binary inputs into 2^n individual output lines.
- They generate minterms using AND (or NAND) gates.
- The Enable pin is crucial for turning the chip on/off and for cascading.
- Used extensively in memory addressing.

Decoders = Binary Code to Distinct Signals

Next Lecture Preview

- Topic: Multiplexers (MUX 2:1, 4:1).
- We will learn how to select one data stream out of many.
- Multiplexers are the 'digital switches' of combinational logic.



Questions?

Lecture 3.6: Multiplexers (2:1 and 4:1)

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

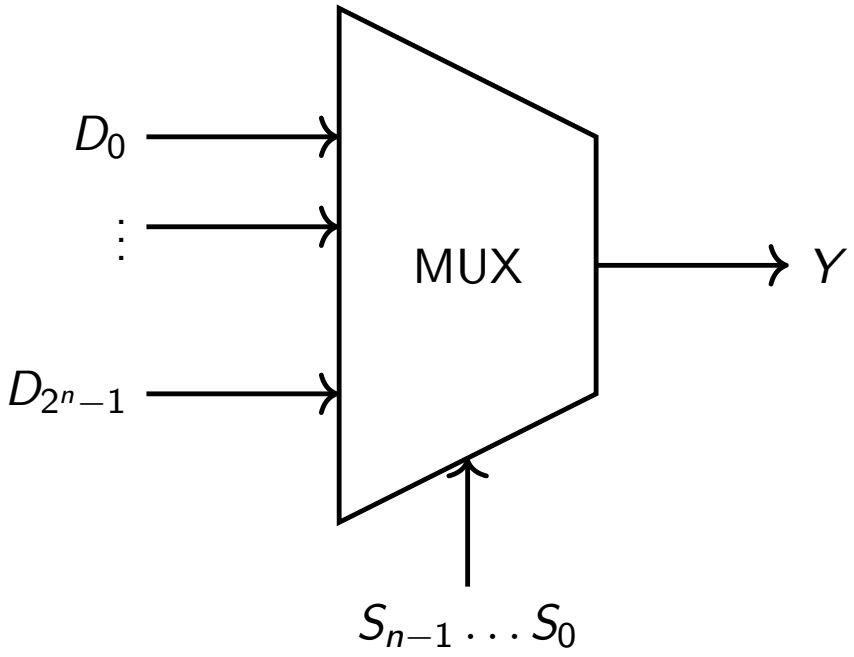
Table of Contents

Agenda

- What is a Multiplexer (MUX)?
- The Analogy of a Rotary Switch
- 2-to-1 Multiplexer: Logic and Circuit
- 4-to-1 Multiplexer: Truth Table and Equation
- 4-to-1 Multiplexer: Logic Circuit
- Applications of Multiplexers

What is a Multiplexer (MUX)?

- A MUX is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line.
- It has 2^n data inputs, n select lines, and 1 output.
- Often called a 'Data Selector'.

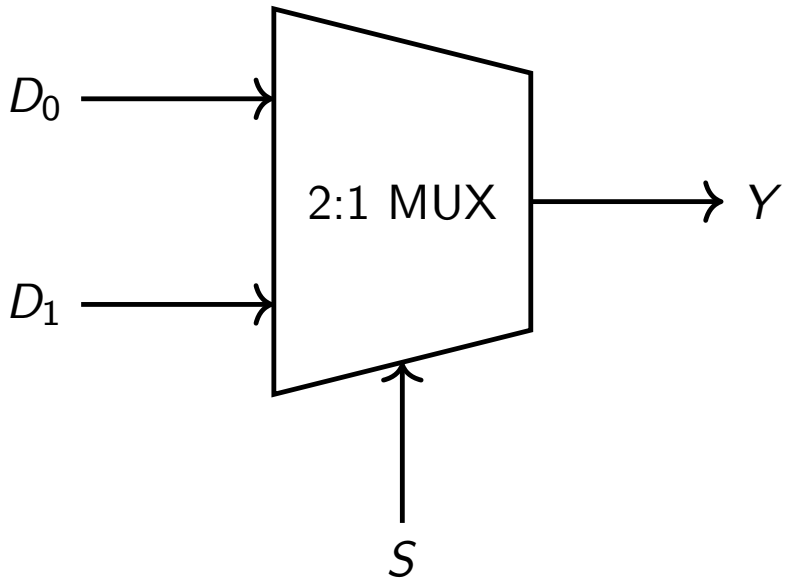


The Analogy: Digital Rotary Switch

- Imagine an old stereo system with inputs for Radio, CD, Aux, and Vinyl.
- A physical rotary knob selects which audio source goes to the speakers.
- A MUX does this electronically. The data lines are the audio sources, the output is the speaker, and the select lines act as the knob.

2-to-1 Multiplexer

- Data Inputs: D_0, D_1
- Select Line: S
- Output: Y
- If $S = 0$, Output $Y = D_0$.
- If $S = 1$, Output $Y = D_1$.



2-to-1 MUX: Logic Equation

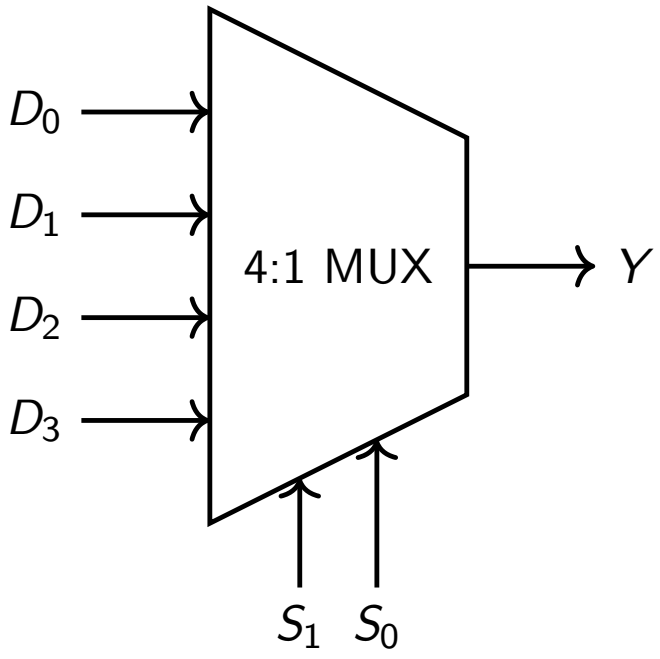
- We can write the Boolean equation directly from the behavioral description.
- $Y = (S' \cdot D_0) + (S \cdot D_1)$
- When $S = 0$, S' is 1, so $Y = D_0$.
- When $S = 1$, S is 1, so $Y = D_1$.

2-to-1 MUX: Logic Circuit

- Requires one NOT gate (for S').
- Requires two 2-input AND gates.
- Requires one 2-input OR gate at the final stage.

4-to-1 Multiplexer

- Data Inputs: D_0, D_1, D_2, D_3
- To select among 4 inputs, we need 2 select lines (S_1, S_0).
- Because $2^2 = 4$.
- Output: Y



4-to-1 MUX: Truth Table

S_1	S_0	Output Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

4-to-1 MUX: Logic Equation

- We build the equation by combining the select line conditions with the corresponding data input.
- $Y = S_1' S_0' D_0 + S_1' S_0 D_1 + S_1 S_0' D_2 + S_1 S_0 D_3$

4-to-1 MUX: Logic Circuit

- Requires two NOT gates for S_1', S_0'.
- Requires four 3-input AND gates.
- Requires one 4-input OR gate to combine the outputs.
- Only one AND gate will output the data; the other three will output 0.

Enable (Strobe) Pin in MUX

- Like decoders, practical MUX ICs (e.g., 74153) have an Enable (E) pin.
- If $E = 0$ (or disabled), the output Y is forced to 0 regardless of S or D.
- If $E = 1$ (or enabled), the MUX functions normally.
- Allows cascading multiple MUX chips together.

Applications of Multiplexers

- Data Routing: Selecting different data paths in a CPU.
- Parallel to Serial Conversion: Loading data in parallel, then using a counter on select lines to send data out one bit at a time.
- Implementing Boolean Functions: A MUX can implement any truth table without using specific logic gates.

Summary

- A Multiplexer routes one of 2^n inputs to a single output.
- Requires n select lines to control the routing.
- Equation format: Sum of (Decoder Minterms ANDed with Data Inputs).
- Functions like a digital rotary switch.

Next Lecture Preview

- Topic: Multiplexer (8:1) and De-multiplexer (1:2).
- We will scale up to an 8-input MUX.
- We will introduce the De-multiplexer (DEMUX), which performs the exact opposite function of a MUX.

Questions?

Lecture 3.7: Multiplexer (8:1) and De-multiplexer (1:2)

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

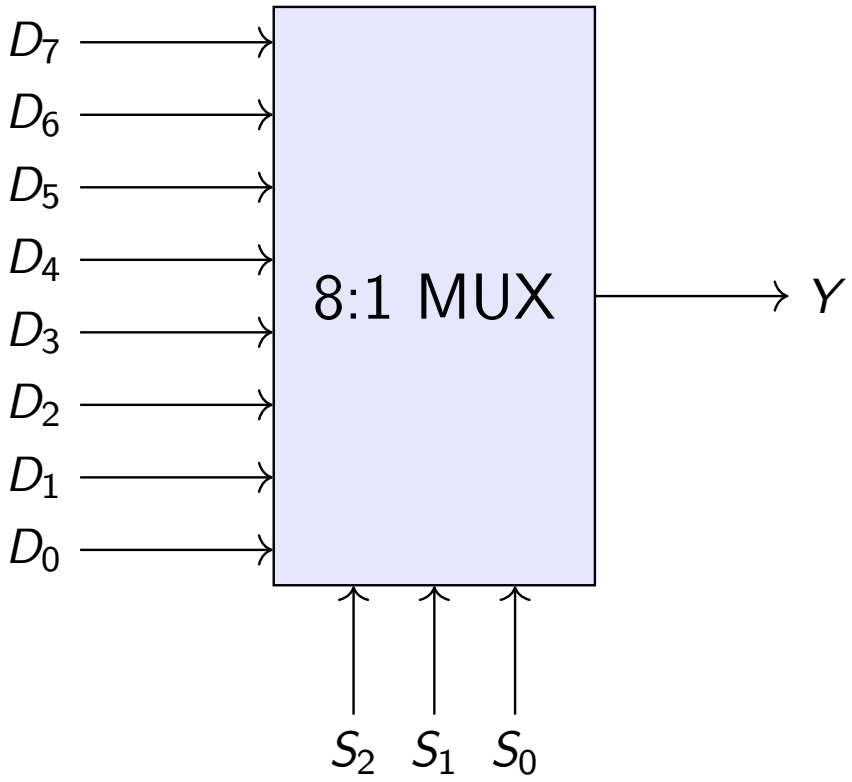
Agenda

- 8-to-1 Multiplexer (MUX)
- Truth Table and Logic Equation for 8:1 MUX
- Building an 8:1 MUX using two 4:1 MUXes
- Introduction to De-multiplexers (DEMUX)
- The Data Distributor Concept
- 1-to-2 De-multiplexer: Logic and Circuit

8-to-1 Multiplexer

8-to-1 Multiplexer

- Data Inputs: 8 lines (D0 through D7).
- Select Lines: 3 lines (S2, S1, S0) because $2^3 = 8$.
- Output: 1 line (Y).
- Selects one of 8 input signals to pass to the output based on a 3-bit code.



8-to-1 MUX: Truth Table Summary

- The binary value of S corresponds directly to the D index.

S_2	S_1	S_0	**Output** Y
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

8-to-1 MUX: Logic Equation

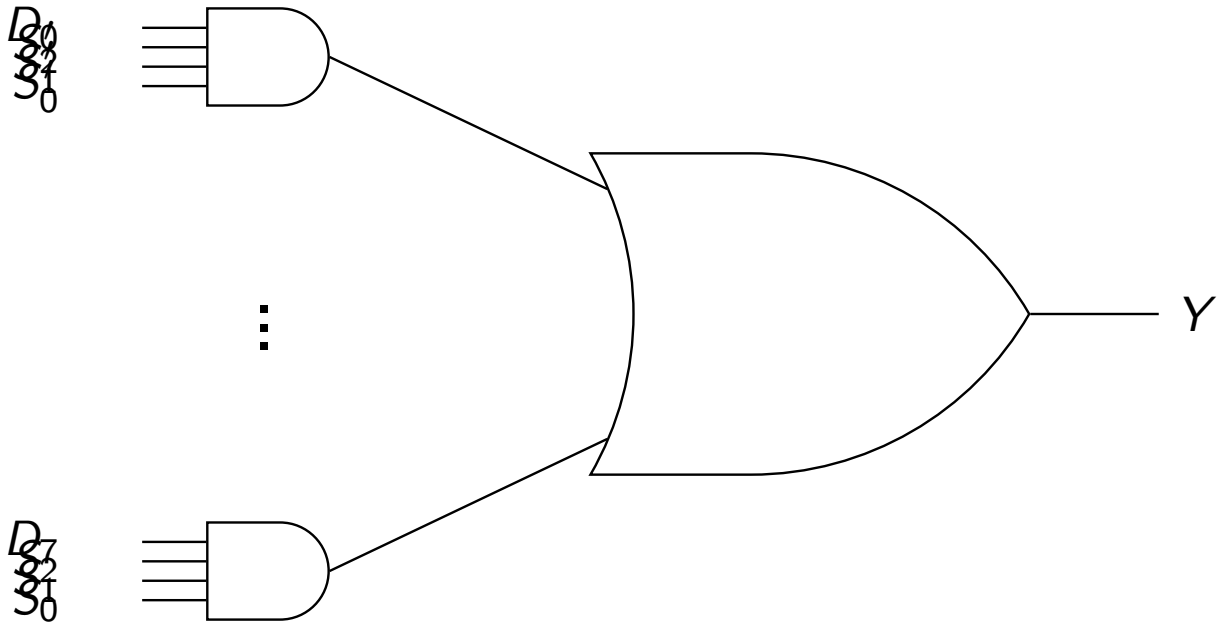
- The Boolean expression is a sum of 8 products.

Boolean Equation

$$Y = S_2' S_1' S_0' D_0 + S_2' S_1' S_0 D_1 + S_2' S_1 S_0' D_2 + S_2' S_1 S_0 D_3 + S_2 S_1' S_0' D_4 + S_2 S_1' S_0 D_5 + S_2 S_1 S_0' D_6 + S_2 S_1 S_0 D_7$$

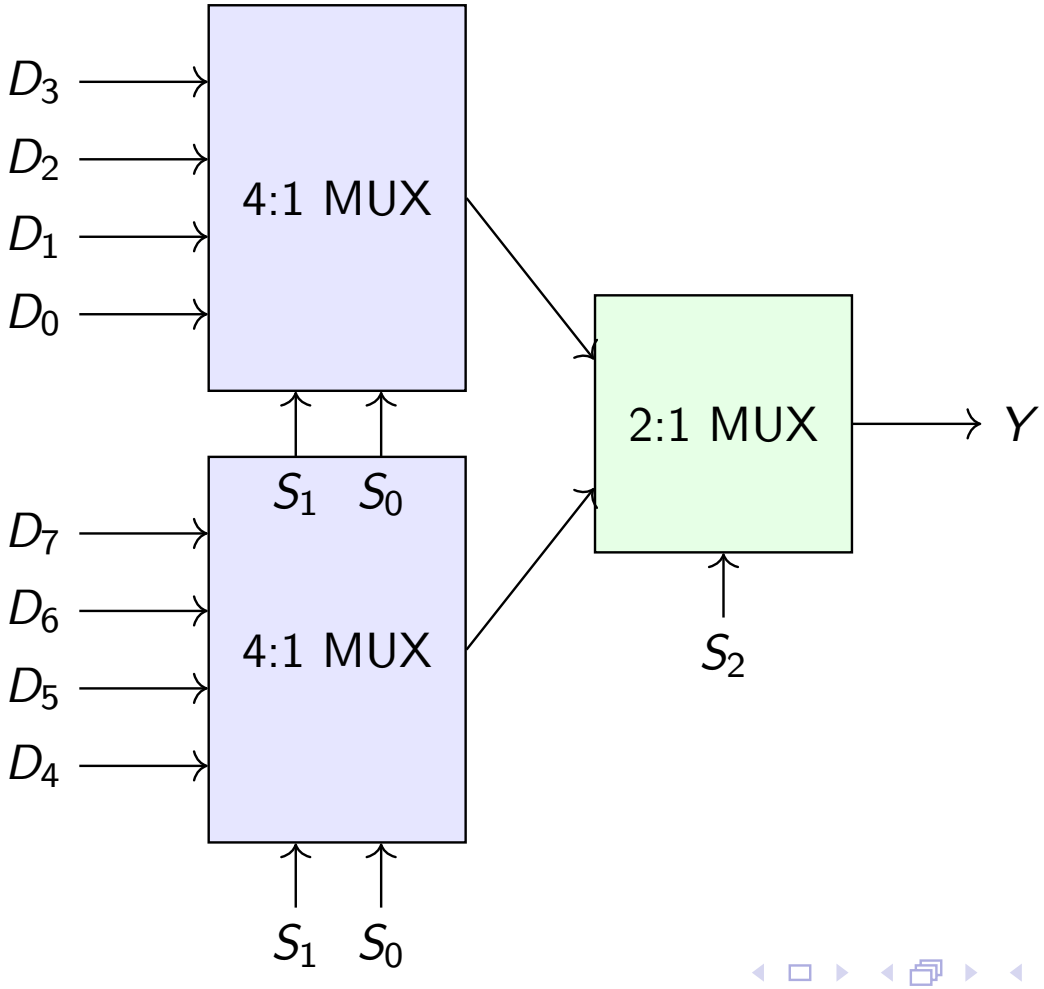
8-to-1 MUX: Logic Circuit

- Requires three NOT gates (for S_2', S_1', S_0').
- Requires eight 4-input AND gates.
- Requires one 8-input OR gate (or cascaded OR gates) for the final output.



Building 8:1 MUX from 4:1 MUXes

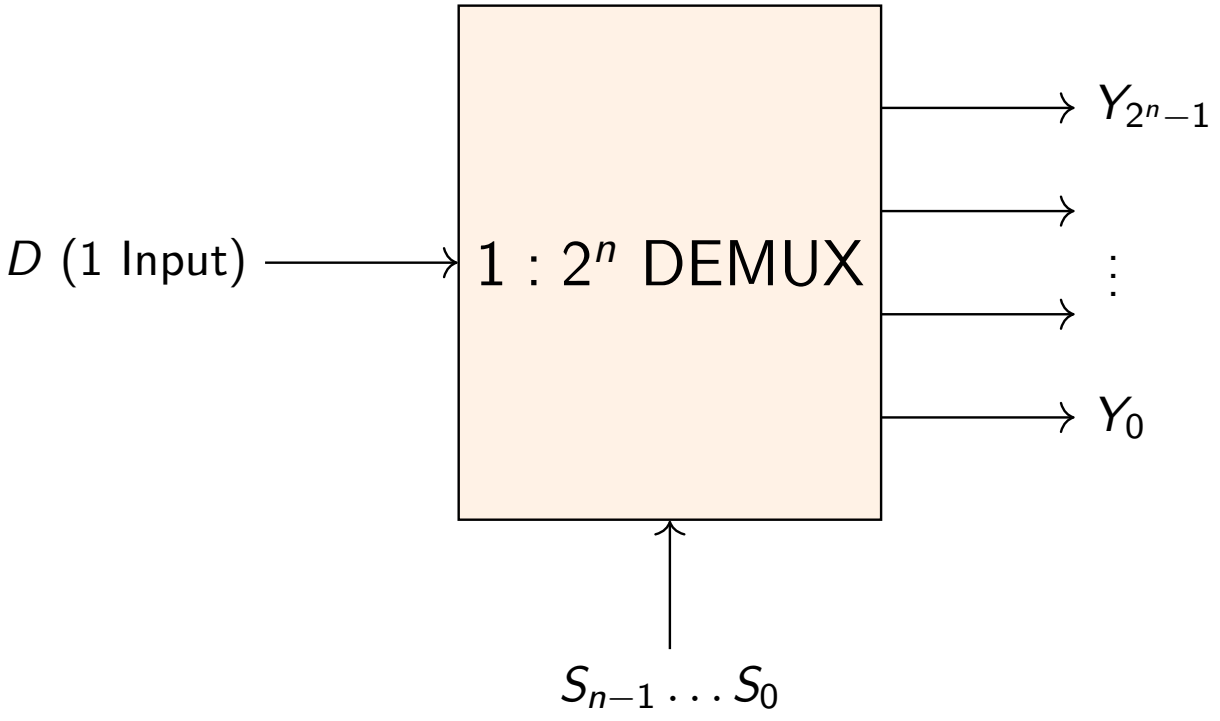
- We can build larger multiplexers using smaller ones.
- To build an 8:1 MUX, use two 4:1 MUXes and one 2:1 MUX.
- S_1 and S_0 select the data within the two 4:1 MUXes.
- S_2 acts as the select line for the final 2:1 MUX.



Introduction to De-multiplexers

What is a De-multiplexer (DEMUX)?

- A DEMUX performs the exact opposite function of a MUX.
- It takes 1 data input line and routes it to one of 2^n output lines.
- It uses n select lines to determine the destination output.
- Also known as a 'Data Distributor'.



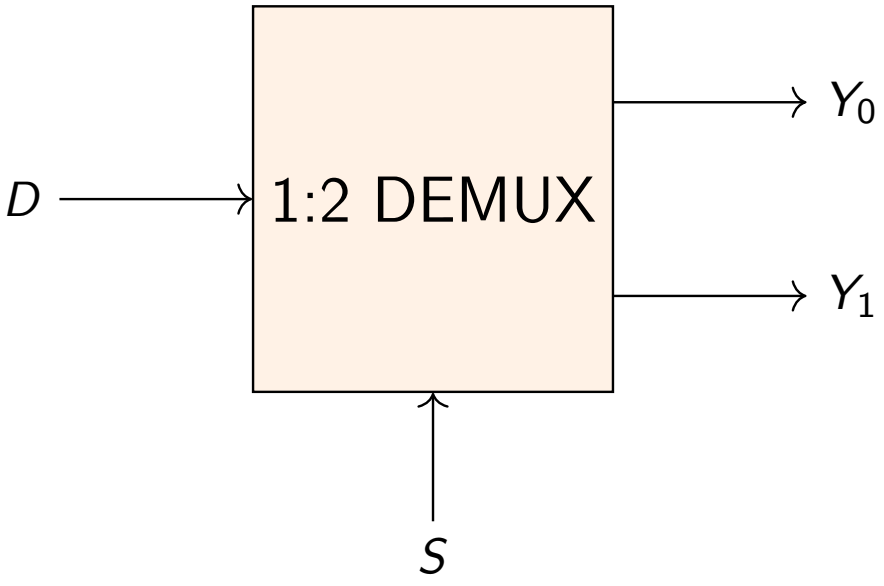
MUX vs DEMUX in Communication

- **MUX (Sender side):** Takes multiple parallel data channels and sends them one by one over a single serial transmission wire.
- **DEMUX (Receiver side):** Takes the serial transmission wire and separates the data back into the original parallel channels.

1-to-2 De-multiplexer

1-to-2 De-multiplexer

- Data Input: 1 line (D).
- Select Line: 1 line (S).
- Outputs: 2 lines (Y_0, Y_1).
- If S=0, Data D is routed to Y_0 (Y_1 remains 0).
- If S=1, Data D is routed to Y_1 (Y_0 remains 0).



1-to-2 DEMUX: Truth Table

- Truth table for 1:2 DEMUX showing D mapping to the active Y output.

Select S	**Data** D	Y_0	Y_1	**Remarks**
0	0	0	0	
0	1	1	0	(D routed to Y_0)
1	0	0	0	
1	1	0	1	(D routed to Y_1)

1-to-2 DEMUX: Logic Equations

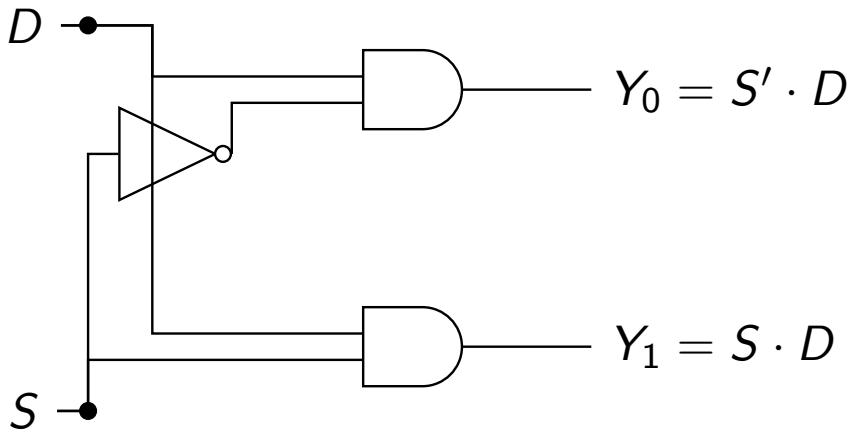
- We need an equation for every output.
- Y_0 is active only when S is 0 and D is 1.
- Y_1 is active only when S is 1 and D is 1.

Equations

$$Y_0 = S' \cdot D$$
$$Y_1 = S \cdot D$$

1-to-2 DEMUX: Logic Circuit

- Implemented using one NOT gate (for S').
- Requires two 2-input AND gates.
- Data (D) is connected to one leg of both AND gates.
- Select (S) enables only one AND gate at a time.



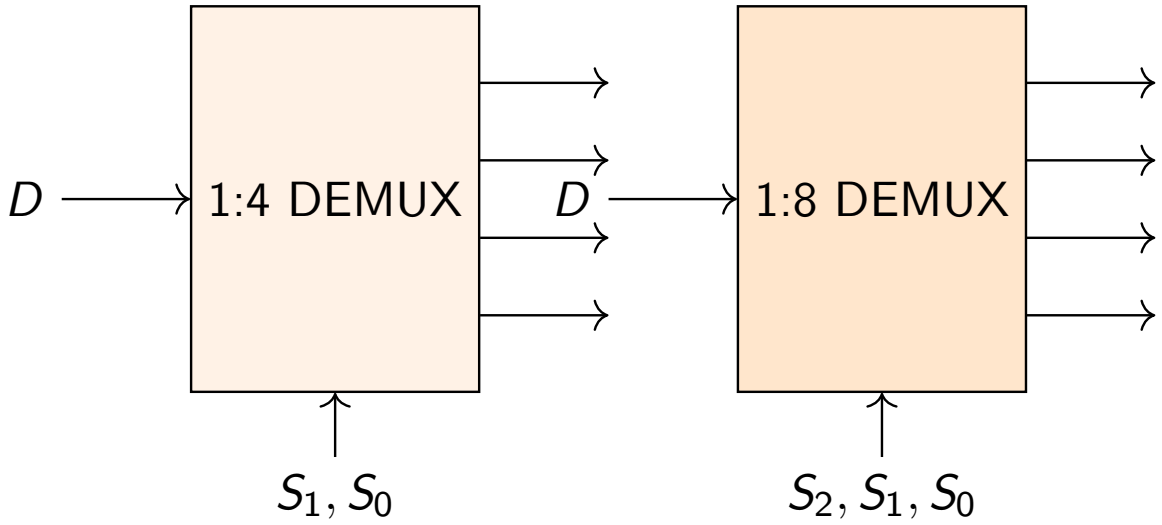
Summary & Preview

Summary

- 8:1 MUX routes 8 inputs to 1 output using 3 select lines.
- A DEMUX distributes 1 input to 2^n outputs using n select lines.
- MUX is a data selector; DEMUX is a data distributor.
- 1:2 DEMUX routes data D to Y_0 or Y_1 based on select line S.

Next Lecture Preview

- Topic: De-multiplexers (1:4, 1:8).
- We will scale up the De-multiplexer to handle 4 and 8 output destinations.
- We will also explore the deep relationship between Decoders and De-multiplexers.



Questions?

Lecture 3.8: De-multiplexers (1:4 and 1:8)

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

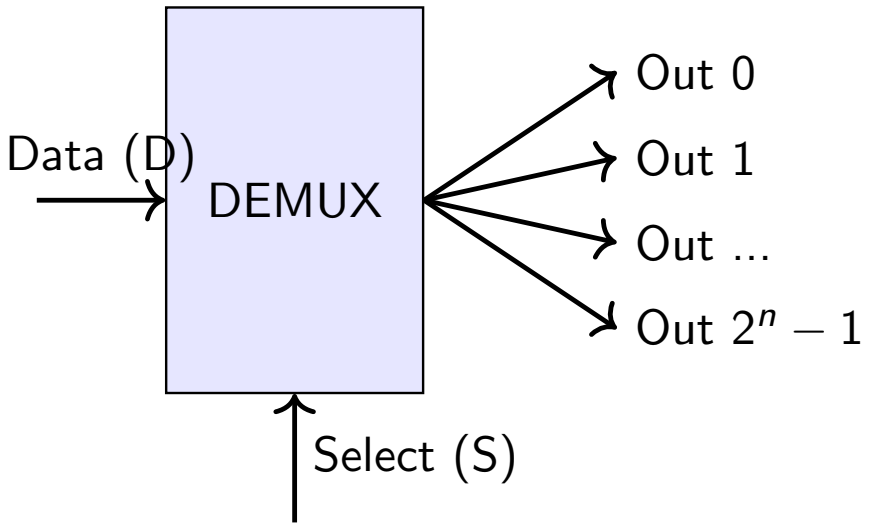
Table of Contents

Agenda

- Review of DEMUX Concept
- 1-to-4 De-multiplexer: Concept and Truth Table
- 1-to-4 DEMUX: Logic Equations and Circuit
- 1-to-8 De-multiplexer Overview
- 1-to-8 DEMUX Logic Circuit
- The Decoder/DEMUX Equivalence
- Summary of MUX and DEMUX

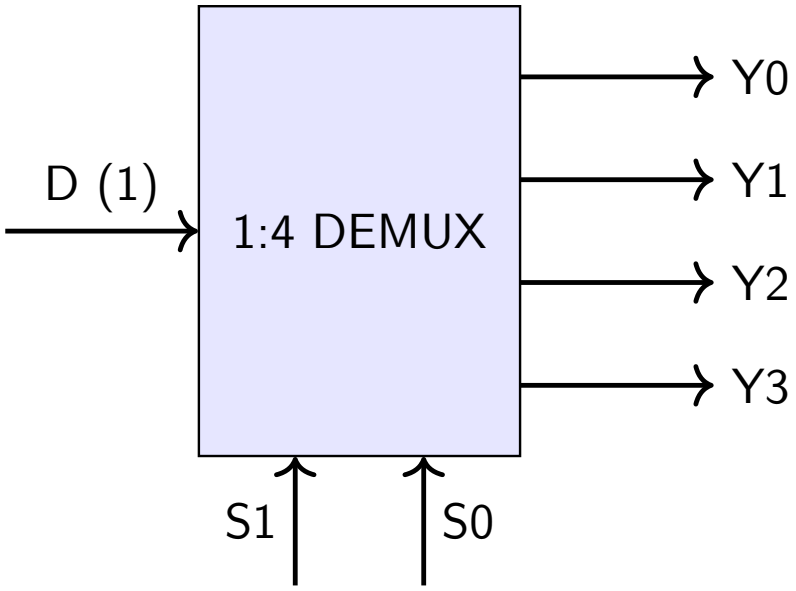
Review of the DEMUX

- Takes 1 Data input (D).
- Uses n Select lines.
- Distributes the Data input to one of 2^n Outputs.
- Unselected outputs remain at Logic 0.



1-to-4 De-multiplexer

- Data Input: 1 line (D).
- Select Lines: 2 lines (S1, S0) because $2^2 = 4$.
- Outputs: 4 lines (Y0, Y1, Y2, Y3).
- The 2-bit select code decides which of the 4 outputs gets the data D.



1-to-4 DEMUX: Truth Table

S1	S0	Y0	Y1	Y2	Y3
0	0	**D**	0	0	0
0	1	0	**D**	0	0
1	0	0	0	**D**	0
1	1	0	0	0	**D**

- 'D' shifts diagonally across the outputs depending on S1 and S0.

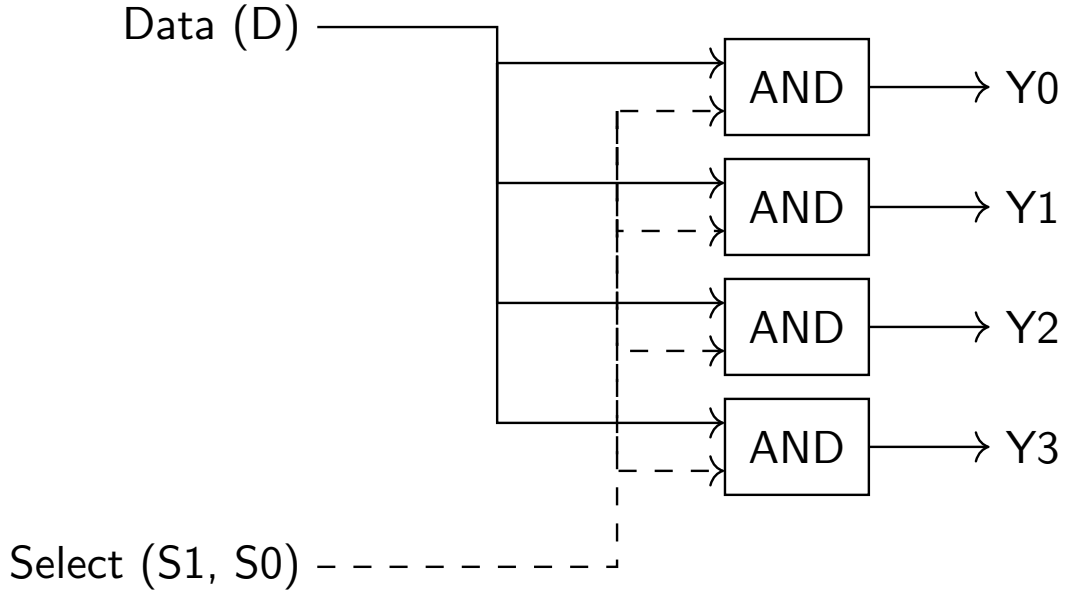
1-to-4 DEMUX: Logic Equations

- We derive an equation for every output based on when it receives D.
- $Y0 = S1' \cdot S0' \cdot D$
- $Y1 = S1' \cdot S0 \cdot D$
- $Y2 = S1 \cdot S0' \cdot D$
- $Y3 = S1 \cdot S0 \cdot D$

Equations derived from S1, S0 minterms ANDed with D.
Each equation represents a path enabling D.

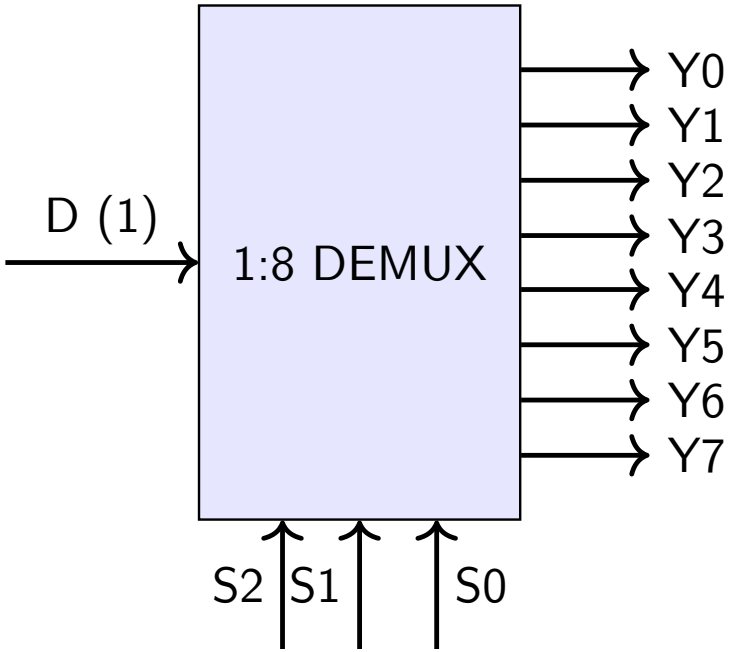
1-to-4 DEMUX: Logic Circuit

- Requires two NOT gates (for S1', S0').
- Requires four 3-input AND gates.
- The Data (D) line connects to all four AND gates.
- The Select lines ensure only one AND gate is fully 'enabled' to pass D.



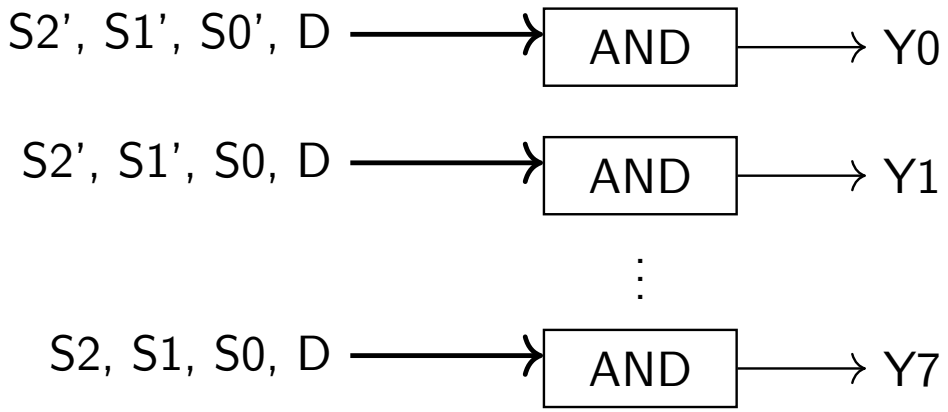
1-to-8 De-multiplexer

- Data Input: 1 line (D).
- Select Lines: 3 lines (S2, S1, S0).
- Outputs: 8 lines (Y0 through Y7).
- If S2 S1 S0 = 1 1 0 (Decimal 6), then Y6 = D, and all other Y = 0.



1-to-8 DEMUX: Logic Implementation

- $Y0 = S2'S1'S0'D$
- $Y1 = S2'S1'S0D$
- ...
- $Y7 = S2S1S0D$
- Requires eight 4-input AND gates.



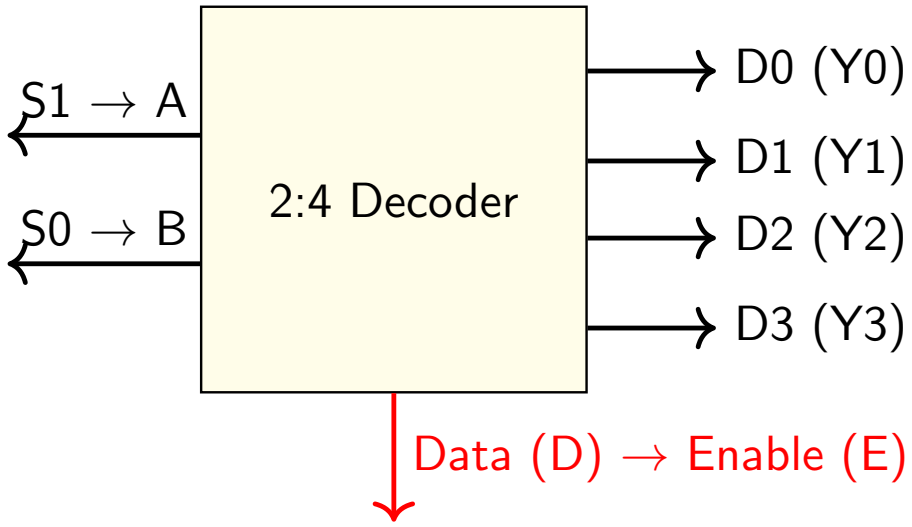
The Decoder / DEMUX Equivalence

- Look closely at a 2-to-4 Decoder with an Enable (E) pin.
- Decoder outputs: $D0 = A'B'E$
- Look at a 1-to-4 DEMUX.
- DEMUX outputs: $Y0 = S1'S0'D$
- They are structurally identical!

Decoder: $D0 = A' \cdot B' \cdot E$ ⟷ Exact Match **DEMUX:** $Y0 = S1' \cdot S0' \cdot D$

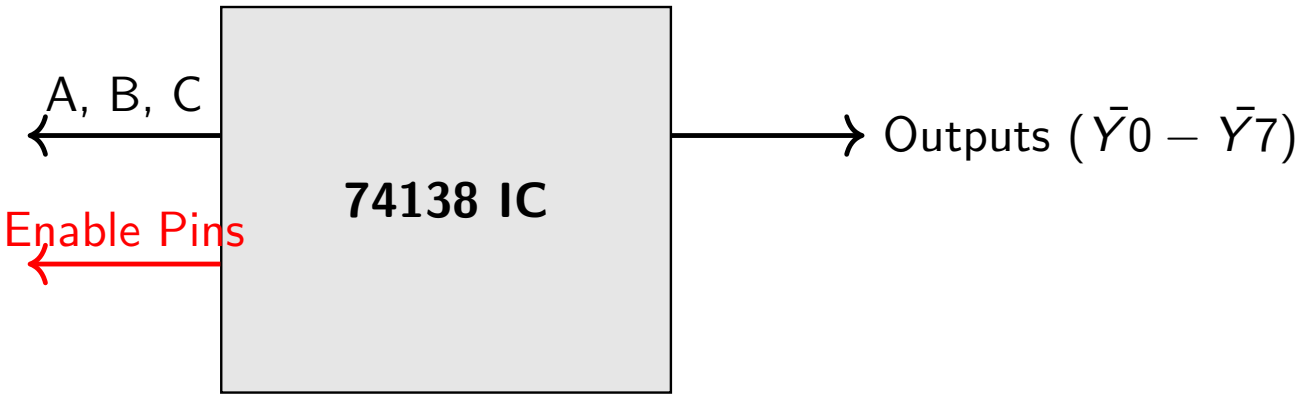
How to Use a Decoder as a DEMUX

- Take a standard 2-to-4 Decoder with Enable.
- Connect the DEMUX 'Select Lines' to the Decoder 'Input lines' (A, B).
- Connect the DEMUX 'Data Line' to the Decoder 'Enable pin' (E).
- The chip now functions perfectly as a De-multiplexer.



