Digital Systems & Microcontrollers

Midsem Answer Key
27th September, 2025

Question-1

Design a 5-to-32 line decoder using four 3-to-8 line decoders with enable inputs and one additional logic gate. **[5 marks]**

Solution:

Step 1: Input Partitioning

A 5-to-32 decoder requires 5 inputs (let's call them A_4, A_3, A_2, A_1, A_0) and produces 32 unique outputs (Y_0 to Y_{31}). The core components available are four 3-to-8 decoders, each having 3 address inputs, 8 outputs, and an enable (E) input. The enable input must be active for the decoder to function; otherwise, all its outputs are disabled.

The five inputs of the main decoder are partitioned to manage the available hardware.

- The three least significant bits (LSBs), $\mathbf{A_2}, \mathbf{A_1}, \mathbf{A_0}$, are used as the common address inputs for all four 3-to-8 decoders. They are connected in parallel to each decoder to select one of the 8 outputs within the enabled block.
- The two most significant bits (MSBs), $\mathbf{A_4}, \mathbf{A_3}$, are used to select which of the four 3-to-8 decoders should be active at any given time.

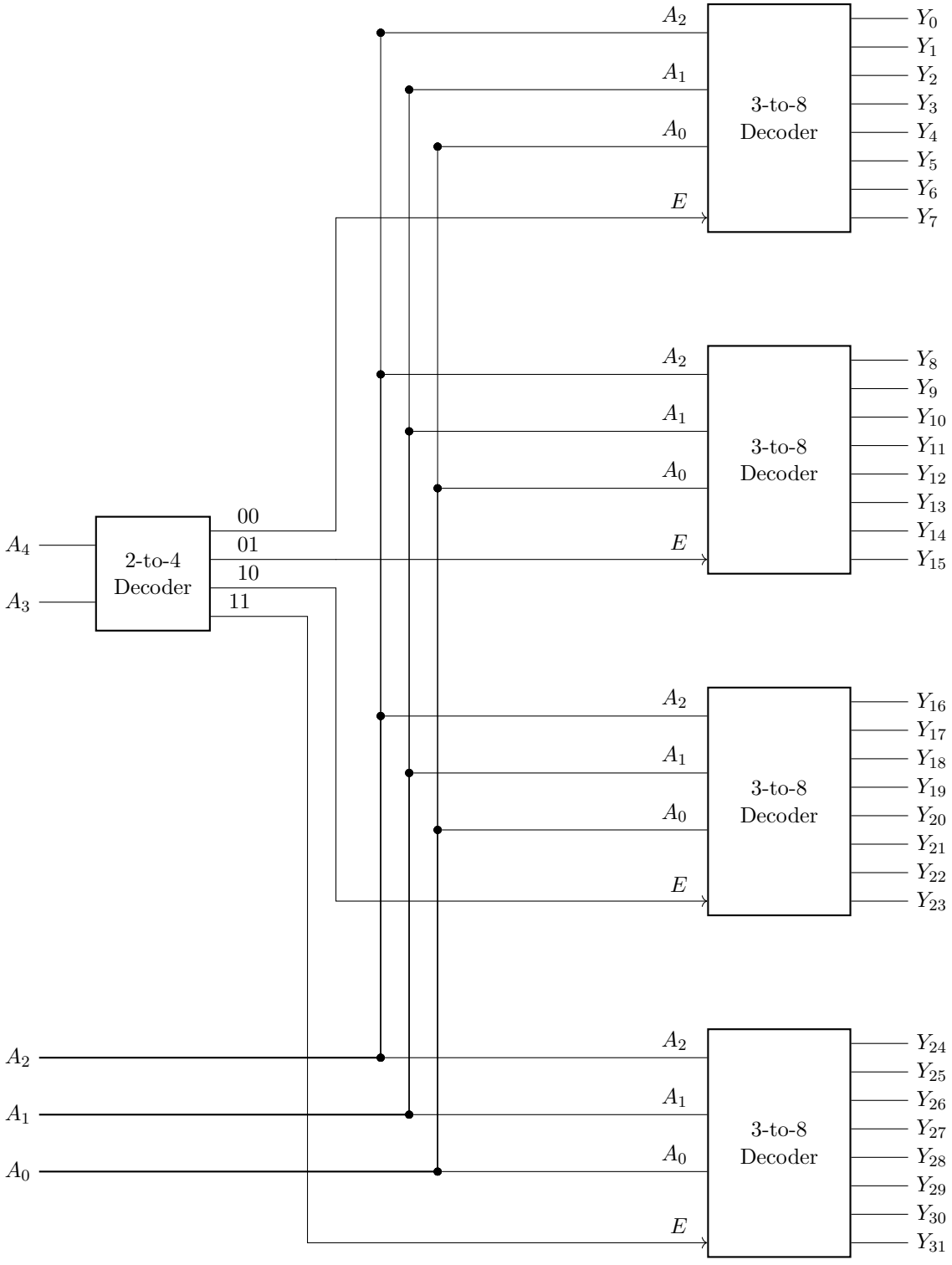
Step 2: Designing the Enable Logic

To select one of the four decoders, the two MSBs (A_4, A_3) are used as inputs to a 2-to-4 decoder. The four outputs of this 2-to-4 decoder are used to drive the enable (E) inputs of the four 3-to-8 decoders. For any given combination of A_4 and A_3, only one of the four enable lines will be asserted.

