Digital Systems & Microcontrollers

Tutorial Quiz Week - 2 Solutions
9th - 11th September, 2025

---

9 September (Tuesday)

Set-A: Design a combinational circuit that compares two 4-bit numbers to check if they are unequal. The circuit output is equal to 1 if the two numbers are not equal and 0 if equal.

Solution: The objective is to design a combinational circuit that compares two 4-bit numbers, $A = A_3A_2A_1A_0$ and $B = B_3B_2B_1B_0$. The circuit's output, Y, should be 1 if the numbers are not equal ($A \neq B$) and 0 if they are equal ($A = B$).

Logical Approach

The most direct way to solve this is by considering the condition for inequality. Two numbers, A and B, are unequal if at least one of their corresponding bits is different. This can be expressed as a logical OR of the inequalities of each bit pair:

$$Y = (A_3 \neq B_3) \text{ OR } (A_2 \neq B_2) \text{ OR } (A_1 \neq B_1) \text{ OR } (A_0 \neq B_0)$$

This modular approach allows us to first analyze a single bit comparison and then combine the results for the full 4-bit comparison. The logical operation that checks if two bits are unequal is the Exclusive OR (XOR) gate.

Truth Table for a 1-bit Comparator

A full truth table for the 8 inputs (A_3, A_2, A_1, A_0 and B_3, B_2, B_1, B_0) would have $2^8 = 256$ rows, which is impractical for manual analysis and K-map simplification. Instead, we can create a truth table for a generic 1-bit comparator for any bit pair (A_i, B_i). The output Y_i is 1 if $A_i \neq B_i$.

A_i	B_i	Y_i **(output is 1 if** $A_i \neq B_i$**)**
0	0	0
0	1	1
1	0	1
1	1	0

As seen from the table, this is the function of an XOR gate. The expression for a single bit comparison is:

$$Y_i = A_i \oplus B_i$$

K-Map for a 1-bit Comparator

We can verify the expression for the 1-bit comparator using a 2-variable K-map.

		B_i
A_i	**0**	**1**
0	0	1
1	1	0

The 1s in the K-map are non-adjacent, meaning they cannot be grouped to simplify the expression further. Reading the minterms directly gives us the Sum-of-Products form:

$$Y_i = \overline{A_i}B_i + A_i\overline{B_i}$$

This is the definition of the XOR operation. Thus, $Y_i = A_i \oplus B_i$.

Final Expression for 4-bit Comparator

The final output Y is 1 if any of the individual bit comparisons yield a 1. Therefore, we OR the outputs of the four 1-bit comparators.

$$Y = Y_3 + Y_2 + Y_1 + Y_0$$

(Here, '+' represents the logical OR operation).

Substituting the XOR expression for each Y_i, we get the final Boolean expression for the 4-bit inequality comparator:

$$Y = (A_3 \oplus B_3) + (A_2 \oplus B_2) + (A_1 \oplus B_1) + (A_0 \oplus B_0)$$

This circuit is implemented using four 2-input XOR gates, with the output of each gate feeding into a single 4-input OR gate to produce the final output Y.

Set-B: Design a combinational circuit that compares two 4-bit numbers to check if they are equal. The circuit output is equal to 1 if the two numbers are equal and 0 if not.

Solution: The goal is to design a combinational circuit that compares two 4-bit numbers, $A = A_3A_2A_1A_0$ and $B = B_3B_2B_1B_0$. The circuit's output, Y, is 1 if the numbers are equal ($A = B$) and 0 otherwise.

Logical Approach

For two numbers to be equal, it is necessary for all of their corresponding bits to be identical. This implies a logical AND operation on the results of each individual bit comparison. The condition for equality can be expressed as:

$$Y = (A_3 = B_3) \text{ AND } (A_2 = B_2) \text{ AND } (A_1 = B_1) \text{ AND } (A_0 = B_0)$$

The logical gate that outputs a 1 when its inputs are equal is the Exclusive-NOR (XNOR) gate. We will use this as the fundamental building block for our comparator.

Truth Table for a 1-bit Comparator

Instead of creating a full 256-row truth table for the eight inputs, we can analyze a single generic bit pair (A_i, B_i). The output of this 1-bit comparator, Y_i, will be 1 only if $A_i = B_i$.