IC 74138: 3-to-8 Line Decoder/Demultiplexer

- A commercially available TTL chip.
- Has 3 select inputs and 8 active-low outputs.
- Has 3 Enable pins for flexibility in cascading.
- Can be used as a 3:8 decoder OR a 1:8 DEMUX.



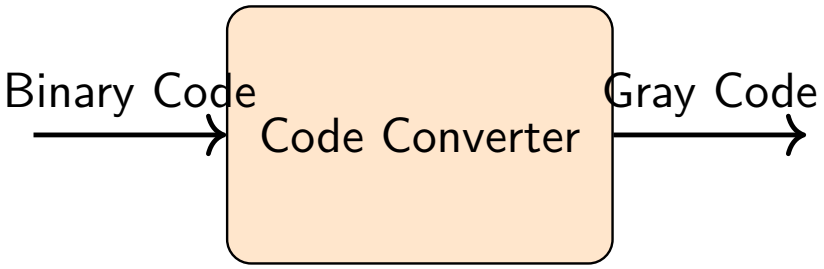
Summary of MUX / DEMUX

- **MUX (Multiplexer):** Many inputs to One output. Used for data selection.
- **DEMUX (De-multiplexer):** One input to Many outputs. Used for data distribution.
- A DEMUX circuit is identical to a Decoder with an Enable pin.
- Select lines dictate the data path in both MUX and DEMUX.

Device	**Inputs**	**Outputs**	**Function**
MUX	2^n	1	Data Selector
DEMUX	1	2^n	Data Distributor
Decoder	n	2^n	Minterm Generator

Next Lecture Preview

- Topic: Binary to Gray Code Converters (up to 4 bits).
- We move to the final topic of Combinational Logic: Code Converters.
- We will learn what Gray code is and design a circuit to convert standard binary into Gray code.



Questions?

Lecture 3.9: Binary to Gray Code Converter

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

Lecture 3.9: Binary to Gray Code Converter

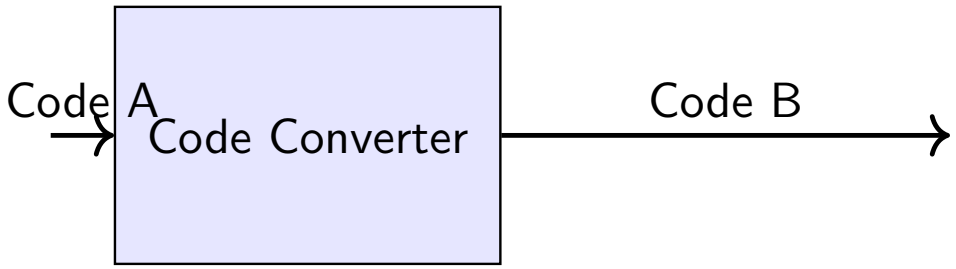
Agenda

- What is a Code Converter?
- Introduction to Gray Code
- Why use Gray Code? (Applications)
- Manual Conversion: Binary to Gray
- Truth Table for 4-bit Converter
- K-Map Simplification
- Logic Circuit Implementation

Focus: 4-bit Binary to Gray Code Conversion

What is a Code Converter?

- A combinational circuit translating data from one binary code to another.
- Inputs represent data in code A, outputs in code B.
- Examples: Binary to Gray, BCD to Excess-3, BCD to 7-Segment.



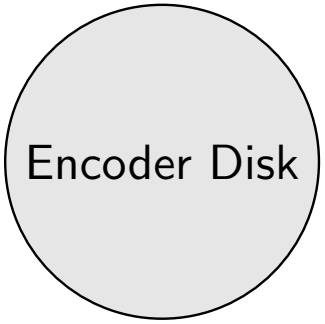
Introduction to Gray Code

- Also known as a 'Reflected Binary Code'.
- Key Property: Only ONE bit changes state from one number to the next sequential number.
- It is an unweighted code (the bits do not have strict place values like 8-4-2-1).

Binary (011 $\rightarrow$ 100)	**Gray (010 $\rightarrow$ 110)**
3 bits change!	1 bit changes!

Why Use Gray Code?

- Reduces switching errors in electromechanical and optical sensors.
- Absolute Encoders (Rotary Encoders): If a physical wheel stops exactly on a boundary, a multi-bit change in binary can cause massive read errors.
- In Gray code, a boundary error only ever causes a 1-position error.
- Also used in K-maps to ensure adjacency!

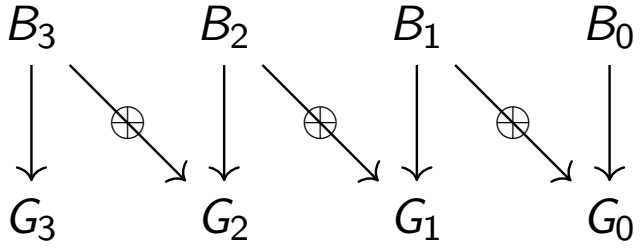


Eliminates multi-bit transition errors

Lecture 3.9: Binary to Gray Code Converter

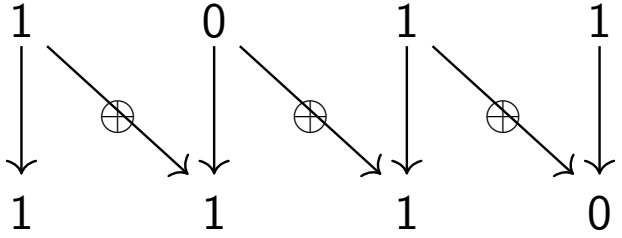
Manual Conversion: Binary to Gray

- Let Binary $= B_3 B_2 B_1 B_0$ and Gray $= G_3 G_2 G_1 G_0$.
- Rule 1: MSB is the same. $G_3 = B_3$.
- Rule 2: $G_2 = B_3 \oplus B_2$ (XOR adjacent binary bits).
- Rule 3: $G_1 = B_2 \oplus B_1$.
- Rule 4: $G_0 = B_1 \oplus B_0$.



Conversion Example

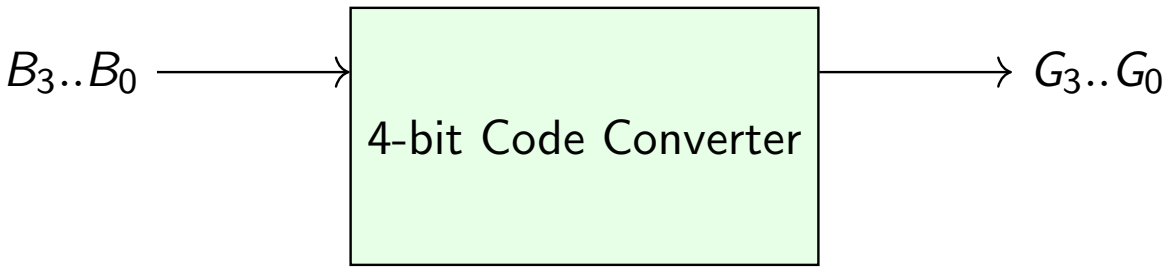
- Convert Binary 1011 to Gray code.
- $B_3 = 1, B_2 = 0, B_1 = 1, B_0 = 1$
- $G_3 = B_3 = 1$
- $G_2 = 1 \oplus 0 = 1$
- $G_1 = 0 \oplus 1 = 1$
- $G_0 = 1 \oplus 1 = 0$
- Result: 1011 (Binary) = 1110 (Gray).



Lecture 3.9: Binary to Gray Code Converter

Designing the 4-bit Converter

- We need a combinational circuit with 4 inputs (B_3, B_2, B_1, B_0) and 4 outputs (G_3, G_2, G_1, G_0).
- Step 1: Create a 16-row Truth Table.
- Step 2: Use four separate 4-variable K-maps to simplify equations for G_3, G_2, G_1, and G_0.
- Step 3: Draw the logic circuit.



Truth Table (First 8 Rows)

Decimal	B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0

K-Map Simplification (G_3 and G_2)

- For G_3: Plotting minterms yields $G_3 = B_3$.
- For G_2: Plotting minterms yields two groups of 4. Equation simplifies to: $G_2 = B_3' B_2 + B_3 B_2' = B_3 \oplus B_2$.

G_3 Map

$B_3B_2 \backslash B_1B_0$	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

G_2 Map

$B_3B_2 \backslash B_1B_0$	00	01	11
00	0	0	0
01	1	1	1
11	0	0	0
10	1	1	1

K-Map Simplification (G_1 and G_0)

- For G_1: Plotting minterms yields two groups of 4. Equation simplifies to: $G_1 = B_2'B_1 + B_2B_1' = B_2 \oplus B_1$.
- For G_0: Plotting minterms yields two groups of 4. Equation simplifies to: $G_0 = B_1'B_0 + B_1B_0' = B_1 \oplus B_0$.

G_1 Map

$B_3B_2 \backslash B_1B_0$	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	1	1	0	0
10	0	0	1	1

G_0 Map

$B_3B_2 \backslash B_1B_0$	00	01	11
00	0	1	0
01	0	1	0
11	0	1	0
10	0	1	0

Lecture 3.9: Binary to Gray Code Converter

Logic Equations Summary

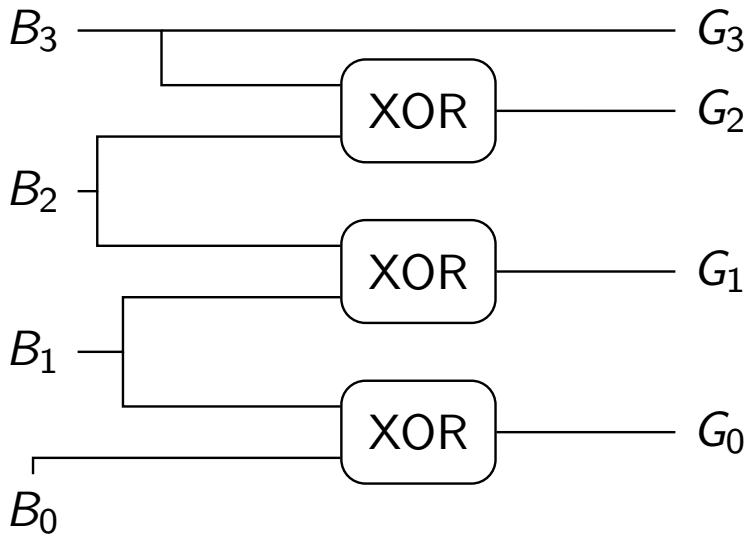
Final Logic Equations

- $G_3 = B_3$
- $G_2 = B_3 \oplus B_2$
- $G_1 = B_2 \oplus B_1$
- $G_0 = B_1 \oplus B_0$

We can implement this entirely using just three 2-input XOR gates!

Logic Circuit Implementation

- Line B_3 connects directly to output G_3.
- An XOR gate takes B_3 and B_2 to output G_2.
- An XOR gate takes B_2 and B_1 to output G_1.
- An XOR gate takes B_1 and B_0 to output G_0.



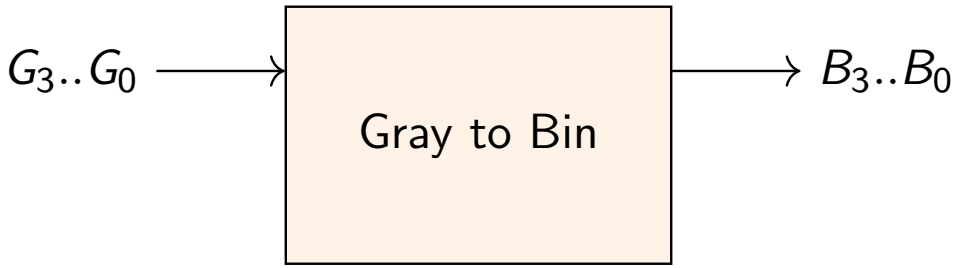
Summary

- Gray code ensures only one bit changes per sequential increment.
- It minimizes errors in physical and optical sensors.
- Binary to Gray conversion is achieved by XORing adjacent binary bits.
- The hardware requires just three XOR gates for a 4-bit converter.

XOR Logic

Next Lecture Preview

- Topic: Gray to Binary Code Converter (up to 4 bits).
- We will reverse the process we learned today.
- We will learn the rules for converting Gray back to Binary and design the circuit.



Questions?

Lecture 3.10: Gray to Binary Code Converter

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

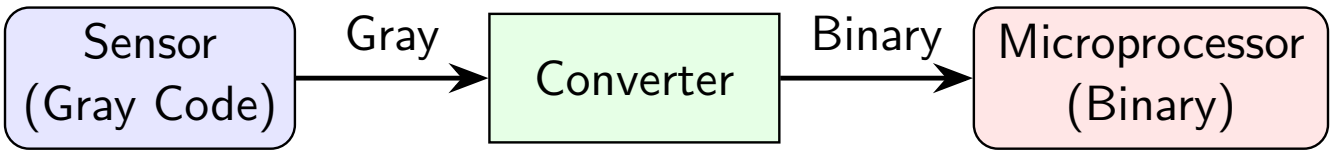
Table of Contents

Agenda

- Review of Gray Code Applications
- Manual Conversion: Gray to Binary
- Step-by-step Conversion Example
- Truth Table formulation
- K-Map Simplification
- Logic Equations for Gray to Binary
- Logic Circuit Implementation

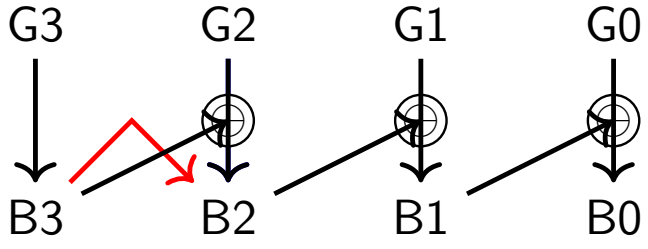
Why Convert Gray back to Binary?

- Gray code is excellent for physical sensors (like robotic arms or CNC machines) to prevent transition errors.
- However, microprocessors cannot easily perform math (addition, subtraction) on Gray code.
- Therefore, sensor data arrives in Gray code and must immediately be converted to Binary before the CPU processes it.



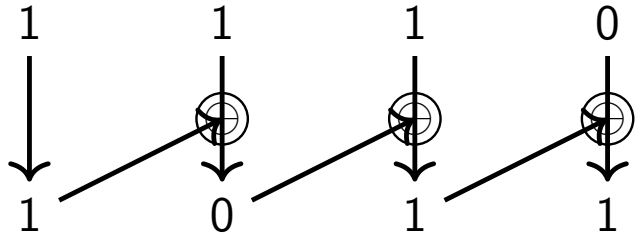
Manual Conversion: Gray to Binary

- Let Gray = G3 G2 G1 G0 and Binary = B3 B2 B1 B0.
- Rule 1: MSB is the same. B3 = G3.
- Rule 2: To get B2, XOR the previous binary bit (B3) with the current Gray bit (G2). B2 = B3 $\oplus$ G2.
- Rule 3: B1 = B2 $\oplus$ G1.
- Rule 4: B0 = B1 $\oplus$ G0.



Conversion Example

- Convert Gray code 1110 to Binary.
- G3 = 1, G2 = 1, G1 = 1, G0 = 0
- B3 = G3 = 1
- B2 = B3 $\oplus$ G2 = 1 $\oplus$ 1 = 0
- B1 = B2 $\oplus$ G1 = 0 $\oplus$ 1 = 1
- B0 = B1 $\oplus$ G0 = 1 $\oplus$ 0 = 1
- Result: 1110 (Gray) = 1011 (Binary).



Formulating the Truth Table

- We need a 16-row Truth Table.
- Inputs: G3, G2, G1, G0.
- Outputs: B3, B2, B1, B0.
- We can fill this table either by manually converting all 16 rows, or by sorting our previous lecture's table by the Gray column.

Gray Code Inputs				Binary Outputs			
G3	G2	G1	G0	B3	B2	B1	B0
...	...	...	...	...	...	...	...

Truth Table (Sample Rows)

- G3 G2 G1 G0 | B3 B2 B1 B0
- 0 0 0 0 | 0 0 0 0
- 0 0 0 1 | 0 0 0 1
- 0 0 1 1 | 0 0 1 0
- 0 0 1 0 | 0 0 1 1
- 0 1 1 0 | 0 1 0 0

Gray Code				Binary			
G3	G2	G1	G0	B3	B2	B1	B0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0

K-Map Simplification (B3 and B2)

- Plotting B3 on a 4-variable K-map:
- Groups perfectly to B3 = G3.

- Plotting B2 on a 4-variable K-map:
- Creates a checkerboard pattern of two groups.
- Simplifies to B2 = G3'G2 + G3G2' = G3 $\oplus$ G2.

B3 K-map

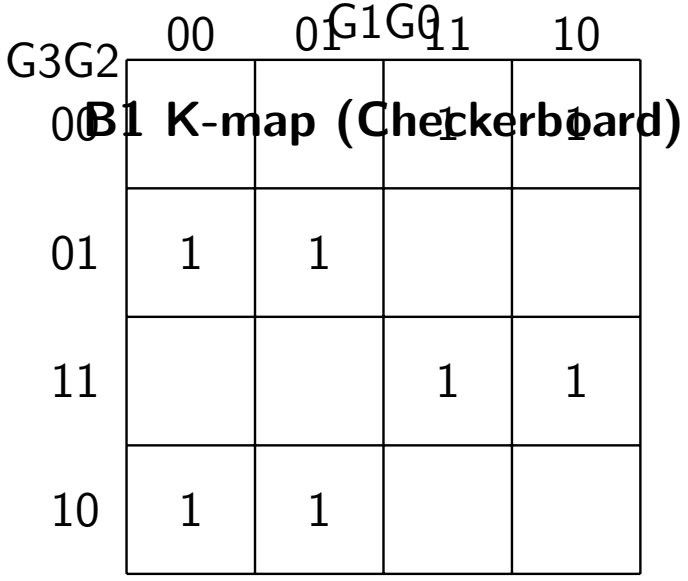
G3G2 \ G1G0	00	01	11	10
00				
01				
11	1	1	1	1
10	1	1	1	1

B2 K-map

G3G2 \ G1G0	00	01	11	10
00				
01	1	1	1	1
11				
10	1	1	1	1

K-Map Simplification (B1)

- Plotting B1 on a 4-variable K-map:
- The 1s are scattered in a complex checkerboard pattern.
- Simplifies to B1 = G3 $\oplus$ G2 $\oplus$ G1.
- Notice that since B2 = G3 $\oplus$ G2, we can rewrite this as: B1 = B2 $\oplus$ G1.



B1 K-map (Checkerboard)

K-Map Simplification (B0)

- Plotting B0 on a 4-variable K-map:
- Yields a full checkerboard of eight 1s.
- Simplifies to $B0 = G3 \oplus G2 \oplus G1 \oplus G0$.
- We can rewrite this as: $B0 = B1 \oplus G0$.

B0 K-map (Full Checkerboard)

G3G2 \ G1G0	00	01	11	10
00		1		1
01	1		1	
11		1		1
10	1		1	

Logic Equations Summary

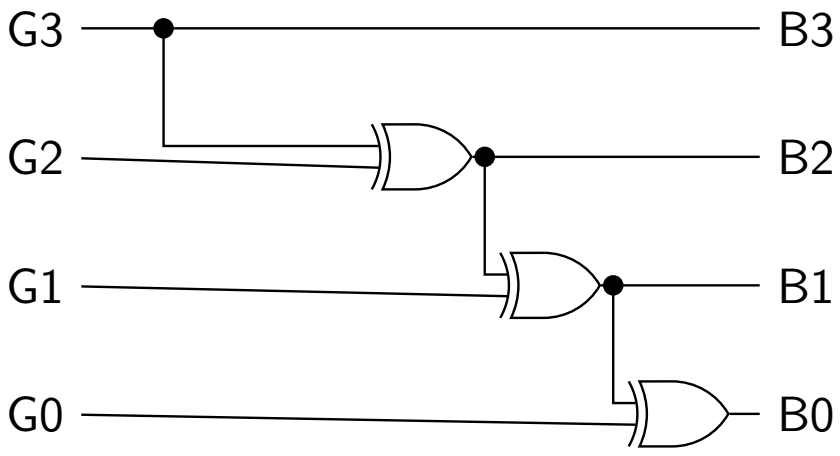
- Hardware optimized equations:
- B3 = G3
- B2 = B3 $\oplus$ G2
- B1 = B2 $\oplus$ G1
- B0 = B1 $\oplus$ G0

Optimized Equations

$$B_3 = G_3$$
$$B_2 = B_3 \oplus G_2$$
$$B_1 = B_2 \oplus G_1$$
$$B_0 = B_1 \oplus G_0$$

Logic Circuit Implementation

- G3 connects straight to B3.
- An XOR gate takes G3 and G2 to produce B2.
- An XOR gate takes the new B2 and G1 to produce B1.
- An XOR gate takes the new B1 and G0 to produce B0.



Comparing the Converters

- Binary to Gray: All XOR gates operate in parallel. Very fast.
- Gray to Binary: XOR gates are cascaded. The signal must ripple through, making it slightly slower due to propagation delay.
- Both require exactly three XOR gates for 4-bit conversion.

Binary to Gray
Parallel XORs
(Fast)

Gray to Binary
Cascaded XORs
(Slower)

Summary

- Gray to Binary conversion is essential for processing sensor data in standard ALUs.
- The manual rule involves XORing the calculated binary bit with the next Gray bit.
- The hardware implementation uses cascaded XOR gates.
- ✓ This completes our study of Combinational Logic Circuits (Unit 3).

Next Lecture Preview

- Topic: Introduction to Sequential Logic (Unit 4).
- We will move beyond combinational circuits and introduce the concept of 'Memory'.
- We will study Latches and Flip-Flops, the basic building blocks of RAM and Registers.

Questions?

Lecture 4.1: Flip-flops: SR and D Flip-flops

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

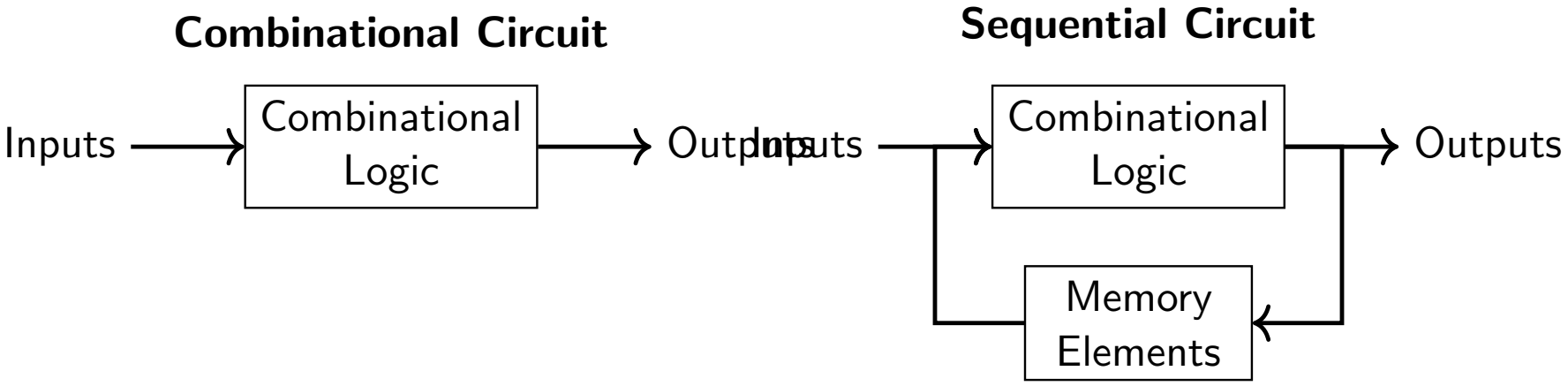
Lecture 4.1: Flip-flops: SR and D Flip-flops

Agenda

- Combinational vs. Sequential Circuits
- Introduction to Latch vs. Flip-flop
- SR (Set-Reset) Latch and Flip-flop
- SR Flip-flop Truth Table and Excitation Table
- The Invalid State Problem in SR
- D (Data/Delay) Flip-flop Introduction
- D Flip-flop Working and Truth Table

Combinational vs. Sequential Circuits

- Combinational: Output depends purely on the current inputs. No memory.
- Sequential: Output depends on current inputs AND previous outputs (past state).
- Sequential circuits require memory elements (like flip-flops) to store past states.
- Feedback loops are an essential characteristic of sequential circuits.



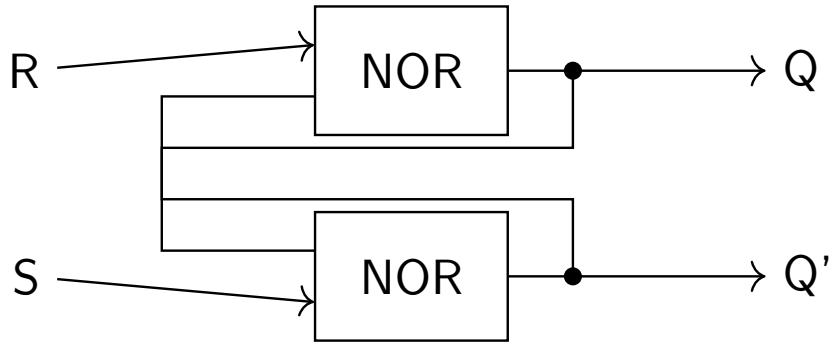
Latches vs. Flip-flops

- Both are basic 1-bit memory elements.
- Latch: Level-triggered. Changes state as long as the enable signal is at an active level (high or low).
- Flip-flop: Edge-triggered. Changes state only at a specific edge of the clock signal (rising or falling).
- Flip-flops provide more reliable synchronous operation in complex circuits.

Lecture 4.1: Flip-flops: SR and D Flip-flops

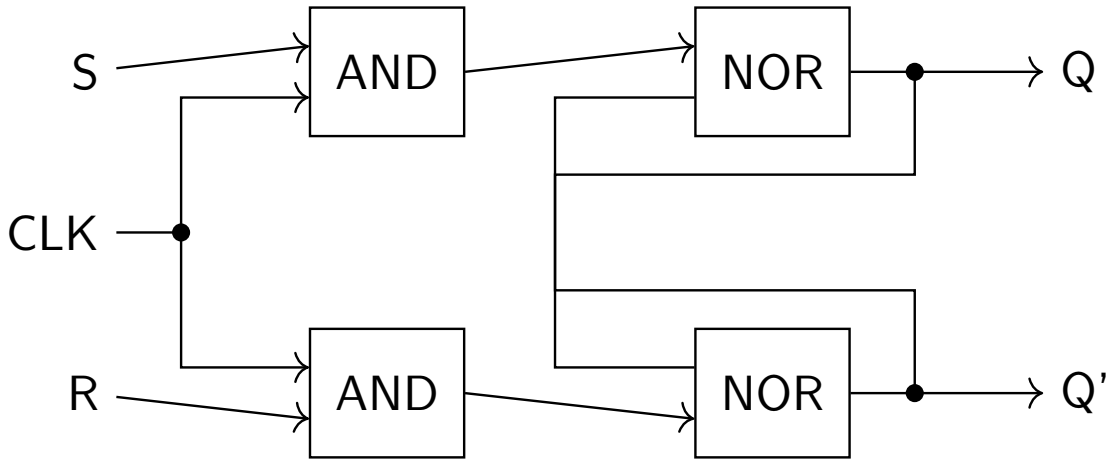
The SR Latch (Active High)

- SR stands for Set-Reset.
- Constructed using two cross-coupled NOR gates.
- S=1, R=0: Sets the output Q to 1.
- S=0, R=1: Resets the output Q to 0.
- S=0, R=0: No change (memory state, Q remains previous Q).
- S=1, R=1: Invalid state (both outputs become 0, contradicting Q and Q').



Clocked SR Flip-flop

- By adding a clock (or enable) signal via AND/NAND gates to the basic latch, we create a synchronous SR flip-flop.
- The S and R inputs are only evaluated when the Clock (CLK) is active.
- When CLK $= 0$, the flip-flop holds its state regardless of S and R.
- When CLK $= 1$, it behaves like the standard SR latch.



SR Flip-flop: Truth Table

CLK	S	R	Q(n+1)	State
0	X	X	Q(n)	No Change
1	0	0	Q(n)	No Change
1	0	1	0	Reset
1	1	0	1	Set
1	1	1	?	Invalid

Excitation Table of SR Flip-flop

- Excitation table tells us: What inputs do I need to get a specific state transition?

Q(n)	**Q(n+1)**	**S**	**R**
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

The S=R=1 Problem

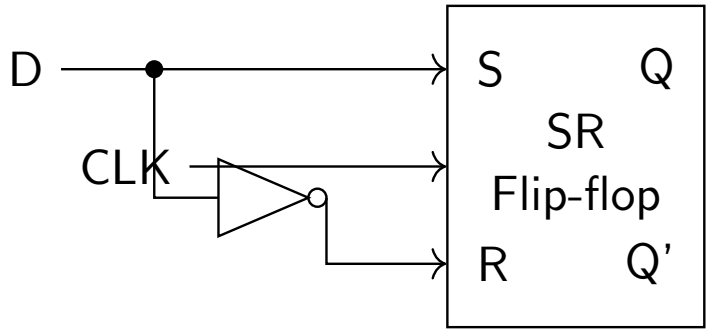
- In an SR flip-flop, asserting both Set and Reset simultaneously leads to an unpredictable/invalid state.
- Outputs Q and Q' might both become the same value, violating the principle that they are complements.
- If S and R both drop back to 0 simultaneously, the final state is a race condition.
- Solution: Ensure S and R are never 1 at the same time.

CLK	S	R	Q(n+1)	State
1	1	1	?	Invalid / Race Condition

Lecture 4.1: Flip-flops: SR and D Flip-flops

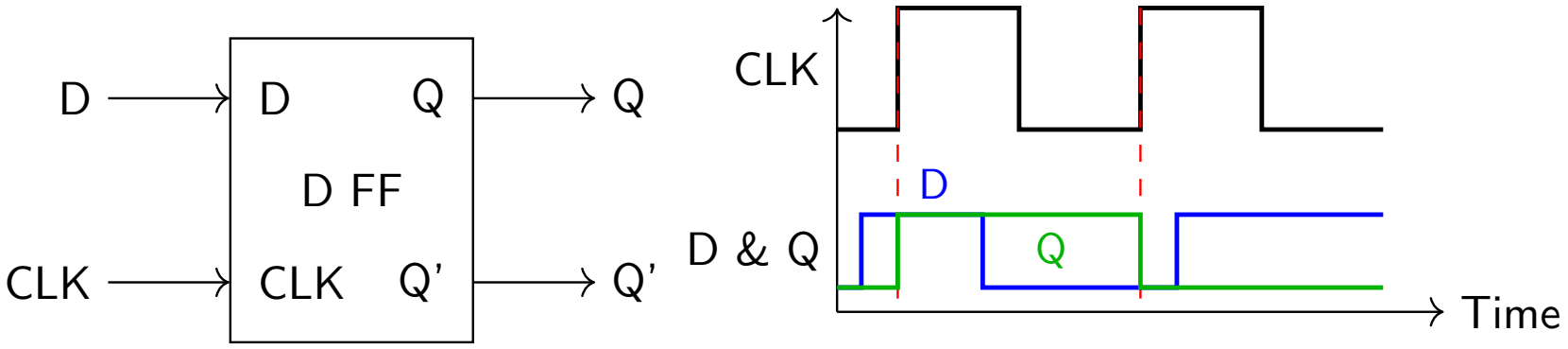
Introduction to the D Flip-flop

- D stands for Data or Delay.
- Created from an SR flip-flop by connecting the S input directly to D, and the R input to D through an inverter (NOT gate).
- This guarantees that S and R are always complements (if S=1, R=0; if S=0, R=1).
- Eliminates the S=R=1 invalid state completely.



Working of the D Flip-flop

- When CLK = 1 (or at clock edge):
- If D = 1 → S=1, R=0 → Flip-flop Sets (Q = 1).
- If D = 0 → S=0, R=1 → Flip-flop Resets (Q = 0).
- The output Q simply follows the input D after a clock pulse.
- Often called a 'Transparent Latch' (if level-triggered) or 'Delay Flip-flop'.



D Flip-flop: Truth Table and Excitation

Truth Table

CLK	D	Q(n+1)	State
0	X	Q(n)	No Change
1	0	0	Reset
1	1	1	Set

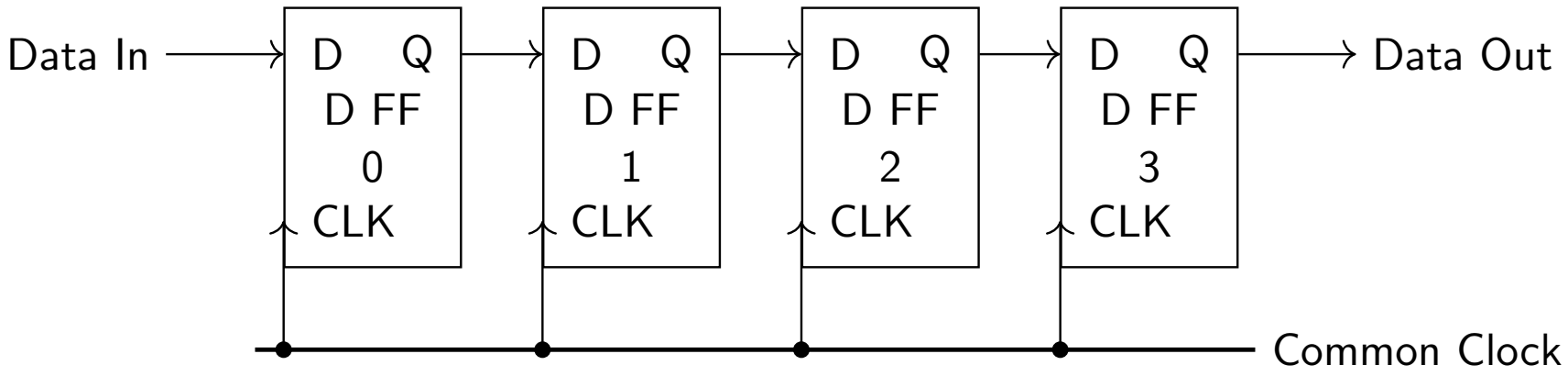
Excitation Table

Q(n)	Q(n+1)	D
0	0	0
0	1	1
1	0	0
1	1	1

Lecture 4.1: Flip-flops: SR and D Flip-flops

Applications of D Flip-flops

- Data storage: The fundamental building block of shift registers and CPU registers.
- Delay lines: Introducing exactly one clock cycle of delay to a signal.
- Synchronizers: Aligning asynchronous external inputs to the internal system clock.
- State machines: Extremely common in modern digital design due to simplicity.



Summary

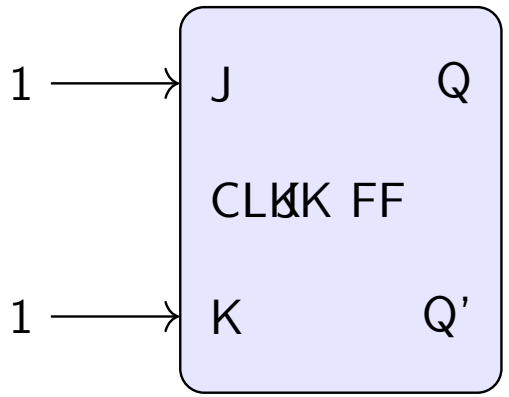
- Sequential circuits use memory to base outputs on both current inputs and past states.
- SR flip-flops can Set (1) or Reset (0), but suffer from an invalid state when S=R=1.
- D flip-flops solve this by feeding D and D' to the S and R inputs, respectively.
- D flip-flops simply delay the input data by one clock cycle, making them ideal for registers.

SR Flip-flop
S=1, R=1 is Invalid

D Flip-flop
Safe & Reliable

Next Lecture Preview

- What happens if we want to toggle the state instead of just setting or resetting?
- Introduction to JK Flip-flops: Solving the SR invalid state differently.
- The T (Toggle) Flip-flop and its role in counters.
- Review today's flip-flop basics for a smooth transition.



Toggle State!
Q toggles on every clock edge

Questions?

Lecture 4.2: Flip-flops: JK and T Flip-flops

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

Agenda

- Recap of the SR Invalid State
- Introduction to the JK Flip-flop
- JK Flip-flop Circuit & Working
- JK Truth Table & Excitation Table
- The Race Around Condition
- Introduction to the T (Toggle) Flip-flop
- T Flip-flop Truth & Excitation Table
- Summary & Preview

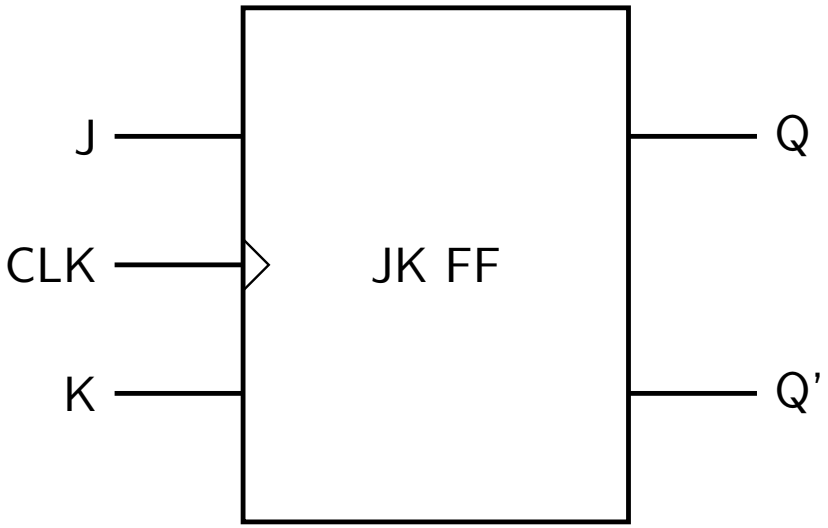
Recap: The Problem with SR

- SR flip-flop has inputs S (Set) and R (Reset).
- When S=1, R=1 simultaneously, the output state becomes invalid.
- Both Q and Q' are driven to the same logic level, violating their complementary nature.
- We need a circuit that safely handles the 1,1 input.

S	R	Q_{n+1}	State
0	0	Q_n	Hold
0	1	0	Reset
1	0	1	Set
1	1	X	Invalid

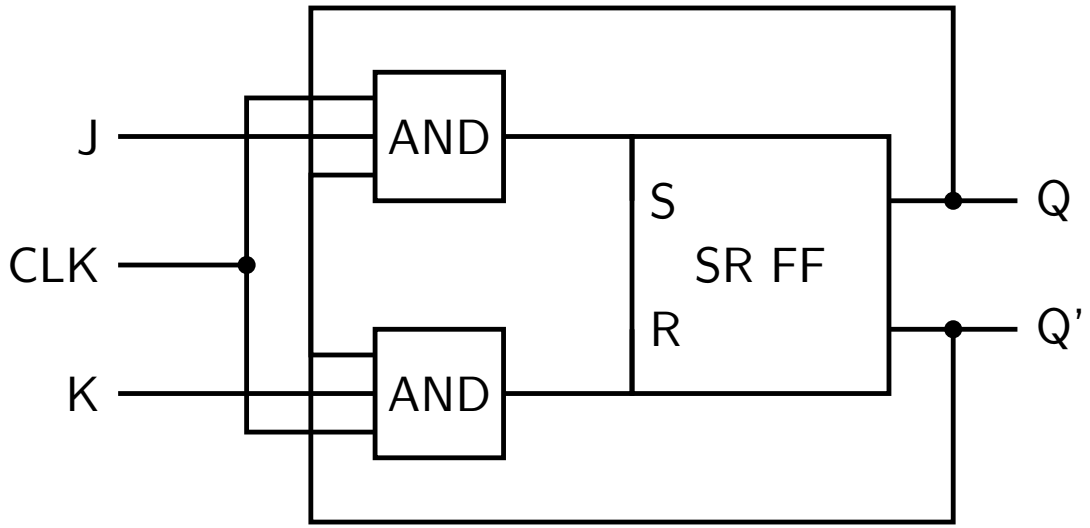
Introduction to JK Flip-flop

- Named after Jack Kilby (inventor of the IC), though historically debatable.
- The JK flip-flop is an enhancement of the SR flip-flop.
- J corresponds to Set (S), and K corresponds to Reset (R).
- The magic: When J=1 and K=1, the flip-flop **TOGGLES** its state instead of becoming invalid.



JK Flip-flop Circuit

- Constructed by adding feedback from the outputs (Q and Q') back to the input AND/NAND gates of an SR flip-flop.
- J input is ANDed with Q' and the clock.
- K input is ANDed with Q and the clock.
- This cross-feedback ensures that if Q=1, only K is enabled; if Q=0, only J is enabled.



JK Flip-flop: Truth Table

- **J — K —— Q(n+1)**
- 0 — 0 —— Q(n) (Hold)
- 0 — 1 —— 0 (Reset)
- 1 — 0 —— 1 (Set)
- 1 — 1 —— Q(n)' (Toggle)

J	**K**	**Q_{n+1}**	**State**
0	0	Q_n	Hold
0	1	0	Reset
1	0	1	Set
1	1	Q_n'	Toggle

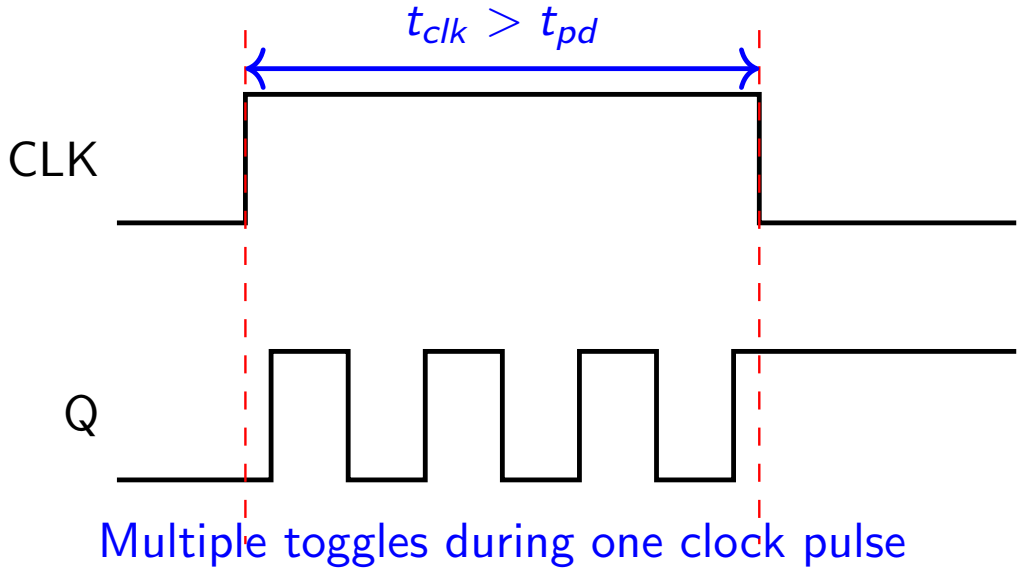
JK Flip-flop: Excitation Table

- **Q(n) $\rightarrow$ Q(n+1) —— J — K**
- 0 $\rightarrow$ 0 —— 0 — X
- 0 $\rightarrow$ 1 —— 1 — X
- 1 $\rightarrow$ 0 —— X — 1
- 1 $\rightarrow$ 1 —— X — 0

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

The Race Around Condition

- Occurs in level-triggered JK flip-flops when J=1, K=1, and the clock pulse (CLK=1) is too long.
- Because of the feedback, the output toggles rapidly: $0 \rightarrow 1 \rightarrow 0 \rightarrow 1 \ldots$
- If the clock stays high longer than the propagation delay of the flip-flop, the final state is unpredictable.
- $t_{clock\_pulse} > t_{propagation\_delay}$ causes this issue.