The overall 32-bit output is generated by combining the outputs of the four smaller decoders. The Enable logic determines which block of outputs is active:

- If $\mathbf{A_4A_3 = 00}$, the first 3-to-8 decoder is enabled, producing outputs $\mathbf{Y_0}$ to $\mathbf{Y_7}$.

- If $\mathbf{A_4A_3 = 01}$, the second 3-to-8 decoder is enabled, producing outputs $\mathbf{Y_8}$ to $\mathbf{Y_{15}}$.
- If $\mathbf{A_4A_3 = 10}$, the third 3-to-8 decoder is enabled, producing outputs $\mathbf{Y_{16}}$ to $\mathbf{Y_{23}}$.
- If $\mathbf{A_4A_3 = 11}$, the fourth 3-to-8 decoder is enabled, producing outputs $\mathbf{Y_{24}}$ to $\mathbf{Y_{31}}$.



Instead of using a 2-to-4 decoder, one can also use basic logic gates to create the enable signals as shown below:

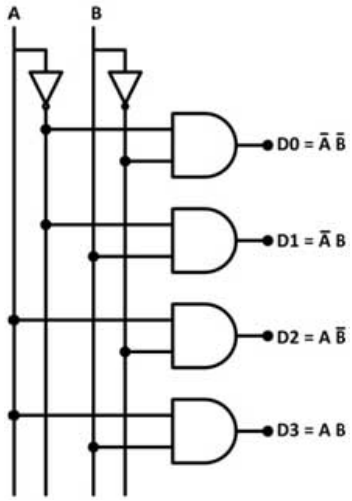


Figure 1: Logic circuit for 2-to-4 Decoder

Marking Scheme for Question 1:

- **[2 Marks]** Correct partitioning of inputs into MSBs and LSBs.
  - Correctly identifying A_4, A_3 as MSBs and A_2, A_1, A_0 as LSBs (1 mark)
  - Connecting them to the decoders in circuit diagram (1 mark)
  - Partial marks: Deduct 1 mark for incorrect partitioning or mislabeling to decoders.
- **[3 Marks]** Enable logic
  - Correct implementation using a 2-to-4 decoder or equivalent logic gates combinational circuit (2 marks)
  - Connecting enable outputs to the correct decoders (1 mark)
  - Partial marks: Deduct 0.5 mark for each wrong wiring or mislabeling

Question-2

Implement the following Boolean function using 8:1 multiplexer by converting it into canonical form. **[5 marks]**

F(A, B, C, D) = A'BD' + ACD + A'C' D +B'CD

Solution:

Step 1: Convert the Function to Canonical Sum-of-Products (SOP) Form

Given:

$$F(A, B, C, D) = A'BD' + ACD + A'C'D + B'CD$$

Let's list the minterms for each term:

- **Term 1:** $A'BD'$
  $A'BD' = A'B(C + C')D' = A'BCD' + A'BC'D'$
  This corresponds to binary values 0110_2 (6) and 0100_2 (4). Minterms: m_6, m_4.
- **Term 2:** ACD
  $ACD = A(B + B')CD = ABCD + AB'CD$
  This corresponds to binary values 1111_2 (15) and 1011_2 (11). Minterms: m_{15}, m_{11}.
- **Term 3:** $A'C'D$
  $A'C'D = A'(B + B')C'D = A'BC'D + A'B'C'D$
  This corresponds to binary values 0101_2 (5) and 0001_2 (1). Minterms: m_5, m_1.
- **Term 4:** $B'CD$
  $B'CD = (A + A')B'CD = AB'CD + A'B'CD$
  This corresponds to binary values 1011_2 (11) and 0011_2 (3). Minterms: m_{11} (duplicate) and m_3.