A_i	B_i	Y_i **(output is 1 if** $A_i = B_i$**)**
0	0	1
0	1	0
1	0	0
1	1	1

This truth table corresponds to the XNOR logical operation. The expression for a single bit equality check is:

$$Y_i = A_i \odot B_i$$

K-Map for a 1-bit Comparator

We can confirm the expression for the 1-bit comparator using a 2-variable Karnaugh map (K-map).

A_i \ B_i	**0**	**1**
0	1	0
1	0	1

The 1s in the K-map correspond to the minterms where A_i and B_i are identical. Reading the minterms directly gives the Sum-of-Products expression:

$$Y_i = \overline{A_i B_i} + A_i B_i$$

This is the definition of the XNOR operation, confirming that $Y_i = A_i \odot B_i$.

Final Expression for 4-bit Comparator

To get the final output Y, we must AND the results from all four of the 1-bit comparators (Y_3, Y_2, Y_1, Y_0), because every bit pair must be equal for the numbers to be equal.

$$Y = Y_3 \cdot Y_2 \cdot Y_1 \cdot Y_0$$

(Here, '$\cdot$' represents the logical AND operation).

Substituting the XNOR expression for each Y_i, we arrive at the final Boolean expression for the 4-bit equality comparator:

$$Y = (A_3 \odot B_3) \cdot (A_2 \odot B_2) \cdot (A_1 \odot B_1) \cdot (A_0 \odot B_0)$$

The resulting circuit consists of four XNOR gates, with the output of each feeding into a single 4-input AND gate to produce the final output Y.

10th September (Wednesday)

Set-A: Design a 3-input majority circuit by finding the circuit's truth table, Boolean equation, and a logic diagram. A majority circuit is a combinational circuit whose output is equal to 1 if the input variables have more 1's than 0's. The output is 0 otherwise.

Solution: A 3-input majority circuit is a combinational circuit with three inputs (let's call them A, B, and C) and one output, Y. The output Y is equal to 1 if the number of 1s in the input variables is greater than the number of 0s (i.e., two or more inputs are 1). The output is 0 otherwise.

Truth Table

First, we construct a truth table that defines the output Y for every possible combination of inputs A, B, and C. The output is 1 for any combination with two or three 1s.

Inputs			**Output**
A	**B**	**C**	**Y**
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Boolean Equation and K-Map

To derive the simplified Boolean equation, we map the outputs from the truth table onto a 3-variable Karnaugh Map (K-map). The minterms where the output is 1 are m_3, m_5, m_6, and m_7.

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

We can form three groups of two adjacent 1s:

- The group of minterms m_3 ($\overline{A}BC$) and m_7 (ABC) simplifies to BC.

- The group of minterms m_5 ($A\overline{B}C$) and m_7 (ABC) simplifies to AC.
- The group of minterms m_6 ($AB\overline{C}$) and m_7 (ABC) simplifies to AB.

Combining these product terms with a logical OR gives the final simplified Sum-of-Products equation:

$$Y = AB + BC + AC$$

This expression is notably the same as the equation for the carry-out (C_{out}) of a full adder.

Logic Diagram

The logic diagram for the circuit is a direct implementation of the final Boolean equation. It consists of:

- Three 2-input AND gates.
- One 3-input OR gate.

The inputs to the AND gates are (A, B), (B, C), and (A, C) respectively. The outputs of these three AND gates are then fed into the 3-input OR gate to produce the final output, Y.

Set-B: Design a 3-input minority circuit by finding the circuit's truth table, Boolean equation, and a logic diagram. A minority circuit is a combinational circuit whose output is equal to 1 if the input variables have more 0's than 1's. The output is 0 otherwise.

Solution: A 3-input minority circuit is a combinational circuit whose output is 1 if the input variables have more 0s than 1s. For three inputs (A, B, and C), this means the output is 1 if there are two or three 0s (which is equivalent to zero or one 1s). The output is 0 otherwise.

Truth Table

We begin by creating a truth table that describes the circuit's function for all possible input combinations. The output Y is 1 for any combination with zero or one 1.