Solving the Race Around Condition

- Method 1: Make the clock pulse shorter than the propagation delay (impractical and hard to control reliably).
- Method 2: Use Edge-Triggering (only evaluating inputs exactly at the rising or falling edge of the clock).
- Method 3: Use a Master-Slave configuration (the most robust hardware solution).

Method 1: Control Pulse Width

Make $t_{clk} < t_{pd}$ (Impractical and unreliable)

Method 2: Edge-Triggering

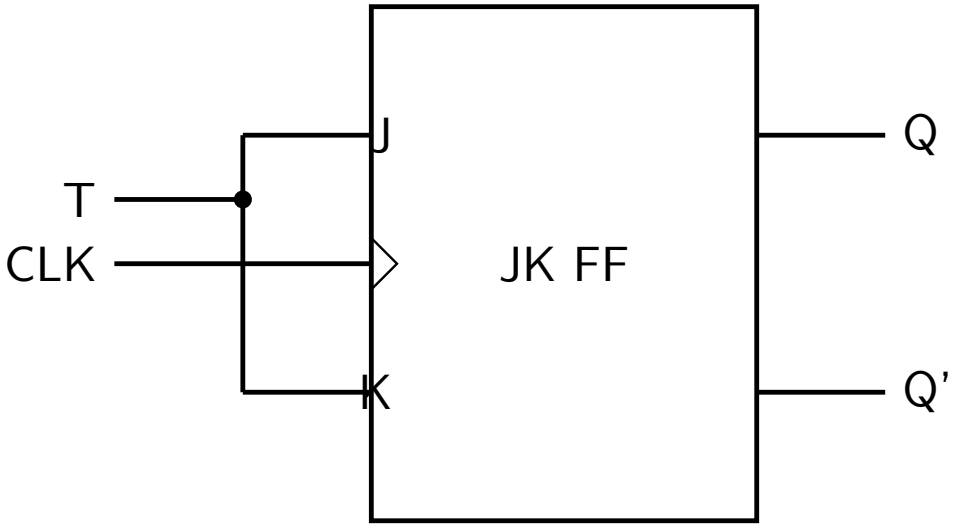
Evaluate inputs only at rising/falling edge

Method 3: Master-Slave Configuration

Most robust hardware solution using two flip-flops

Introduction to the T (Toggle) Flip-flop

- T stands for Toggle.
- Created directly from a JK flip-flop by tying J and K inputs together into a single input T.
- When T=0, J=0 & K=0 $\rightarrow$ Hold state.
- When T=1, J=1 & K=1 $\rightarrow$ Toggle state.



T Flip-flop: Truth Table

- **T —— Q(n+1)**
- 0 —— Q(n) (Hold)
- 1 —— Q(n)' (Toggle)

T	**Q_{n+1}**	**State**
0	Q_n	Hold
1	Q_n'	Toggle

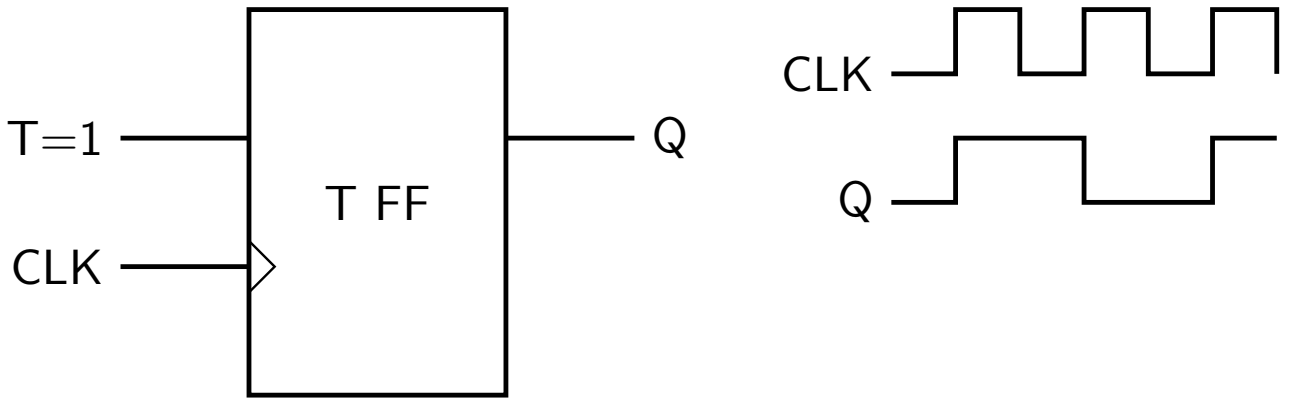
T Flip-flop: Excitation Table

- **Q(n) $\rightarrow$ Q(n+1) —— T**
- $0 \rightarrow 0$ —— 0
- $0 \rightarrow 1$ —— 1
- $1 \rightarrow 0$ —— 1
- $1 \rightarrow 1$ —— 0

Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Applications of T Flip-flops

- Frequency Dividers: A T flip-flop with T=1 toggles on every clock edge, halving the clock frequency.
- Binary Counters: Cascading T flip-flops naturally creates a circuit that counts in binary.
- Used extensively wherever sequential state progression (0-1-2-3) is needed.



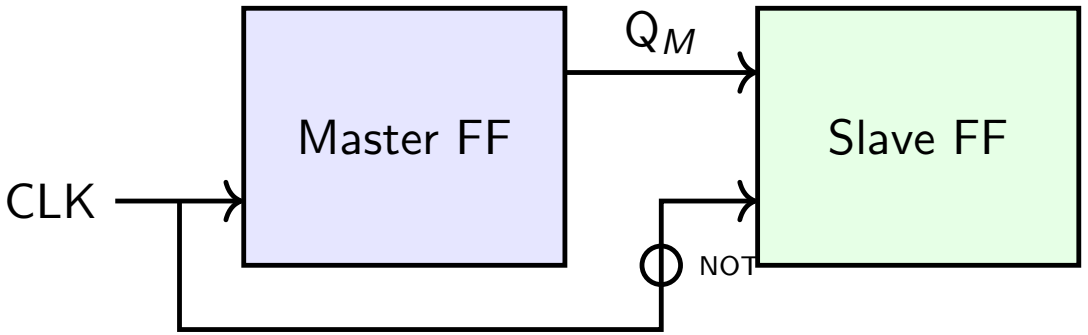
Summary

- JK flip-flop solves the SR 1,1 problem by turning it into a useful toggle function.
- Feedback from Q and Q' enables this toggle behavior.
- Level-triggered JK flip-flops suffer from the Race Around Condition if the clock pulse is too wide.
- T flip-flop is a simplified JK flip-flop used specifically for toggling and counting.

JK FF	Set, Reset, and **Toggle** functionality. Solves SR 1,1 state.
T FF	Simplified JK for **counting**. Toggles when T=1.

Next Lecture Preview

- How do we completely eliminate the race around condition in hardware?
- Introduction to the Master-Slave JK Flip-flop.
- Understanding how two cascaded flip-flops operating on opposite clock phases solve the timing issue.



Questions?

Lecture 4.3: Flip-flops: Master Slave JK

Complete Lecture Deck – All 45 Lectures

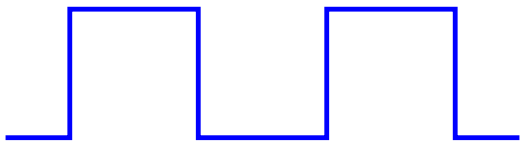
Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

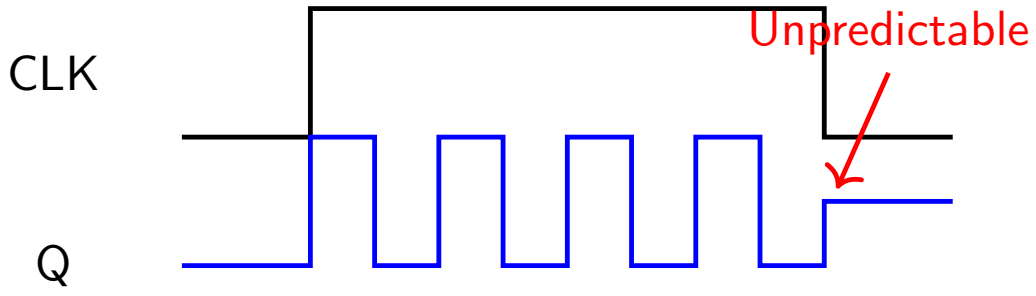
Agenda

- Review of Race Around Condition
- The Master-Slave Concept
- Circuit Construction
- Working during Clock HIGH (Master Active)
- Working during Clock LOW (Slave Active)
- Timing Diagram Analysis
- How it solves Race Around



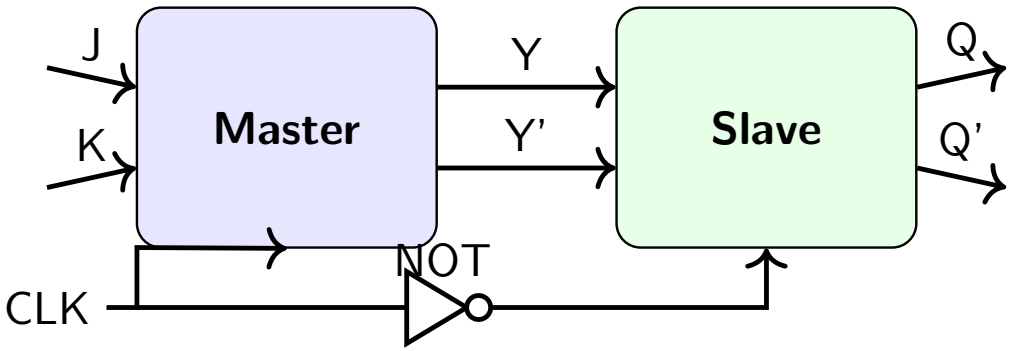
The Problem: Race Around Condition

- In a standard level-triggered JK flip-flop, if J=1, K=1, it toggles.
- If the clock pulse width (t_p) is greater than the propagation delay (t_{pd}), the output will toggle multiple times during one clock pulse.
- Result: Unpredictable final state when the clock goes low.
- We need a way to ensure the output only changes state ONCE per clock cycle.



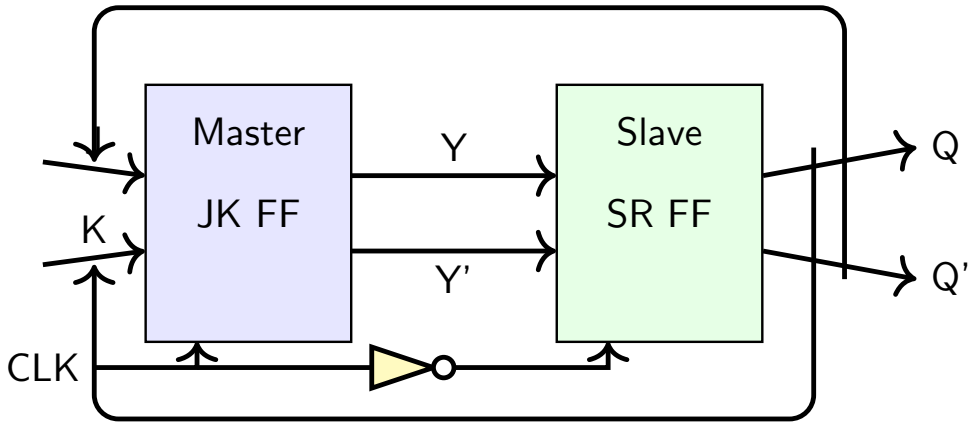
The Master-Slave Concept

- Use TWO flip-flops in series.
- The first is the 'Master', receiving the external J and K inputs.
- The second is the 'Slave', taking its inputs from the Master's outputs.
- Crucial trick: They operate on opposite phases of the clock.
- Master is active on Clock HIGH; Slave is active on Clock LOW.



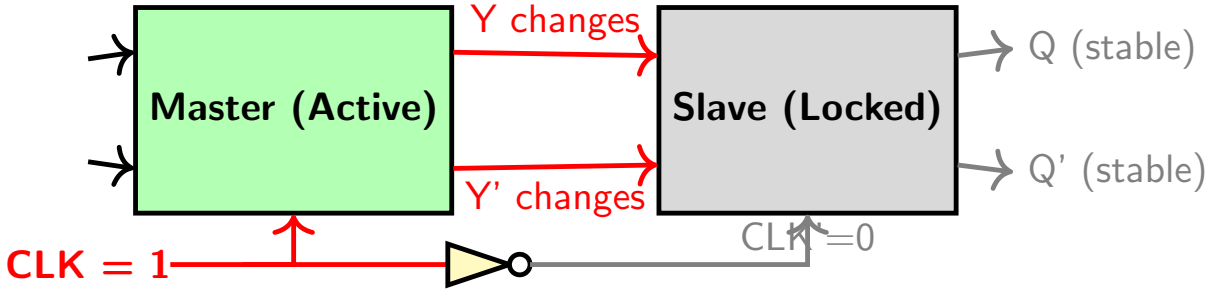
Circuit Construction

- Master: A JK flip-flop driven by the main CLK signal.
- Slave: An SR (or D) flip-flop driven by the inverted CLK signal (CLK').
- The outputs of the Master (Y and Y') are fed into the Slave.
- The final outputs of the Slave (Q and Q') are fed back to the input gates of the Master.



Phase 1: Clock is HIGH (CLK = 1)

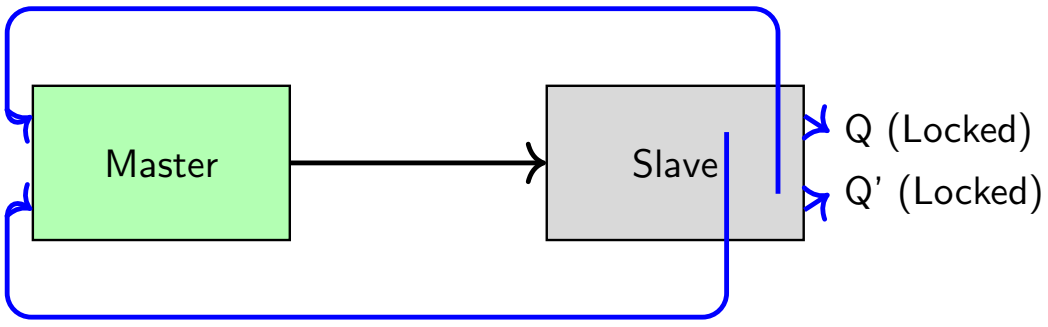
- Main CLK = 1, so the Master is **ENABLED**.
- Inverted CLK to Slave = 0, so the Slave is **DISABLED** (isolated).
- The Master reacts to J and K inputs and updates its intermediate outputs (Y, Y').
- The final outputs Q and Q' DO NOT CHANGE because the Slave is locked.



Phase 1: No Feedback Racing

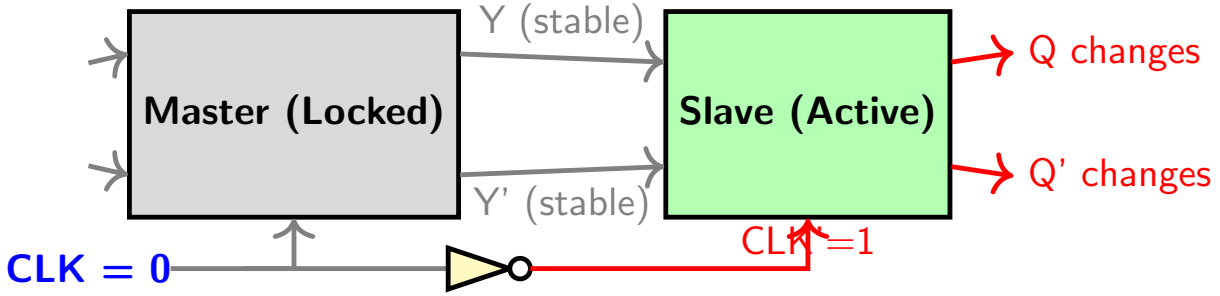
- Even if J=1 and K=1, the Master evaluates based on the current final outputs Q and Q'.
- Because Q and Q' are locked (Slave is disabled), they do not change during this phase.
- Therefore, the Master's inputs remain stable.
- The Master toggles exactly once and waits.

Stable Feedback → No Racing



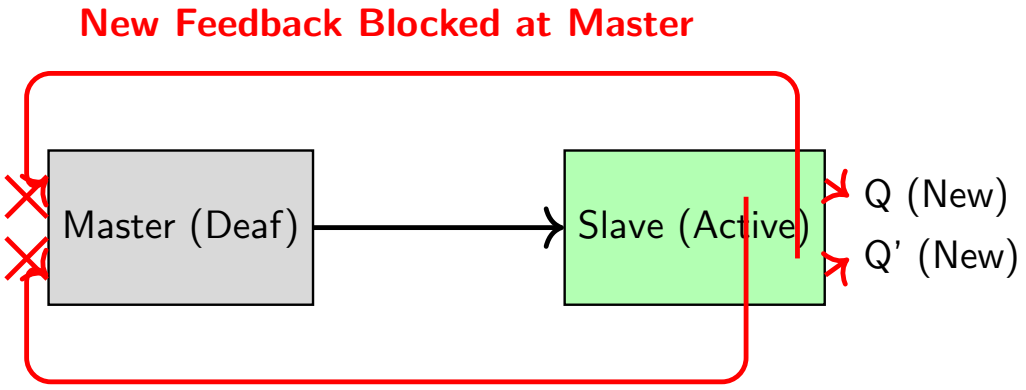
Phase 2: Clock is LOW (CLK = 0)

- Main CLK = 0, so the Master is **DISABLED** (locked).
- Inverted CLK to Slave = 1, so the Slave is **ENABLED**.
- The Slave now reads the Master's outputs (Y, Y') and copies them to the final outputs Q, Q'.
- The external outputs finally update.



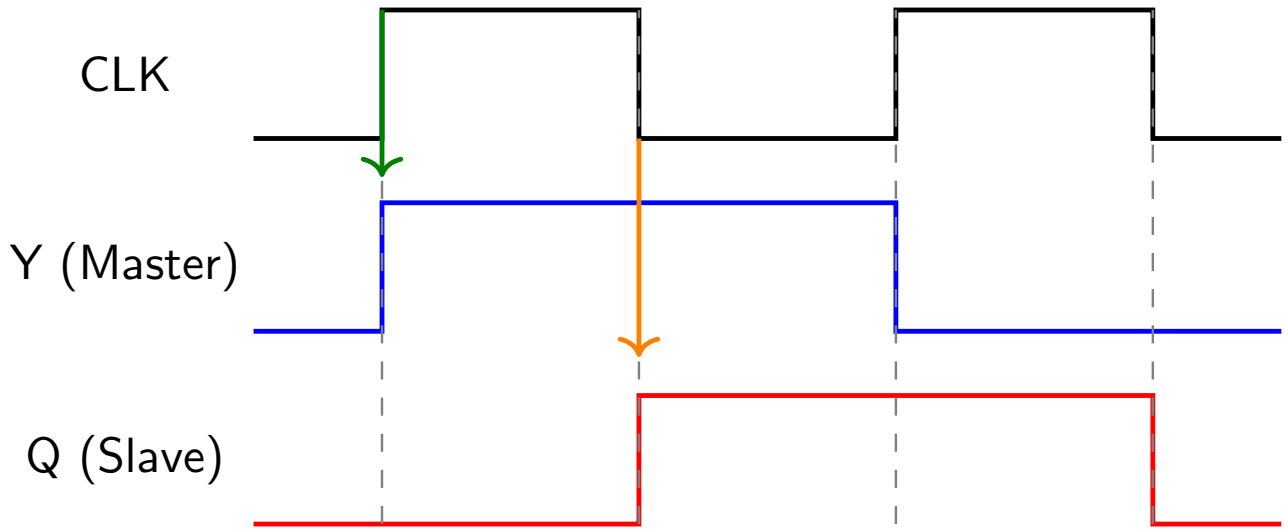
Phase 2: Safe Feedback Update

- When Q and Q' update, they change the signals fed back to the Master.
- However, the Master is currently disabled (CLK=0), so it ignores these new feedback values.
- Racing is completely prevented.
- The cycle is complete, with exactly one clean toggle.



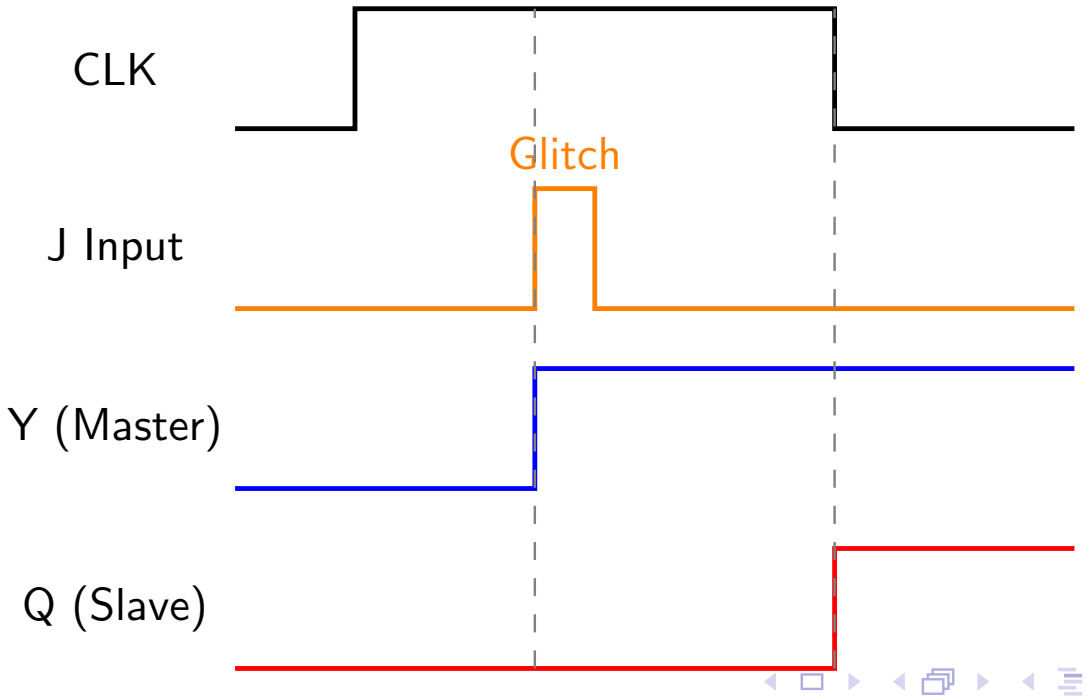
Timing Diagram Analysis

- Observe the relationship between CLK, Master Output (Y), and Final Output (Q).
- When CLK goes High $\rightarrow$ Y changes.
- When CLK goes Low $\rightarrow$ Q changes to match Y.
- The overall circuit behaves like a Negative Edge-Triggered flip-flop.



1s Catching (Master-Slave Glitch)

- While Master-Slave solves racing, older designs had a quirk called '1s catching'.
- If J=0, K=0, but J glitches to 1 momentarily while CLK=1, the Master 'catches' the 1 and toggles.
- Because it's a flip-flop itself, the Master holds this error until the Slave copies it on the falling edge.
- Modern true edge-triggered flip-flops solve this by isolating inputs faster.



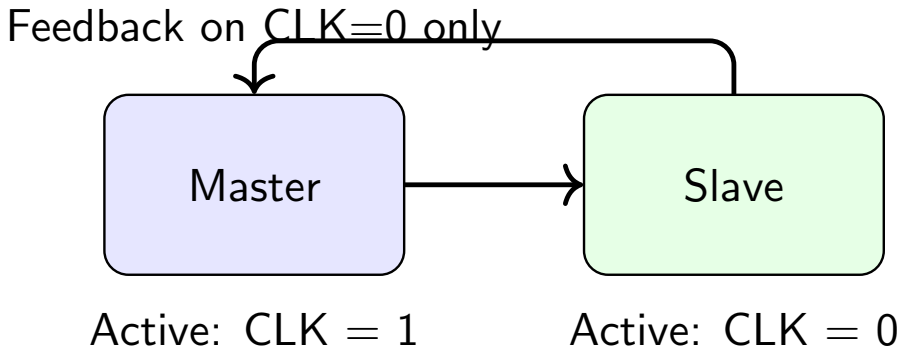
Master-Slave vs. Edge-Triggered

- Both solve the race around condition.
- True edge-triggered designs (using fewer gates) are more common in modern ICs.

Feature	Master-Slave	Edge-Triggered
Construction	Composed of two level-triggered latches	Specialized gate design to isolate inputs
Trigger Timing	Master evaluates on level, Slave updates on falling edge	Evaluates ONLY at the infinitesimally small moment of clock transition
1s Catching Glitch	Susceptible (catches glitches during active level)	Immune (inputs are ignored during the level)

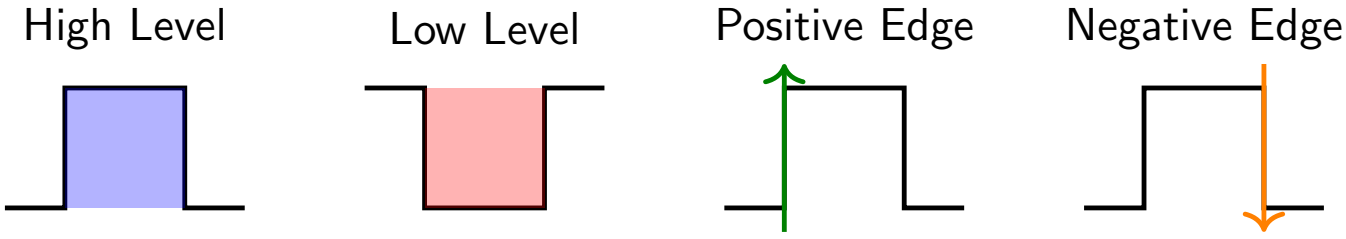
Summary

- The Race Around Condition occurs in level-triggered JK flip-flops due to continuous feedback during a long clock pulse.
- Master-Slave JK uses two flip-flops working on opposite clock phases.
- Master accepts inputs on Clock HIGH; Slave updates outputs on Clock LOW.
- This decoupling strictly ensures only one state change per clock cycle.



Next Lecture Preview

- We've discussed 'level-triggered' and 'edge-triggered'.
- Next lecture: Triggering Methods.
- We will formally define and compare High-level, Low-level, Positive-edge, and Negative-edge triggering.
- Understanding how clocks synchronize massive digital systems.



Questions?

Lecture 4.4: Triggering Methods

Complete Lecture Deck – All 45 Lectures

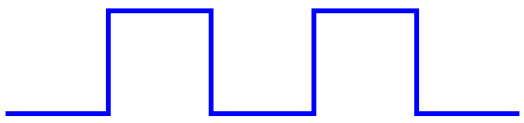
Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

Agenda

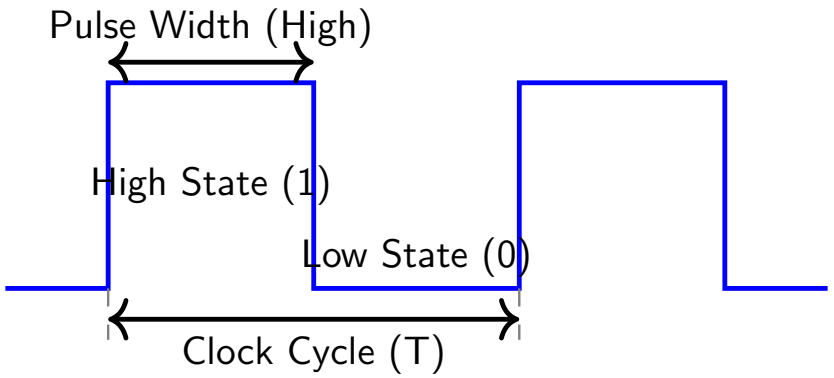
- The Role of the Clock Signal
- What is Triggering?
- Level Triggering (High & Low)
- Drawbacks of Level Triggering
- Edge Triggering (Positive & Negative)
- Edge-Detector Circuits
- Symbols for Triggering Types
- Summary & Preview



Clock Signal

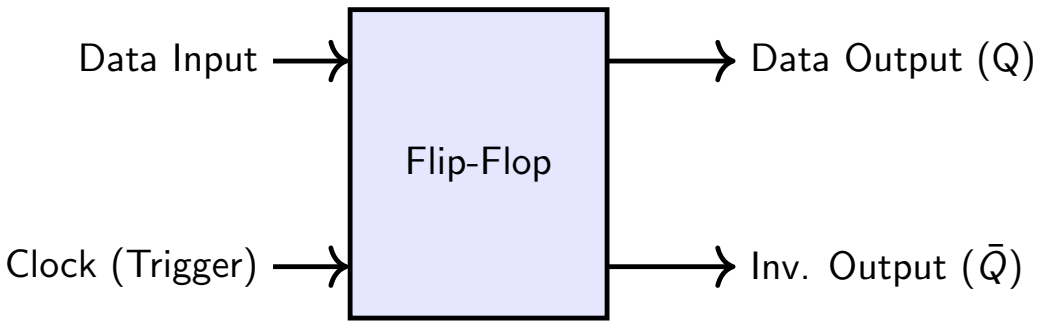
The Role of the Clock Signal

- A clock is a continuous square wave signal oscillating between High (1) and Low (0).
- It acts as a metronome for the digital system.
- Ensures all parts of a complex sequential circuit change state simultaneously (Synchronous operation).
- Prevents intermediate 'garbage' states from propagating through the system.



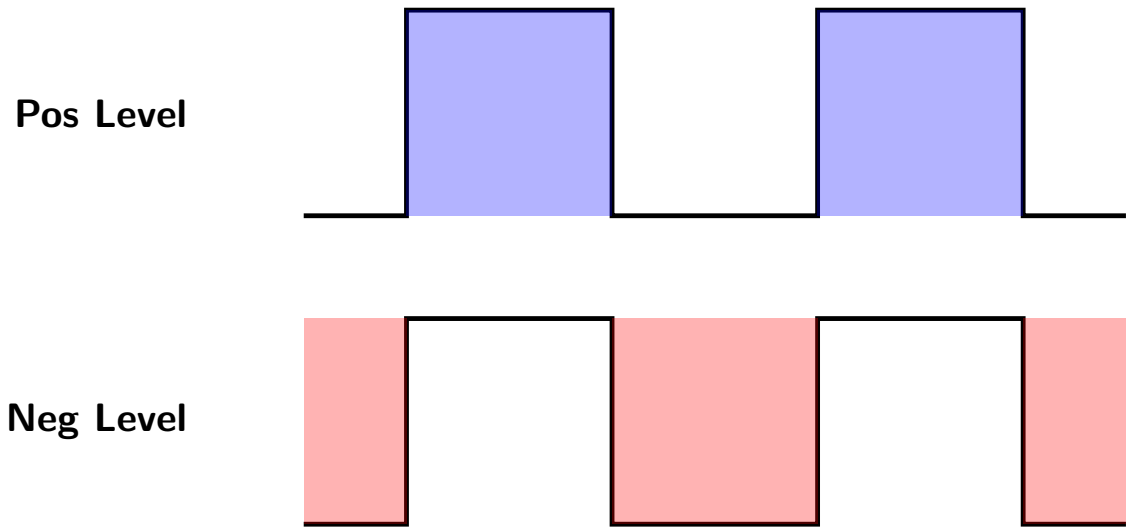
What is Triggering?

- Triggering refers to the specific moment when a memory element (latch or flip-flop) evaluates its inputs and changes its output state.
- Until the triggering event occurs, the circuit holds its previous state (Memory).
- Two main categories:
- 1. Level Triggering
- 2. Edge Triggering



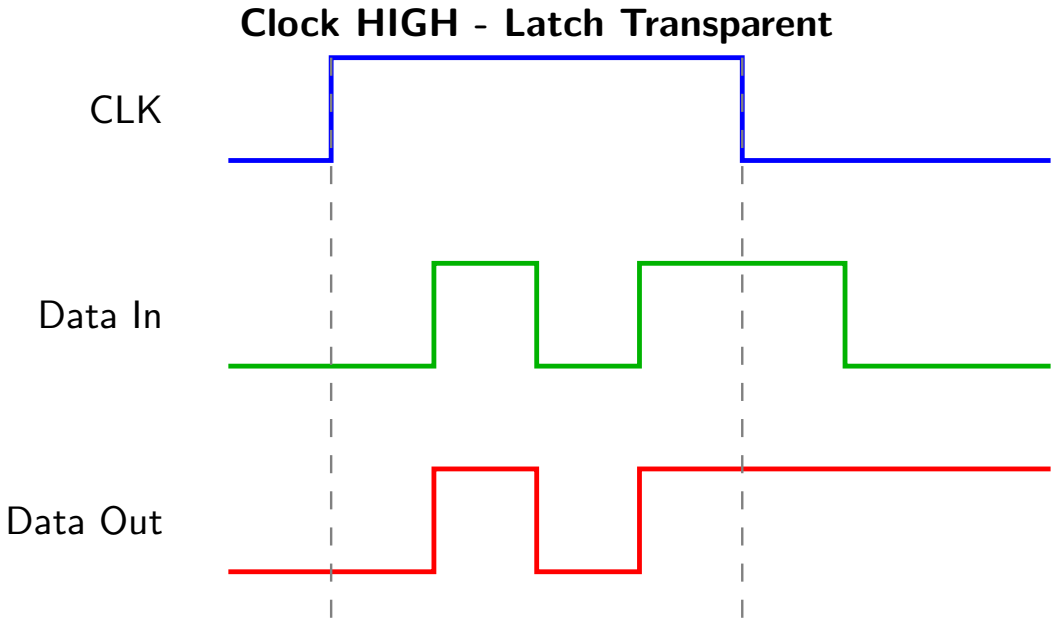
Level Triggering

- The circuit responds to inputs as long as the clock signal is at a specific voltage level.
- Positive Level Triggering: Active when Clock is HIGH (1).
- Negative Level Triggering: Active when Clock is LOW (0).
- Often called 'Latches' rather than flip-flops.



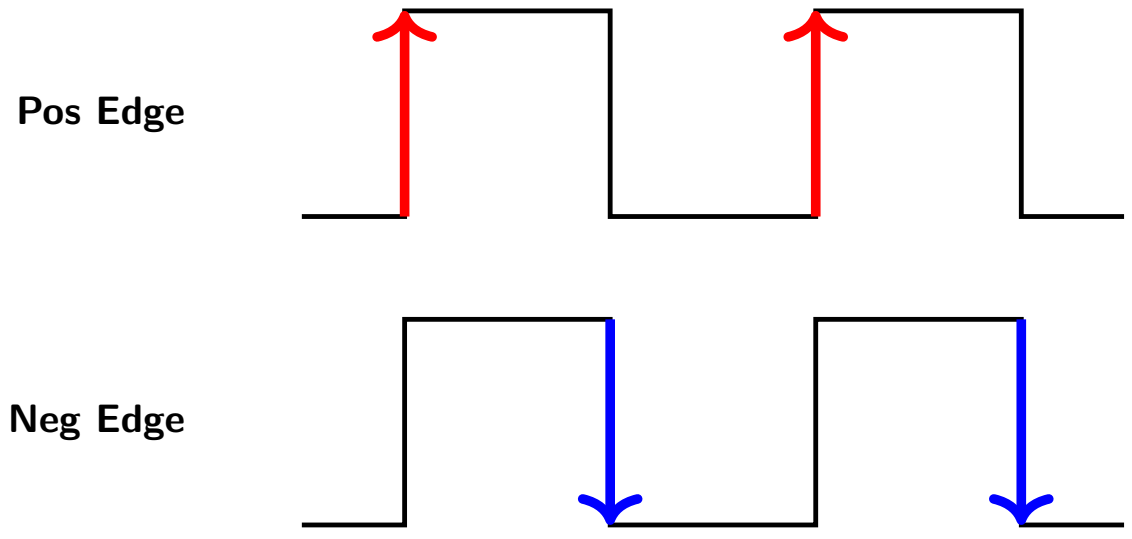
Drawbacks of Level Triggering

- Transparency: If inputs change multiple times while the clock is active, the output changes multiple times.
- Can lead to Race Conditions in feedback circuits (like the JK flip-flop issue).
- Makes timing analysis in complex circuits very difficult, as delays add up.
- Requires very strict control over input stability.



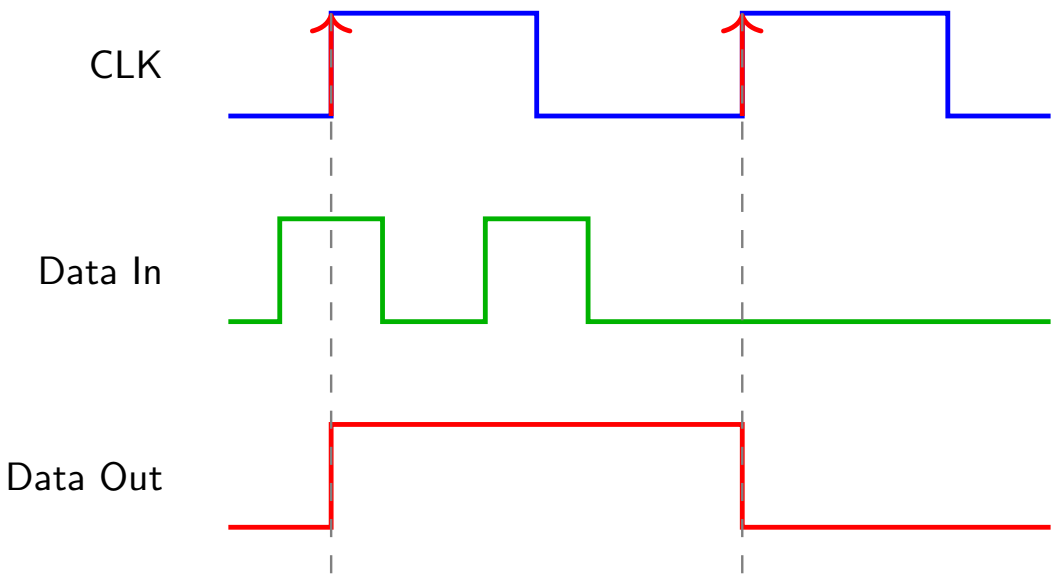
Edge Triggering

- The circuit responds to inputs ONLY during the instantaneous transition of the clock signal.
- Positive Edge Triggering (Rising Edge): Active only when Clock transitions from LOW to HIGH ($0 \rightarrow 1$).
- Negative Edge Triggering (Falling Edge): Active only when Clock transitions from HIGH to LOW ($1 \rightarrow 0$).
- These are true 'Flip-flops'.



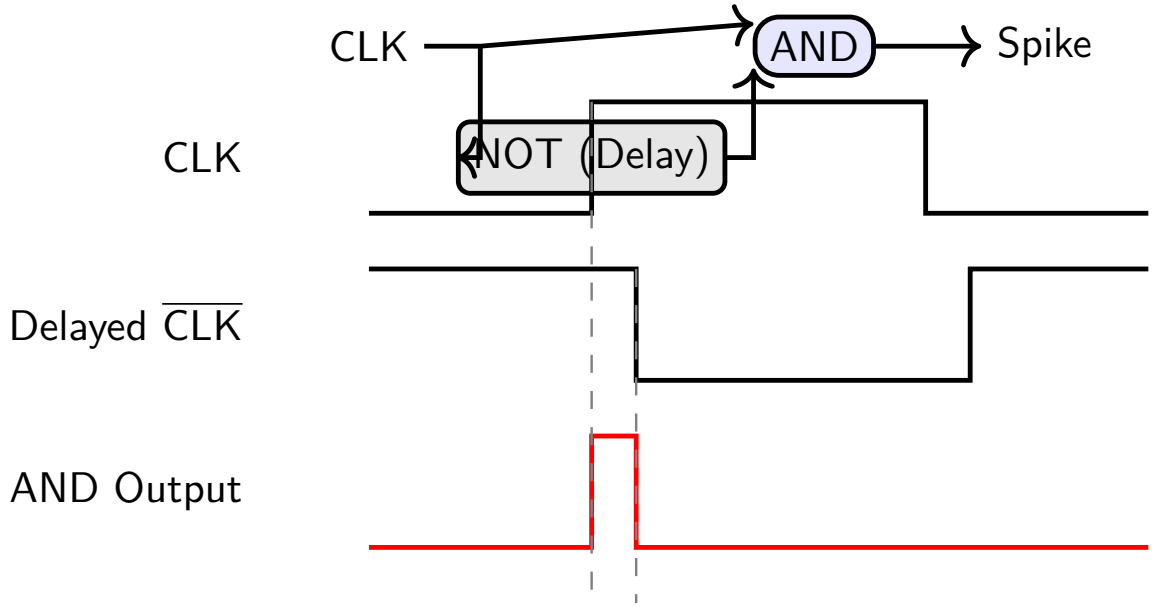
Advantages of Edge Triggering

- Input changes before or after the edge are completely ignored.
- Solves race around conditions easily.
- Allows for precise synchronous design—all flip-flops update at the exact same instant.
- Glitches on data lines during the clock cycle do not affect the output.