So, Combining all the minterms, function can be expressed in canonical SOP form:

$$\begin{aligned} F(A, B, C, D) &= A'B(C + C')D' + A(B + B')CD + A'(B + B')C'D + (A + A')B'CD \\ &= A'BCD' + A'BC'D' + ABCD + \underbrace{AB'CD} + A'BC'D + A'B'C'D + \underbrace{AB'CD} \\ &\quad + A'B'CD \\ &= A'BCD' + A'BC'D' + ABCD + AB'CD + A'BC'D + A'B'C'D + A'B'CD \end{aligned}$$

Or, equivalently (listing minterms) (removing duplicates), the canonical SOP form can be written as:

$$F(A, B, C, D) = \sum m(1, 3, 4, 5, 6, 11, 15)$$

Step 2:Implementation using 8:1 Multiplexer

To use an 8:1 MUX, select three variables as select lines and express F in terms of the remaining variable.

Let A, B, and C be the select lines. For each combination of A, B, C, determine F as a function of D:

Data Input	**Select Lines**			**Minterms Involved**	**Required MUX Input**
	A (S_2)	**B** (S_1)	**C** (S_0)	**(for D=0, D=1)**	**F in terms of D**
I_0	0	0	0	m_0, m_1	D
I_1	0	0	1	m_2, m_3	D
I_2	0	1	0	m_4, m_5	1
I_3	0	1	1	m_6, m_7	D'
I_4	1	0	0	m_8, m_9	0
I_5	1	0	1	m_{10}, m_{11}	D
I_6	1	1	0	m_{12}, m_{13}	0
I_7	1	1	1	m_{14}, m_{15}	D

Table 1: 8:1 MUX Implementation Table

How to read the table: For each combination of select lines (A, B, C), we check our minterm list $F = \sum m(1, 3, 4, 5, 6, 11, 15)$.

- **For I_0 (A'B'C'):** The possible minterms are m_0 (when D=0) and m_1 (when D=1). Since only m_1 is in our function, the output F should be 1 only when D=1. Therefore, $I_0 = D$.
- **For I_2 (A'BC'):** The possible minterms are m_4 (D=0) and m_5 (D=1). Since both are in our function, the output F is always 1 for this combination of select lines. Therefore, $I_2 = 1$.
- **For I_3 (A'BC):** The possible minterms are m_6 (D=0) and m_7 (D=1). Since only m_6 is in our function, the output F should be 1 only when D=0. Therefore, $I_3 = D'$.
- **For I_4 (AB'C'):** The possible minterms are m_8 (D=0) and m_9 (D=1). Since neither is in our function, the output F is always 0. Therefore, $I_4 = 0$.

This logic is applied to all rows.

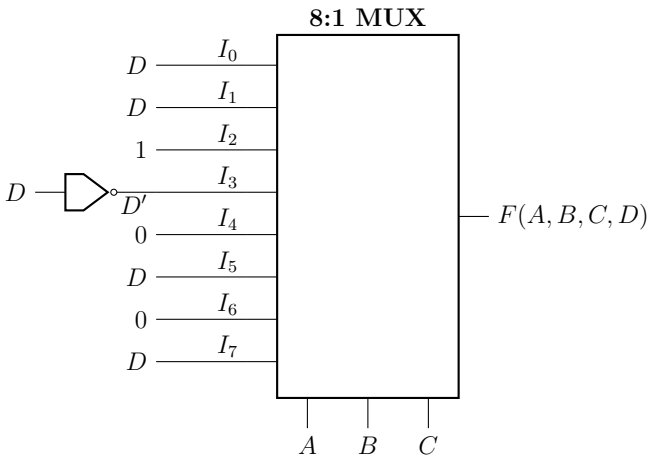


Figure 2: 8:1 MUX with A, B, C as select lines

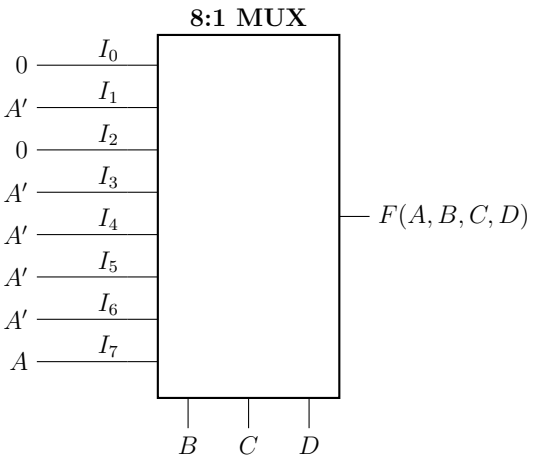


Figure 3: 8:1 MUX with B, C, D as select lines

Marking Scheme for Question 2: [5 Marks]

- **[2 Marks]** Converting to Canonical Form.
  - Full marks (2): Correct expression in either Σm notation or expanded product terms
  - Partial marks: Deduct 0.5 marks for each incorrect minterm, with minimum total of 0 marks
- **[2 Marks]** MUX Implementation
  - Full marks (2): Correctly identified select lines, data inputs, and their corresponding values
  - Partial marks: Deduct 0.5 marks for each incorrect connection or mislabelled input/output
- **[1 Mark]** Circuit Diagram
  - Full marks (1): Correct diagram with all proper labelled inputs and select lines.
  - Half marks (0.5): Minor errors ($\leq$ 3 labels incorrect) or unclear labeling

