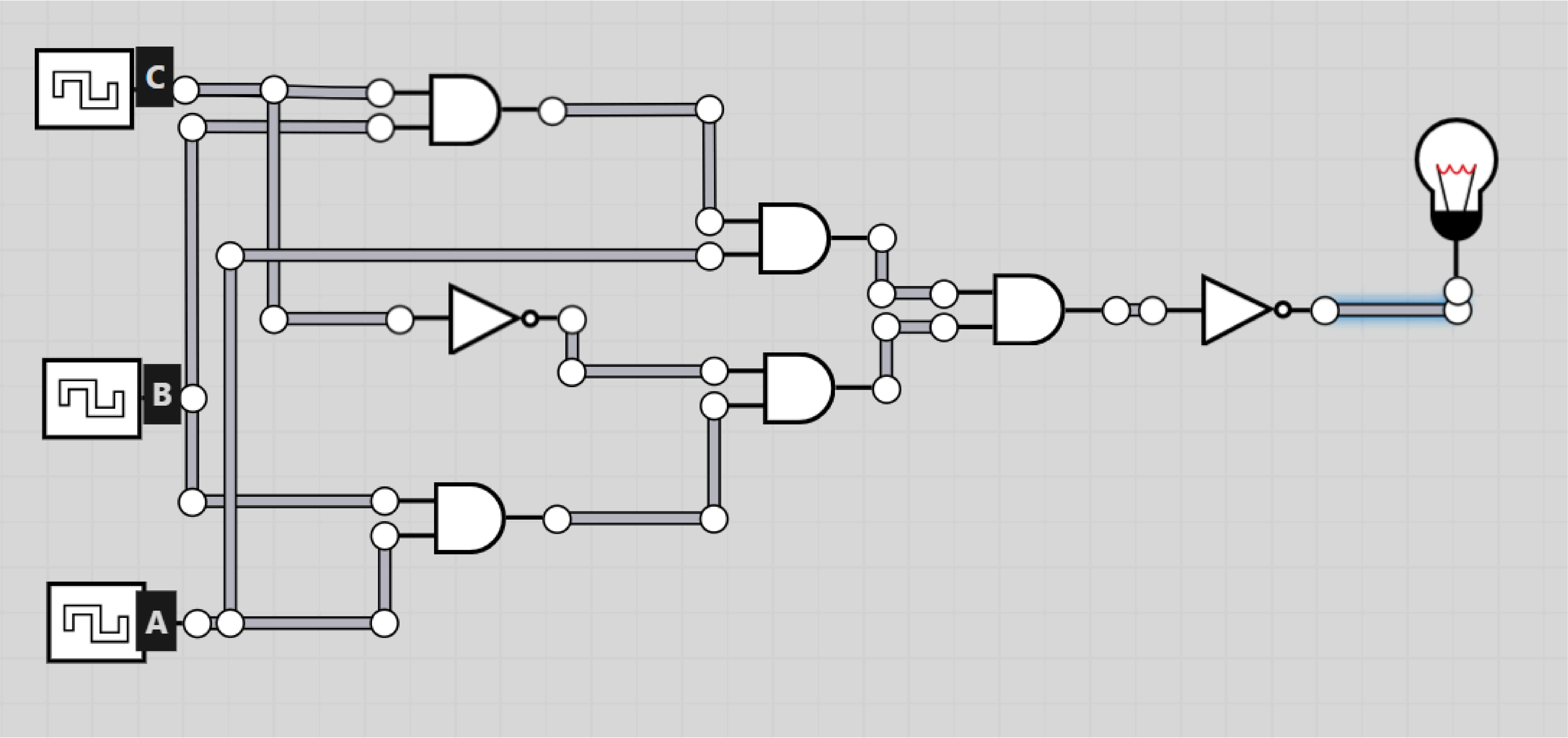
Task (2)

PHM311s - Discrete Mathematics

DIGITAL LOGIC CIRCUIT SIMPLIFICATION

01 CIRCUIT



02 REQUIRED

Write a code in c++ that:

01 Simplify the expression of the digital logic circuit to the simplest form. (this step is done analytically on paper)

02 Calculates the truth table of the logical expression of the logic circuit given.

03 Calculates the truth table simplified logical expression of the simplest form.

02 REQUIRED

04 Gives an output message showing whether the 2 expressions are equivalent or not.

05 Find the value of the inputs which makes each of the 2 logic circuits expression satisfiable.

06 If the circuit is unsatisfiable or tautology, change one gate to make it satisfiable and repeat step 1-4.

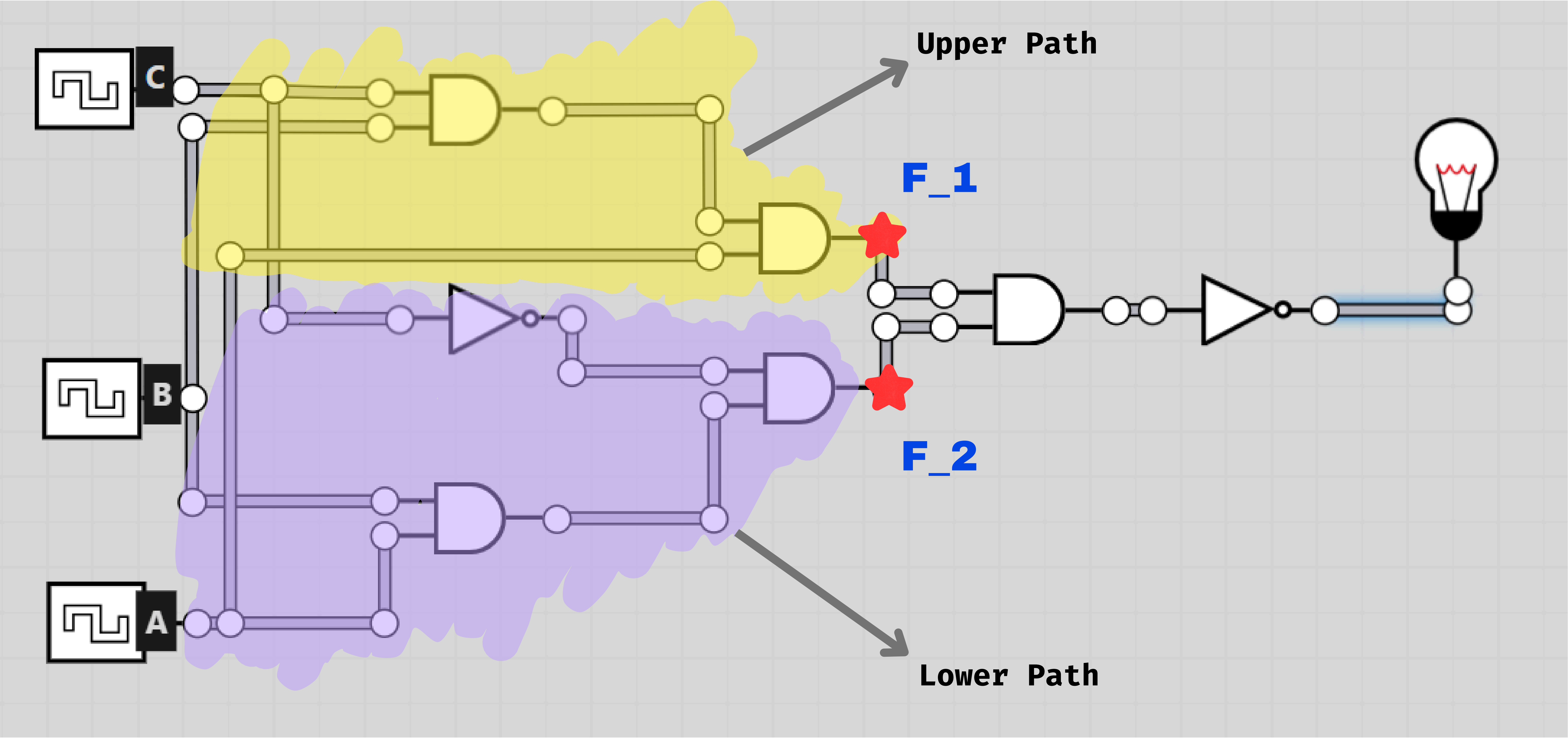
03 GUIDE FOR SOLUTION

01 Build the truth table, equivalent functions for the assigned logic circuit defined in your code.

02 Modify the code by taking the logic circuit from the user by asking questions about the circuit. (for example: How many inputs to the circuit? How many levels are the logic circuit?... etc)