How to Create an Edge-Detector

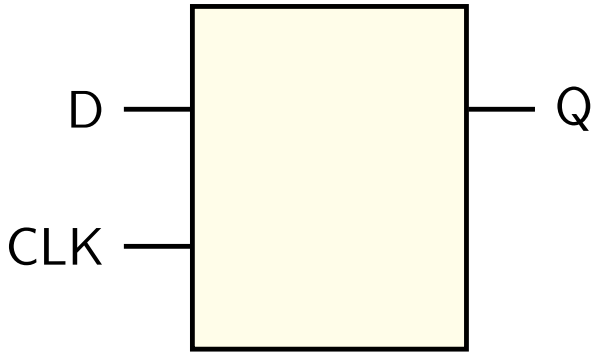
- How does hardware detect an 'instantaneous' edge?
- Use a small delay (e.g., an RC circuit or an inverter chain) and an AND gate.
- By ANDing the clock signal with a slightly delayed, inverted version of itself, you generate a very narrow spike.
- This narrow spike acts as the internal enable for the latches.



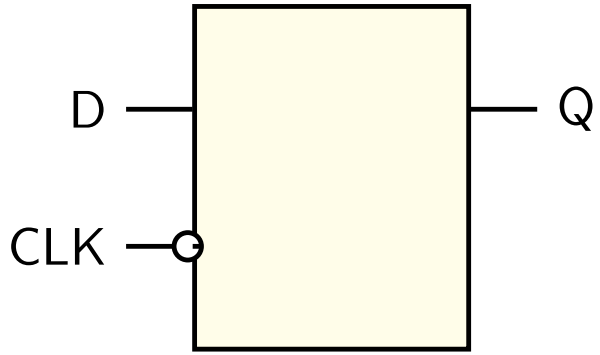
Schematic Symbols: Level Triggered

- Positive Level Triggered (Latch): Straight line into the block.
- Negative Level Triggered: Straight line with a 'bubble' (inversion circle) at the clock input.
- Whenever you see a clock input without a triangle, it is level-triggered.

Pos. Level (Latch)

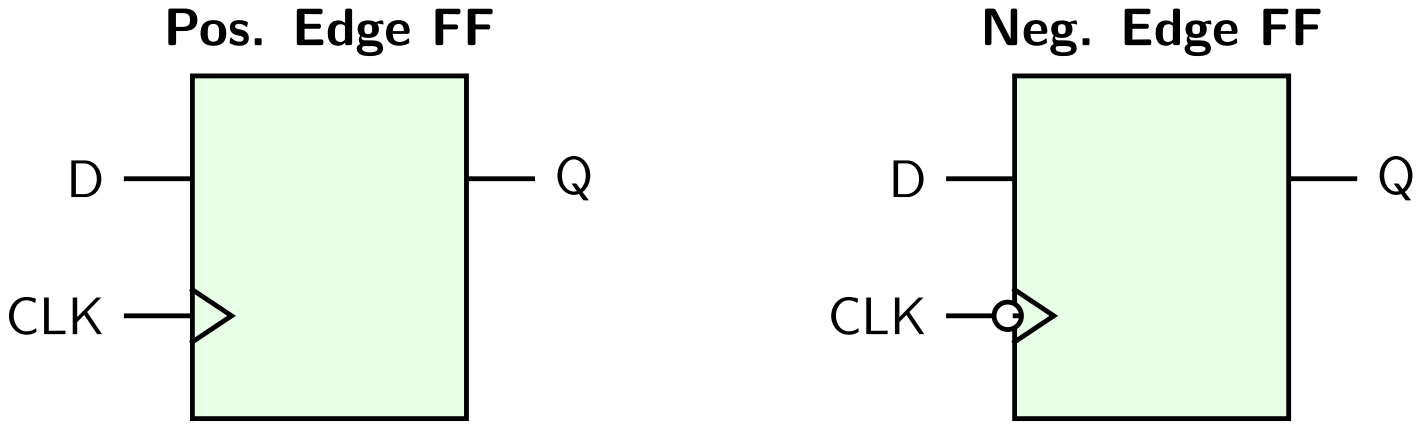


Neg. Level (Latch)



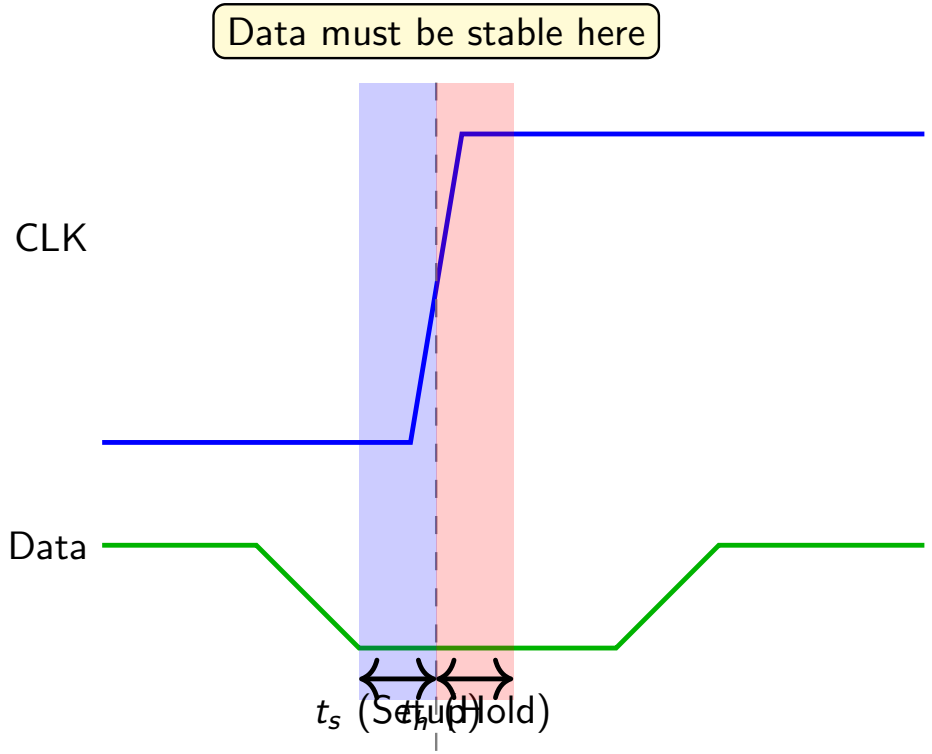
Schematic Symbols: Edge Triggered

- Positive Edge Triggered: Indicated by a small triangle (dynamic indicator) at the clock input inside the block.
- Negative Edge Triggered: Indicated by a small triangle AND a bubble (inversion circle) at the clock input.
- The triangle means 'edge sensitive'.



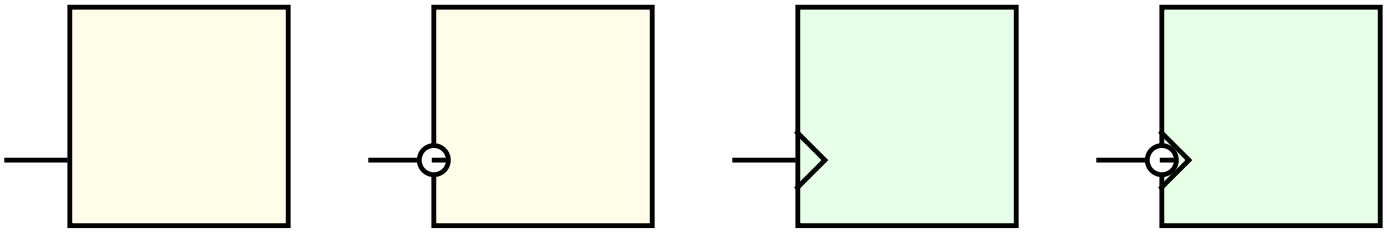
Setup and Hold Times

- Even edge triggering isn't magic; it needs time.
- Setup Time (t_s): Minimum time the data input must be stable BEFORE the clock edge.
- Hold Time (t_h): Minimum time the data input must remain stable AFTER the clock edge.
- Violating these causes Metastability (unpredictable outputs).



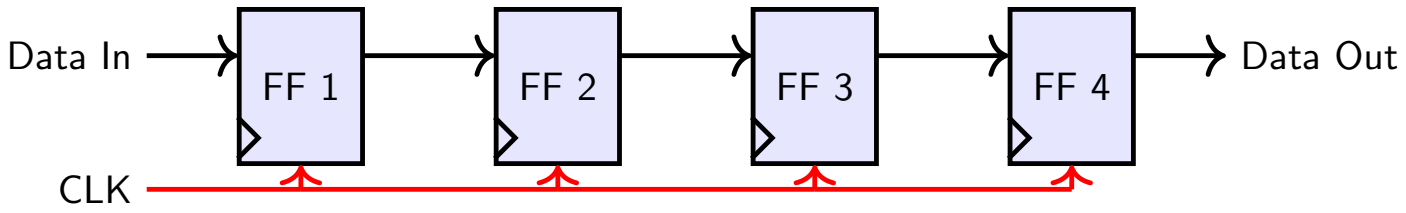
Summary

- Clock signals synchronize sequential logic.
- Level triggering evaluates inputs as long as the clock is High or Low (prone to racing and glitches).
- Edge triggering evaluates inputs only during the precise transition (Rising or Falling).
- Edge-triggered devices are denoted by a triangle symbol on the clock pin.
- Setup and hold times must be respected for reliable operation.



Next Lecture Preview

- Now that we know how flip-flops store 1 bit of data synchronously, how do we store 8 bits? Or 32 bits?
- Introduction to Registers and Shift Registers.
- Specifically, Serial-in Serial-out (SISO) and Serial-in Parallel-out (SIPO) shift registers.
- How to move data sequentially through a circuit.



Questions?

Lecture 4.5: Shift Registers: SISO and SIPO

Complete Lecture Deck – All 45 Lectures

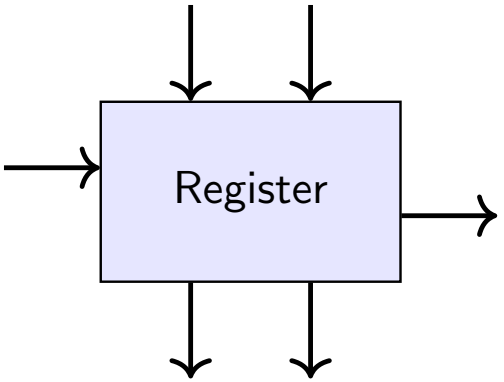
Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

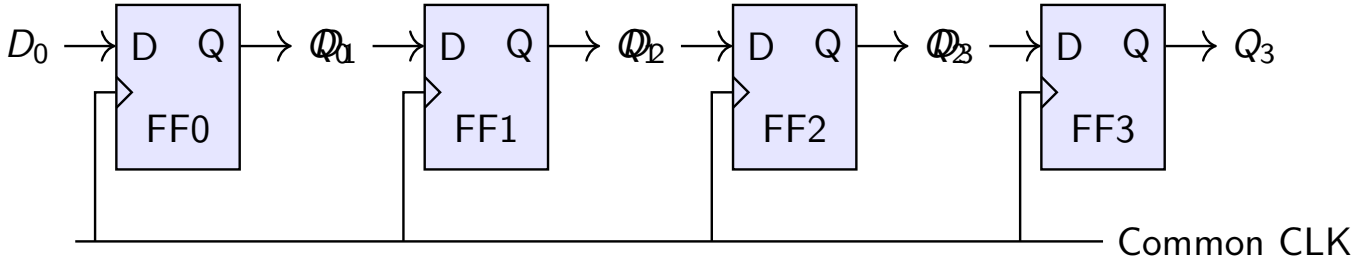
Agenda

- What is a Register?
- What is a Shift Register?
- Types of Shift Registers
- Serial-In Serial-Out (SISO) Architecture
- SISO Operation & Timing Diagram
- Serial-In Parallel-Out (SIPO) Architecture
- SIPO Operation & Applications
- Summary & Preview



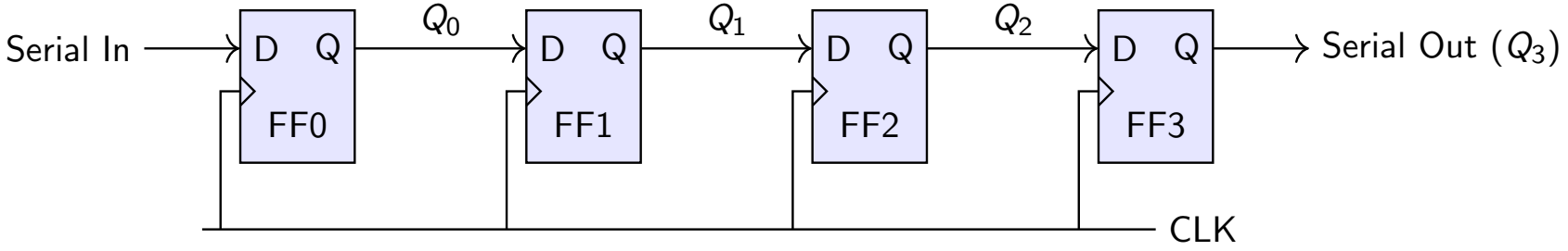
What is a Register?

- A flip-flop can store 1 bit of information.
- An n-bit register is a group of n flip-flops used to store an n-bit word.
- Example: To store an 8-bit byte, you need a register containing 8 flip-flops.
- All flip-flops in a register share a common clock signal for synchronous operation.



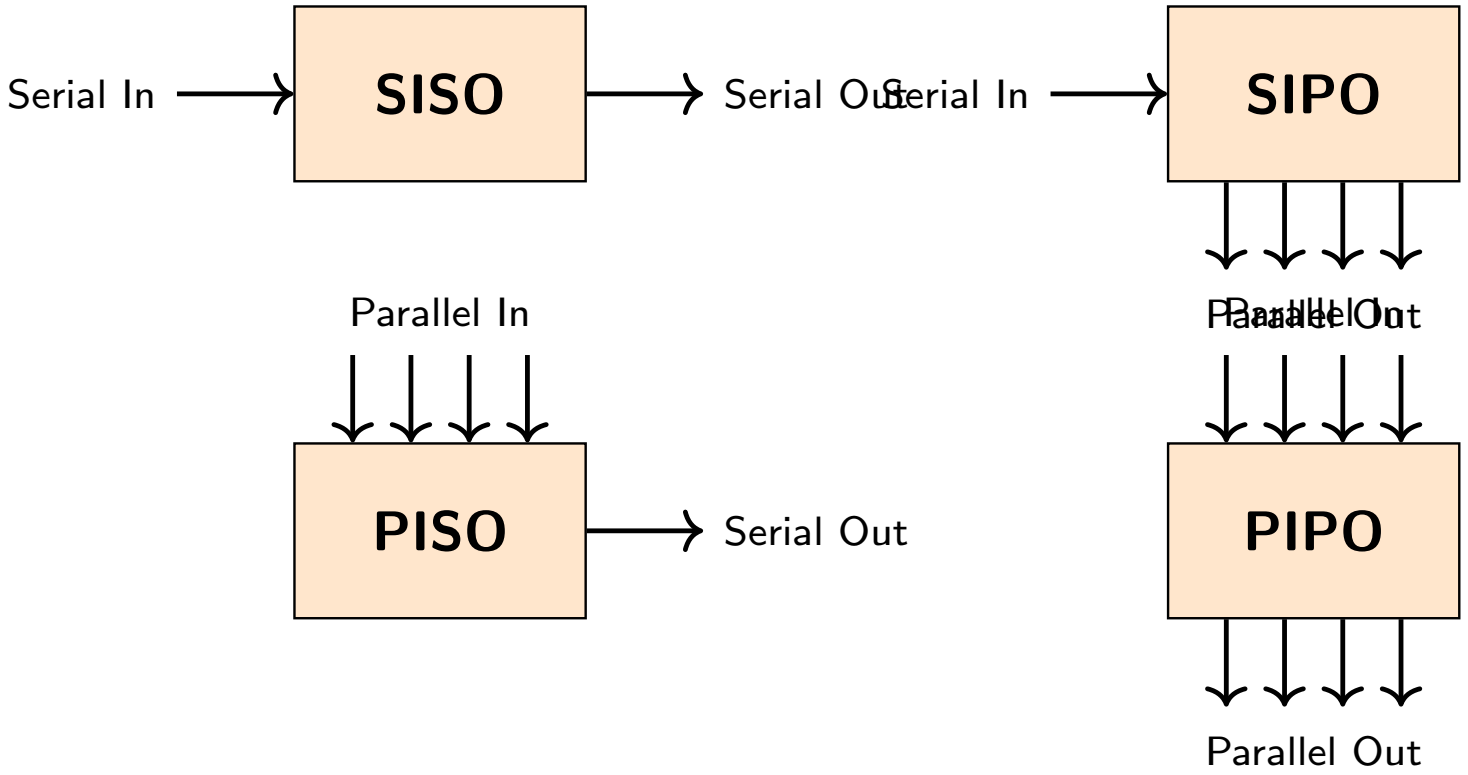
What is a Shift Register?

- A Shift Register can not only store data, but also shift it left or right.
- Flip-flops are cascaded: the output (Q) of one is connected to the input (D) of the next.
- With each clock pulse, data moves one position down the chain.
- Used for data transfer, arithmetic operations (multiply/divide by 2), and data formatting.



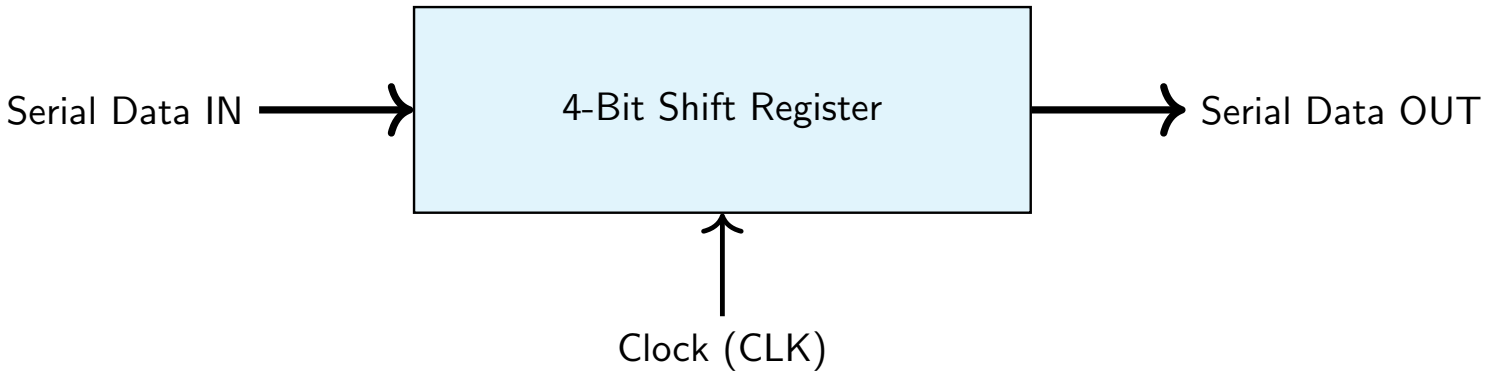
Types of Data Movement

- Based on how data goes IN and OUT, there are 4 main types:
- 1. SISO (Serial In, Serial Out)
- 2. SIPO (Serial In, Parallel Out)
- 3. PISO (Parallel In, Serial Out)
- 4. PIPO (Parallel In, Parallel Out)
- Today we focus on the first two.



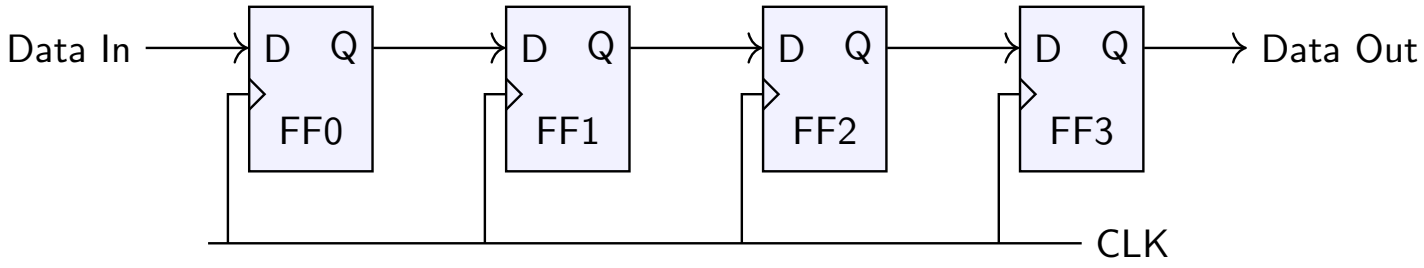
Serial-In Serial-Out (SISO)

- Data enters sequentially (one bit per clock cycle) at the first flip-flop.
- Data leaves sequentially (one bit per clock cycle) at the last flip-flop.
- To store an n-bit word, it takes n clock cycles.
- To read the stored n-bit word, it takes another n clock cycles.



SISO: Circuit Diagram

- Usually built using D flip-flops.
- Input Data $\rightarrow$ D0.
- Q0 $\rightarrow$ D1, Q1 $\rightarrow$ D2, Q2 $\rightarrow$ D3 (Output).
- All flip-flops share the same Edge-Triggered CLK.



SISO: Operation Example

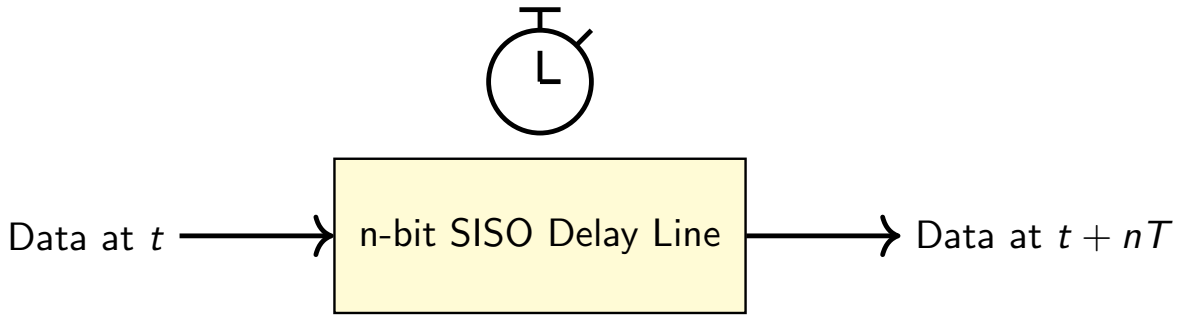
Let's shift in the binary sequence **1011** into a 4-bit SISO register (initially 0000).

Clock Pulse	Data Input	Q0	Q1	Q2	Q3 (Output)
Initial	-	0	0	0	0
1	1	**1**	0	0	0
2	1	**1**	1	0	0
3	0	**0**	1	1	0
4	1	**1**	0	1	1

The data 1011 is now stored in the register.

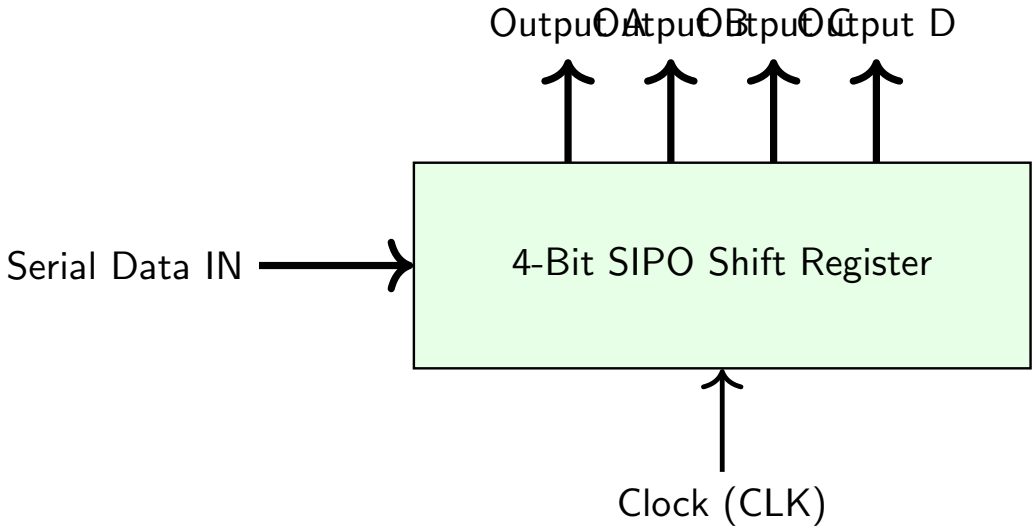
SISO: Applications

- Mainly used for introducing a specific time delay to a digital signal.
- An n-bit SISO delays the serial data by 'n' clock periods.
- Used in digital communication systems for data buffering.
- Requires the fewest number of pins/wires.



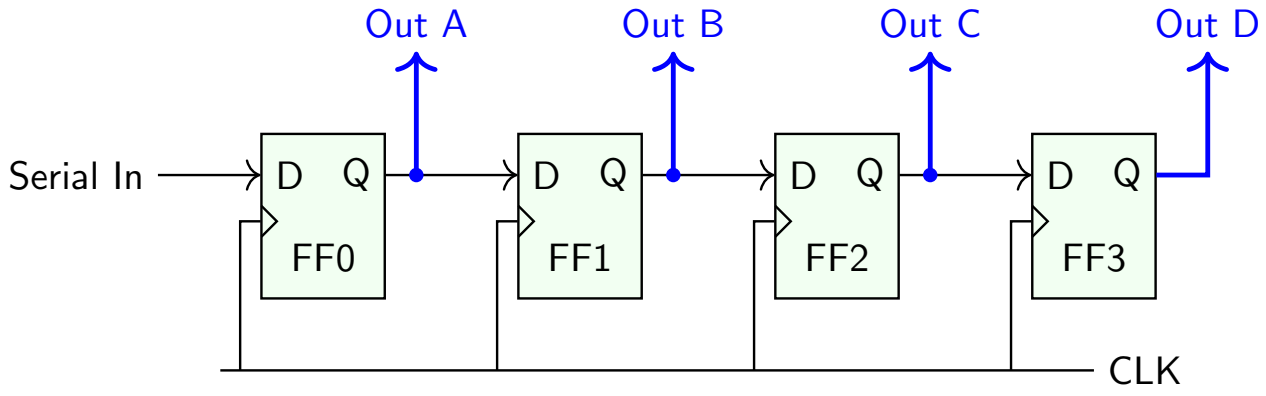
Serial-In Parallel-Out (SIPO)

- Data enters sequentially (one bit per clock cycle), just like SISO.
- Data is available in parallel at all times on the output pins.
- To load an n-bit word, it takes n clock cycles.
- Once loaded, all n bits can be read simultaneously in 1 clock cycle.



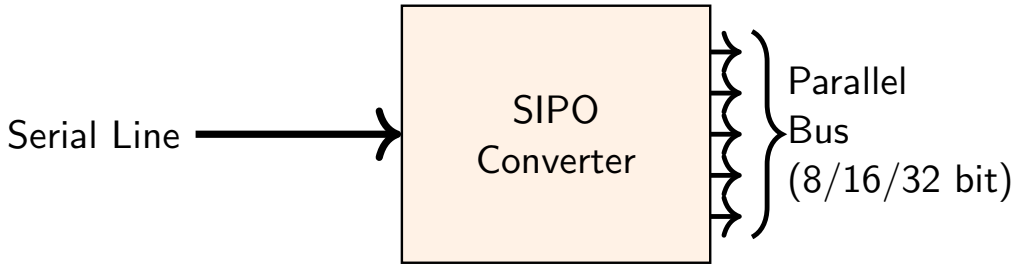
SIPO: Circuit Diagram

- Internal structure is identical to SISO.
- The difference is that Q0, Q1, Q2, and Q3 are brought out to external pins.
- Often includes an 'Output Enable' or buffer stage to prevent reading garbage while shifting is occurring.



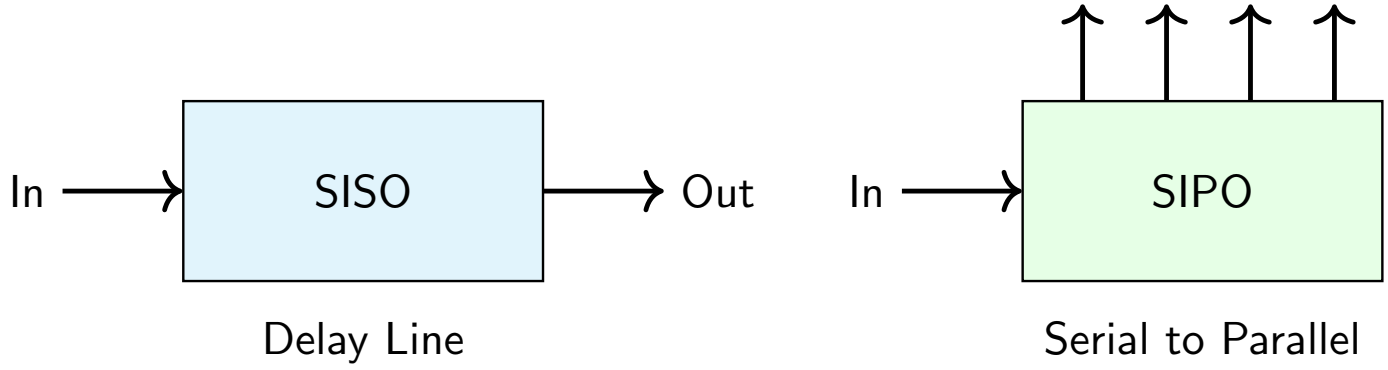
SIPO: Applications

- **Serial-to-Parallel Conversion**: The most common application.
- **USB to CPU**: USB sends data sequentially (serial), but the CPU processes it in 8/16/32 bit chunks (parallel).
- **Driving LED displays**: Microcontroller sends a few pins of serial data to light up 8 or more LEDs simultaneously.



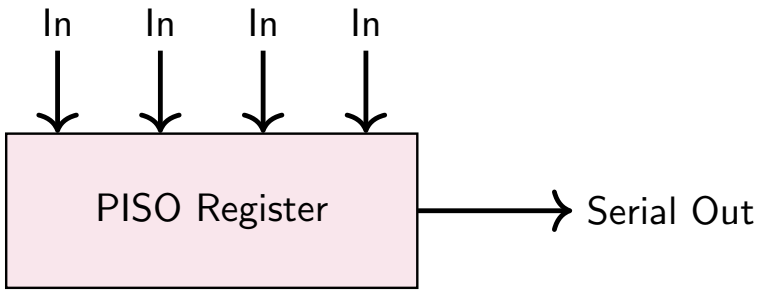
Summary

- Registers store multiple bits; Shift Registers can also move data.
- SISO takes n clocks to load and n clocks to read. Used for delays.
- SIPO takes n clocks to load but provides all data instantly. Used for serial-to-parallel conversion.
- Both are built by cascading D flip-flops with a common clock.



Next Lecture Preview

- What if we have 8 bits of data ready on a bus, and we want to send it over a single serial wire?
- Next Lecture: Parallel-In Serial-Out (PISO).
- We'll also look at Parallel-In Parallel-Out (PIPO).
- Understanding how to construct parallel loading circuits.



Questions?

Lecture 4.6: Shift Registers: PISO and PIPO

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

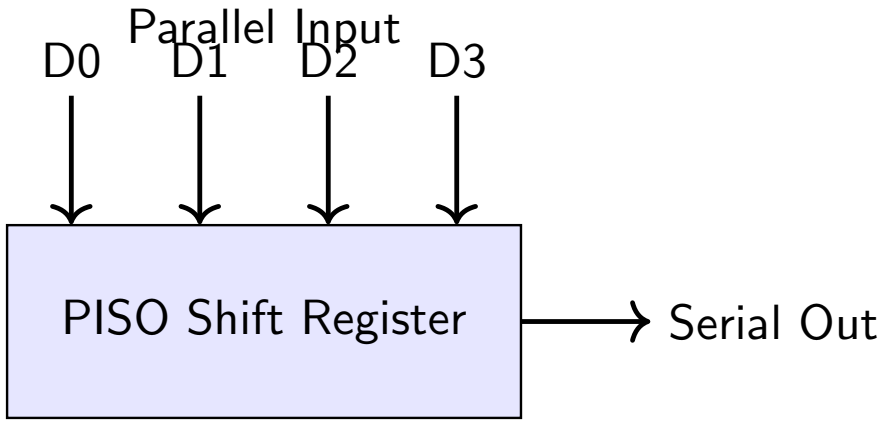
Agenda

- Parallel-In Serial-Out (PISO) Overview
- The Load/Shift Control Signal
- PISO Circuit Diagram (with Logic Gates)
- PISO Operation & Timing
- Parallel-In Parallel-Out (PIPO) Overview
- PIPO Circuit and Operation
- Comparison of all Shift Registers
- Summary & Preview

Lecture 4.6: Shift Registers: PISO and PIPO

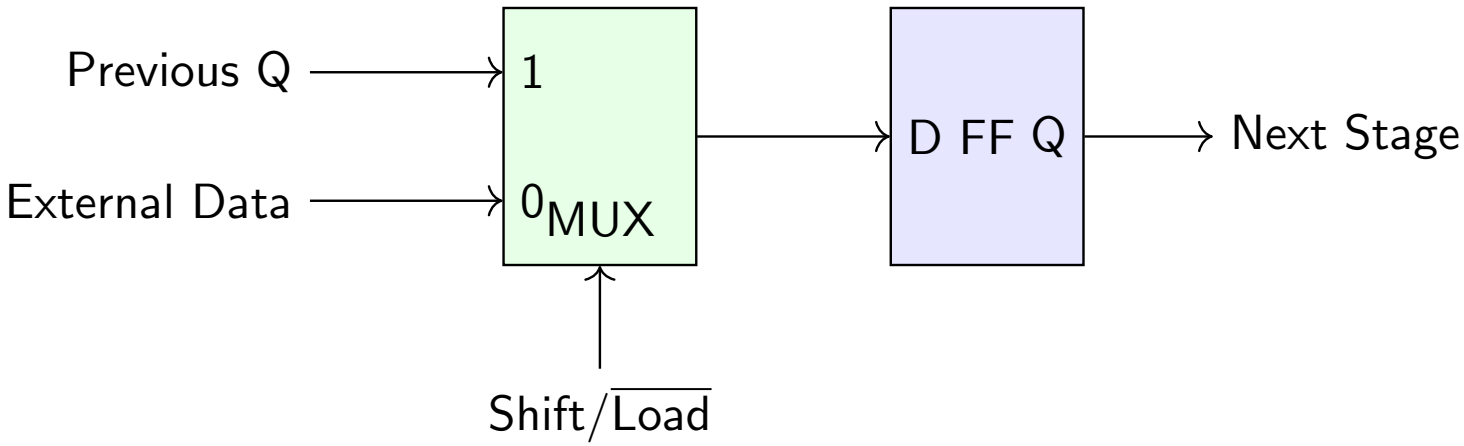
Parallel-In Serial-Out (PISO)

- Data is loaded into the register simultaneously on all flip-flops (Parallel In).
- Data is then read out sequentially, one bit per clock cycle (Serial Out).
- Takes 1 clock cycle to load n bits.
- Takes n clock cycles to read out n bits.
- Primary use: Parallel-to-Serial Conversion.



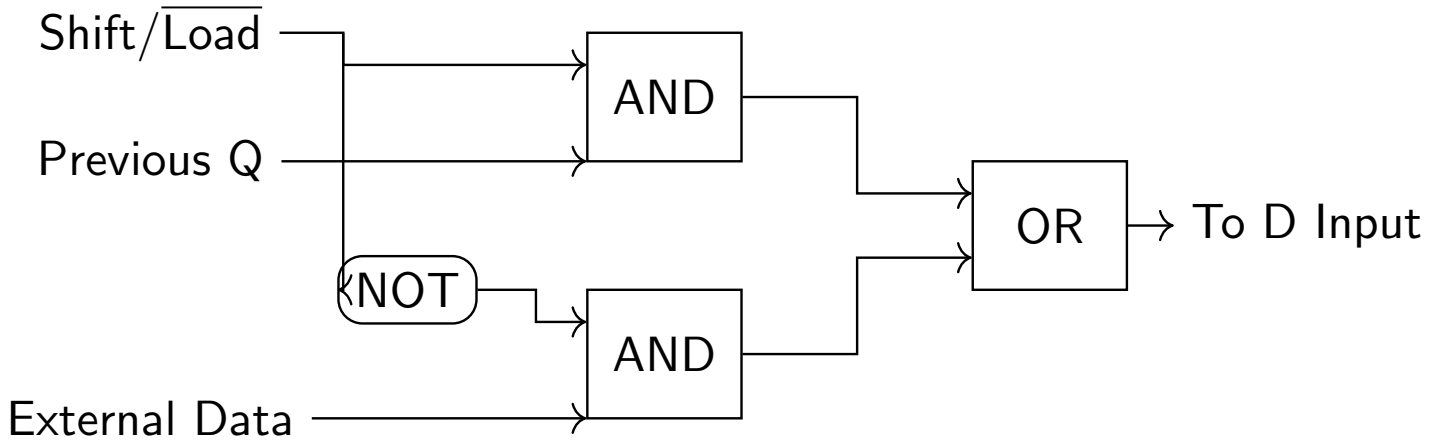
The Challenge with PISO

- The D input of each flip-flop needs to do two different things:
- 1. During loading, D must connect to the external parallel data input.
- 2. During shifting, D must connect to the Q output of the preceding flip-flop.
- How do we switch between two inputs?
- We need a Multiplexer (or equivalent AND/OR logic) at each flip-flop's input!



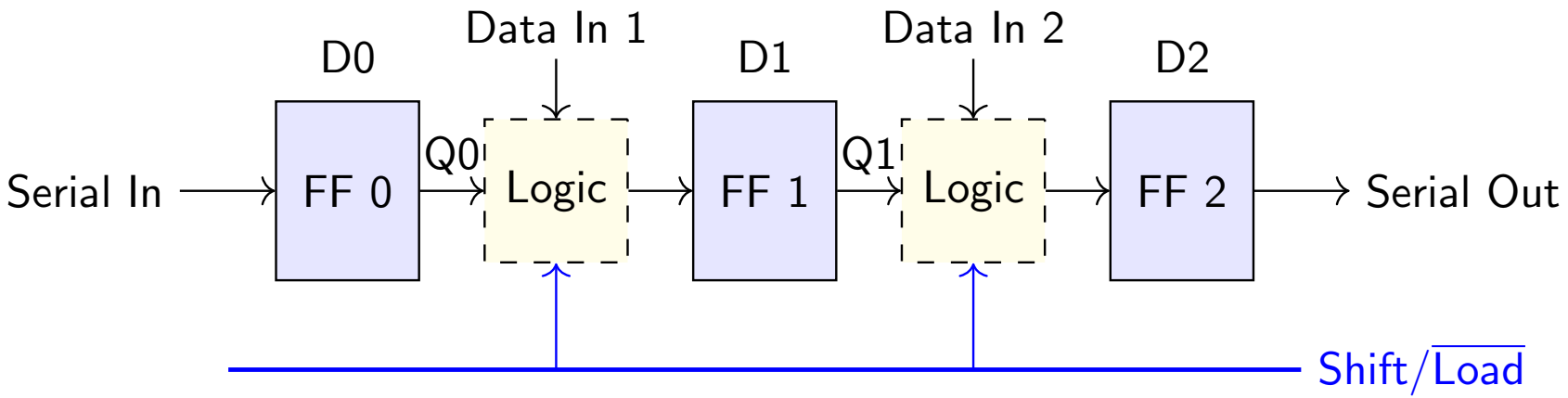
The Shift/Load Control Line

- A control signal called Shift/Load' (Shift active high, Load active low) selects the mode.
- When Shift/Load' = 0: The AND gates connected to the parallel data lines are active. Data is LOADED.
- When Shift/Load' = 1: The AND gates connected to the previous Q outputs are active. Data is SHIFTED.



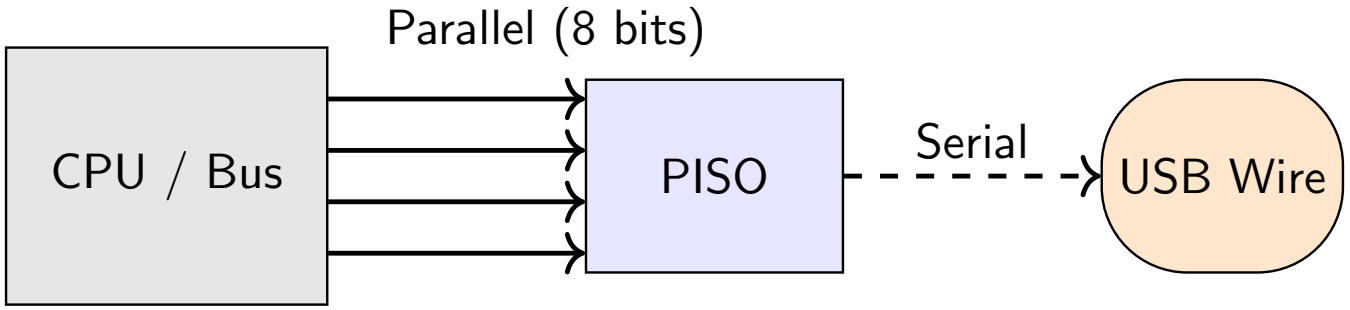
PISO Circuit Diagram

- Each stage (except the first) consists of: a D flip flop, two AND gates, one OR gate, and one NOT gate for the control signal.
- Stage 0 only needs to shift in 0s, or it can be part of a larger chain.
- This combinational logic routing is required to enable the dual-mode functionality.