Question-3

Game of Keys for Children: Two kids have access to two individual BCD keyboards that have keys labelled from 0 to 9. One keyboard is Red in Color and the other one is Green in Color given to the two kids. Design a circuit that says whether the key pressed on the Red keyboard is greater than or equal to or lesser than the Green keyboard. Clearly mention the building blocks or combinatorial modules that comprise this design. **[7 marks]**

Solution: The problem requires designing a digital circuit to compare two decimal numbers (0-9) from two different BCD keyboards. The design can be broken down into two main combinatorial modules:

1. **Encoders:** Two 10-line-to-4-line encoders are needed to convert the decimal key press from each keyboard into its 4-bit Binary Coded Decimal (BCD) equivalent.
2. **4-Bit Magnitude Comparator:** This module takes the two 4-bit BCD outputs from the encoders and determines if one number is greater than, equal to, or less than the other. It will have three corresponding outputs.

1. Encoder Block

Each keyboard has 10 input lines (one for each key, 0-9) and needs to produce a 4-bit binary output. A 10-line-to-4-line priority encoder is the ideal component. We will need two of these circuits, one for each keyboard.

- The first encoder will produce the 4-bit number $A = A_3A_2A_1A_0$.
- The second encoder will produce the 4-bit number $B = B_3B_2B_1B_0$.

The circuit diagram for a standard 10-to-4 encoder is assumed to be a known building block.

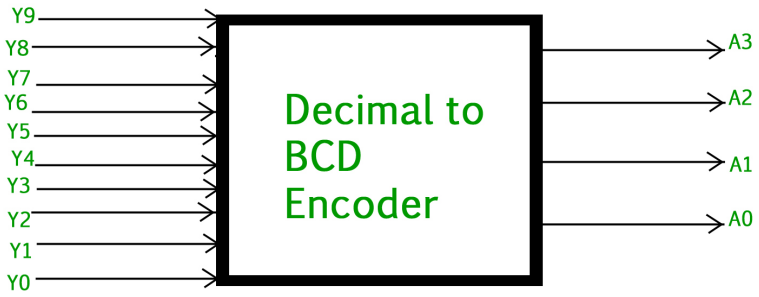


Figure 4: Block diagram showing encoders for both keyboards.

2. 4-Bit Magnitude Comparator Block

This block compares the two 4-bit numbers $A(A_3A_2A_1A_0)$ and $B(B_3B_2B_1B_0)$ and generates three outputs: $O_{A>B}$, $O_{A=B}$, and $O_{A<B}$.

To derive the logic, we first define a helper variable, x_i, for bitwise equality (XNOR gate):

$$x_i = A_i \odot B_i = A_iB_i + \overline{A_iB_i}$$

Logic for Equality ($O_{A=B}$): The two numbers A and B are equal if and only if all their corresponding bits are equal.

$$\boldsymbol{O_{A=B} = x_3 \cdot x_2 \cdot x_1 \cdot x_0}$$

Logic for Greater Than ($O_{A>B}$): The number A is greater than B if the most significant bit where they differ is a 1 in A and a 0 in B. This can be expressed as a cascading logic:

- $A_3 > B_3$ (i.e., $A_3 = 1, B_3 = 0$), OR
- $A_3 = B_3$ AND $A_2 > B_2$, OR
- $A_3 = B_3$ AND $A_2 = B_2$ AND $A_1 > B_1$, OR
- $A_3 = B_3$ AND $A_2 = B_2$ AND $A_1 = B_1$ AND $A_0 > B_0$

This translates to the following Boolean expression:

$$\boldsymbol{O_{A>B} = A_3\overline{B_3} + x_3A_2\overline{B_2} + x_3x_2A_1\overline{B_1} + x_3x_2x_1A_0\overline{B_0}}$$

Logic for Less Than ($O_{A<B}$): Similarly, A is less than B if the most significant bit where they differ is a 0 in A and a 1 in B.

$$\boldsymbol{O_{A<B} = \overline{A_3}B_3 + x_3\overline{A_2}B_2 + x_3x_2\overline{A_1}B_1 + x_3x_2x_1\overline{A_0}B_0}$$

The complete circuit would be a direct implementation of these three equations using logic gates.