Inputs			Output
A	**B**	**C**	**Y**
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Boolean Equation and K-Map

To find the simplest Boolean expression, we transfer the outputs from the truth table to a 3-variable Karnaugh Map (K-map). The minterms where the output is 1 are m_0, m_1, m_2, and m_4.

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

We can form three groups of two adjacent 1s to cover all the minterms:

- The group of minterms m_0 ($\overline{A}\overline{B}\overline{C}$) and m_1 ($\overline{A}\overline{B}C$) simplifies to $\overline{A}\overline{B}$.
- The group of minterms m_0 ($\overline{A}\overline{B}\overline{C}$) and m_2 ($\overline{A}B\overline{C}$) simplifies to $\overline{A}\overline{C}$.
- The group of minterms m_0 ($\overline{A}\overline{B}\overline{C}$) and m_4 ($A\overline{B}\overline{C}$) simplifies to $\overline{B}\overline{C}$.

Combining these terms using a logical OR gives the final simplified Sum-of-Products equation:

$$Y = \overline{A}\overline{B} + \overline{A}\overline{C} + \overline{B}\overline{C}$$

This expression is the logical inverse of the 3-input majority circuit ($AB + AC + BC$).

Logic Diagram

The logic diagram is a graphical implementation of the derived Boolean equation. The circuit requires:

- Three NOT gates (inverters) to generate $\overline{A}$, $\overline{B}$, and $\overline{C}$.
- Three 2-input AND gates.
- One 3-input OR gate.

The inputs to the three AND gates are $(\overline{A}, \overline{B})$, $(\overline{A}, \overline{C})$, and $(\overline{B}, \overline{C})$. The outputs of these three AND gates are then connected as inputs to the 3-input OR gate, which produces the final output, Y.

11th September (Thursday) Slot-1

Set-A: Design a combinational circuit with three inputs, x , y , and z , and three outputs, A, B , and C. When the binary input is 0, 1, 2, or 3, the binary output is one greater than the input. When the binary input is 4, 5, 6, or 7, the binary output is two less than the input.

Solution: The problem requires designing a 3-input (x, y, z), 3-output (A, B, C) combinational circuit. The circuit's function is defined by two rules based on the decimal value of the binary input 'xyz':

- If the input is 0, 1, 2, or 3, the binary output is one greater than the input.
- If the input is 4, 5, 6, or 7, the binary output is two less than the input.

We will derive the solution by first creating a truth table, then using K-maps to find the simplified Boolean equations for each output.

Truth Table

First, we construct the complete truth table by calculating the output for each of the 8 possible input combinations according to the given rules.

Decimal Input	Inputs x	y	z	Decimal Output	Outputs A	B	C
0	0	0	0	0 + 1 = 1	0	0	1
1	0	0	1	1 + 1 = 2	0	1	0
2	0	1	0	2 + 1 = 3	0	1	1
3	0	1	1	3 + 1 = 4	1	0	0
4	1	0	0	4 - 2 = 2	0	1	0
5	1	0	1	5 - 2 = 3	0	1	1
6	1	1	0	6 - 2 = 4	1	0	0
7	1	1	1	7 - 2 = 5	1	0	1

K-Maps and Boolean Equations

Using the truth table, we create a separate K-map for each output (A, B, and C) to derive the simplified logic equations.

Equation for Output A The minterms for A=1 are m_3, m_6, and m_7.

		yz			
		00	01	11	10
x	0	0	0	1	0
	1	0	0	1	1

We can form two groups: one combining m_3 and m_7 to get yz, and another combining m_6 and m_7 to get xy. The resulting equation is:

$$A = xy + yz$$

Equation for Output B The minterms for B=1 are m_1, m_2, m_4, and m_5.

		yz			
		00	01	11	10
x	0	0	1	0	1
	1	1	1	0	0

The minterms in this map do not group conveniently in a standard Sum-of-Products form. We can group m_1 and m_5 to get $\overline{y}z$. The remaining minterms m_2 ($\overline{x}y\overline{z}$) and m_4 ($x\overline{y}\overline{z}$) cannot be grouped. The simplest expression involves XOR operations:

$$B = \overline{y}z + \overline{z}(x \oplus y)$$

Equation for Output C The minterms for C=1 are m_0, m_2, m_5, and m_7.