PISO Application: Communication

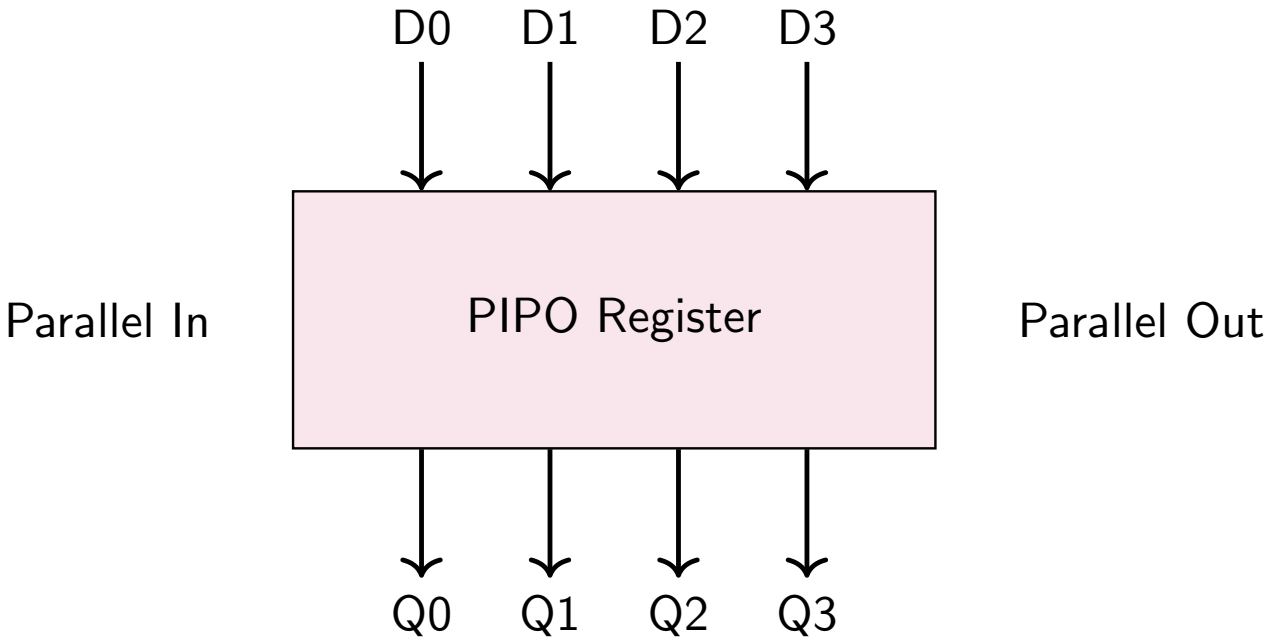
- Essential in transmitters (UART, SPI, I2C).
- A computer wants to send a keystroke (8-bit ASCII) over a single USB wire.
- It dumps the 8 bits into a PISO register instantly.
- The clock then ticks 8 times to push the bits serially onto the USB wire.



Lecture 4.6: Shift Registers: PISO and PIPO

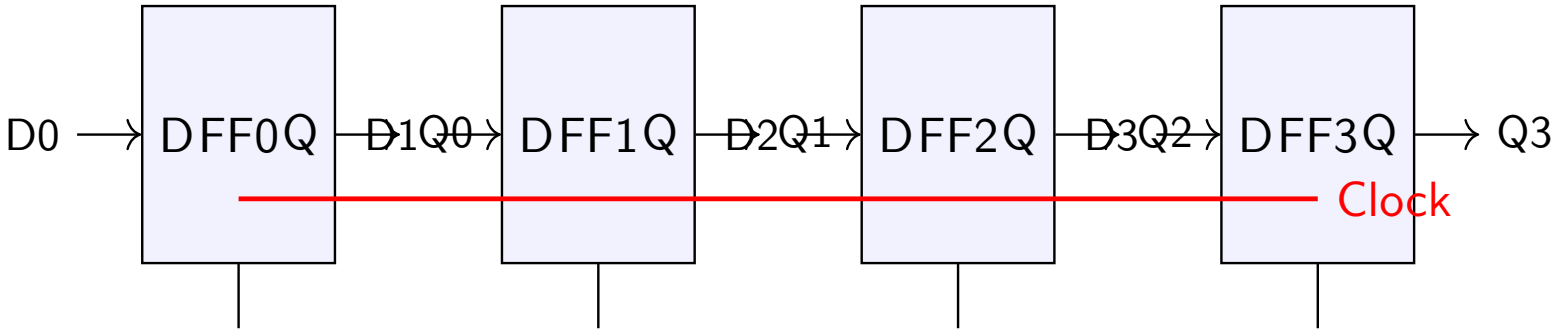
Parallel-In Parallel-Out (PIPO)

- Data is loaded simultaneously into all flip-flops.
- Data is read simultaneously from all flip-flops.
- Does not actually 'shift' data left or right.
- Acts purely as a multi-bit storage register (like a CPU register).



PIPO Circuit and Operation

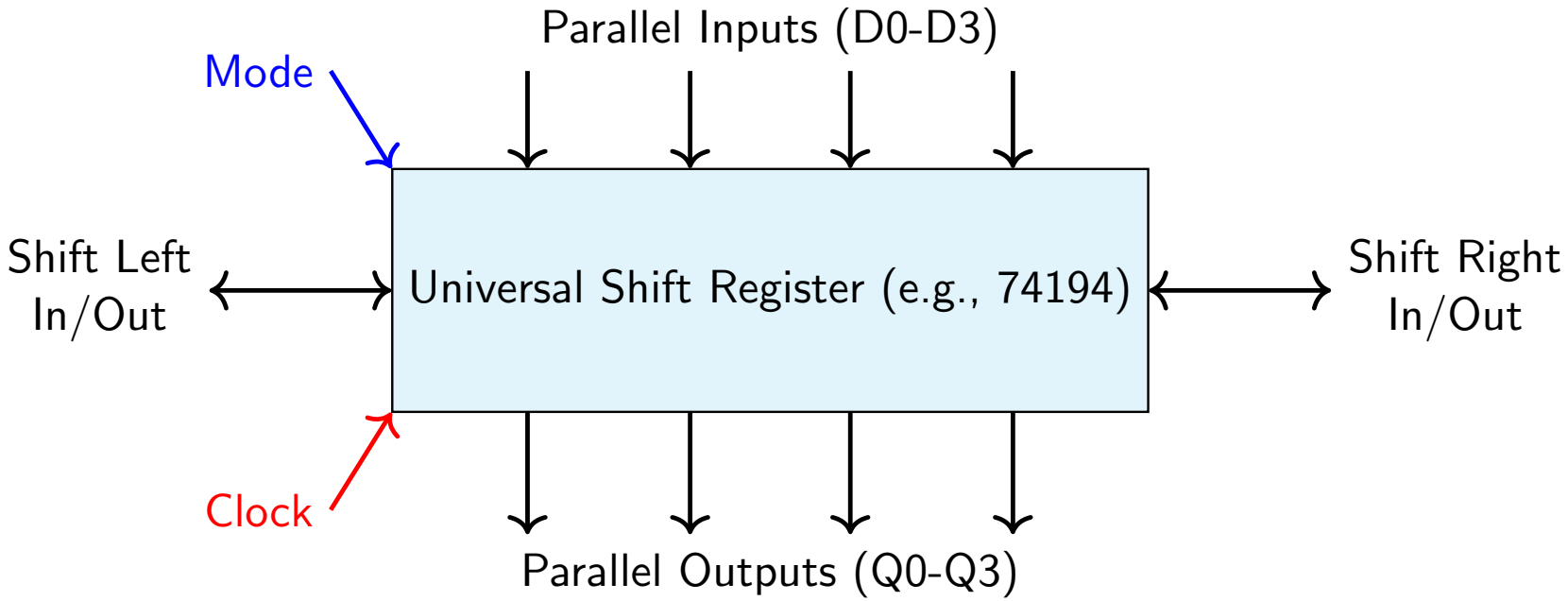
- Circuit: n parallel D flip-flops.
- Inputs: D0, D1, D2, D3 connected directly to external data.
- Outputs: Q0, Q1, Q2, Q3 connected directly to external bus.
- Requires 1 clock cycle to load, 0 clock cycles to read.



Lecture 4.6: Shift Registers: PISO and PIPO

Universal Shift Register

- Combines all functions: SISO, SIPO, PISO, PIPO.
- Can shift left AND right.
- Uses larger multiplexers (4-to-1) at the input of each flip-flop to select:
  1. Parallel Input
  2. Shift Right
  3. Shift Left
  4. Hold Data



Lecture 4.6: Shift Registers: PISO and PIPO

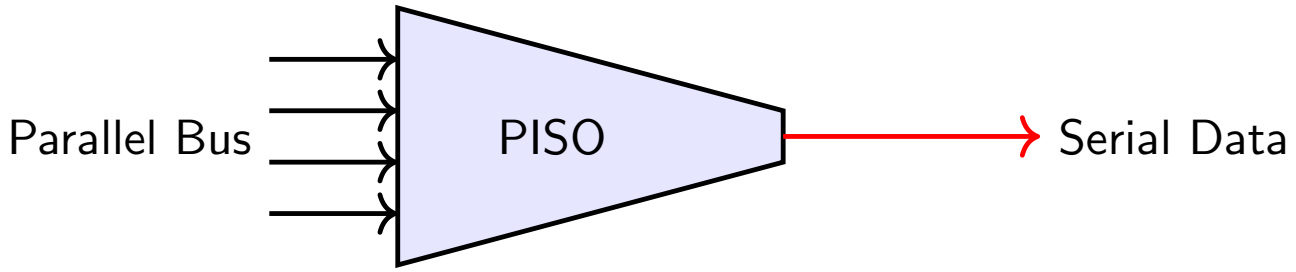
Summary Comparison

- SISO: 1 bit in, 1 bit out. (Delay)
- SIPO: 1 bit in, n bits out. (Serial to Parallel)
- PISO: n bits in, 1 bit out. (Parallel to Serial)
- PIPO: n bits in, n bits out. (Data Storage)

Input Type	**Output Type**	
	Serial Out	**Parallel Out**
Serial In	SISO (Delay / Storage)	SIPO (Demultiplexing)
Parallel In	PISO (Multiplexing)	PIPO (Fast Register)

Summary

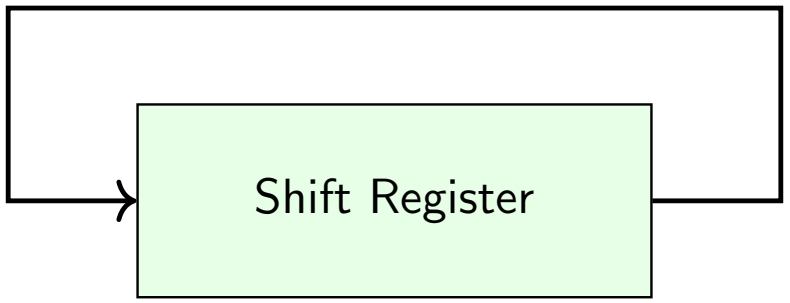
- PISO registers require steering logic (MUX/Gates) to switch between parallel loading and serial shifting.
- PISO is critical for transmitting data over serial communication lines.
- PIPO registers are the standard building blocks for data memory and CPU registers, prioritizing speed.
- Universal shift registers can perform all these tasks and shift bidirectionally.



Next Lecture Preview

- We've moved data. Now let's count.
- Introduction to Counters.
- We will start by modifying shift registers to create specialized counters:
- The Ring Counter and the Johnson Counter.

Feedback Loop (Ring Counter)



Questions?

Lecture 4.7: Counters: Ring and Johnson Counter

Complete Lecture Deck – All 45 Lectures

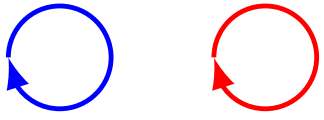
Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

Agenda

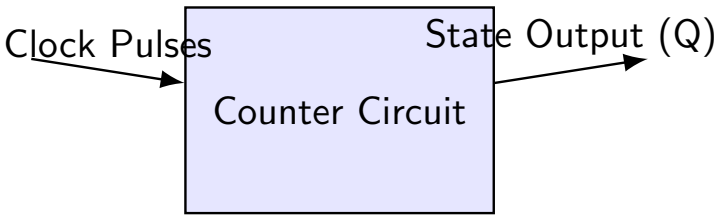
- Introduction to Counters
- Shift Register Counters Overview
- The Ring Counter: Circuit & Operation
- Ring Counter State Table & Waveforms
- Drawbacks of the Ring Counter
- The Johnson Counter: Circuit & Operation
- Johnson Counter State Table
- Comparison: Ring vs Johnson



Lecture 4.7: Counters: Ring and Johnson Counter

Introduction to Counters

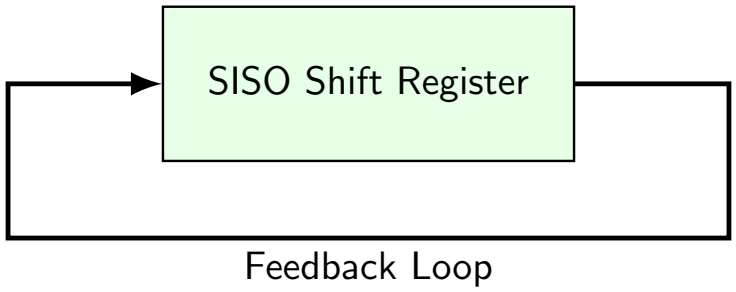
- A counter is a sequential circuit designed to go through a prescribed sequence of states upon the application of input pulses.
- The sequence may follow binary numbers (0,1,2,3...), or any other specific pattern.
- Used for counting events, dividing frequencies, and controlling timing sequences.



Lecture 4.7: Counters: Ring and Johnson Counter

Shift Register Counters

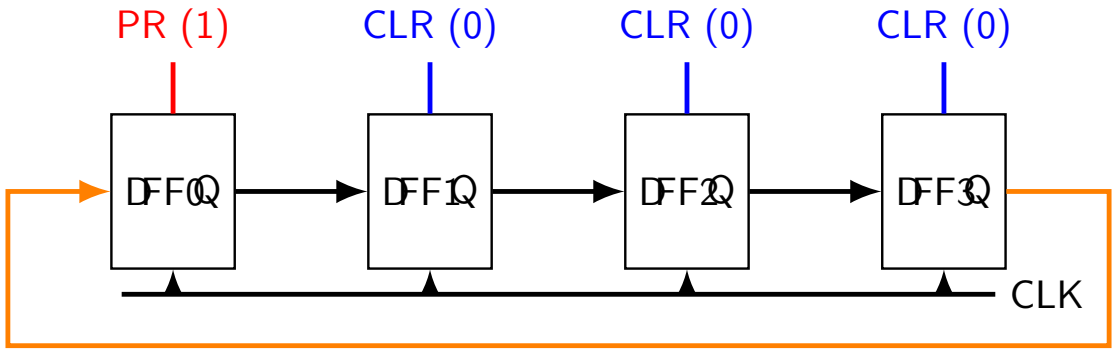
- Normally, data shifts right out of a shift register and is lost.
- If we take the output of the last flip-flop and feed it back to the input of the first, we create a closed loop.
- The data bit pattern simply circulates endlessly.
- The two most common types are Ring Counters and Johnson Counters.



Lecture 4.7: Counters: Ring and Johnson Counter

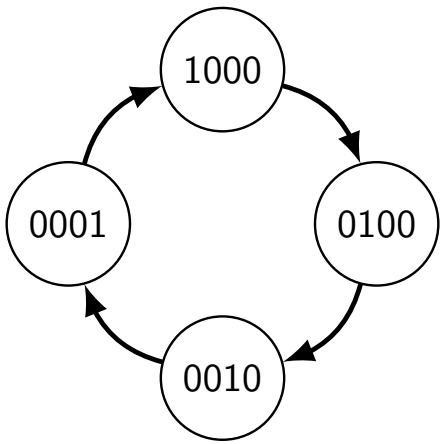
The Ring Counter

- Constructed using a standard shift register.
- The normal output (Q) of the final flip-flop is connected directly to the D input of the first flip-flop.
- Requires an initial 'preset' state. Usually, one flip-flop is set to 1, and the rest are cleared to 0.
- Example initialization: 1000.



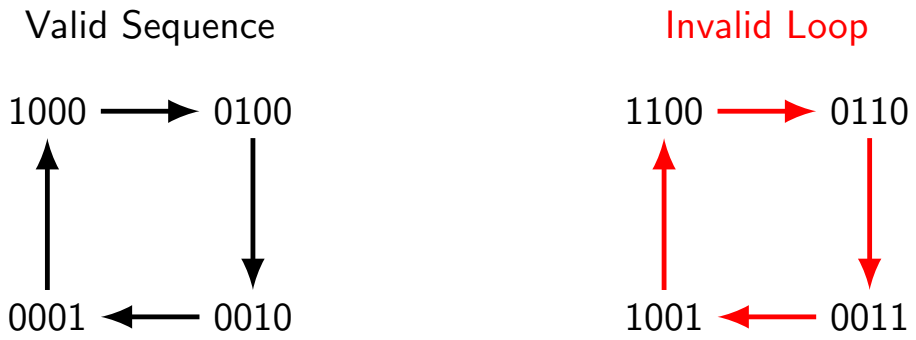
Ring Counter: State Sequence

- Assume a 4-bit counter initialized to Q0=1, Q1=0, Q2=0, Q3=0.
- CLK 0: 1000
- CLK 1: 0100 (The 1 shifts right. Q3's 0 loops to Q0)
- CLK 2: 0010
- CLK 3: 0001
- CLK 4: 1000 (Loops back to start)
- Modulus (number of states) = N (where N is the number of flip-flops).



Ring Counter: Drawbacks

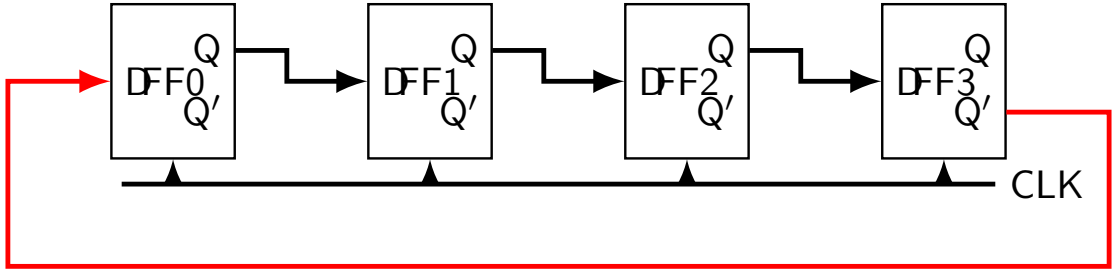
- Inefficient use of flip-flops. 4 flip-flops can theoretically have 16 states, but a Ring counter only uses 4.
- Self-starting problem: If a noise glitch sets it to 1100, it will circulate $1100 \rightarrow 0110 \rightarrow 0011 \rightarrow 1001$ endlessly, never recovering the single '1' sequence.
- Requires extra decoding logic to force it back to 1000 if it enters an invalid state.



Lecture 4.7: Counters: Ring and Johnson Counter

The Johnson (Twisted-Ring) Counter

- An improvement over the Ring counter to get more states.
- The INVERTED output (Q') of the final flip-flop is connected to the D input of the first flip-flop.
- Initial state is usually all zeros: 0000.



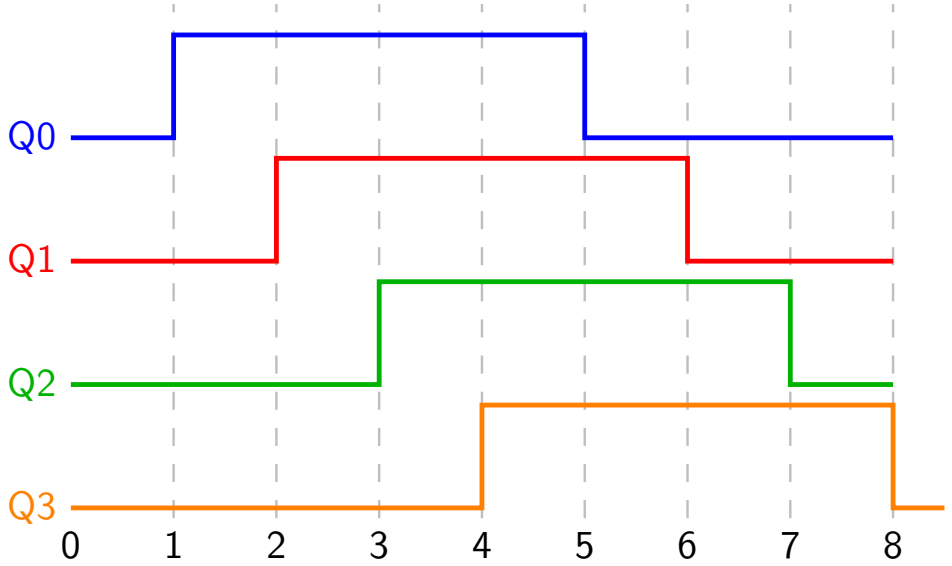
Johnson Counter: State Sequence

- 4-bit sequence starting at 0000:
- CLK 0: 0000 (Q3 is 0, so Q3′ is 1. Feed 1 in)
- CLK 1-4: 1000 $\rightarrow$ 1100 $\rightarrow$ 1110 $\rightarrow$ 1111
- CLK 5-8: 0111 $\rightarrow$ 0011 $\rightarrow$ 0001 $\rightarrow$ 0000 (Back to start)
- Modulus = 2N.

CLK	Q0	Q1	Q2	Q3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1
8	0	0	0	0

Johnson Counter Advantages

- Double the states (2N) compared to a Ring counter (N) for the same hardware.
- Only one bit changes at a time between states (similar to Gray code).
- This means decoding the states using logic gates produces no glitches.
- Often used to generate multi-phase clocks or step motor control sequences.



Decoding a Johnson Counter

- To know what 'number' the counter is at, we use 2-input AND gates.
- Because of the unique sequence, we only need to look at two adjacent flip-flops to uniquely identify any of the 2N states.
- Example: State 1 (1000) is the only state where Q0=1 AND Q1=0.

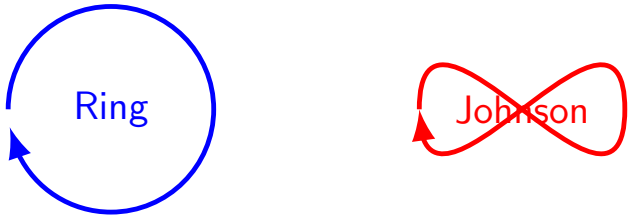
Lecture 4.7: Counters: Ring and Johnson Counter

Summary Comparison

Feature	Ring Counter	Johnson Counter
Feedback	Q to D	Q' to D
Modulus (States)	N	2N
Initial State	One '1' (e.g. 1000)	All '0's (e.g. 0000)
Efficiency	Low	Medium (Still $< 2^N$)
Use Case	Simple scanning	Multi-phase generation

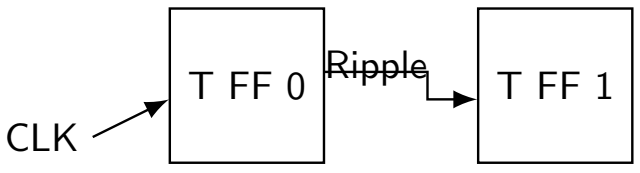
Summary

- By looping the output of a shift register back to its input, we create a sequential sequence generator.
- Ring counters circulate a single active bit, yielding N states.
- Johnson counters cross the feedback wire (Q' to D), yielding a 2N state sequence of filling and emptying.
- Both are simple to design but do not count in standard binary.



Next Lecture Preview

- How do we build a counter that actually counts in binary (0000, 0001, 0010...)?
- Next Lecture: 4-bit Asynchronous (Ripple) Counter.
- We will use T-flip-flops or JK flip-flops to create a true binary sequence.
- Understanding 'ripple' delay.



Questions?

Lecture 4.8: 4-bit Asynchronous (Ripple) Counter

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

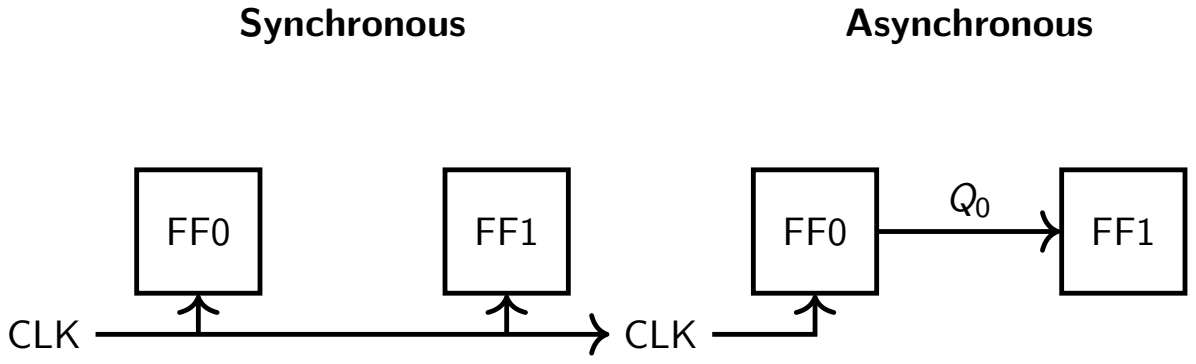
Agenda

- Synchronous vs. Asynchronous Circuits
- The Toggle (T) Flip-Flop Revisit
- Architecture of a Ripple Counter
- Step-by-Step Operation
- Timing Diagram and Waveforms
- The 'Ripple' Effect and Propagation Delay
- Modulus of a Counter
- Summary & Preview

Lecture 4.8: 4-bit Asynchronous (Ripple) Counter

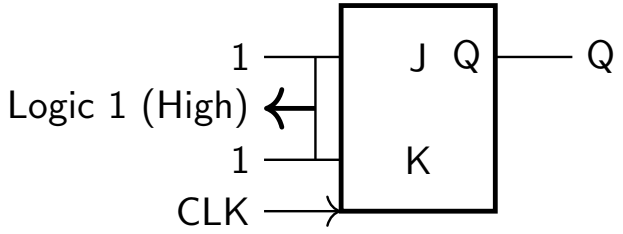
Synchronous vs. Asynchronous

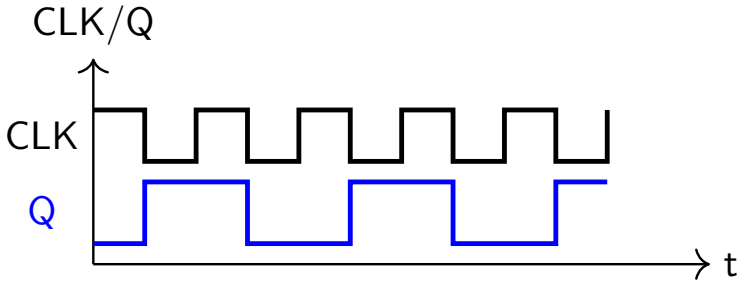
- **Synchronous**: Every flip-flop in the circuit receives the exact same clock signal. All state changes happen simultaneously.
- **Asynchronous**: Only the first flip-flop receives the external clock. Subsequent flip-flops are clocked by the outputs of the previous ones.
- State changes 'ripple' through the circuit one after the other.



Revisiting the T Flip-flop

- We build ripple counters using Toggle (T) flip-flops, or JK flip-flops with $J = 1$ and $K = 1$.
- If $T = 1$, the output toggles state on every active clock edge.
- A T flip-flop essentially divides the input clock frequency by 2.
- Two T flip-flops in series divide by 4. Three divide by 8.

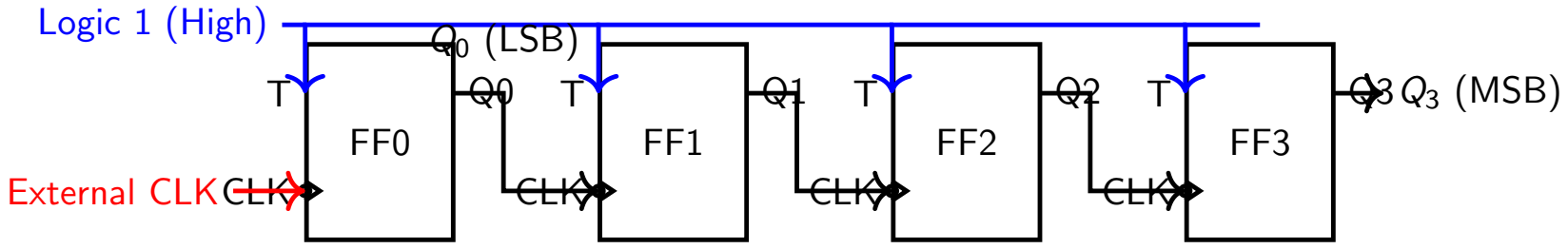




Lecture 4.8: 4-bit Asynchronous (Ripple) Counter

4-bit Ripple Up-Counter Architecture

- Requires 4 Negative Edge-Triggered T flip-flops ($T = 1$ always).
- External CLK connected ONLY to FF0 (LSB - Q0).
- Output Q0 connected to CLK input of FF1.
- Output Q1 connected to CLK input of FF2.
- Output Q2 connected to CLK input of FF3 (MSB - Q3).



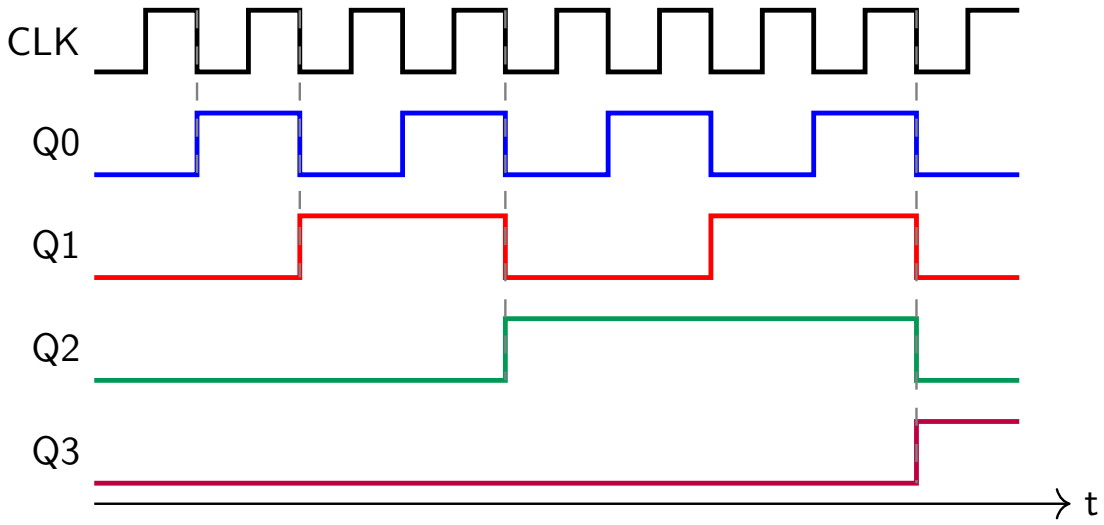
How It Counts (The Math)

- Let's trace the binary sequence:
- **State 0**: 0000
- **Clk 1** $\rightarrow$ Q0 toggles to 1. (0001). Since Q0 went 0 $\rightarrow$ 1, FF1 is NOT triggered.
- **Clk 2** $\rightarrow$ Q0 toggles to 0. Since Q0 went 1 $\rightarrow$ 0 (falling edge), FF1 IS triggered and Q1 toggles to 1. (0010).
- **Clk 3** $\rightarrow$ Q0 toggles to 1. (0011).
- **Clk 4** $\rightarrow$ Q0 toggles to 0, triggering Q1 to toggle to 0, which triggers Q2 to toggle to 1. (0100).

Clock Pulse	Q3	Q2	Q1	Q0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0

Timing Diagram

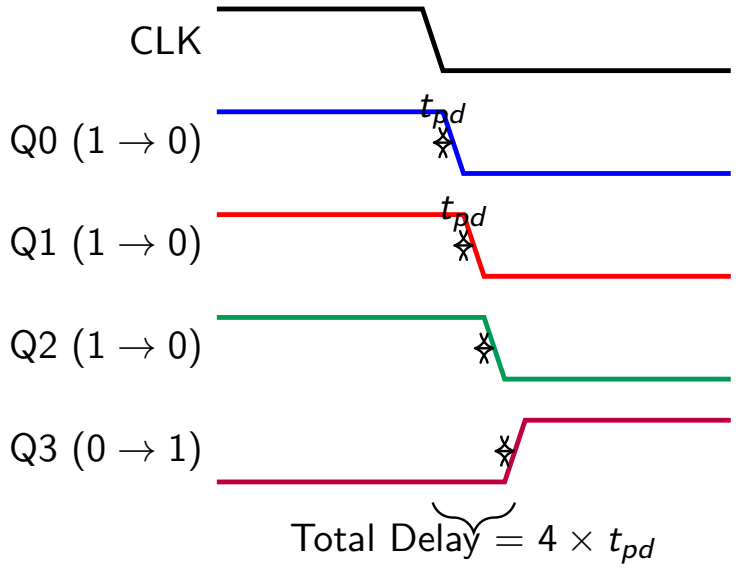
- The timing diagram visually proves the binary sequence.
- Q0 toggles on every falling edge of CLK. Q1 toggles on falling edge of Q0.
- Q2 toggles on falling edge of Q1. Q3 toggles on falling edge of Q2.



Lecture 4.8: 4-bit Asynchronous (Ripple) Counter

The 'Ripple' Delay Problem

- Each flip-flop has a small propagation delay (t_{pd}) between receiving a clock edge and changing its output.
- In asynchronous counters, these delays add up.
- When going from 0111 (7) to 1000 (8), Q0 must flip, then Q1, then Q2, then Q3.
- Total delay $= 4 \times t_{pd}$.
- During this delay, outputs temporarily show incorrect 'garbage' values.



Maximum Frequency Limit

- Because of the cumulative delay, the ripple counter cannot be clocked too fast.
- The entire ripple must settle before the next external clock pulse arrives.
- $T_{clock} > N \times t_{pd}$
- $f_{max} = \frac{1}{N \times t_{pd}}$
- where N is the number of bits.

$$f_{max} \propto \frac{1}{N}$$

As the number of bits (N) increases, the maximum operating frequency (f_{max}) decreases.

Lecture 4.8: 4-bit Asynchronous (Ripple) Counter

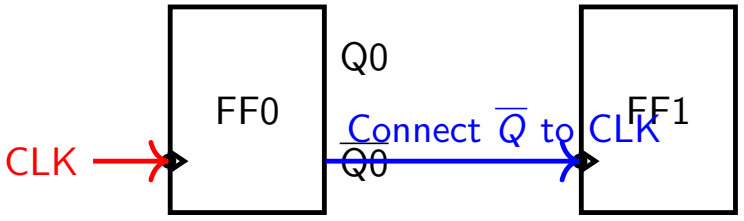
Modulus (MOD) of a Counter

- The Modulus is the number of unique states a counter goes through before repeating.
- An n-bit binary counter has a maximum Modulus of 2^n.
- A 4-bit counter is a MOD-16 counter (counts 0 to 15).
- We can create a counter of ANY modulus (e.g., MOD-10) by forcing it to reset early using an asynchronous CLEAR signal.

4-bit Counter
$2^4 = 16$ states
MOD-16

Ripple Down-Counter

- To make the circuit count backwards (15, 14, 13...):
- Use Positive Edge-Triggered flip-flops (clocked by Q).
- OR
- Keep Negative Edge-Triggered flip-flops, but connect the CLOCK of the next stage to the $\overline{Q}$ (inverted) output instead of Q.
- The logic inverts, causing subtraction instead of addition.



Advantages and Disadvantages

Advantages

- ✓ Extremely simple hardware design.
- ✓ Uses the minimum number of gates/connections.

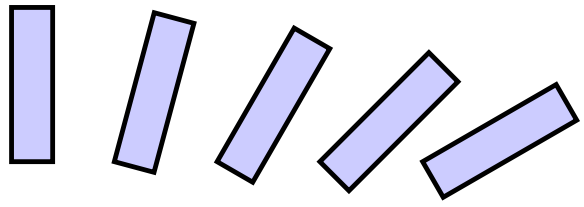
Disadvantages

- ✗ Slow. Cumulative propagation delay limits maximum speed.
- ✗ Generates decoding glitches during the ripple transition.
- ✗ Difficult to design complex state machines around.

Lecture 4.8: 4-bit Asynchronous (Ripple) Counter

Summary

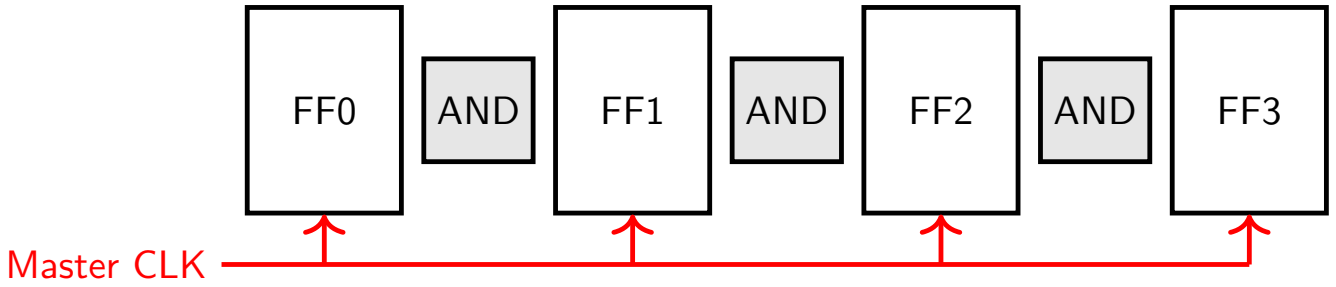
- Asynchronous counters cascade the clock signal through the flip-flops.
- A 4-bit ripple up-counter uses 4 T-flip-flops, with each Q driving the next CLK.
- It naturally counts in binary from 0 to 15 (MOD-16).
- The ripple effect causes cumulative propagation delays, severely limiting its maximum operating frequency.



Domino Effect of Propagation Delay

Next Lecture Preview

- How do we solve the ripple delay?
- **Next Lecture:** 4-bit Synchronous Up/Down Counter.
- We will connect the clock to EVERY flip-flop simultaneously.
- We will use combinational logic to decide when each flip-flop should toggle.



Questions?

Lecture 4.9: 4-bit Synchronous Up/Down Counter

Complete Lecture Deck – All 45 Lectures

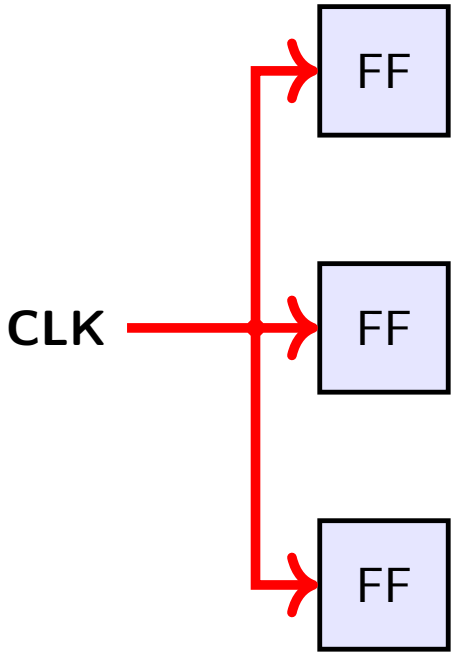
Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

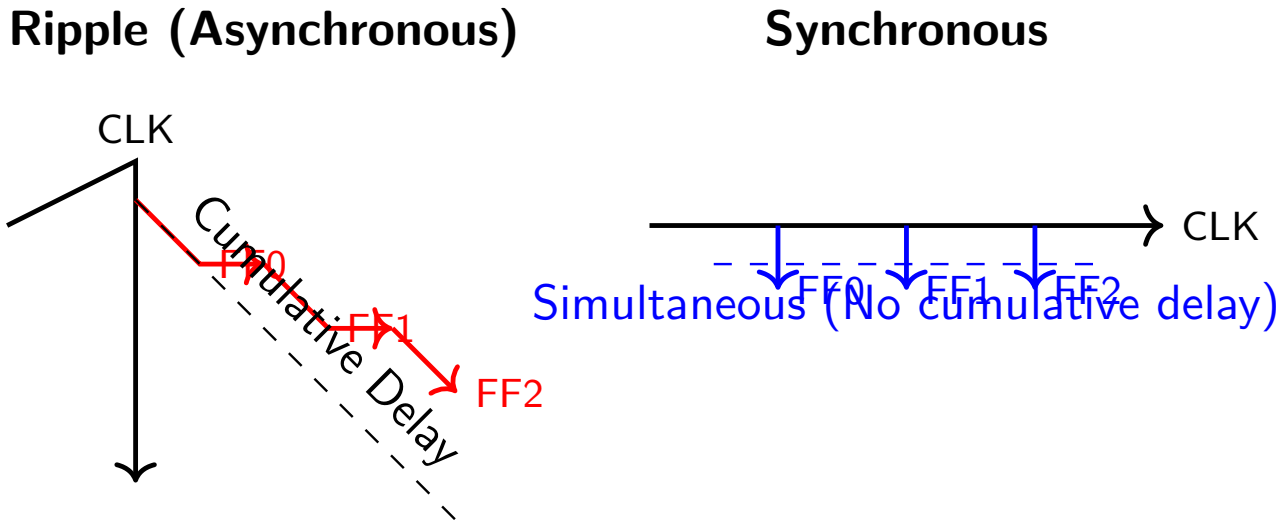
Agenda

- The Synchronous Advantage
- Designing a Synchronous Up-Counter
- Combinational Logic for Toggling
- Designing a Synchronous Down-Counter
- The Up/Down Control Signal
- Combined Architecture
- Performance Comparison
- Summary & Preview



The Synchronous Advantage

- External clock signal is connected to the CLK pin of EVERY flip-flop.
- Flip-flops meant to change state will do so at the exact same instant.
- Total delay = One FF delay + combinational logic gate delay.
- Result: Higher max clock frequency; no intermediate 'glitch' states.