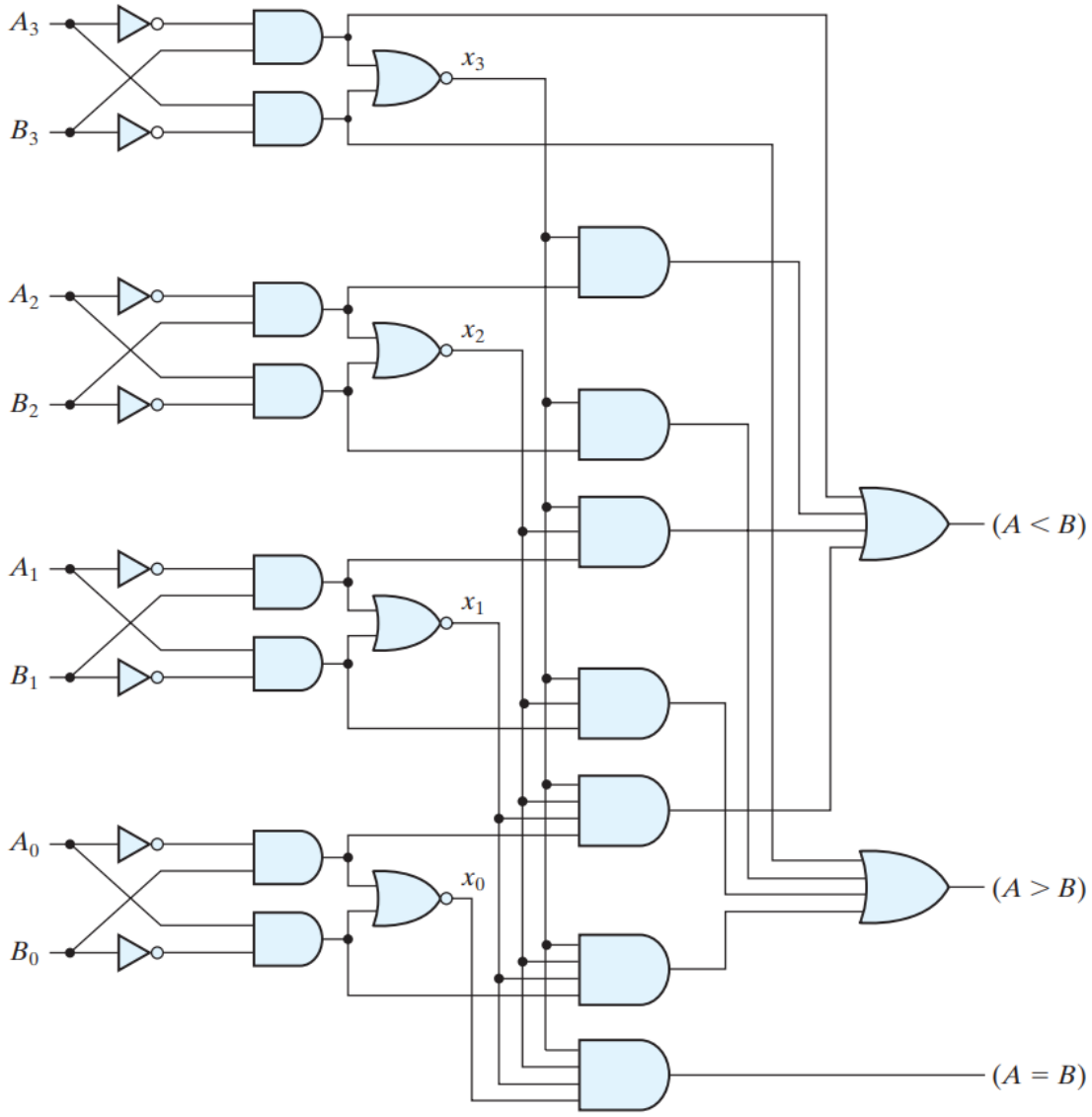


Figure 5: Logic diagram for the 4-bit Magnitude Comparator.

Marking Scheme for Question 3: [7 Marks + Bonus]

- **[1 Mark]** Correctly identifying the necessary combinatorial blocks/circuit diagram as a black box.
  - **(1 mark)** Circuit/some basic block diagram, even a black box comparator having 8 inputs (4 for each keyboard) and 3 outputs will be enough with proper labelling.
  - **(0.5 mark)** If circuit diagram not properly labelled.
- **[6 Marks]** (2×3) Derivation and implementation for $O_{A=B}$, $O_{A<B}$, $O_{A>B}$ (2 marks for each output). Distribution of 2 marks is as following:
  - **(2 marks)** Correct Boolean equation and a corresponding valid circuit diagram.
  - **(1.5 marks)** Correct equation, but the circuit has one minor error (e.g., one missing/wrong gate or connection).
  - **(1 mark)** Circuit has one minor error (e.g., one missing/wrong gate or connection).
  - **(1 mark)** Correct equation, but the circuit diagram is wrong or missing.
  - **(0.5 marks)** Correct truth table or logic description in words, but the equation and circuit are incorrect.