		yz			
		00	01	11	10
x	0	1	0	0	1
	1	0	1	1	0

The checkerboard pattern indicates an XNOR relationship. Grouping m_0 and m_2 gives $\overline{x}\overline{z}$. Grouping m_5 and m_7 gives xz. The equation is:

$$C = xz + \overline{x}\overline{z}$$

This is the definition of the XNOR operation:

$$C = x \odot z$$

Final Equations and Logic Diagram

The final simplified Boolean equations for the circuit are:

- $A = xy + yz$
- $B = \overline{y}z + \overline{z}(x \oplus y)$
- $C = x \odot z$

A logic diagram for this circuit would be constructed using the necessary AND, OR, NOT, XOR, and XNOR gates to implement these three equations from the inputs x, y, and z.

Set-B: Design a combinational circuit with three inputs, x , y , and z , and three outputs, A, B , and C. When the binary input is 0, 1, 2, or 3, the binary output is two greater than the input. When the binary input is 4, 5, 6, or 7, the binary output is one less than the input.

Solution: The task is to design a combinational circuit with three inputs (x, y, z) and three outputs (A, B, C). The circuit's behavior is governed by the following rules based on the decimal value of the binary input 'xyz':

- If the input is 0, 1, 2, or 3, the binary output is two greater than the input.
- If the input is 4, 5, 6, or 7, the binary output is one less than the input.

The design process involves creating a truth table from these rules and then using K-maps to derive the simplified logic equations for each output.

Truth Table

First, we establish the relationship between the inputs and outputs for all possible combinations in a truth table.

Decimal Input	**Inputs**			**Decimal Output**	**Outputs**		
	x	**y**	**z**		**A**	**B**	**C**
0	0	0	0	0 + 2 = 2	0	1	0
1	0	0	1	1 + 2 = 3	0	1	1
2	0	1	0	2 + 2 = 4	1	0	0
3	0	1	1	3 + 2 = 5	1	0	1
4	1	0	0	4 - 1 = 3	0	1	1
5	1	0	1	5 - 1 = 4	1	0	0
6	1	1	0	6 - 1 = 5	1	0	1
7	1	1	1	7 - 1 = 6	1	1	0

K-Maps and Boolean Equations

We will now create a K-map for each output (A, B, and C) to find the most simplified Boolean expression.

Equation for Output A The minterms where A=1 are m_2, m_3, m_5, m_6, and m_7.

		yz			
		00	01	11	10
x	0	0	0	1	1
	1	0	1	1	1

We can cover the 1s with two groups. A group of four (m_2, m_3, m_6, m_7) simplifies to y. The remaining 1 at m_5 can be grouped with m_7 to form the term xz. The resulting equation is:

$$A = y + xz$$

Equation for Output B The minterms where B=1 are m_0, m_1, m_4, and m_7.

		yz			
		00	01	11	10
x	0	1	1	0	0
	1	1	0	1	0

These minterms do not group easily. We can group m_0 and m_1 to get $\overline{xy}$. We can group m_0 and m_4 to get $\overline{yz}$. The minterm m_7 (xyz) is left and cannot be grouped. The simplest Sum-of-Products expression is:

$$B = \overline{xy} + \overline{yz} + xyz$$

Equation for Output C The minterms where C=1 are m_1, m_3, m_4, and m_6.

		yz			
		00	01	11	10
x	0	0	1	1	0
	1	1	0	0	1

The distinct checkerboard pattern indicates an XOR relationship. The minterms cannot be grouped. The expression derived from the minterms is $\overline{xy}z+\overline{x}yz+x\overline{yz}+xy\overline{z}$. This simplifies to:

$$C = \overline{x}z + x\overline{z}$$

This is the definition of the XOR operation:

$$C = x \oplus z$$

Final Equations and Logic Diagram

The set of simplified Boolean equations that describe the combinational circuit is:

- $A = y + xz$
- $B = \overline{xy} + \overline{yz} + xyz$
- $C = x \oplus z$

A logic diagram for this circuit would be constructed using the AND, OR, NOT, and XOR gates required to implement these three expressions from the primary inputs x, y, and z.

11th September (Thursday) Slot-2

Set-A: A logic circuit implements $F(A, B, C, D) = \sum(1, 3, 7, 11, 15)$. Simplify using K-map. Redesign the function using only 2-input multiplexers.