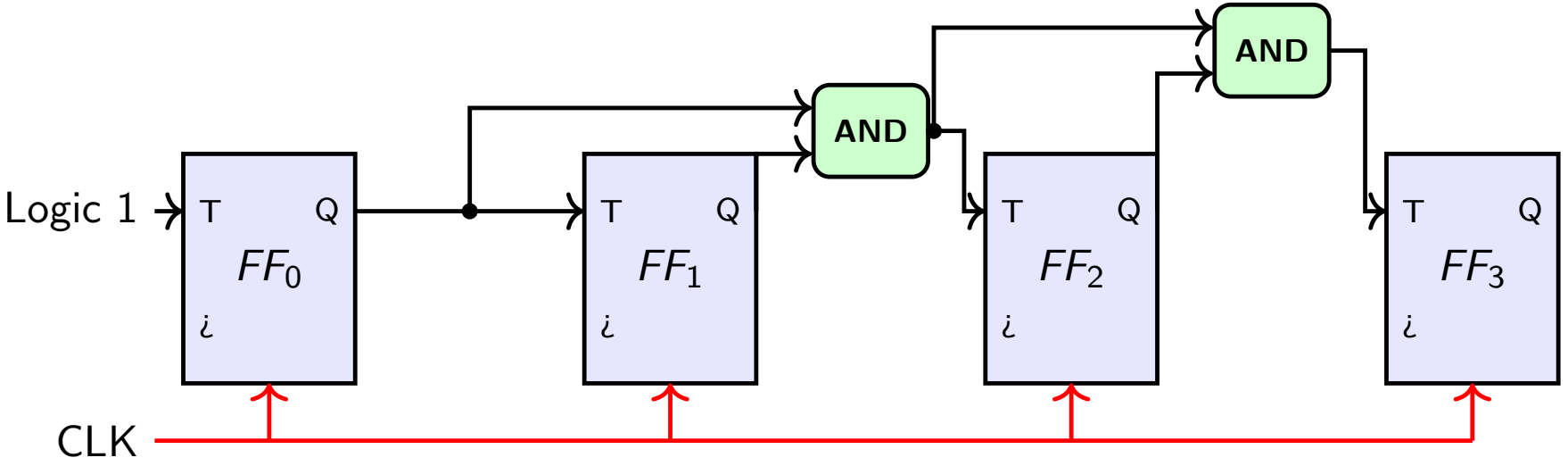
How to Count Synchronously

- How do we stop them from all toggling every cycle?
- Use T (Toggle) or JK inputs to control WHO toggles next.
- Use Combinational Logic (AND gates) based on current state (Q).
- **Rule for Up-Counting:** A bit toggles only if ALL preceding bits are 1.

Current State				Next State				Action
Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	
0	0	1	1	0	1	0	0	Q_2 toggles because Q_0 AND $Q_1 = 1$

Synchronous Up-Counter Circuit

- FF0 (LSB): $T_0 = 1$ (Toggles every clock).
- FF1: $T_1 = Q_0$ (Toggles only when Q_0 is 1).
- FF2: $T_2 = Q_0 \cdot Q_1$ (Toggles only when both are 1).
- FF3 (MSB): $T_3 = Q_0 \cdot Q_1 \cdot Q_2$.



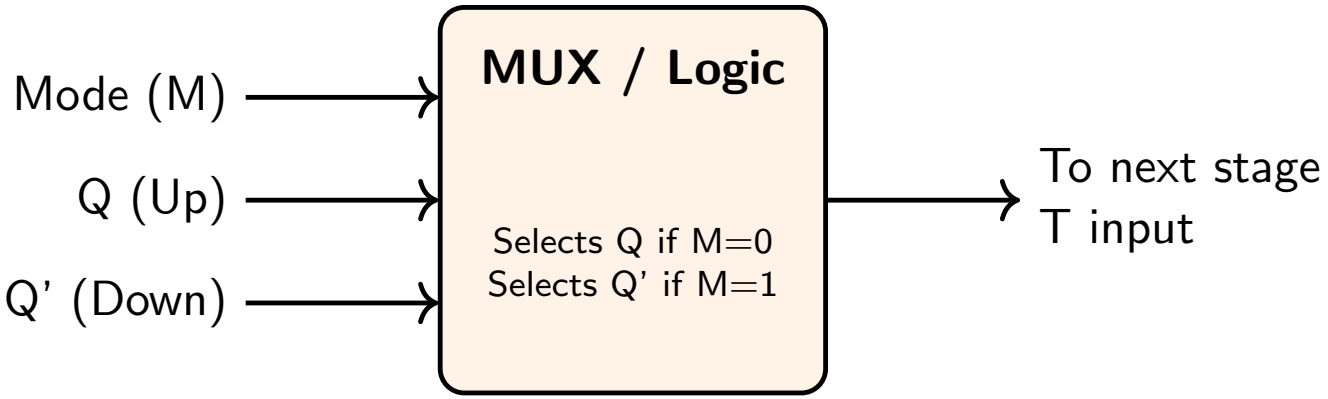
Synchronous Down-Counter Logic

- How do we count backward (15, 14, 13...)?
- **Rule for Down-Counting:** A bit toggles only if ALL preceding bits are 0.
- Instead of looking at Q, we look at the inverted Q' (Q-bar) outputs.
- FF0: $T_0 = 1$ FF1: $T_1 = Q_0'$ FF2: $T_2 = Q_0' \cdot Q_1'$

Current State				Next State				Action
Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	
1	0	0	0	0	1	1	1	Q_3 toggles to 0 because $Q_0, Q_1, Q_2 = 0$

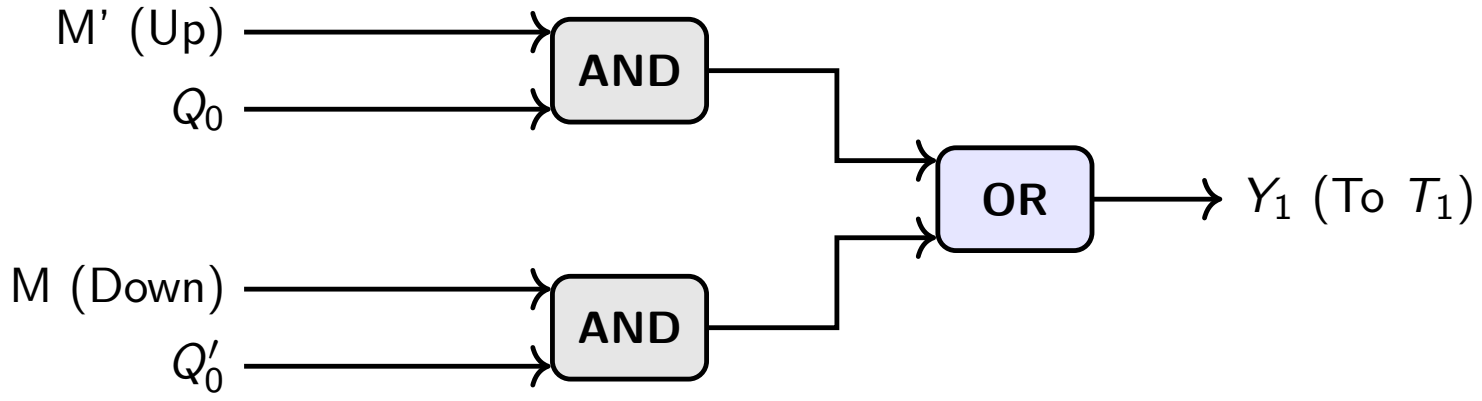
The Up/Down Control Signal

- We want ONE circuit that can do both, controlled by a switch 'M' (Mode).
- If $M = 0$: Count Up (Use Q outputs for logic).
- If $M = 1$: Count Down (Use Q' outputs for logic).
- Requires 2-to-1 Multiplexer logic (AND-OR gates) at each stage.



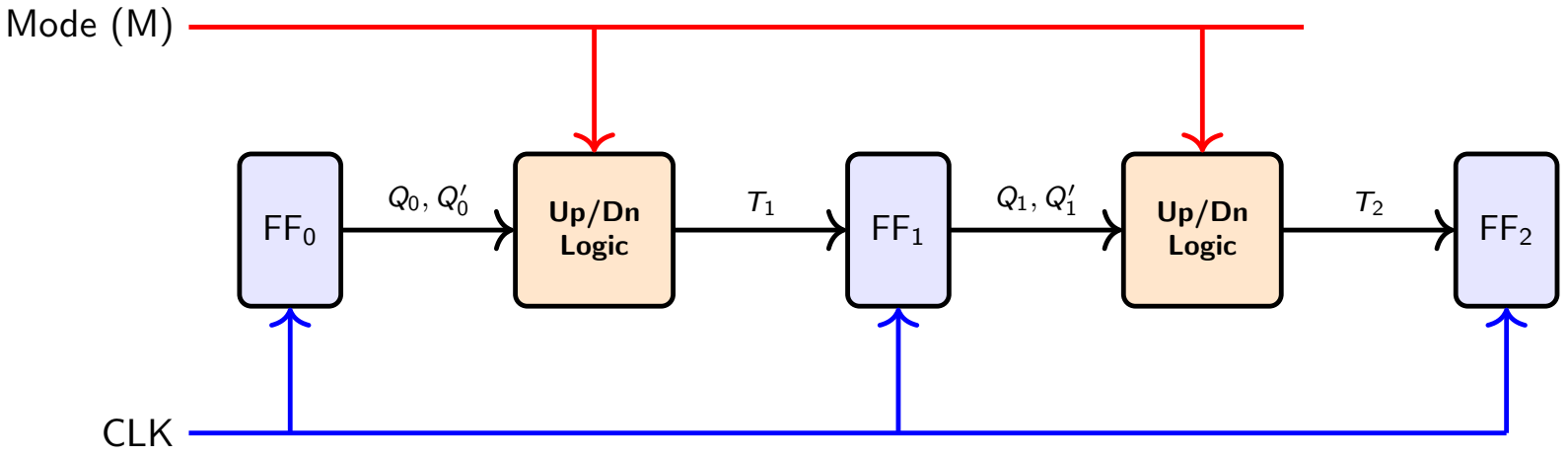
Up/Down Multiplexer Logic

- For stage 1, the toggle signal Y_1 is:
- $Y_1 = (M' \cdot Q_0) + (M \cdot Q_0')$
- If M=0 (Up), $Y_1 = Q_0$.
- If M=1 (Down), $Y_1 = Q_0'$.
- This logic is repeated for every stage, cascaded through AND gates.



Combined Up/Down Circuit Architecture

- Final schematic combines synchronous clocking, cascaded AND logic, and MUX.
- Highly reliable, high-speed, bidirectional counting.
- Commonly found in ICs like the 74LS191 (4-bit Synchronous Up/Down Counter).



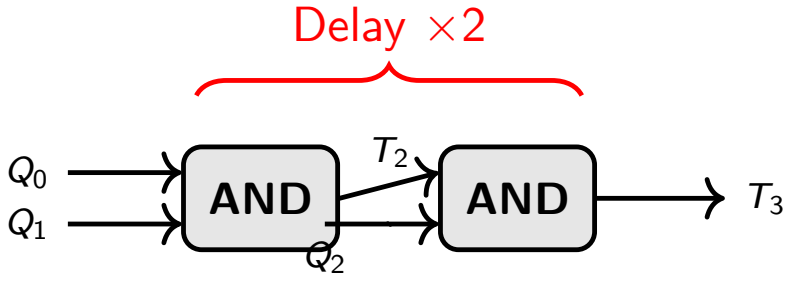
Performance: Ripple vs Synchronous

Feature	Ripple (Asynchronous)	Synchronous
Hardware	Simpler (fewer logic gates)	Complex (many AND/OR gates)
Clocking	Cascaded (Output to next CLK)	Simultaneous (Common CLK)
Delay	$N \times t_{pd}$ (slow for large N)	$1 \times t_{pd}$ + Logic delay (fast)
Glitches	Generates glitches	No glitches, strict timing

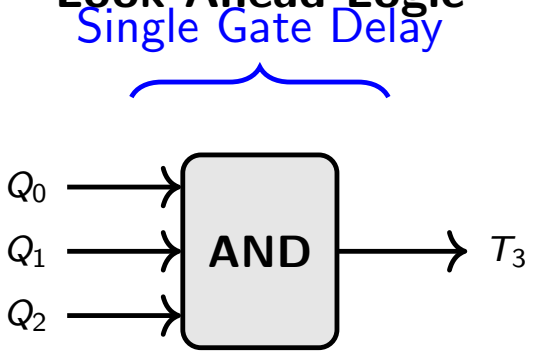
Look-Ahead Carry in Counters

- AND gates cascade. T_3 signal must pass through T_1, then T_2 AND gates.
- For very large counters (e.g., 32-bit), this gate delay becomes significant.
- 'Look-Ahead' logic calculates T_3 directly from Q_0, Q_1, Q_2 to eliminate ripple.

Cascaded Logic (Ripple)

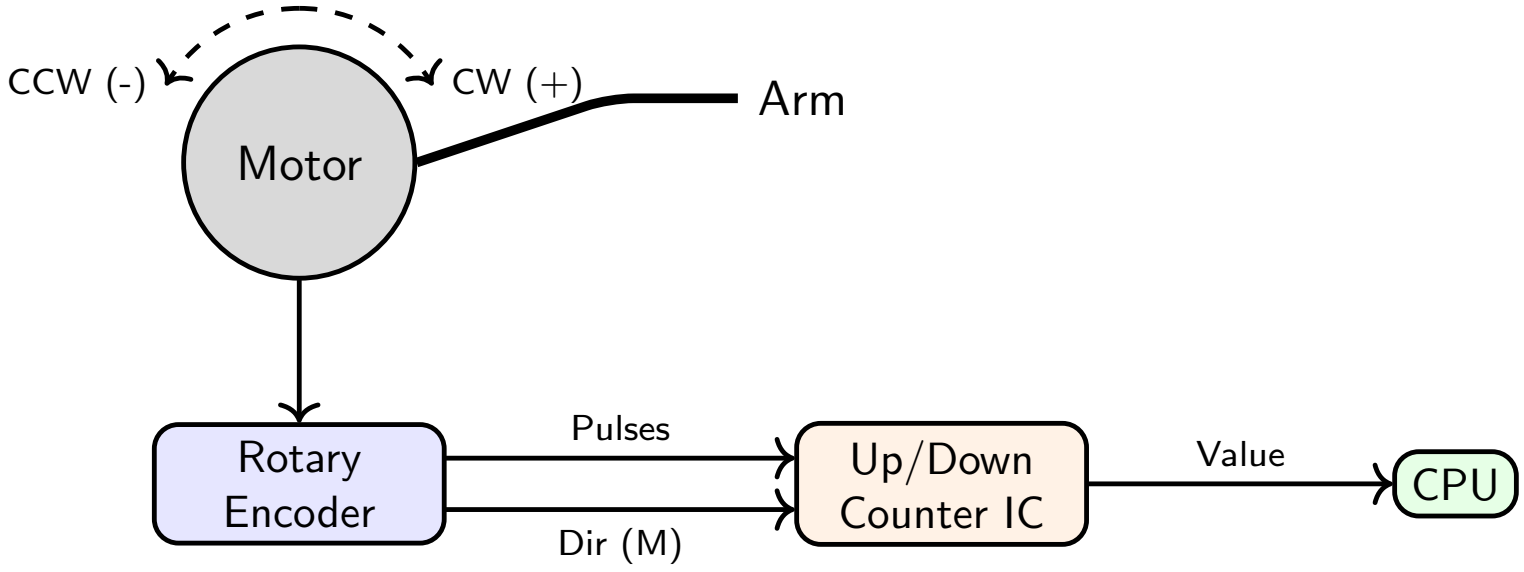


Look-Ahead Logic



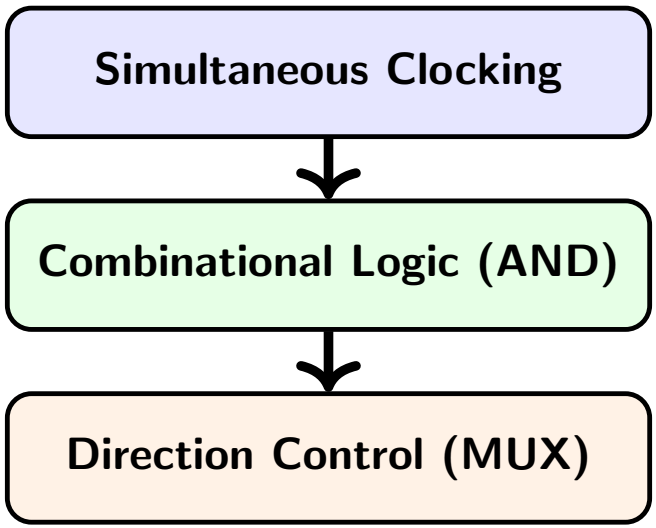
Applications of Up/Down Counters

- **Digital Timers:** Counting down from a set value to 0.
- **Position Encoders:** Tracking a motor spinning clockwise (up) or counter-clockwise (down).
- **Analog-to-Digital Converters:** Adjusting binary value up or down to match analog voltage.



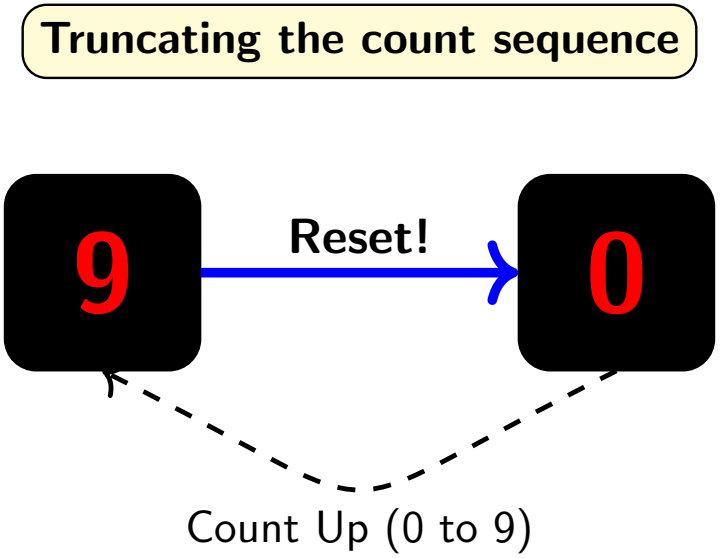
Summary

- Synchronous counters eliminate ripple delay by clocking all FFs simultaneously.
- Toggle decisions are handled by combinational logic networks.
- Up-counting: Q outputs used. Down-counting: Q' outputs used.
- A Mode switch and MUX logic allow dynamic bidirectional counting.



Next Lecture Preview

- We've counted 0 to 15 (MOD-16). But humans count in base-10 (0 to 9).
- **Next Lecture:** BCD / Decade Counter.
- How to force a 4-bit counter to reset back to 0000 after 1001 (9).
- Understanding asynchronous resets in digital design.



Questions?

Lecture 4.10: BCD / Decade Counter

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

Agenda

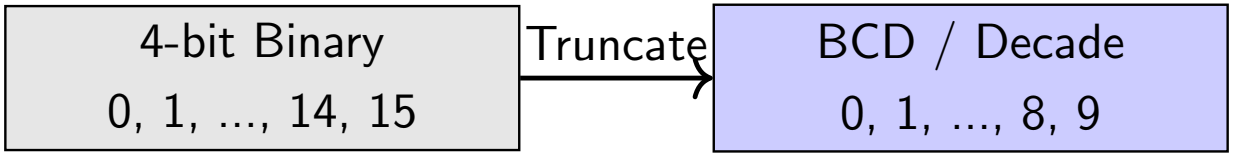
- What is a Decade (MOD-10) Counter?
- The Binary Coded Decimal (BCD) Sequence
- Truncating a Counter Sequence
- Asynchronous Reset Logic
- Designing the BCD Ripple Counter
- State Diagram & Waveforms
- Applications (Digital Clocks/Displays)
- Summary & Unit 4 Conclusion

MOD-10

Lecture 4.10: BCD / Decade Counter

What is a Decade / MOD-10 Counter?

- A standard 4-bit counter has 16 states (0 to 15) $\rightarrow$ MOD-16.
- A Decade counter only has 10 states (0 to 9) $\rightarrow$ MOD-10.
- It counts: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, and then rolls over directly back to 0.
- Since 9 is 1001 in binary, we need 4 flip-flops (3 flip-flops only count up to 7).



10–15 Crossed Out!

Lecture 4.10: BCD / Decade Counter

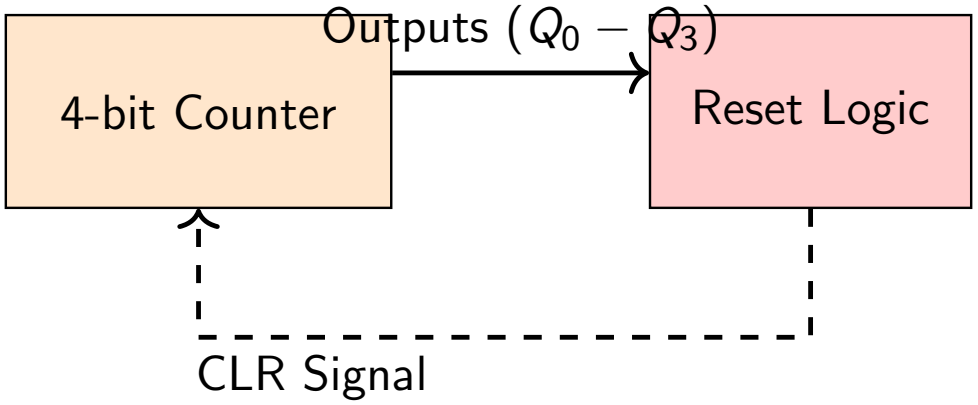
The BCD Sequence

- BCD stands for Binary Coded Decimal.
- It represents each decimal digit (0-9) with a 4-bit binary code.
- 0 = 0000, 1 = 0001, ..., 9 = 1001.
- States 1010 (10) through 1111 (15) are INVALID states in BCD.
- A BCD counter is simply a counter that perfectly follows this sequence.

Decimal	BCD	Decimal	BCD
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

How to Truncate the Sequence

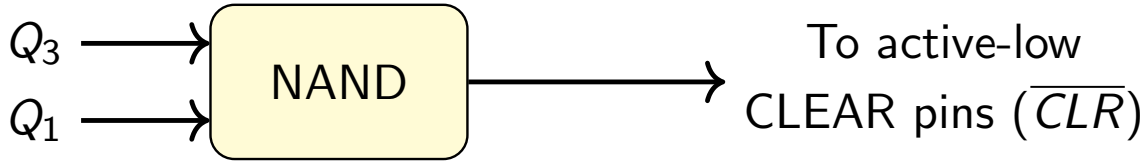
- We start with a standard 4-bit Asynchronous (Ripple) Up-Counter.
- We want it to count normally up to 9 (1001).
- On the next clock pulse, it will naturally try to go to 10 (1010).
- We must detect the moment it hits 1010, and instantly use the CLEAR (CLR) pins on all flip-flops to force them back to 0000.



Lecture 4.10: BCD / Decade Counter

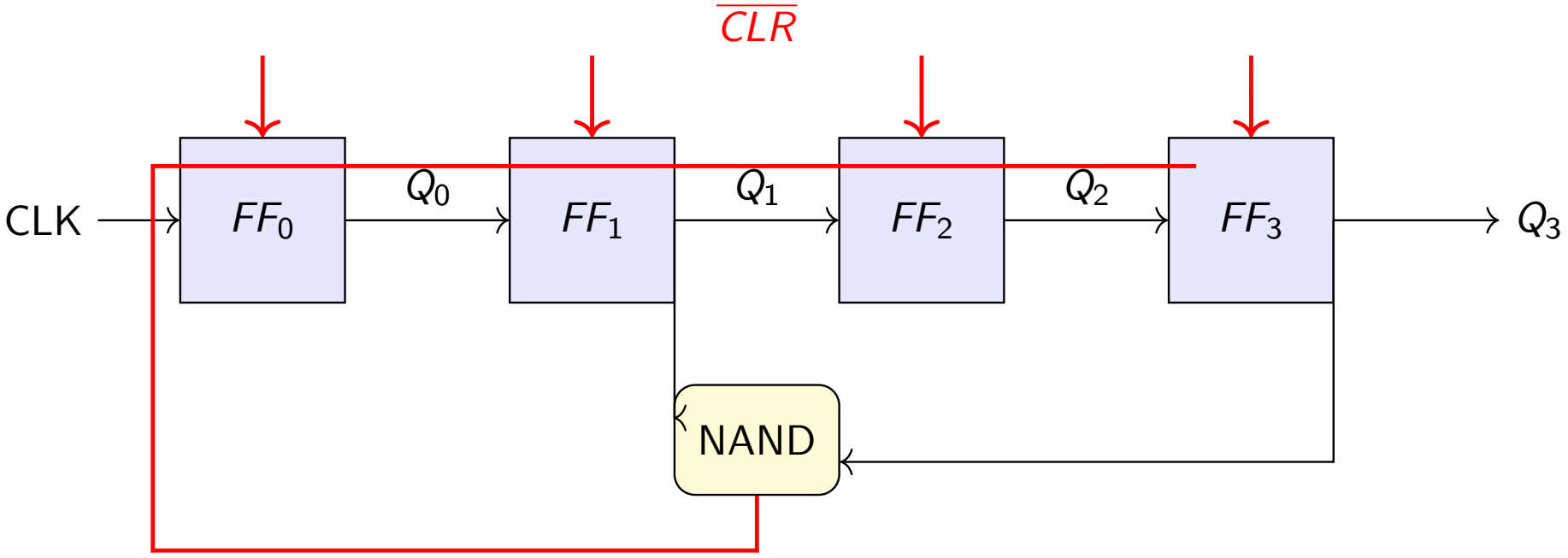
Designing the Reset Logic

- Target state to detect: 10 (Binary: 1010).
- In 1010, $Q_3 = 1$, $Q_2 = 0$, $Q_1 = 1$, $Q_0 = 0$.
- We need a logic gate that outputs a Reset signal ONLY when this specific combination occurs.
- We use a NAND gate connected to Q_3 and Q_1.
- Why only Q_3 and Q_1? Because 1010 is the FIRST time in the counting sequence (0-9) that both Q_3 and Q_1 are high simultaneously.



BCD Ripple Counter Circuit

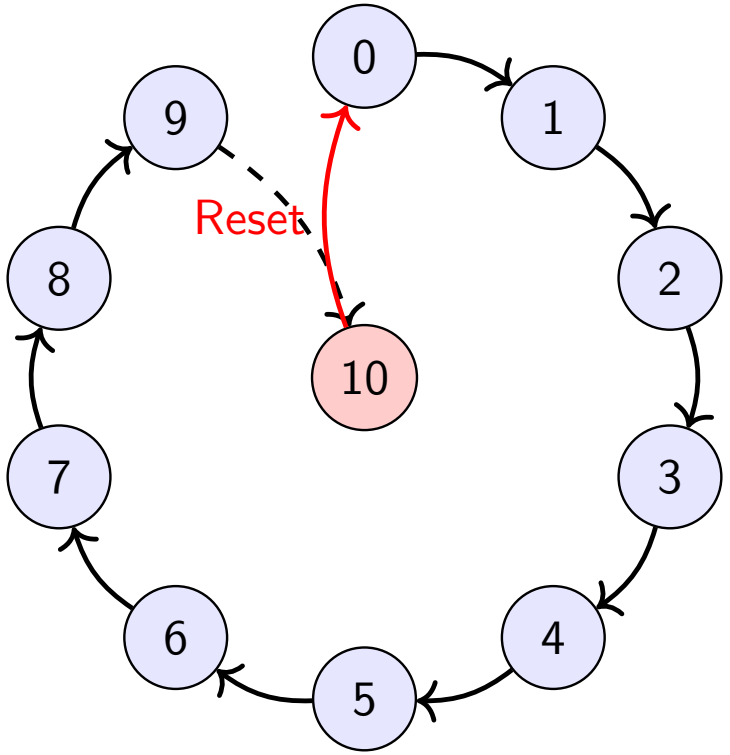
- 4 Negative-edge triggered T flip-flops acting as a ripple counter.
- A 2-input NAND gate takes inputs from Q_3 and Q_1.
- The output of the NAND gate connects to the active-low CLR' (Clear) inputs of all 4 flip-flops.
- When $Q_3 = 1$ and $Q_1 = 1$, NAND outputs 0, clearing all flip-flops to 0.



Lecture 4.10: BCD / Decade Counter

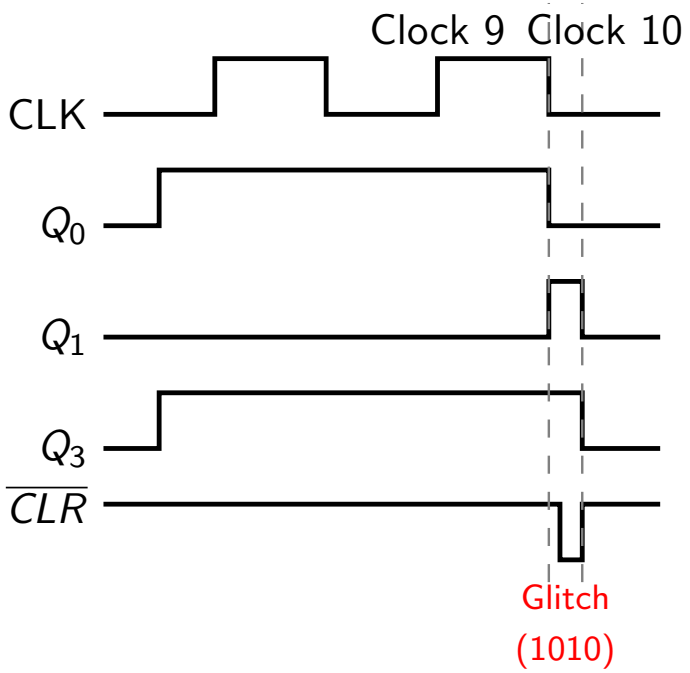
State Diagram

- The state diagram shows a main circular loop from 0000 to 1001.
- From 1001, the arrow points to 1010.
- However, 1010 is a transient state. It exists only for the propagation delay time of the NAND gate.
- An immediate forced transition arrow goes from 1010 to 0000.



Timing Diagram of the Reset

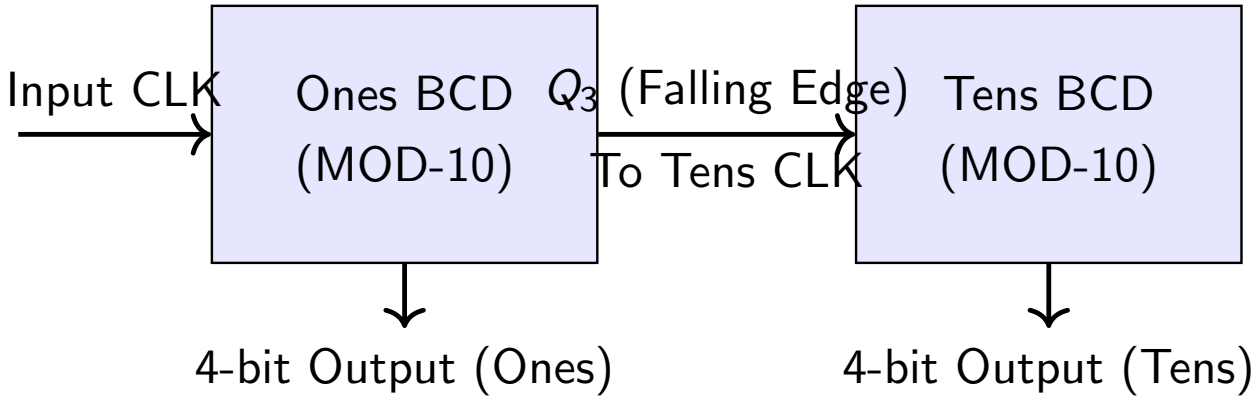
- Observe the transition from Clock 9 to 10.
- At Clock 9: Output is 1001.
- At Clock 10 falling edge: Q_0 toggles to 0, which toggles Q_1 to 1. Output is briefly 1010.
- NAND gate detects $Q_3 = 1, Q_1 = 1$, sends CLR=0.
- Q_3 and Q_1 are instantly forced to 0. Output becomes 0000.
- The "glitch" at 1010 is very short and usually imperceptible.



Lecture 4.10: BCD / Decade Counter

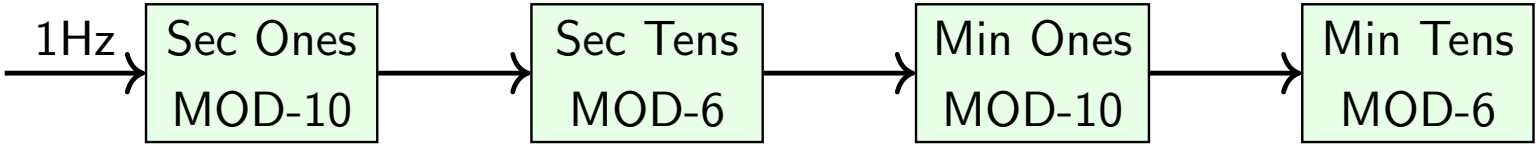
Cascading BCD Counters

- How do we count to 99?
- We cascade two BCD counters.
- The first counter counts the "Ones" (0-9).
- We take the Q_3 output of the "Ones" counter (which drops from 1 to 0 when resetting from 9 to 0).
- We feed this falling edge into the CLOCK input of the "Tens" counter.



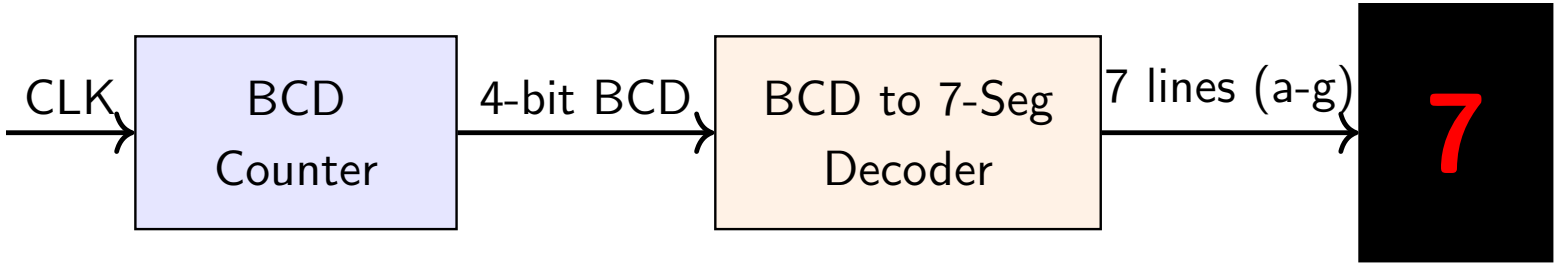
Applications: Digital Clocks

- Digital clocks are just cascaded truncated counters.
- Seconds (Ones): BCD counter (MOD-10) $\rightarrow$ counts 0-9.
- Seconds (Tens): MOD-6 counter (resets at 6) $\rightarrow$ counts 0-5.
- Together they count 00 to 59.
- The rollover of the Seconds drives the Minutes counter.



Driving a 7-Segment Display

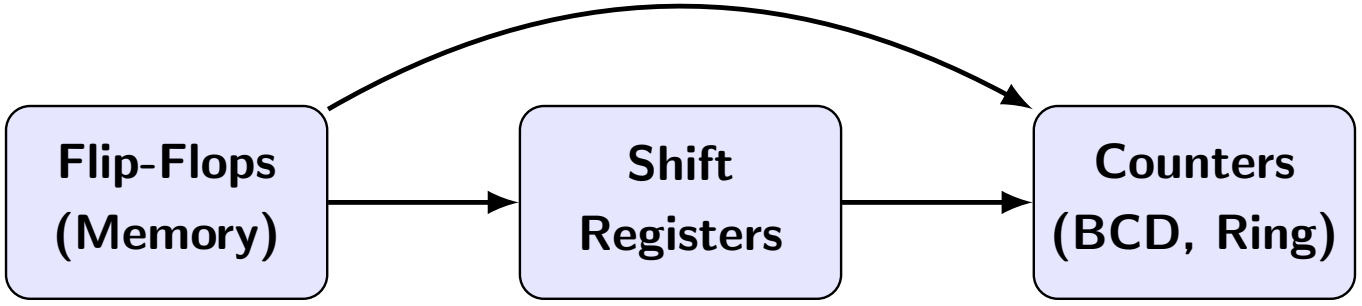
- The 4-bit output of the BCD counter (e.g., 0111 for 7) cannot directly light up the LEDs of a display.
- We use a "BCD to 7-Segment Decoder" combinational IC (like the 7447).
- It translates the 4-bit BCD into 7 distinct signals to light up the correct segments (a,b,c,d,e,f,g).



Lecture 4.10: BCD / Decade Counter

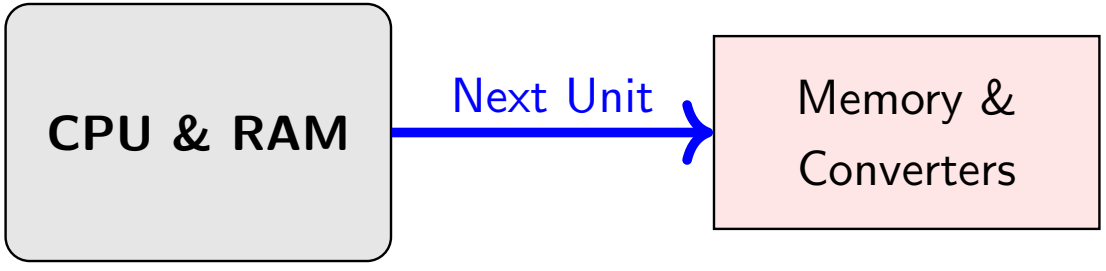
Summary of Unit 4

- We started with flip-flops (SR, D, JK, T) adding memory to circuits.
- We learned about clock triggering (level vs edge) and Master-Slave configs.
- We cascaded flip-flops to move data (Shift Registers: SISO, SIPO, PISO, PIPO).
- We added feedback to create Counters (Ring, Johnson, Ripple, Synchronous, and BCD).



Course Next Steps Preview

- Congratulations on completing Unit 4: Sequential Logic Circuits.
- You now understand both Combinational (Unit 3) and Sequential (Unit 4) logic.
- These are the foundational building blocks of modern CPUs, RAM, and all digital computing.
- Next units will explore Memory Devices and A/D D/A Converters.



Questions?

Lecture 5.1: Introduction and Types of Memory: RAM (SRAM, DRAM)

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

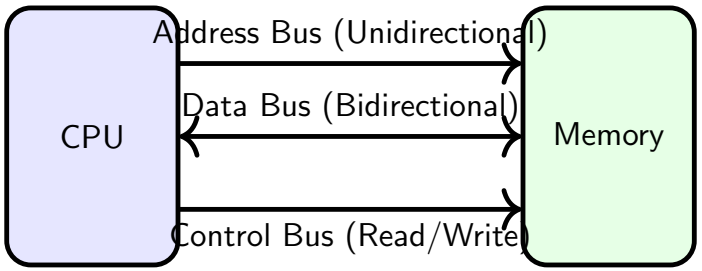
Table of Contents

Agenda

- Introduction to Digital Memory
- Memory Organization (Bits, Bytes, Words)
- Classification of Memory Devices
- Introduction to RAM
- Static RAM (SRAM) - Concept & Structure
- Dynamic RAM (DRAM) - Concept & Structure
- SRAM vs. DRAM Comparison

Introduction to Digital Memory

- Digital Memory is a physical device capable of storing information temporarily or permanently.
- It consists of electronic components that store digital data in the form of binary bits (0s and 1s).
- Memory is a critical component of any digital system or computer, providing the workspace for processing data.
- **Read Operation**: Retrieving stored data from memory.
- **Write Operation**: Storing new data into memory.



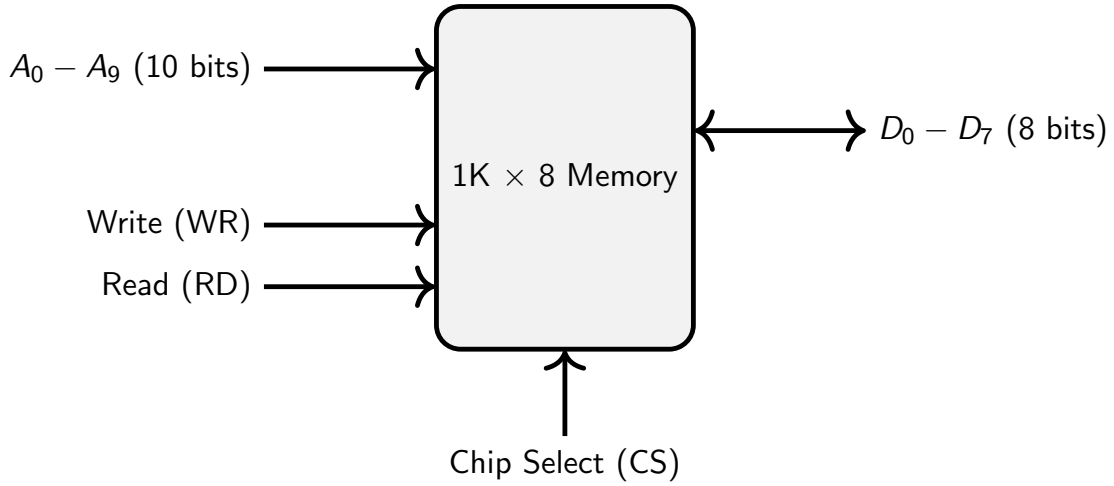
Basic Memory Concepts

- **Bit**: The smallest unit of data, a binary digit (0 or 1).
- **Byte**: A group of 8 bits.
- **Word**: A group of bits that a digital system processes simultaneously (e.g., 16-bit, 32-bit, or 64-bit word).
- **Memory Capacity**: Total number of bits or bytes a memory device can store.
- **Address**: A unique binary number that identifies a specific location in memory.