  Note: If any two of the three outputs are used to get the third one, it will be considered as the correct approach and will fetch full marks for correct implementation.
- **[1 Bonus Mark]** For correctly including the Encoder Blocks in circuit diagram with proper labelling of inputs/outputs.

Question-4

You are in control of Chandrayaan and it is communicating data to you which can be corrupted due to radiation/solar flares. How can you device a simple scheme to be able to know if there has been one bit flip (one bit data corruption). You can assume data size to be three bits. Implement the mechanism using digital logic. Also, suggest a simple mechanism (scheme) so that China cannot read your data correctly. **[8 marks]**

Solution: This problem has two distinct parts. The first part requires designing a simple error-detection scheme to identify a single-bit corruption in a 3-bit data transmission from Chandrayaan. This involves adding redundant information to the original data. The second part asks for a simple encryption (obfuscation) scheme to prevent an unauthorized party from correctly interpreting the transmitted data.

Part 1: Single Bit-Flip Detection

Intuition: The core idea behind detecting a bit flip is to add one or more redundant bits (check bits) to the original data before transmission. These check bits are calculated based on the original data bits. The receiver performs the same calculation on the received data bits and compares its result with the received check bit(s). If a single bit (either data or check bit) flips during transmission, the calculated check bit at the receiver's end will not match the received check bit, thus signaling an error.

Any digital logic circuit and corresponding Boolean expression that correctly implements this principle of creating and checking redundant information will be considered a valid solution and awarded marks.

Implementation using a Parity Bit: The simplest and most common method for single-bit error detection is using a single parity bit. Let the 3-bit data be D_2, D_1, D_0. We can use an even parity scheme, where we add a parity bit, P, such that the total number of 1s in the transmitted 4-bit message (D_2, D_1, D_0, P) is even.

Transmitter Logic: The parity bit P is generated by XORing the data bits. If the number of 1s in the data is odd, the XOR result is 1, making the total count of 1s even. If the number of 1s is already even, the XOR result is 0. The expression for the parity bit is:

$$P = D_2 \oplus D_1 \oplus D_0$$

The transmitter sends the 4-bit packet: $\{D_2, D_1, D_0, P\}$.

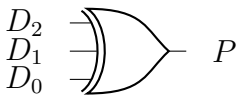


Figure 6: Transmitter Circuit: Parity Bit Generation

Receiver Logic: The receiver gets a (potentially corrupted) 4-bit packet: $\{D_2', D_1', D_0', P'\}$. To check for errors, it calculates a check bit, C, by XORing all the received bits.

$$C = D_2' \oplus D_1' \oplus D_0' \oplus P'$$

- If $C = 0$, no single-bit error occurred. The received data is considered valid.
- If $C = 1$, a single-bit error was detected. The data is corrupt.

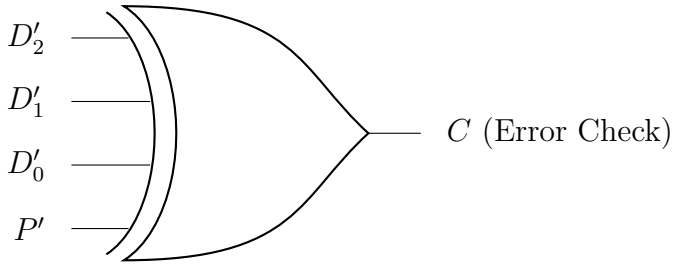


Figure 7: Receiver Circuit: Error Detection (Spaced Out)

Other Methods: Other valid, though more complex, methods include:

- **Repetition Code:** Transmitting the data chunk twice (e.g., $D_2D_1D_0D_2D_1D_0$) and checking for equality at the receiver.
- **Checksum:** Calculating a sum of the data bits and transmitting it.
- **Hamming Code:** A more advanced technique that can not only detect but also correct single-bit errors.

Part 2: Simple Encryption Scheme

Intuition: The goal of a simple encryption scheme is to obfuscate the data so that it is unreadable to an eavesdropper, but easily recoverable by the intended recipient. This is achieved by applying a reversible transformation to the data using a secret key that is known only to the sender and receiver. Any strategy that ensures the transmitted information is not the original data and can be easily decrypted by the receiver is a valid solution.

Implementation using XOR Cipher: A very simple and effective method is to XOR the data bits with a secret 3-bit key, $K = K_2K_1K_0$. Let the transmitted, encrypted data be $C = C_2C_1C_0$.

Encryption (at Transmitter): Each data bit is XORed with the corresponding key bit.

$$C_2 = D_2 \oplus K_2 \quad | \quad C_1 = D_1 \oplus K_1 \quad | \quad C_0 = D_0 \oplus K_0$$

The transmitter sends the encrypted data $C_2C_1C_0$ (along with its parity bit, if combined with the error detection scheme).