Solution: The given logic function is $F(A, B, C, D) = \sum(1, 3, 7, 11, 15)$, which means the function outputs 1 for minterms 1, 3, 7, 11, and 15.

K-Map Simplification

AB \ CD	00	01	11	10
00	0	1	1	0
01	0	0	1	0
11	0	0	1	0
10	0	0	1	0

$$\Rightarrow \quad \begin{aligned} \text{F(A,B,C,D)} &= A'B'D + CD \\ &= (A'B' + C).D \end{aligned}$$

Now we look for groups of adjacent 1s:

- Group 1: m_1 and m_3 can be grouped (adjacent in the same row). This simplifies to $A'B'D$.
- Group 2: m_3, m_7, m_{11}, and m_{15} form a column of four 1s in the $CD = 11$ column. This simplifies to CD.

However, we need to check if there's any overlap. The minterm m_3 is covered by both groups, which is acceptable in Boolean algebra.

The simplified Boolean expression is:

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

We can factor this as:

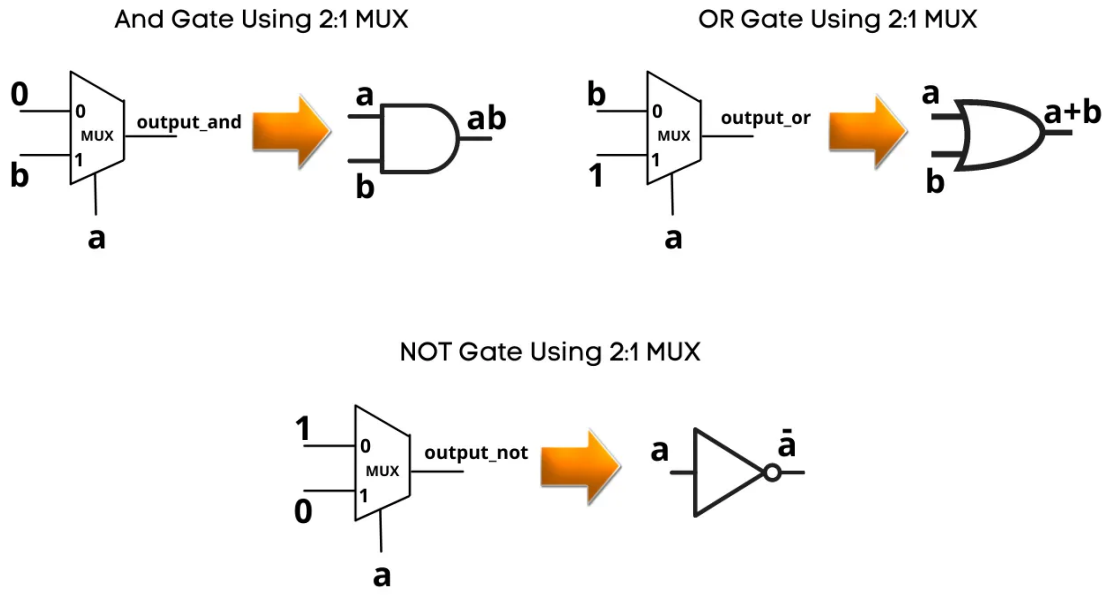
$$F(A, B, C, D) = D(A'B' + C)$$

Implementation Using 2-Input Multiplexers

A 2-input multiplexer has the function: $Y = \overline{S} \cdot I_0 + S \cdot I_1$, where S is the select line, I_0 and I_1 are the data inputs, and Y is the output.

To implement $F(A, B, C, D) = D(A'B' + C)$, we can use the following approach:

Method 1: Gates using MUX We can implement the AND, OR and NOT operations using 2-input multiplexers as shown below:



For AND gate: $Y = A.B$

- When $A = 0$: $Y = 0$
- When $A = 1$: $Y = B$

Therefore, $Y = \overline{A} \cdot I_0 + A \cdot I_1$, where $I_0 = 0$ and $I_1 = B$. (Also known as Shannon's Expansion)

Similarly for other gates:

For OR gate: $Y = \overline{A} \cdot I_0 + A \cdot I_1$, where $I_0 = A$ and $I_1 = 1$.