Unit	**Size**
Bit	1 binary digit (0 or 1)
Nibble	4 bits
Byte	8 bits
Word	Typically 16, 32, or 64 bits
Double Word	2 $\times$ Word size

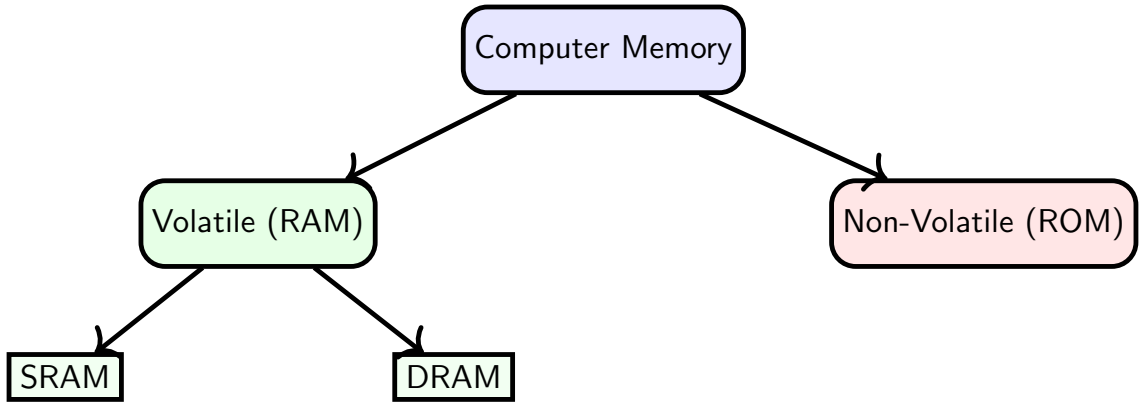
Memory Capacity and Organization

- Memory is organized as an array of locations or 'words'.
- If a memory has N address lines, it can access 2^N locations.
- If each location stores M bits, the total capacity is $2^N \times M$ bits.
- Example: A memory chip with 10 address lines and 8 data lines has a capacity of $2^{10} \times 8 = 1024 \times 8 = 8\text{K}$ bits (or 1 KByte).



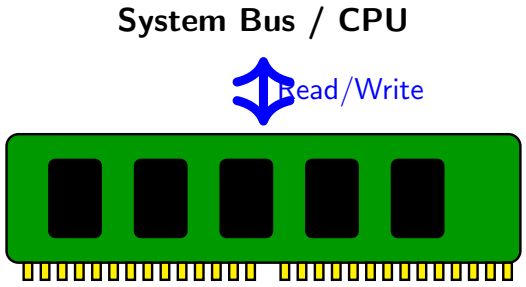
Types of Memory

- **Volatile Memory**: Loses its stored data when power is turned off (e.g., RAM).
- **Non-Volatile Memory**: Retains its stored data even when power is turned off (e.g., ROM, Flash memory).
- **Random Access Memory (RAM)**: Any location can be accessed in the same amount of time, regardless of its physical position.
- **Sequential Access Memory**: Locations must be accessed in sequence (e.g., Magnetic Tape).



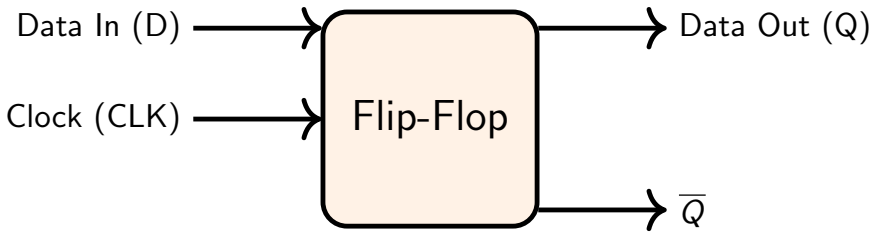
Introduction to RAM

- RAM stands for Random Access Memory.
- It is the main memory in computer systems used for active processing.
- RAM is read/write memory; data can be easily modified.
- It is volatile: power loss means data loss.
- Two main types of RAM: Static RAM (SRAM) and Dynamic RAM (DRAM).



Static RAM (SRAM)

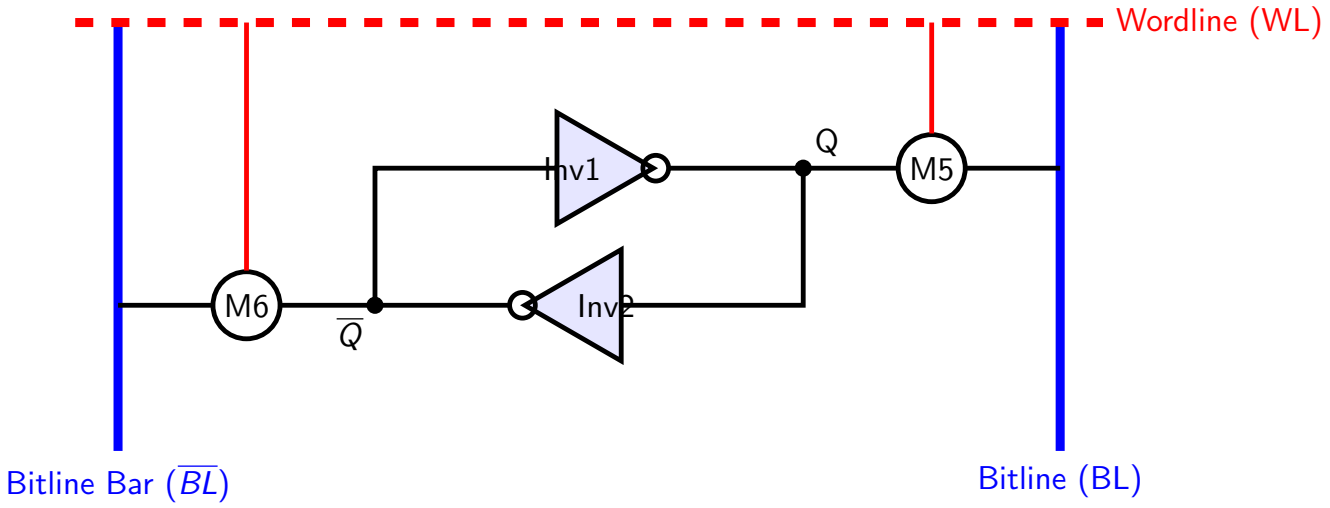
- Stores data using flip-flop logic circuits.
- Data remains 'static' and is held as long as power is continuously supplied.
- Does not require periodic refreshing to retain data.
- Very fast access times compared to DRAM.
- Consumes more power and takes up more silicon area per bit.



SRAM uses flip-flops to latch and hold data

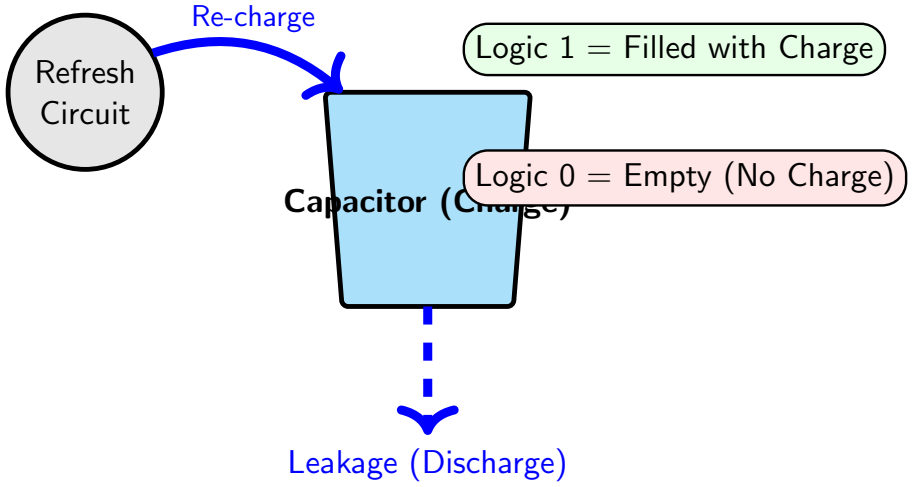
SRAM Cell Design

- A typical SRAM cell uses 6 transistors (6T).
- Four transistors form two cross-coupled inverters to store the state (0 or 1).
- Two additional access transistors control read and write operations during access.
- Because of the 6 transistors, SRAM density is relatively low (fewer bits per chip).



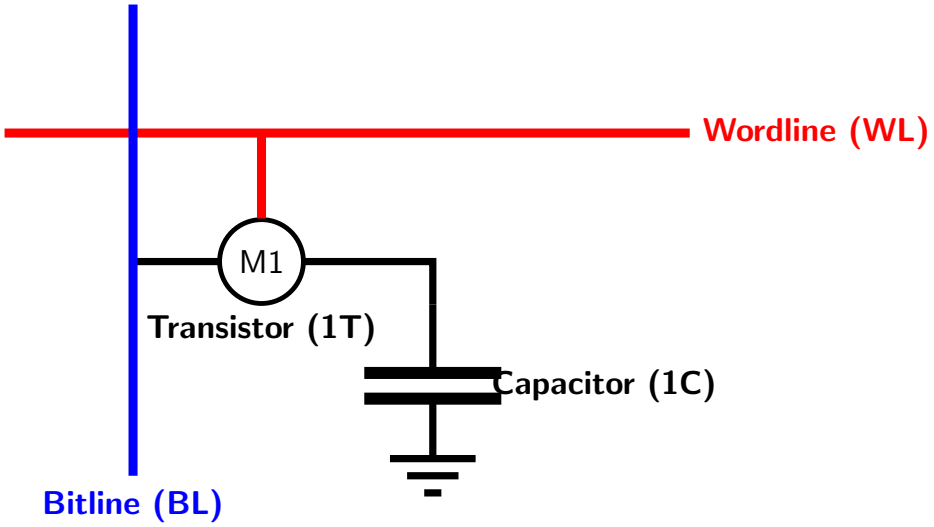
Dynamic RAM (DRAM)

- Stores data as an electrical charge on a tiny capacitor.
- Presence of charge = Logic 1; Absence of charge = Logic 0.
- Capacitors slowly leak charge over time.
- Requires periodic 'refreshing' (reading and rewriting) thousands of times per second to retain data.
- Much higher storage density than SRAM.



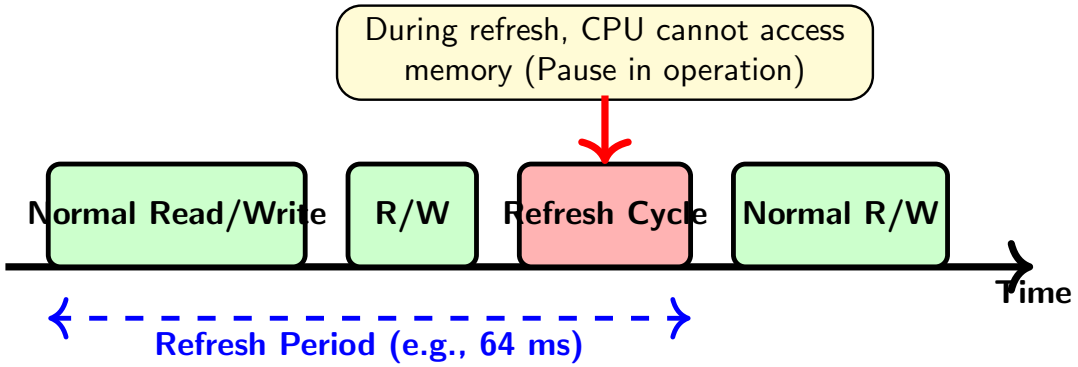
DRAM Cell Design

- A typical DRAM cell uses just 1 transistor and 1 capacitor (1T1C).
- The transistor acts as a switch to let charge in or out of the capacitor.
- Fewer components mean smaller cell size.
- Smaller cell size allows for massive memory capacities on a single chip (Gigabytes).



DRAM Refresh Operation

- Memory controllers must periodically read every row in the DRAM.
- The sense amplifiers read the weakened charge and write back a full charge.
- This process typically happens every few milliseconds (e.g., 64ms).
- During a refresh cycle, the memory cannot be accessed by the CPU, causing slight delays.



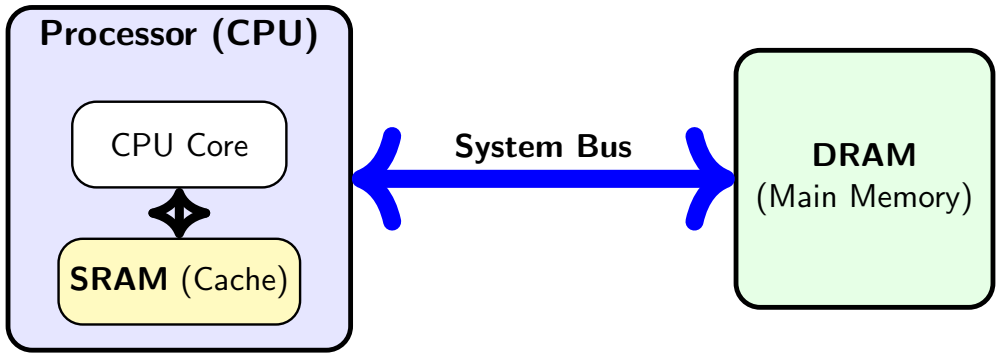
Comparison: SRAM vs DRAM

- **Storage Mechanism**: SRAM uses Flip-Flops; DRAM uses Capacitors.
- **Speed**: SRAM is faster; DRAM is slower.
- **Density**: SRAM is low (6T/bit); DRAM is high (1T1C/bit).
- **Refresh Requirement**: SRAM does not need refresh; DRAM requires periodic refresh.
- **Power Consumption**: SRAM has higher power consumption; DRAM has lower power consumption.
- **Cost**: SRAM is expensive per bit; DRAM is cheaper per bit.

Feature	SRAM	DRAM
Storage Mechanism	Flip-Flops (6T)	Capacitors (1T1C)
Speed	Very Fast	Slower
Density	Low (Fewer bits per chip)	High (More bits per chip)
Refresh Requirement	Not Required	Periodically Required
Power Consumption	Higher	Lower
Cost per Bit	Expensive	Cheaper
Typical Use	CPU Cache (L1, L2, L3)	Main Memory (RAM)

Applications of SRAM and DRAM

- **SRAM Applications**:
  - CPU Cache memory (L1, L2, L3 caches) due to high speed.
  - Registers in microprocessors.
  - High-performance networking equipment.
- **DRAM Applications**:
  - Main system memory (RAM in PCs, laptops, smartphones) due to high density and low cost.
  - Video graphics memory (VRAM).



Summary

- Memory is categorized by organization (address/data lines) and volatility.
- RAM is volatile, read/write memory.
- SRAM uses 6-transistor flip-flops, requires no refresh, and is very fast but expensive.
- DRAM uses a 1-transistor 1-capacitor design, requires constant refreshing, is slower, but very dense and cost-effective.
- SRAM is typically used for Cache; DRAM for Main Memory.

SRAM

✓ **Speed**

✗ Cost & Density

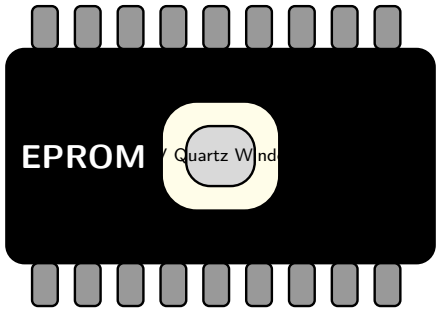
DRAM

✓ **Density & Cost**

✗ Speed & Refresh

Preview of Next Lecture

- In the next lecture, we will explore Non-Volatile Memory.
- We will cover Read-Only Memory (ROM) and its variants.
- Topics include: PROM, EPROM, and EEPROM.
- We'll see how data can be retained even when the power is switched off.



Questions?

Lecture 5.2: Types of Memory: ROM (PROM, EPROM, EEPROM)

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

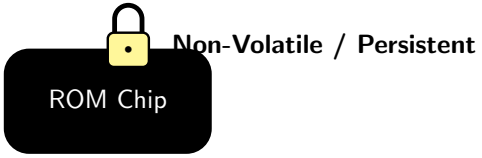
Table of Contents

Agenda

- Recap: Volatile vs Non-Volatile
- Introduction to ROM
- Internal Architecture
- Mask ROM (MROM)
- Programmable ROM (PROM)
- Erasable PROM (EPROM)
- EEPROM
- Comparison and Applications

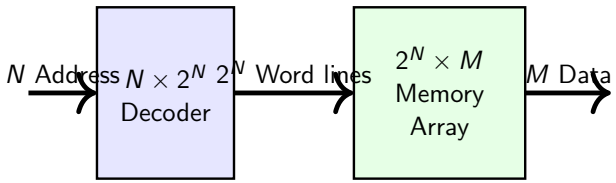
Introduction to ROM (Read-Only Memory)

- ROM is a type of non-volatile memory.
- It retains stored information even when the power is turned off.
- Traditionally, data in ROM is 'hardwired' or programmed once.
- Used to store critical firmware, boot instructions, and look-up tables.
- Modern ROM variations allow for erasing and reprogramming.



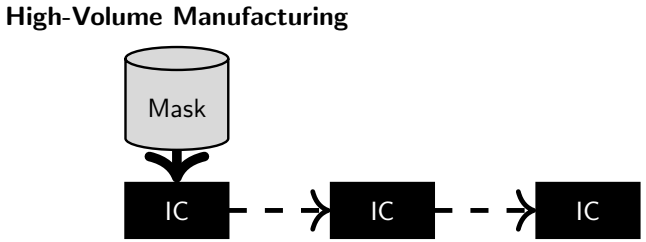
ROM Internal Architecture

- A ROM essentially acts as a combinational logic circuit.
- It consists of an integrated decoder and a memory array (OR gates).
- An N-bit address activates one of 2^N word lines via the decoder.
- The word line triggers connections, outputting an M-bit data word.



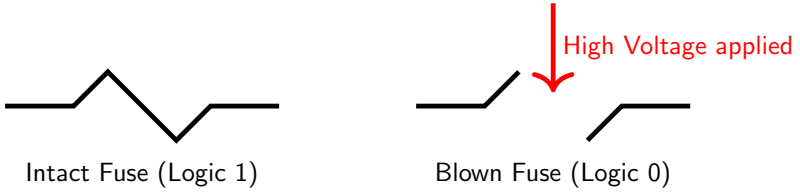
Mask ROM (MROM)

- The oldest and simplest type of ROM.
- Data is physically embedded into the IC using a photographic mask.
- Cannot be modified or erased after manufacture.
- Extremely cheap for high-volume production.
- Inflexible: Any bug in the code requires a completely new batch.



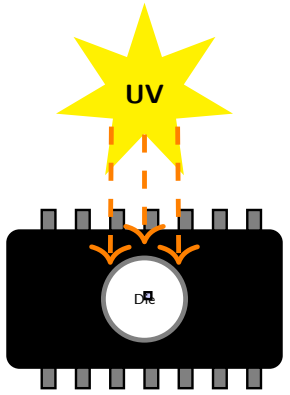
Programmable ROM (PROM)

- Allows the user to program the memory themselves after manufacturing.
- Chips are manufactured entirely blank (all 1s or all 0s).
- Uses microscopic 'fusible links' at each memory cell.
- Programming uses a high voltage to 'blow' specific fuses (writes 0s).
- One-Time Programmable (OTP): Once blown, it cannot be restored.



Erasable Programmable ROM (EPROM)

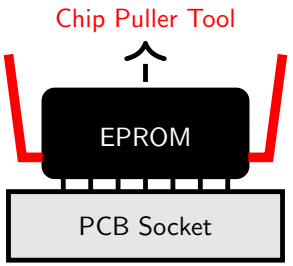
- Overcomes the 'one-time' limitation of PROM.
- Can be programmed, erased, and reprogrammed multiple times.
- Uses Floating-Gate Transistors to store charge.
- Erased by exposing the entire chip to strong UV light for 15-20 minutes.
- UV light gives electrons enough energy to escape the floating gate.



EPROM Erasure under UV Light

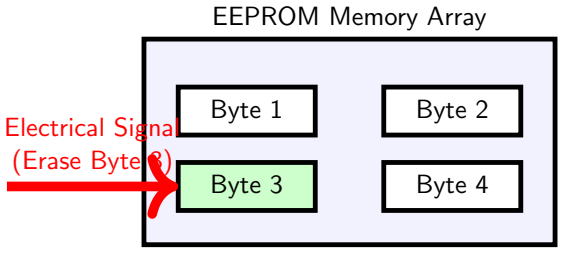
EPROM Limitations

- The erasure process is cumbersome and takes a long time (minutes).
- The entire chip must be erased at once; cannot selectively erase a byte.
- The chip must be physically removed from its circuit board.
- Requires specialized equipment (UV eraser and programmer).



Electrically Erasable PROM (EEPROM)

- Solves the physical removal and bulk-erase issues of EPROM.
- Can be erased and reprogrammed electrically, directly within the circuit.
- Erasing and writing can be done on a byte-by-byte basis (selectively).
- Uses higher voltage to tunnel electrons on/off the floating gate.
- Slower write/erase cycles compared to RAM, and limited lifespan.



EEPROM vs. Flash Memory

- Flash memory is a specific type of EEPROM.
- Standard EEPROM erases data byte-by-byte.
- Flash memory erases data in large blocks (sectors).
- Flash is cheaper to produce and provides much higher density.

Feature	**Standard EEPROM**	**Flash Memory**
Erase Granularity	Byte-by-byte	Large Blocks (Sectors)
Erase Speed	Slow (per byte)	Fast (per block)
Density	Low	Very High
Cost per Bit	Higher	Lower
Typical Application	Small Config Data	USB Drives, SSDs

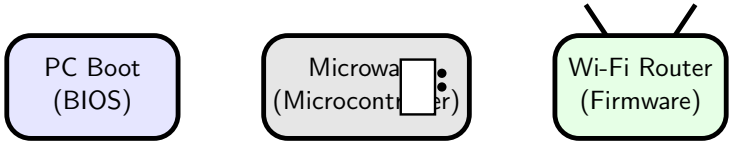
Comparison of ROM Types

- Mask ROM: Programmed by manufacturer, permanent, cheap in bulk.
- PROM: Programmed by user (once), permanent, uses blown fuses.
- EPROM: Programmed electrically, erased by UV light (bulk).
- EEPROM: Programmed/erased electrically (byte level), in-circuit.

ROM Type	Programmability	Erase Method	Reusability
Mask ROM	Factory Programmed	Cannot be erased	Permanent (No)
PROM	User Programmed (Once)	Cannot be erased	Permanent (No)
EPROM	User Programmed (Elec.)	UV Light (Bulk Erase)	Yes (Requires removal)
EEPROM	User Programmed (Elec.)	Electrical (Byte Erase)	Yes (In-circuit)

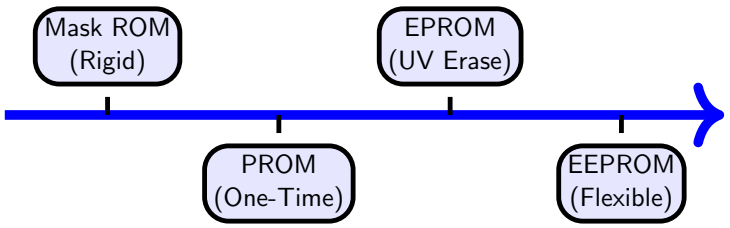
Applications of ROM

- Firmware storage (e.g., PC BIOS, router operating systems).
- Microcontroller program memory (e.g., code for microwaves).
- Look-Up Tables (LUTs) in digital signal processing (e.g., sine wave).
- Storing calibration data and MAC addresses in networking devices.



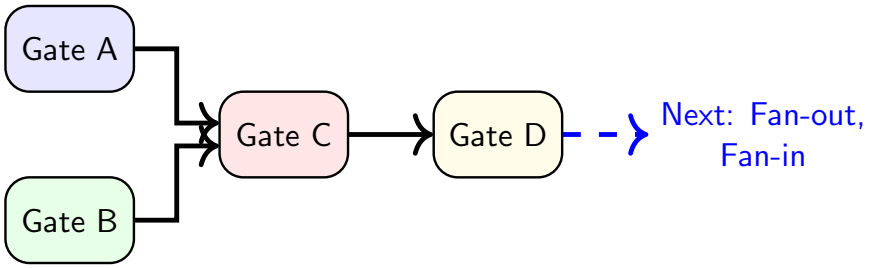
Summary

- ROM is non-volatile memory used for permanent or semi-permanent storage.
- Mask ROM is hardcoded at the factory.
- PROM is user-programmable but only once via fuses.
- EPROM is erasable using UV light, requiring chip removal.
- EEPROM is electrically erasable byte-by-byte, in-circuit.



Preview of Next Lecture

- Moving on to the final part of Unit 5: Logic Families.
- We will learn how standard ICs are classified and built internally.
- We will define key characteristics: Fan-in, Fan-out, and Noise Margin.
- These metrics dictate how we connect different digital chips together.



Questions?

Lecture 5.3: Overview of Logic Family Definitions: Fan-in, Fan-out, Noise Margin

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

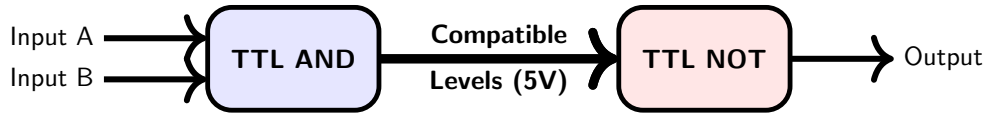
Table of Contents

Agenda

- What is a Digital Logic Family?
- Importance of Electrical Characteristics
- Definition and Implications of Fan-in
- Definition and Calculation of Fan-out
- Current Sourcing and Sinking
- Introduction to Electrical Noise
- Understanding Noise Margin
- Summary

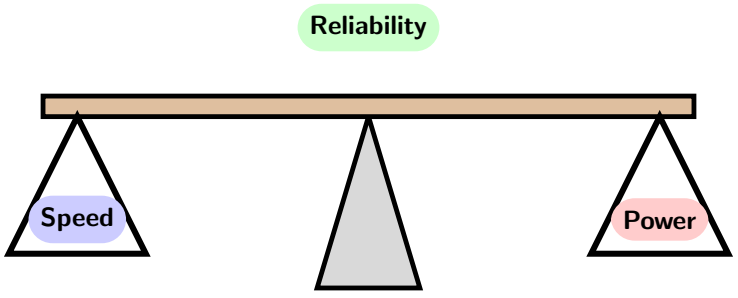
Introduction to Logic Families

- Set of ICs implementing logic functions using basic electronic building blocks.
- Chips within the same family are perfectly compatible.
- Share same logic levels, supply voltages, and connect directly.
- Examples: TTL (Transistor-Transistor Logic), CMOS.



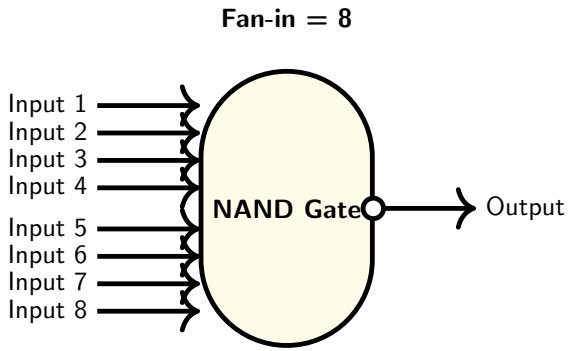
Importance of Logic Characteristics

- Logic gates are not ideal; they have physical limitations.
- Constraints based on voltages, currents, and internal capacitance.
- Reliable systems require understanding connectivity limits, speed, and power.
- Today's focus: Fan-in, Fan-out, and Noise Margin.



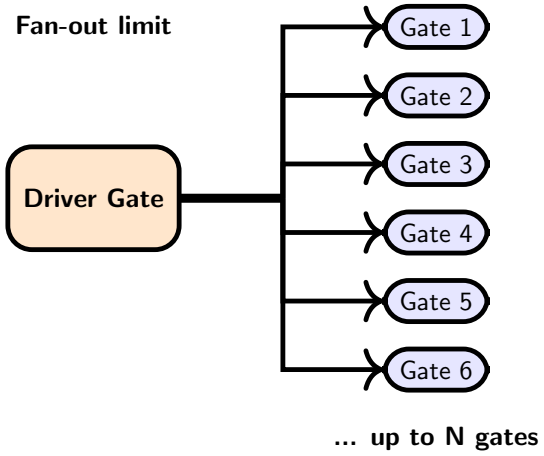
Definition: Fan-in

- **Fan-in:** Maximum number of inputs a single logic gate can handle.
- Example: A 2-input AND gate has fan-in = 2; an 8-input NAND has fan-in = 8.
- Adding more inputs increases internal capacitance and propagation delay.
- Standard ICs typically have a fan-in ranging from 2 to 8.



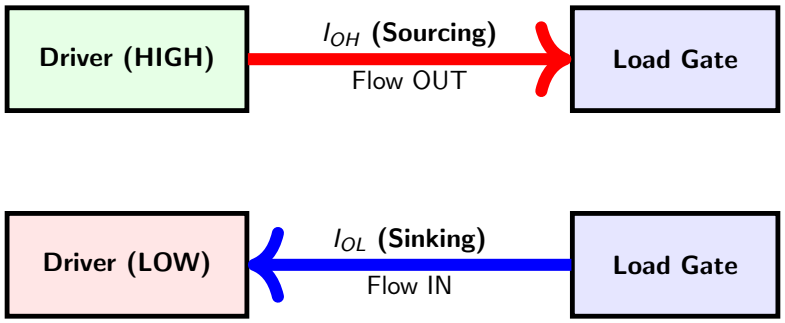
Definition: Fan-out

- **Fan-out:** Maximum number of standard inputs a single output can drive safely.
- E.g., Fan-out of 10 means one output drives up to 10 gate inputs.
- Exceeding fan-out causes output voltage to drop below acceptable logic levels.
- Determined by output current capability and input current requirements.



Current Sourcing and Sinking

- **Current Sourcing (Logic High):** Driving gate supplies (sources) current to the driven gates (I_{OH}).
- **Current Sinking (Logic Low):** Driving gate absorbs (sinks) current from driven gates (I_{OL}).
- Input requirements are denoted as I_{IH} (for High) and I_{IL} (for Low).



Calculating Fan-out

- Fan-out is calculated for both High and Low states. The actual fan-out is the **lower** of the two values.

Fan-out Formulas

$$\text{High State Fan-out} = \frac{I_{OH(max)}}{I_{IH(max)}} \quad \text{Low State Fan-out} = \frac{I_{OL(max)}}{I_{IL(max)}}$$

Example Calculation

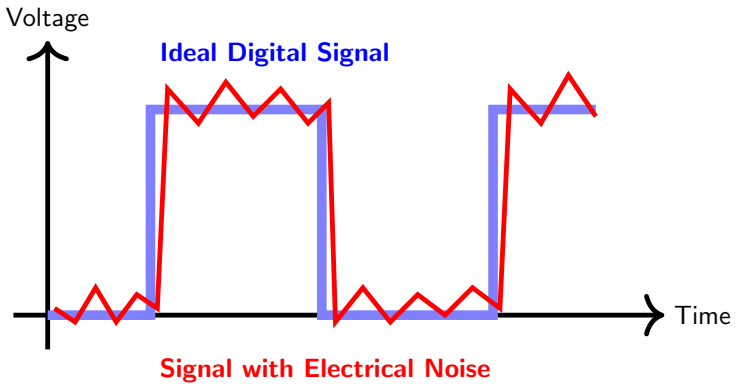
Given: $I_{OH} = 400\mu A$, $I_{IH} = 40\mu A \implies$ High Fan-out $= 400/40 = 10$.

Given: $I_{OL} = 16mA$, $I_{IL} = 1.6mA \implies$ Low Fan-out $= 16/1.6 = 10$.

Overall Fan-out = 10.

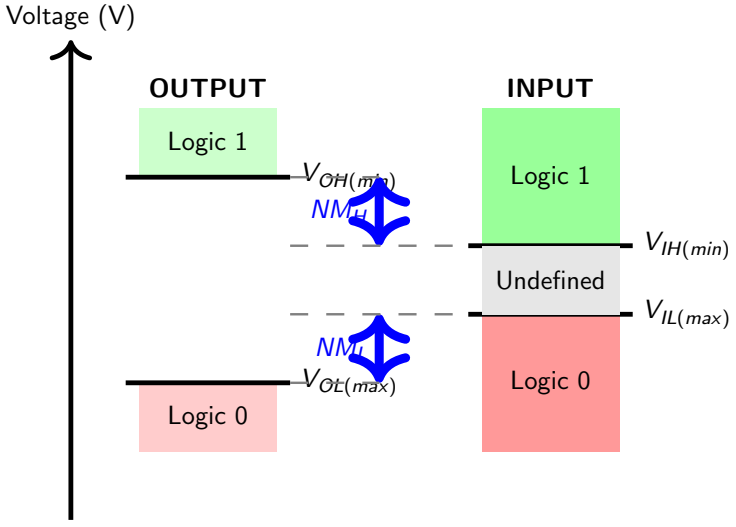
What is Electrical Noise?

- **Noise:** Unwanted voltage fluctuations induced by external interference (motors, fields) or internal switching transients.
- Noise can cause a logic signal to temporarily dip or spike.
- If a spike is large enough, a logic '0' might be mistakenly interpreted as a logic '1'.



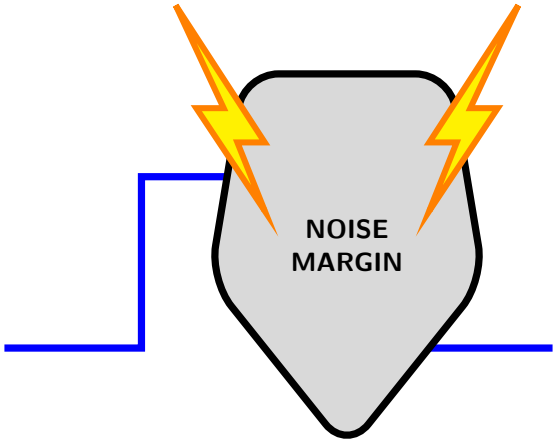
Voltage Levels in Logic Families

- $V_{OH(min)}$: Minimum voltage guaranteed for a Logic 1 output.
- $V_{IH(min)}$: Minimum voltage recognized as a Logic 1 input.
- $V_{OL(max)}$: Maximum voltage guaranteed for a Logic 0 output.
- $V_{IL(max)}$: Maximum voltage recognized as a Logic 0 input.



Definition: Noise Margin

- **Noise Margin:** A quantitative measure of a logic family's immunity to noise.
- Maximum noise voltage that can be added without causing errors.
- Higher noise margin = higher reliability in noisy environments.



Calculating Noise Margin

- Calculated for High-level (NM_H) and Low-level (NM_L).

Noise Margin Formulas

$$NM_H = V_{OH(min)} - V_{IH(min)}$$

$$NM_L = V_{IL(max)} - V_{OL(max)}$$

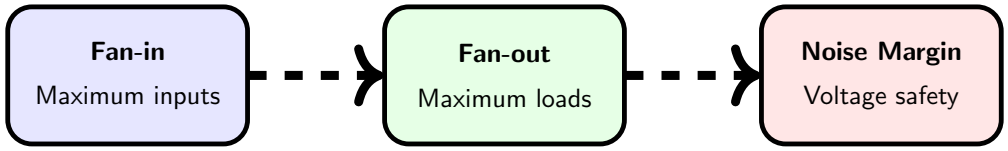
Example Calculation (Standard TTL)

Given: $V_{OH} = 2.4V$, $V_{IH} = 2.0V \implies NM_H = 0.4V$

Given: $V_{IL} = 0.8V$, $V_{OL} = 0.4V \implies NM_L = 0.4V$

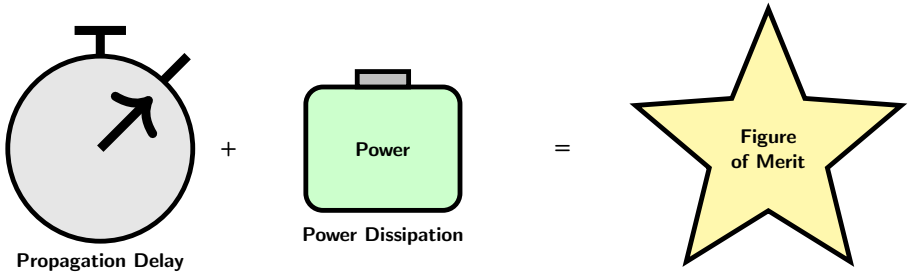
Summary

- **Logic families** group compatible digital ICs.
- **Fan-in** is the number of inputs a gate has.
- **Fan-out** is the number of standard inputs a single output can safely drive.
- **Noise Margin** defines the maximum voltage interference a signal can endure.



Preview of Next Lecture

- In the next lecture, we will explore:
- **Propagation Delay:** How fast gates can operate.
- **Power Dissipation:** How much energy they consume.
- **Figure of Merit:** The Speed-Power Product.



Questions?

Lecture 5.4: Overview of Logic Family Definitions: Propagation Delay, Power Dissipation, Figure of Merit

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

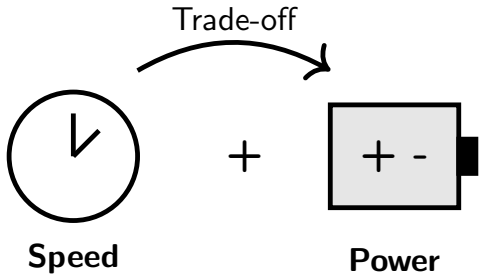
Table of Contents

Agenda

- Introduction to Speed and Power Constraints
- Definition of Propagation Delay
- High-to-Low and Low-to-High Delays
- Impact of Delay on System Clock Speed
- Definition of Power Dissipation
- Static vs. Dynamic Power
- The Speed-Power Trade-off
- Figure of Merit (Speed-Power Product)
- Summary

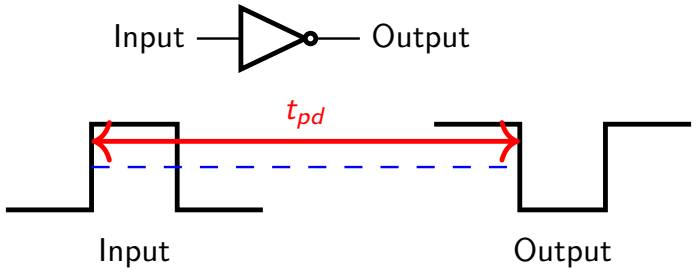
Introduction to Dynamic Constraints

- Fan-out and Noise Margin are mostly 'static' or DC characteristics.
- When logic gates switch states, we must consider 'dynamic' or AC characteristics.
- Transitions take time (**Delay**) and require moving electrons, which consumes energy (**Power**).
- **Goal:** Switch states as fast as possible using the least amount of power.



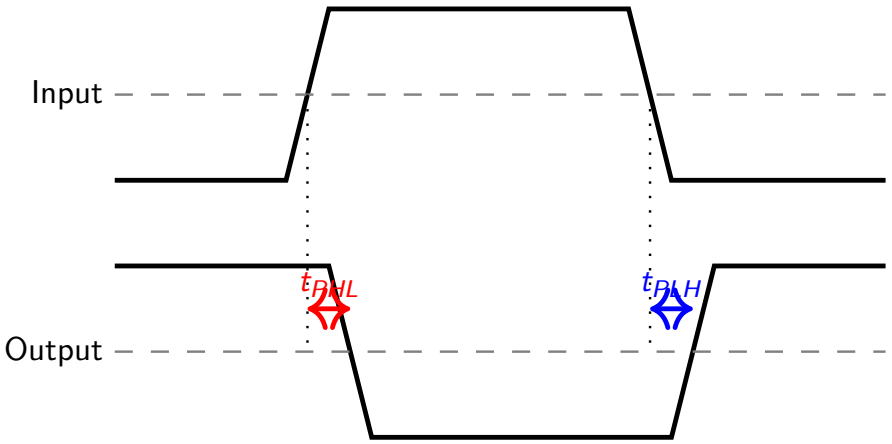
Definition: Propagation Delay

- **Propagation delay (t_{pd})**: Time required for a change in the input signal to produce a change at the output.
- Caused by internal parasitic capacitances and transistor switching times.
- Measured in nanoseconds (ns) or picoseconds (ps). Lower t_{pd} = faster gate.



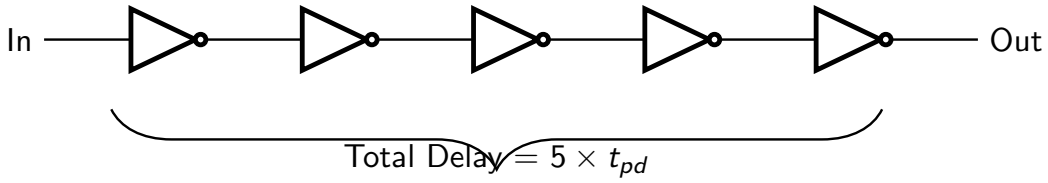
Types of Propagation Delay

- Measured at the **50% voltage level** of the transitions.
- t_{PHL}: Time for output to transition High-to-Low.
- t_{PLH}: Time for output to transition Low-to-High.
- Average: $t_{pd} = \frac{t_{PHL}+t_{PLH}}{2}$



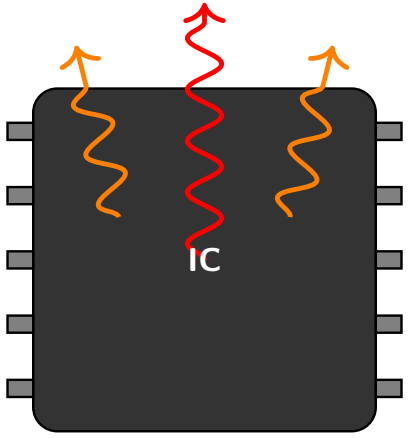
Impact on System Clock Speed

- Propagation delay is **cumulative**.
- 5 gates in series = total delay of $5 \times t_{pd}$.
- Max clock frequency (f_{max}) is limited by the longest delay path.
- $f_{max} \approx \frac{1}{\text{Total Propagation Delay}}$



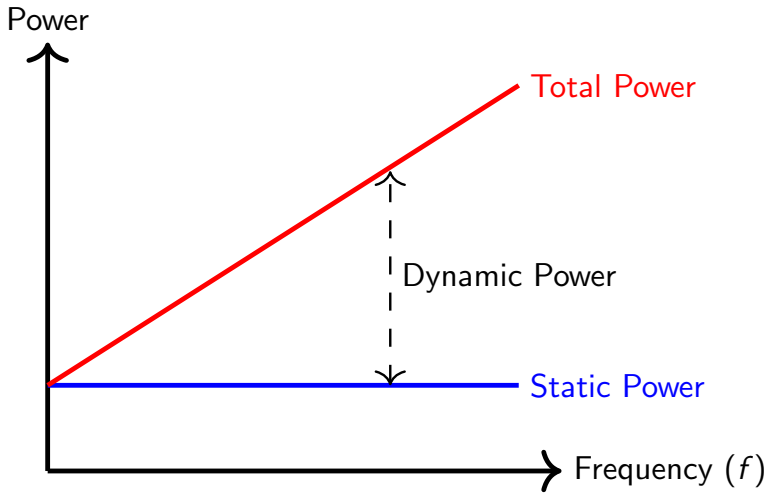
Definition: Power Dissipation

- **Power dissipation**: Electrical power consumed by a logic gate and converted into heat.
- Measured in milliwatts (mW) or microwatts (μW) per gate.
- Formula: $P = V_{cc} \times I_{cc}$ (Supply Voltage $\times$ Supply Current).
- High power $\rightarrow$ Chip runs hot, requires cooling.
- Low power is critical for battery-operated devices.