03 Extend the code to be able to handle different gates other than the ones in your assigned argument (the maximum you can handle is limited by the values in your assigned circuit)

04 CIRCUIT ANALYSIS

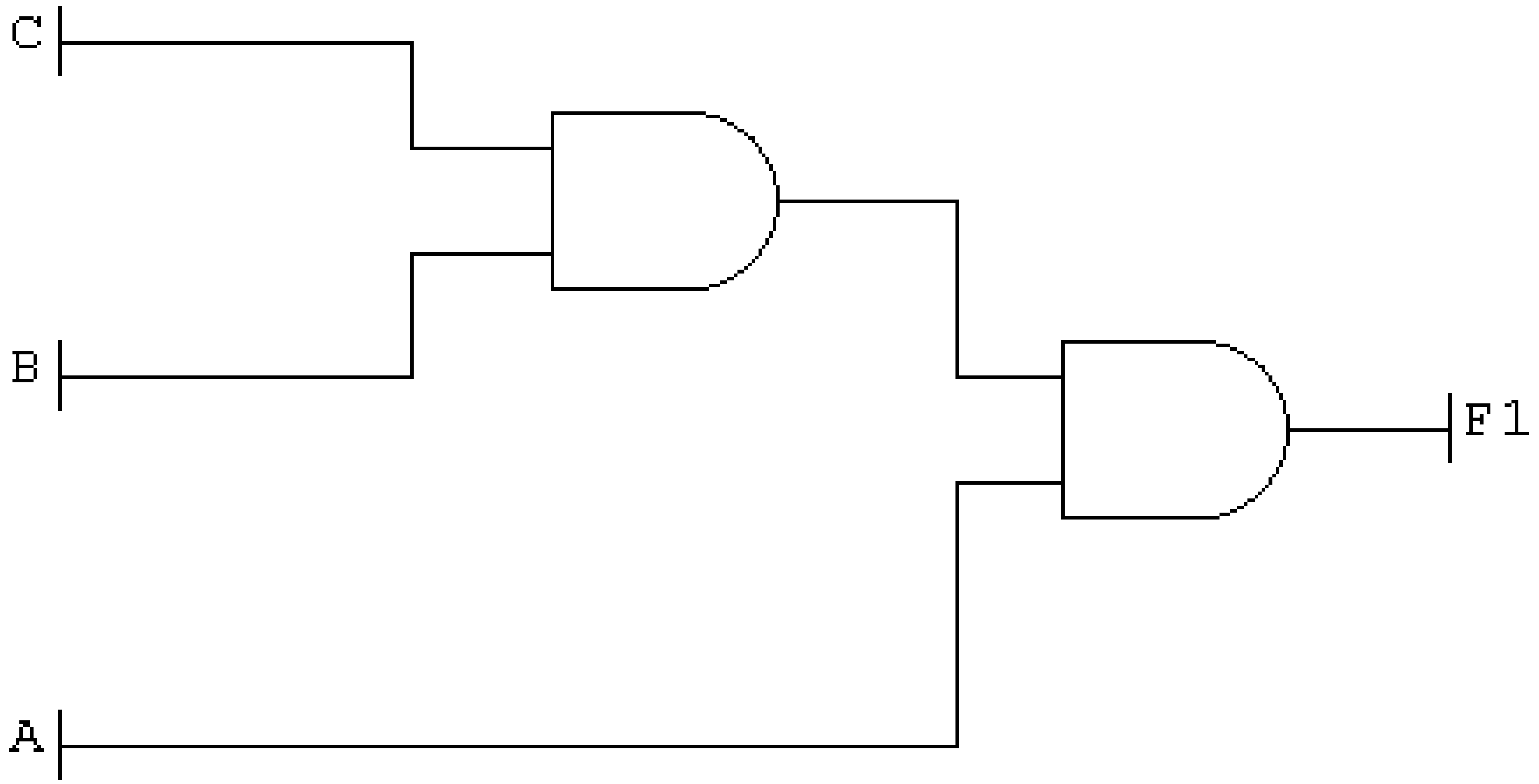