Decryption (at Receiver): The receiver, knowing the secret key K, performs the exact same XOR operation on the received ciphertext to recover the original data. This works because $A \oplus B \oplus B = A$.

$$D_2 = C_2 \oplus K_2 \quad | \quad D_1 = C_1 \oplus K_1 \quad | \quad D_0 = C_0 \oplus K_0$$

Other Methods: Other valid schemes include modular arithmetic (e.g., adding a secret number to the data and taking modulo-8) or bit-swapping/permutation based on a secret pattern. Any correct solution that is explained clearly will fetch marks.

Marking Scheme for Question 4: [8 Marks]

- **[6 Marks]** Single Bit-Flip Detection
  - **[2 Marks]** Explanation/Intuition:
    - Full marks (2): A clear explanation of using redundant information (like a parity bit) and checking this information at the receiver to detect a mismatch.
    - Partial marks (1): A partially correct or vague idea, such as adding an extra bit to check without explaining the mechanism.
  - **[4 Marks]** Circuit and Expression:
    - Full marks (4): Correct Boolean expressions and corresponding circuit diagrams for both the transmitter (generation) and receiver (checking). A correct circuit without an explicit expression also gets full marks.
    - 3 marks: A minor error in both the expression and circuit (e.g., one wrong gate, one wrong connection) OR a correct expression with a minor error in the circuit.
    - 3 marks: If both the expressions are correct but only 1 circuit is drawn of the two.
    - 2.5 marks: A minor error in the circuit and the expression is missing.
    - 2 marks: Only 1 side's circuit and expression are given even when the logic required both.
    - 2 marks: Correct expression(s) but the circuit is missing or entirely incorrect.
    - 1.5 marks: A minor error in the expression(s) and the circuit is missing.
    - 1 mark: Only 1 side's expression is given and no circuit is drawn.
    - NOTE: If the encryption does not require boolean logic (like retransmitting the same message twice), only receiver's side circuit and the complete explanation will also fetch full marks
- **[2 Marks]** Encryption Mechanism
  - Full marks (2): A complete and correct mechanism (like XOR cipher) is explained. An explanation is sufficient, but providing a correct expression or circuit is also valid.
  - 1.5 marks: A correct mechanism is proposed, but the expression or circuit provided has a minor error.
  - Partial marks (1): A partially correct mechanism is proposed (e.g., one that is not easily reversible or has logical flaws).

Question-5

Implement in 3 bit binary the function in y= 2x-1 where x, y are input 'x' decimal numbers 1 to 4 using NAND gates. **[5 marks]**

Solution:

Step 1: Truth Table Construction

First, we map the decimal inputs x and outputs y to their 3-bit binary equivalents. Let the input bits be x_2, x_1, x_0 and the output bits be y_2, y_1, y_0. The valid input range for x is [1, 2, 3, 4]. Other binary inputs (0, 5, 6, 7) are considered "don't care" conditions (denoted by 'X'), which can be used for simplification.

Decimal	**Binary Input**			**Decimal**	**Binary Output**		
Input (x)	x_2	x_1	x_0	**Output** $(y = 2x - 1)$	y_2	y_1	y_0
0	0	0	0	(Don't Care)	X	X	X
1	**0**	**0**	**1**	**1**	**0**	**0**	**1**
2	**0**	**1**	**0**	**3**	**0**	**1**	**1**
3	**0**	**1**	**1**	**5**	**1**	**0**	**1**
4	**1**	**0**	**0**	**7**	**1**	**1**	**1**
5	1	0	1	(Don't Care)	X	X	X
6	1	1	0	(Don't Care)	X	X	X
7	1	1	1	(Don't Care)	X	X	X

Table 2: Truth Table for the function $y = 2x - 1$

Step 2: Boolean Expression Derivation (K-Maps)

We create a K-map for each output bit (y_2, y_1, y_0) to find the simplest Sum-of-Products (SOP) expression.

K-Map for y_2

The minterms for $y_2 = 1$ are $m(3, 4)$ with don't cares at $d(0, 5, 6, 7)$.

	x_1x_0			
x_2	00	01	11	10
0	X	0	**1**	0
1	**1**	X	X	X

Grouping the '1's with the don't cares gives two terms:

- A group for the entire bottom row: x_2
- A group for the column where $x_1x_0 = 11$: x_1x_0

The simplified expression is: $\boldsymbol{y_2 = x_2 + x_1 x_0}$

K-Map for y_1

The minterms for $y_1 = 1$ are $m(2, 4)$ with don't cares at $d(0, 5, 6, 7)$.

	x_1x_0			
x_2	00	01	11	10
0	X	0	0	**1**
1	**1**	X	X	X

The simplified expression is: $\boldsymbol{y_1 = \overline{x_0}}$ or $\boldsymbol{y_1 = x_2 + x_1\overline{x_0}}$ (Both expressions are correct)