For NOT gate: $Y = \overline{A} \cdot I_0 + A \cdot I_1$, where $I_0 = 1$ and $I_1 = 0$.

Method 2: Direct Implementation We can use D as the select line for the main multiplexer:

- When $D = 0$: $F = 0$ (since the expression has D as a factor)
- When $D = 1$: $F = A'B' + C$

So we need:

- $I_0 = 0$ (constant)
- $I_1 = A'B' + C$

To implement $A'B' + C$, we can use another 2-input multiplexer with C as the select line:

- When $C = 0$: $A'B' + C = A'B'$
- When $C = 1$: $A'B' + C = 1$

For the second multiplexer:

- $I_0 = A'B'$ (which requires a NAND gate followed by a NOT gate, or two NOT gates and an AND gate)
- $I_1 = 1$ (constant)

Complete Implementation The complete implementation requires:

1. Two NOT gates to generate $\overline{A}$ and $\overline{B}$
2. One AND gate to compute $A'B'$
3. One 2-input multiplexer with C as select, $A'B'$ as I_0, and 1 as I_1
4. One 2-input multiplexer with D as select, 0 as I_0, and the output of the first multiplexer as I_1

Set-B: A logic circuit implements $F(A, B, C, D) = \Pi(0, 1, 4, 5, 9, 13)$. Simplify using K-map. Redesign the function using only 2-input multiplexers.

Solution: The given logic function is $F(A, B, C, D) = \Pi(0, 1, 4, 5, 9, 13)$, which means the function is expressed as a Product of Maxterms. The function outputs 0 for maxterms 0, 1, 4, 5, 9, and 13, and outputs 1 for all other combinations.

Converting to Sum of Minterms

First, we need to identify which minterms make the function equal to 1. Since we have maxterms 0, 1, 4, 5, 9, and 13, the function is 0 for these values and 1 for all others.

For a 4-variable function, we have minterms from 0 to 15. The minterms where $F = 1$ are:

$$\begin{aligned} F(A, B, C, D) &= \prod(0, 1, 4, 5, 9, 13) \\ &= \sum(2, 3, 6, 7, 8, 10, 11, 12, 14, 15) \end{aligned}$$

K-Map Simplification

We construct a 4-variable K-map and place 1s in the cells corresponding to the minterms where the function outputs 1.

AB \ CD	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	1	0	1	1
10	1	0	1	1

$\Rightarrow$ F(A,B,C,D) = $AC'D' + C$

Now we look for groups of adjacent 1s:

- Group 1: m_8 and m_{12} (adjacent vertically) simplify to $AC'D'$.

- Group 2: Remaining minterms form a 4×2 block, simplifying to C.

The most efficient grouping gives us:

$$F(A, B, C, D) = AC'D' + C$$

Method 1: Direct Implementation We can use C as the select line for the main multiplexer:

- When $C = 0$: $F = A\overline{D}$
- When $C = 1$: $F = 1$

So we need:

- $I_0 = A\overline{D}$
- $I_1 = 1$ (constant)

To implement $A\overline{D}$, we can use another 2-input multiplexer with A as the select line:

- When $A = 0$: $A\overline{D} = 0$
- When $A = 1$: $A\overline{D} = \overline{D}$

For the second multiplexer:

- $I_0 = 0$ (constant)
- $I_1 = \overline{D}$ (requires one NOT gate)

Method 2: Using Gate Implementations with MUXes As shown in Set-A, we can implement basic gates using 2-input multiplexers:

For AND gate: $Y = A \cdot B$ using MUX with A as select, $I_0 = 0$, $I_1 = B$

For OR gate: $Y = A + B$ using MUX with A as select, $I_0 = B$, $I_1 = 1$

For NOT gate: $Y = \overline{A}$ using MUX with A as select, $I_0 = 1$, $I_1 = 0$

Complete Implementation The most efficient implementation requires:

1. One NOT gate to generate $\overline{D}$
2. One 2-input multiplexer with A as select, 0 as I_0, and $\overline{D}$ as I_1 (implements $A\overline{D}$)
3. One 2-input multiplexer with C as select, output of first MUX as I_0, and 1 as I_1 (implements final function)