Static vs. Dynamic Power

- **Static Power**: Consumed while gate is maintaining a state (not switching). Caused by leakage.
- **Dynamic Power**: Consumed only during transition (0 to 1, or 1 to 0).
- Dynamic power $\propto$ Switching frequency (f). Total = Static + Dynamic.



The Speed-Power Trade-off

- Inescapable trade-off between speed and power.
- Faster switching (lower delay) $\rightarrow$ Requires more current.
- More current $\rightarrow$ Higher power dissipation.
- Less current (save battery) $\rightarrow$ Slower switching.

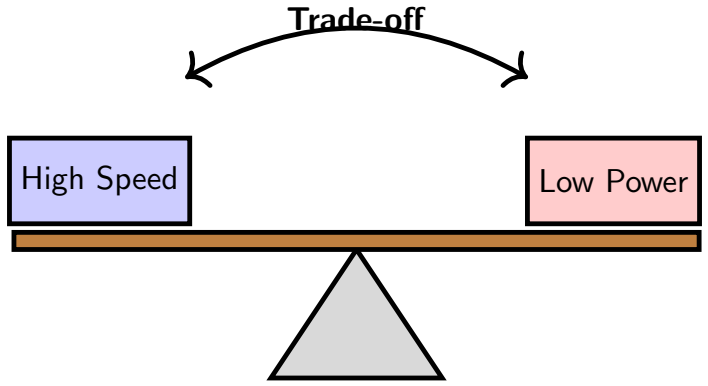


Figure of Merit (Speed-Power Product)

- The **Speed-Power Product (SPP)** is used as a comprehensive Figure of Merit.
- Comparing just on speed or just on power is unfair.
- Units: ns $\times$ mW = picojoules (pJ).
- SPP represents the Energy required to switch a gate once.

Figure of Merit Formula

$$SPP = t_{pd} \times P_D$$

Speed-Power Product = Propagation Delay $\times$ Power Dissipation

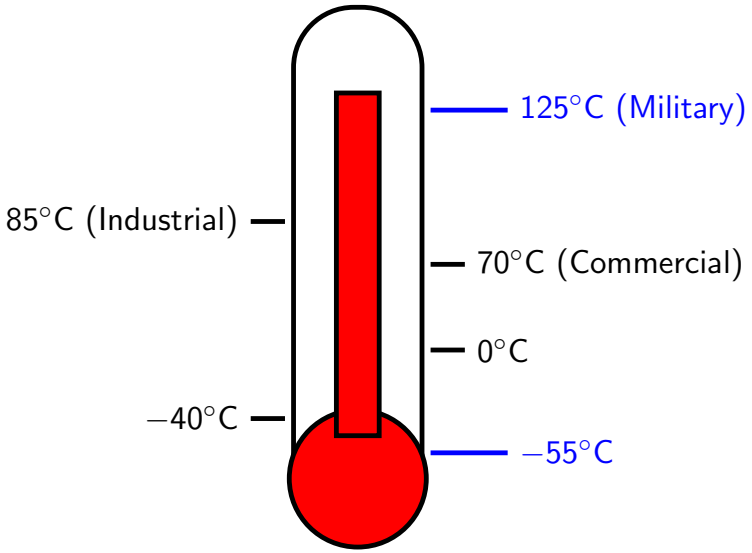
Significance of the Figure of Merit

- A **lower** Speed-Power Product is always better.
- It means the logic family is efficient (fast without excessive power).
- Engineers look for the lowest possible SPP when evaluating technology.

Logic Family	**Delay (t_{pd})**	**Power (P_D)**	**SPP (Figure of Merit)**
Family A	10 ns	10 mW	100 pJ
Family B	5 ns	2 mW	**10 pJ (Better!)**

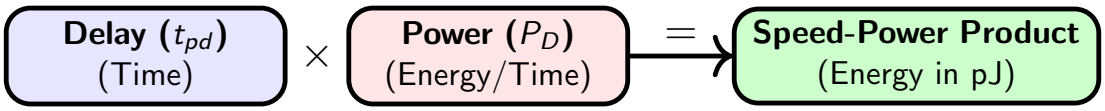
Operating Temperature Range

- Temperature range over which the IC is guaranteed to operate properly.
- Temperature affects delay and power.
- **Commercial:** 0°C to 70°C (Standard consumer electronics).
- **Industrial:** −40°C to 85°C (Automotive, factory equipment).
- **Military:** −55°C to 125°C (Aerospace, defense).



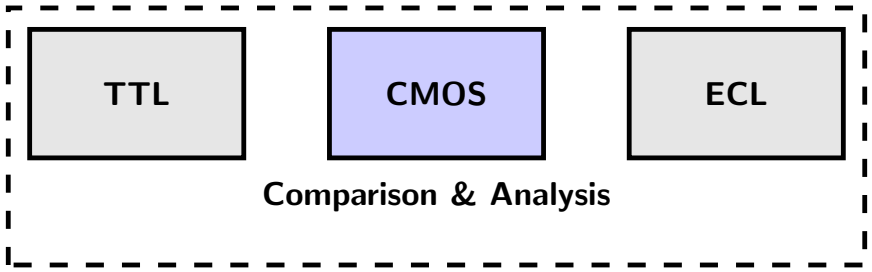
Summary

- **Propagation Delay (t_{pd})** limits system speed.
- **Power Dissipation** dictates battery life and cooling.
- Fundamental physical trade-off between high speed and low power.
- **Figure of Merit (SPP)** combines both metrics.
- A lower Figure of Merit indicates a more efficient, advanced logic family.



Preview of Next Lecture

- In the next lecture, we will look at specific Logic Families: TTL, CMOS, and ECL.
- We will see how they are constructed internally.
- We will apply the definitions learned today to compare them.
- We will discover why CMOS dominates modern digital electronics.



Questions?

Lecture 5.5: Comparison of different logic families (TTL, CMOS, ECL) - Part 1

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

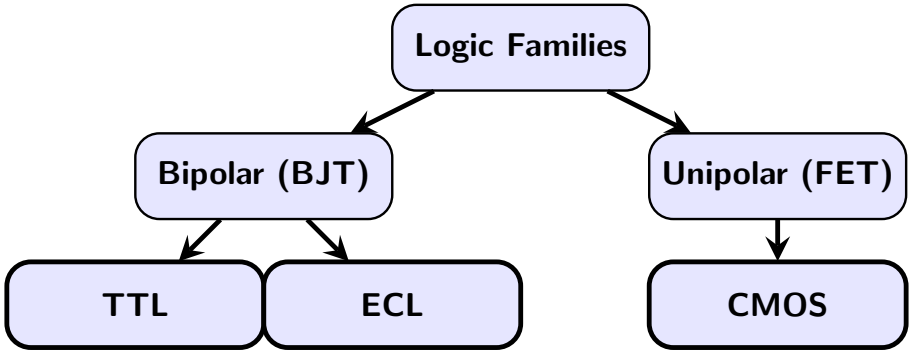
Lecturer, GP Palanpur

Agenda

- **Introduction:** The Major Logic Families
- **TTL (Bipolar):**
  - Transistor-Transistor Logic
  - Characteristics and Sub-families
- **CMOS (Unipolar):**
  - Complementary Metal-Oxide Semiconductor
  - Inverter Operation & Characteristics
- **ECL (High Speed Bipolar):**
  - Emitter-Coupled Logic
  - ECL Characteristics
- **Conclusion:** Summary

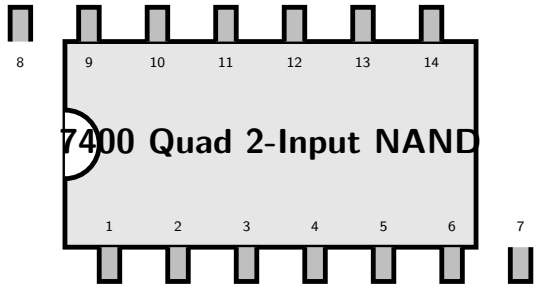
The Major Logic Families

- Logic families are categorized by the type of electronic components they use internally.
- Bipolar Logic Families (using BJT transistors):
  - TTL (Transistor-Transistor Logic)
  - ECL (Emitter-Coupled Logic)
- Unipolar Logic Families (using Field Effect Transistors - FETs):
  - CMOS (Complementary Metal-Oxide Semiconductor)



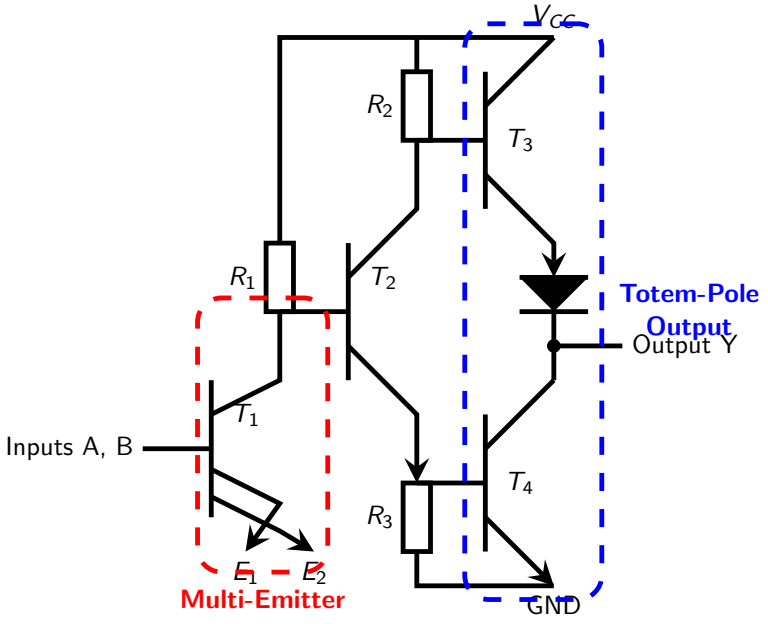
Transistor-Transistor Logic (TTL)

- Introduced by Texas Instruments in 1964 (the famous 7400 series).
- Uses Bipolar Junction Transistors (BJTs) for both logic processing and output buffering.
- Standard power supply voltage is strictly 5.0V.
- TTL was the dominant logic family for decades and set the standard for logic levels (0V-0.8V for Low, 2.0V-5V for High).



TTL Internal Circuit (NAND Gate)

- A basic TTL NAND gate features a multi-emitter input transistor.
- The output stage uses a 'Totem-Pole' arrangement.
- The Totem-Pole output provides active pull-up and active pull-down.



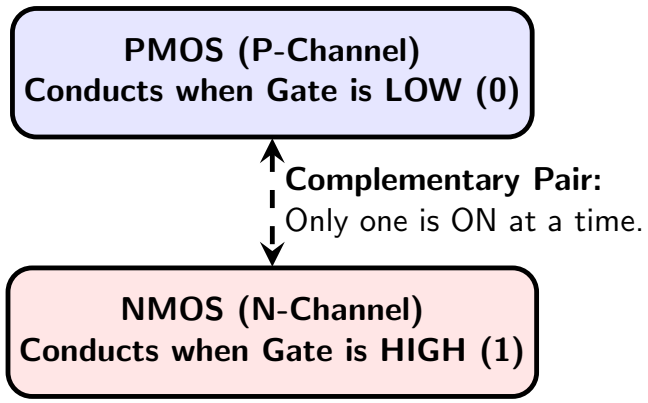
TTL Characteristics & Sub-families

- Moderate speed ($t_{pd} \approx 10$ns).
- Relatively high power dissipation ($P_D \approx 10$mW per gate).
- Sub-families evolved to improve the speed-power product:
  - **74LS (Low-power Schottky):** Better SPP, became the industry standard.
  - **74S (Schottky):** Very fast, but high power.
  - **74ALS (Advanced Low-power Schottky):** Further optimization.

Sub-family	**Prefix**	**Delay (ns)**	**Power (mW)**
Standard TTL	7400	10	10
Low-Power Schottky	74LS	9.5	2
Schottky	74S	3	19
Advanced LS	74ALS	4	1

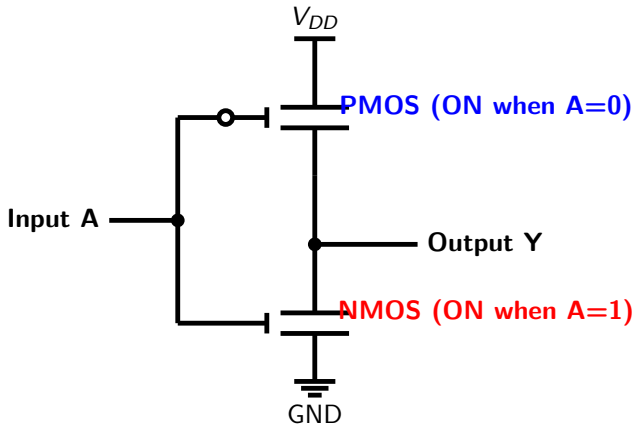
Complementary Metal-Oxide Semiconductor (CMOS)

- The dominant technology in modern digital ICs (microprocessors, memory).
- Uses 'Complementary' pairs of N-channel (NMOS) and P-channel (PMOS) MOSFET transistors.
- Wide power supply range (typically 3V to 15V for older series, down to 1.8V or lower for modern CPUs).
- Extremely high input impedance (draws virtually no input current).



CMOS Inverter Operation

- A CMOS inverter uses one PMOS (top) and one NMOS (bottom) transistor.
- **When Input = High:** NMOS turns ON, PMOS turns OFF $\rightarrow$ Output = Low.
- **When Input = Low:** PMOS turns ON, NMOS turns OFF $\rightarrow$ Output = High.
- **Key aspect:** At any given steady state, one transistor is ALWAYS off. Therefore, no direct path exists from V_{CC} to Ground.

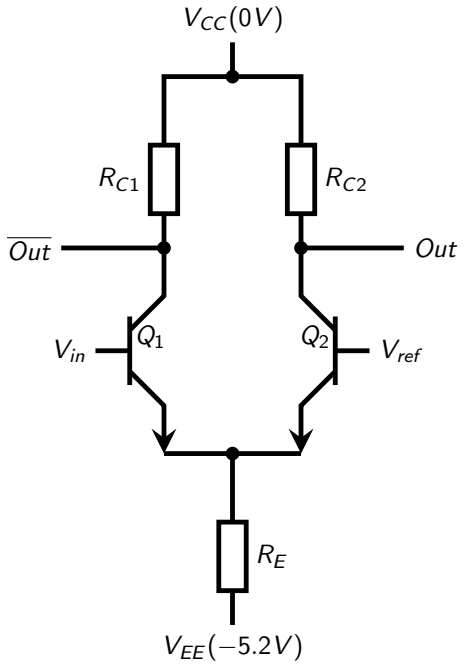


CMOS Characteristics

- Ultra-low static power dissipation (microwatts).
- Excellent Noise Margin (often 45% of the supply voltage).
- High Fan-out (can drive many other CMOS gates because inputs draw almost zero current).
- Slower than TTL in early iterations (4000 series), but modern CMOS is incredibly fast.
- Sensitive to Electrostatic Discharge (ESD) damage during handling.

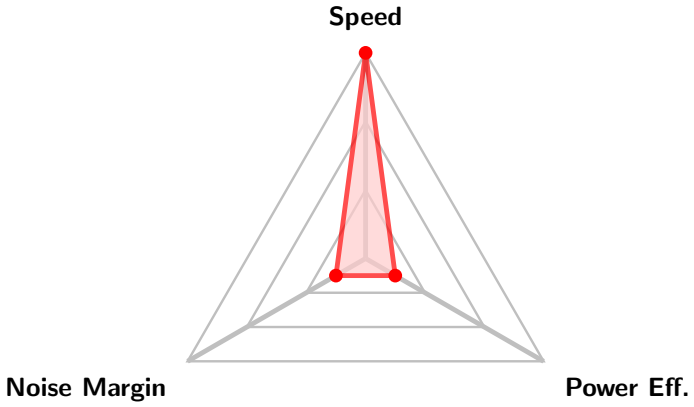
Emitter-Coupled Logic (ECL)

- A specialized bipolar logic family designed for absolute maximum speed.
- Uses a differential amplifier architecture.
- Transistors operate in the 'active' region and are never allowed to fully saturate.
- Preventing saturation eliminates the delay of discharging stored charge, resulting in ultra-fast switching.



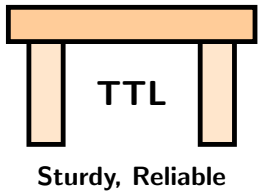
ECL Characteristics

- Fastest traditional logic family (t_{pd} often < 1ns).
- Highest power dissipation of all families (transistor pairs constantly draw current).
- Uses negative power supply voltages (e.g., $V_{CC} = 0V$, $V_{EE} = -5.2V$) to reduce noise generation.
- Poor noise margin.
- Historically used in supercomputers and high-frequency communication equipment.

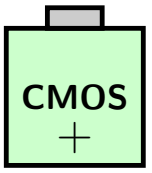


Summary of Basic Architectures

- **TTL** uses bipolar transistors with multi-emitter inputs and totem-pole outputs; it's a historical standard.
- **CMOS** uses pairs of FETs (NMOS/PMOS) ensuring one is always off, resulting in near-zero static power.
- **ECL** uses non-saturating bipolar differential pairs to achieve the highest possible switching speeds at the cost of massive power usage.



Sturdy, Reliable



Low Power

High Speed

Preview of Next Lecture

- In Part 2, we will perform a direct, side-by-side comparative analysis of these families.
- We will evaluate them against the metrics defined in Lecture 42 (Speed, Power, Fan-out, Noise Margin).
- We will also discuss how to interface a TTL chip with a CMOS chip in the same circuit.

Parameter	**TTL**	**CMOS**	**ECL**
Speed	Moderate	Varies	Fastest
Power	High	Very Low	Highest
Fan-out	~ 10	~ 50	~ 25
Noise Margin	Good	Excellent	Poor

Questions?

Lecture 5.6: Comparison of different logic families (TTL, CMOS, ECL) - Part 2

Complete Lecture Deck – All 45 Lectures

Milav Dabgar

Lecturer, GP Palanpur

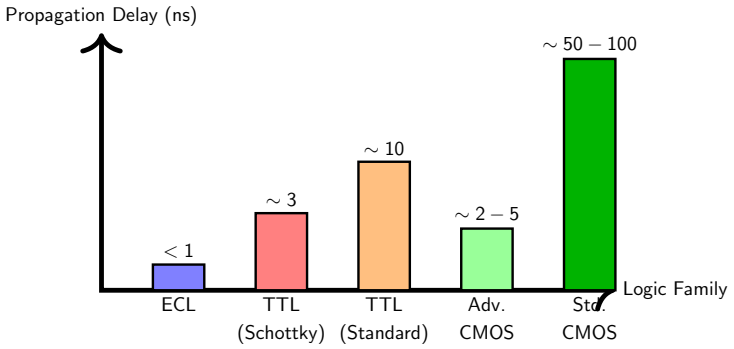
Table of Contents

Agenda

- Comparative Framework
- Speed (Propagation Delay) Comparison
- Power Dissipation Comparison
- Figure of Merit (SPP) Comparison
- Noise Margin and Fan-out Comparison
- Density and Cost Constraints
- Interfacing Logic Families
  - Interfacing TTL to CMOS
  - Interfacing CMOS to TTL
- Summary

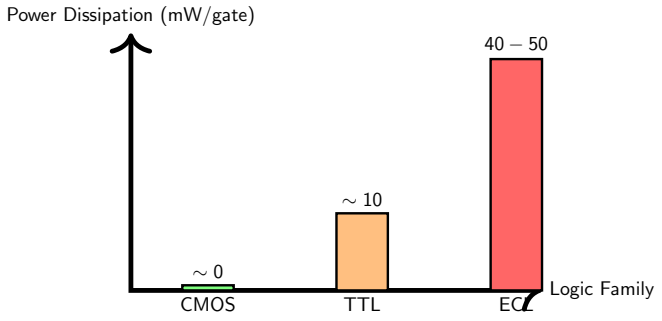
Speed Comparison (Propagation Delay)

- **ECL:** Fastest. Delay is often < 1 ns. Operates easily at microwave frequencies.
- **TTL:** Moderate to fast. Delay ranges from 10 ns (standard) down to 3 ns (Schottky).
- **CMOS:** Slowest (Standard 4000 series, 50-100 ns). Modern Advanced CMOS is very fast.
- **Ranking:** ECL $\rightarrow$ Advanced TTL $\rightarrow$ Advanced CMOS $\rightarrow$ Standard CMOS.



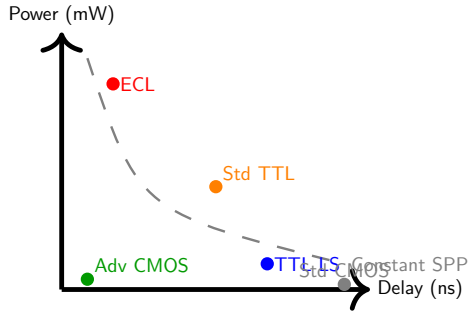
Power Dissipation Comparison

- **CMOS:** Outstanding. Static power is practically zero (μW). Dynamic power increases with frequency but remains highly efficient.
- **TTL:** High. Standard TTL dissipates about 10 mW per gate continuously.
- **ECL:** Very High. Dissipates 40-50 mW per gate. Requires massive cooling systems in large computers.
- **Ranking:** CMOS $\rightarrow$ TTL $\rightarrow$ ECL (Lowest to Highest Power).



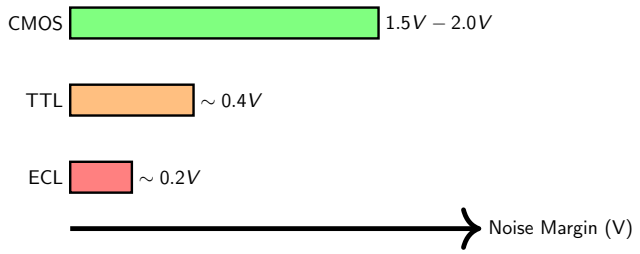
Speed-Power Product (Figure of Merit)

- Recall: SPP = Delay (ns) $\times$ Power (mW) (lower is better, measured in pJ).
- **ECL:** Fast but power-hungry $\rightarrow$ Moderate/High SPP (e.g., 50 pJ).
- **Standard TTL:** Moderate speed, high power $\rightarrow$ High SPP (e.g., 100 pJ).
- **TTL LS:** Low-power Schottky offers better balance (e.g., 20 pJ).
- **Modern CMOS:** Moderate/Fast speed, ultra-low power $\rightarrow$ Excellent SPP (< 1 pJ).
- **Conclusion:** Advanced CMOS offers the best overall efficiency.



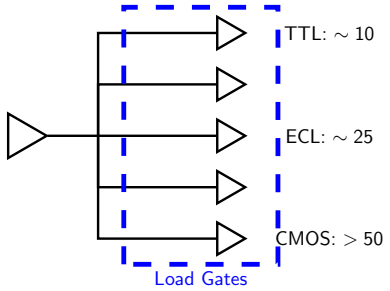
Noise Margin Comparison

- **CMOS:** Excellent noise margin (typically 45% of V_{CC}). At 5V, noise margin is roughly 1.5V to 2V.
- **TTL:** Moderate noise margin (typically 0.4V for both High and Low states).
- **ECL:** Poor noise margin (around 0.2V to 0.25V). Highly sensitive to noise.
- **Ranking:** CMOS $\rightarrow$ TTL $\rightarrow$ ECL (Best to Worst).



Fan-out Comparison

- **CMOS:** Extremely high fan-out (> 50). Because MOSFET inputs act like capacitors and draw no DC current.
- **TTL:** Moderate fan-out (typically 10). Limited by current sinking requirements of BJT inputs.
- **ECL:** High fan-out (typically 25). Low output impedance allows driving many gates.
- **Ranking:** CMOS $\rightarrow$ ECL $\rightarrow$ TTL.

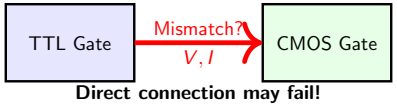


Overall Summary Table

Parameter	TTL	CMOS	ECL
Basic Component	BJT	MOSFET	BJT
Speed	Fast ($\sim$ 10ns)	Moderate/Fast	Fastest ($\sim$ 1ns)
Power Dissipation	High ($\sim$ 10mW)	Lowest ($<$ 1mW)	Highest ($\sim$ 40mW)
Noise Margin	Good (0.4V)	Excellent (1.5V)	Poor (0.2V)
Fan-out	10	$>$ 50	25
Packing Density	Low	High	Low

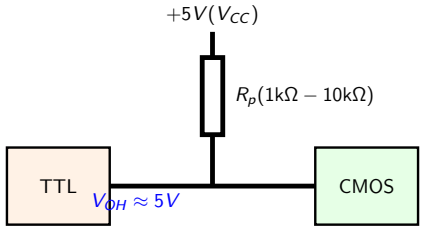
Interfacing Logic Families

- Often, a system requires mixing TTL and CMOS chips.
- **Problem:** They have different guaranteed voltage levels and current capabilities.
- A TTL 'High' output might not be high enough for a CMOS input to recognize it as a 'High'.
- A CMOS output might not be able to sink enough current to pull a TTL input to a valid 'Low'.
- **Solution:** We must analyze voltage (V_{OH}, V_{IH}) and current (I_{OL}, I_{IL}) specifications.



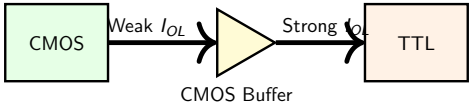
Interfacing TTL to CMOS

- **Issue:** TTL guarantees $V_{OH(min)} = 2.4V$.
- CMOS requires $V_{IH(min)} = 3.5V$ (when operating at 5V).
- If TTL outputs 2.4V, CMOS might misinterpret it or enter an undefined state.
- **Solution:** Use a **Pull-up Resistor**.
- Connect a resistor (e.g., 1kΩ to 10kΩ) between the TTL output and the 5V supply (V_{CC}).
- This pulls the output voltage up to near 5V when the TTL gate is High.



Interfacing CMOS to TTL

- **Issue:** Voltage levels are usually compatible (CMOS $V_{OH} >$ TTL V_{IH}).
- The problem is **Current**. Standard TTL inputs require significant current to be pulled Low ($I_{IL} \approx 1.6$mA).
- Standard CMOS outputs have very weak current sinking capability ($I_{OL} \approx 0.4$mA).
- A standard CMOS gate cannot drive a standard TTL gate (Fan-out < 1).
- **Solution:** Use a specialized **CMOS Buffer IC** (e.g., 4049/4050) which has enhanced current sinking capabilities.



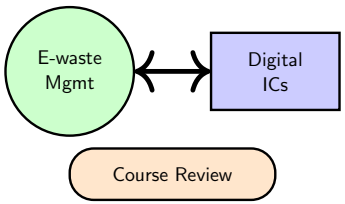
Summary

- **ECL** provides maximum speed at the cost of extreme power dissipation.
- **TTL** was the reliable middle-ground for decades but is largely obsolete.
- **CMOS** dominates modern design due to its ultra-low power, high density, and excellent noise margin.
- Interfacing different logic families requires careful attention to:
  - Voltage level mismatches (use pull-up resistors for TTL $\rightarrow$ CMOS).
  - Current driving limits (use buffers for CMOS $\rightarrow$ TTL).

Key Takeaway:
Always match V_{OH}/V_{IH} and I_{OL}/I_{IL} specifications!

Preview of Next Lecture

- In our final lecture for the course, we will step back from circuit design.
- We will discuss the impact of digital ICs and E-waste.
- Finally, we will review the entire Digital Logic course to prepare you for finals.



Questions?

Lecture 5.7: E-waste Management of Digital ICs/Chips and Course Recap

Complete Lecture Deck – All 45 Lectures

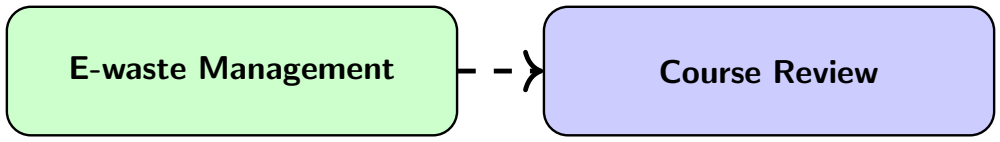
Milav Dabgar

Lecturer, GP Palanpur

Table of Contents

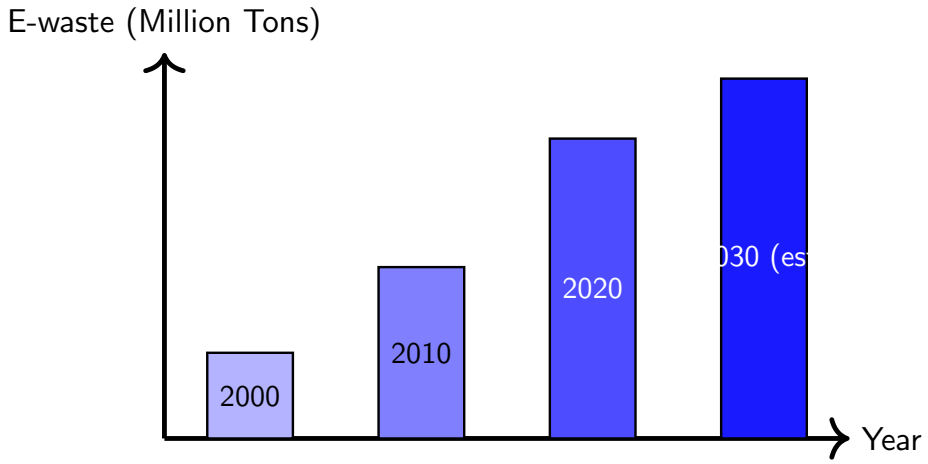
Agenda

- Introduction to E-waste
- Hazardous Materials in ICs
- E-waste Management Strategies (3 Rs)
- The IC Recycling Process
- Course Recap: Unit 1 to 5
- Final Conclusion



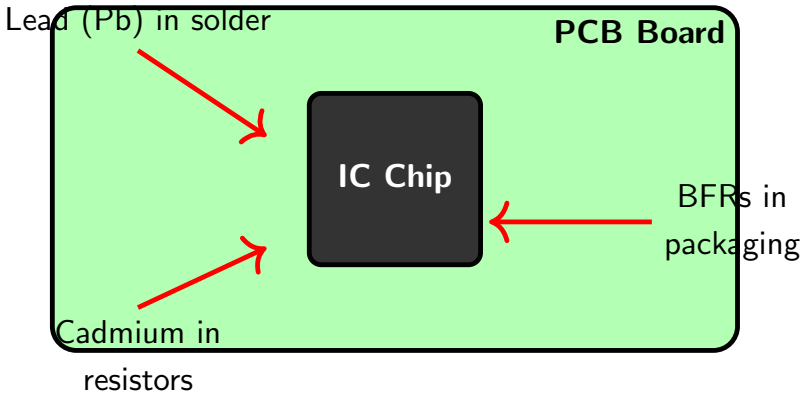
Introduction to E-waste

- **E-waste** encompasses discarded electrical or electronic devices.
- Rapid technological advancement leads to shorter lifecycles for digital devices.
- Millions of tons of obsolete systems are discarded annually.
- Improper disposal leads to severe environmental and health consequences.



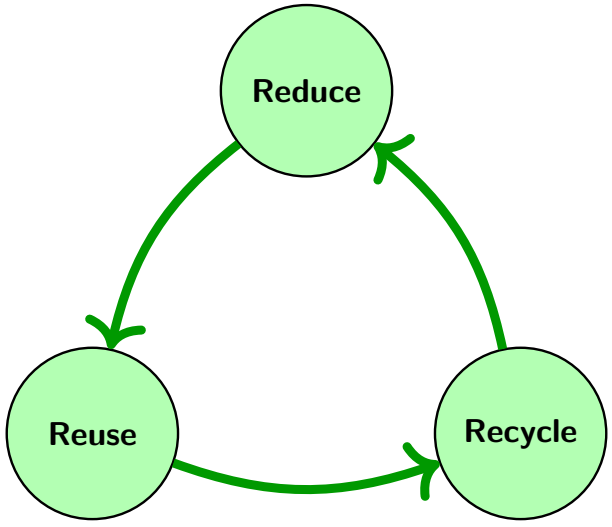
Hazardous Materials in Digital ICs

- While silicon is harmless, ICs and PCBs contain highly toxic heavy metals and chemicals.
- **Lead (Pb):** Historically used in solder; causes neurological damage.
- **Mercury (Hg):** Found in older displays and switches; highly toxic.
- **Cadmium (Cd):** Used in chip resistors/semiconductors; causes kidney damage.
- **Brominated Flame Retardants (BFRs):** Used in plastic chip packaging.



E-waste Management Strategies (The 3 Rs)

- **Reduce:** Design systems with longer lifespans. Implement 'Green Design' (e.g., RoHS compliance - Restriction of Hazardous Substances).
- **Reuse:** Refurbish older systems. Repurpose outdated chips for less demanding applications (e.g., IoT sensors).
- **Recycle:** Extract valuable materials (Gold, Silver, Copper) and safely dispose of toxic elements through formal channels.

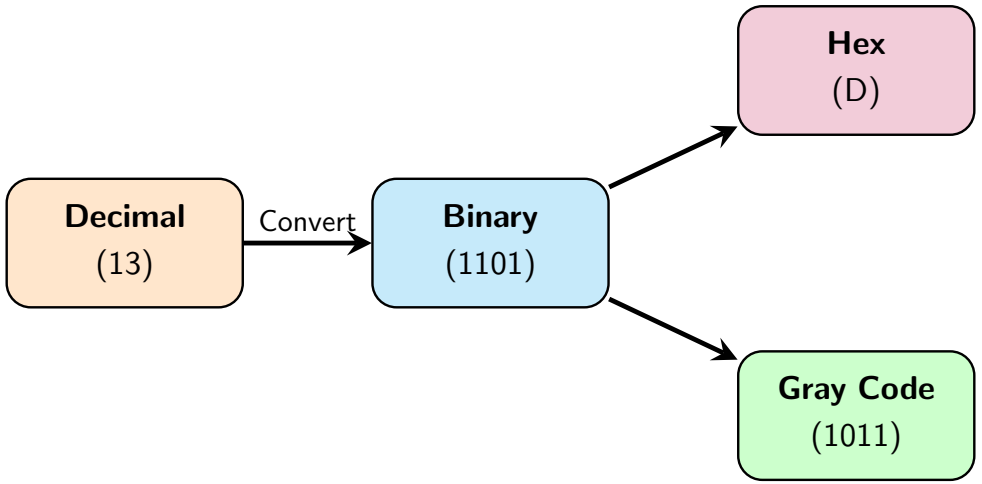


The IC Recycling Process

- **Collection & Sorting:** Separating PCBs from plastics and batteries.
- **Shredding:** Breaking boards down into small fragments.
- **Separation:** Magnetic & water methods to separate plastics from metals.
- **Metallurgical Processing:** Smelting or hydrometallurgy to recover precious metals (Gold, Palladium) from chip pins and bonding wires.
- **Safe Disposal:** Neutralizing toxic sludge.

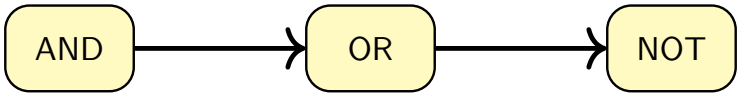
Course Recap: Unit 1 - Number Systems

- The language of computers: Binary (Base-2), Octal (Base-8), Hexadecimal (Base-16).
- Conversion techniques between bases.
- Binary Arithmetic (Addition, Subtraction).
- 1's and 2's Complement representation for negative numbers.
- Digital Codes: BCD, Gray Code (used in K-maps), and ASCII.

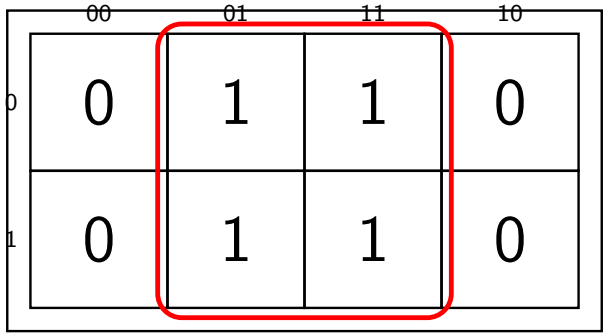


Course Recap: Unit 2 - Logic Gates & Boolean Algebra

- **Basic Gates:** AND, OR, NOT.
- **Universal Gates:** NAND, NOR (Can build any circuit).
- **Exclusive Gates:** XOR, XNOR (Parity generation).
- Boolean Algebra laws (DeMorgan's Theorems).
- Karnaugh Maps (K-maps) for systematic logic minimization (SOP/POS).

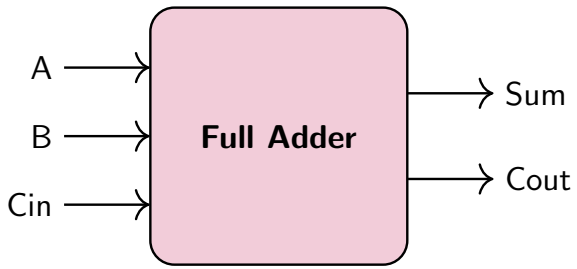


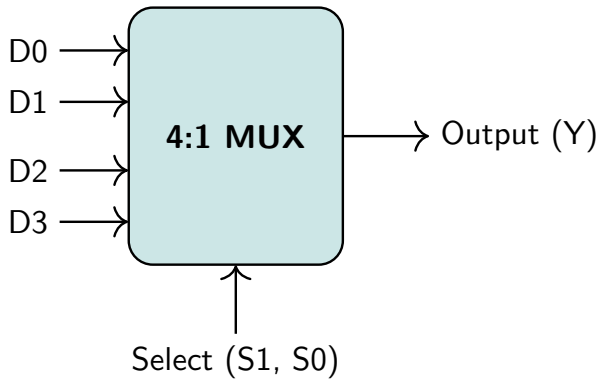
K-map Grouping



Course Recap: Unit 3 - Combinational Logic

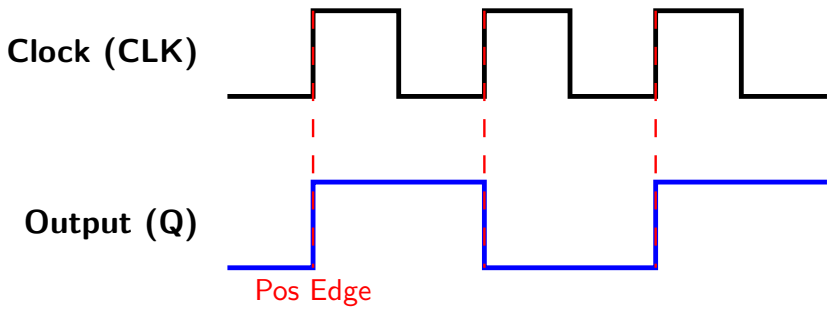
- **Combinational circuits:** Output depends strictly on current inputs (No memory).
- **Arithmetic Circuits:** Half/Full Adders, Half/Full Subtractors.
- **Data Routing:** Multiplexers (Data Selectors) and Demultiplexers.
- **Encoding/Decoding:** Decoders (Binary to Octal/Display), Encoders.





Course Recap: Unit 4 - Sequential Logic

- **Sequential circuits:** Output depends on current inputs AND past states (Memory).
- **Flip-Flops:** SR, D, JK, and T types. Edge-triggered devices.
- **Registers:** Shift registers (SISO, SIPO, PISO, PIPO).
- **Counters:** Asynchronous (Ripple) and Synchronous counters.
- **A/D and D/A Converters:** Bridging the analog real world with digital domain.

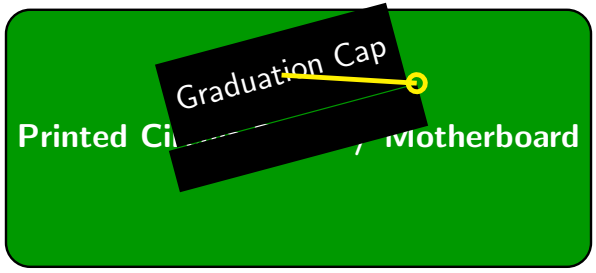


Course Recap: Unit 5 - Memory & Logic Families

- **Memory Devices:** RAM (Volatile, SRAM/DRAM) vs ROM (Non-volatile).
- **Logic Families:** Grouping compatible ICs (TTL, CMOS, ECL).
- **Key parameters:** Fan-in, Fan-out, Noise Margin, Delay, Power Dissipation.
- **Comparison:** CMOS (low power, dominant) vs TTL (legacy) vs ECL (ultra-fast).
- Interfacing rules and E-waste management.

Summary

- Engineers must consider the environmental lifecycle (E-waste) of their designs.
- Digital Logic Design provides the fundamental building blocks of all modern computing.
- From basic binary numbers to complex memory architectures, we have covered the journey from theory to physical implementation.
- Mastery of these concepts is essential for further study in computer architecture and embedded systems.



Course Conclusion

- Thank you for your hard work and participation this semester.
- Review your notes, practice K-maps and circuit diagrams, and understand the trade-offs in logic families.
- Good luck on your final examinations!
- Keep building, and design responsibly.

Thank You!

```
10101011 00101010 11001100
/* End of Course */
```

Questions?