K-Map for y_0

The minterms for $y_0 = 1$ are $m(1, 2, 3, 4)$. As seen in the truth table, y_0 is always 1 for all valid inputs. The K-map confirms this.

	x_1x_0			
x_2	00	01	11	10
0	X	**1**	**1**	**1**
1	**1**	X	X	X

Since all valid inputs result in $y_0 = 1$, the simplified expression is: $\boldsymbol{y_0 = 1}$

Step 3: Conversion to NAND Logic

The NAND gate can be used to create NOT, AND, and OR functions as follows:

- $\overline{A} = \overline{A \cdot A}$
  NOT A = A NAND A
- $A \cdot B = \overline{\overline{A \cdot B} \cdot \overline{A \cdot B}}$
  A AND B = (A NAND B) NAND (A NAND B)
- $A + B = \overline{\overline{A} \cdot \overline{B}} = \overline{\overline{A \cdot A} \cdot \overline{B \cdot B}}$
  A OR B = (NOT A) NAND (NOT B) = (A NAND A) NAND (B NAND B)

Now, we convert each SOP expression into a circuit that uses only NAND gates. We use the property $A + B = \overline{\overline{A} \cdot \overline{B}} = \text{NAND}(\text{NAND}(A, A), \text{NAND}(B, B))$ and De Morgan's laws.

Expression for y_2: $y_2 = x_2 + x_1x_0$
Apply double negation: $y_2 = \overline{\overline{(x_2 + x_1x_0)}}$ Apply De Morgan's theorem: $y_2 = \overline{\overline{x_2} \cdot \overline{(x_1x_0)}}$
This is the NAND-NAND implementation. $\overline{A}$ is $\text{NAND}(A, A)$ and $\overline{A \cdot B}$ is $\text{NAND}(A, B)$.

$$y_2 = \text{NAND}(\text{NAND}(x_2, x_2), \text{NAND}(x_1, x_0))$$

Expression for y_1: $y_1 = \overline{x_0}$ or $y_1 = x_2 + x_1\overline{x_0}$
Apply double negation and De Morgan's theorem: $y_1 = \overline{\overline{x_2} \cdot \overline{(x_1\overline{x_0})}}$ In NAND form:

$$y_1 = \text{NAND}(\text{NAND}(x_2, x_2), \text{NAND}(x_1, \text{NAND}(x_0, x_0)))$$

or simply,

$$y_1 = \overline{x_0} = \text{NAND}(x_0, x_0)$$

Expression for y_0: $y_0 = 1$
To generate a constant '1' using NAND gates from an input line, we can use the identity $A + \overline{A} = 1$. In NAND logic, this is $1 = \text{NAND}(\text{NAND}(A, \overline{A}), \text{NAND}(A, \overline{A}))$. A simpler form is to use the property that $\text{NAND}(A, \overline{A}) = \overline{A \cdot \overline{A}} = \overline{0} = 1$. Let's use x_0 as the source input:

$$y_0 = \text{NAND}(x_0, \overline{x_0}) = \text{NAND}(x_0, \text{NAND}(x_0, x_0))$$

If assuming VDD is available (which is in most cases), y = 1 is also considered correct. The final 3-bit binary implementation of $y = 2x - 1$ using only NAND gates is given by the following three equations:

$$y_2 = \text{NAND}(\text{NAND}(x_2, x_2), \text{NAND}(x_1, x_0))$$
$$y_1 = \text{NAND}(\text{NAND}(x_2, x_2), \text{NAND}(x_1, \text{NAND}(x_0, x_0))) = \text{NAND}(x_0, x_0)$$
$$y_0 = 1 = \text{NAND}(x_0, \text{NAND}(x_0, x_0))$$

These equations describe the complete logic circuit required to realize the function.

Marking Scheme for Question 5:

- **[1.5 Marks]** Boolean Expression Derivation
  - Full marks (1.5): Correct K-maps and simplified expressions for y_2, y_1, and y_0.
  - Partial marks: Deduct 0.5 marks for each incorrect expression. Award 0.25 marks in case the K-map (if made) is correct but the expression is incorrect.
- **[2.5 Marks]** NAND Gate Implementation
  - Full marks (2.5): Correct conversion of all three expressions into NAND-only logic. 1 mark each for y_2 and y_1, 0.5 marks for y_0.
  - Partial marks: 1 mark if conversions (≥ 2) are not correct, but showed the correct implementation of AND, NOT, OR logic using NAND.
- **[1 Mark]** Circuit Diagram
  - Full marks (1): Correct diagram with all proper labelled inputs and outputs.
  - Half marks (0.5): Minor errors (≤ 2 wrong wires connections) or unclear labeling.