04 CIRCUIT ANALYSIS

1. Upper path:

$$F\_1 = (C \land B) \land A$$

$$= (A \land B) \land C \quad \text{“Associative law”}$$

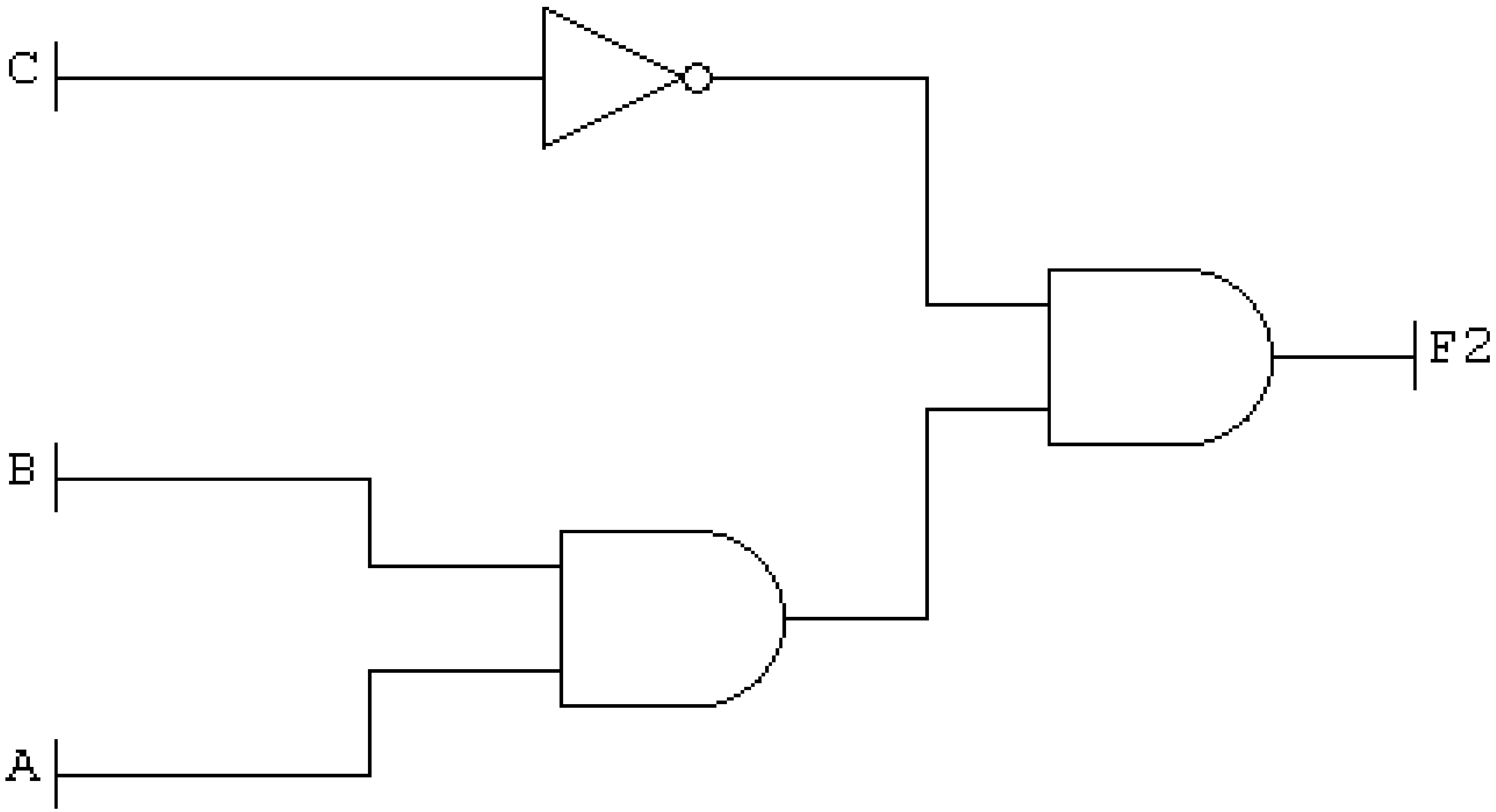


$F\_1 = (A \land B) \land C$

04

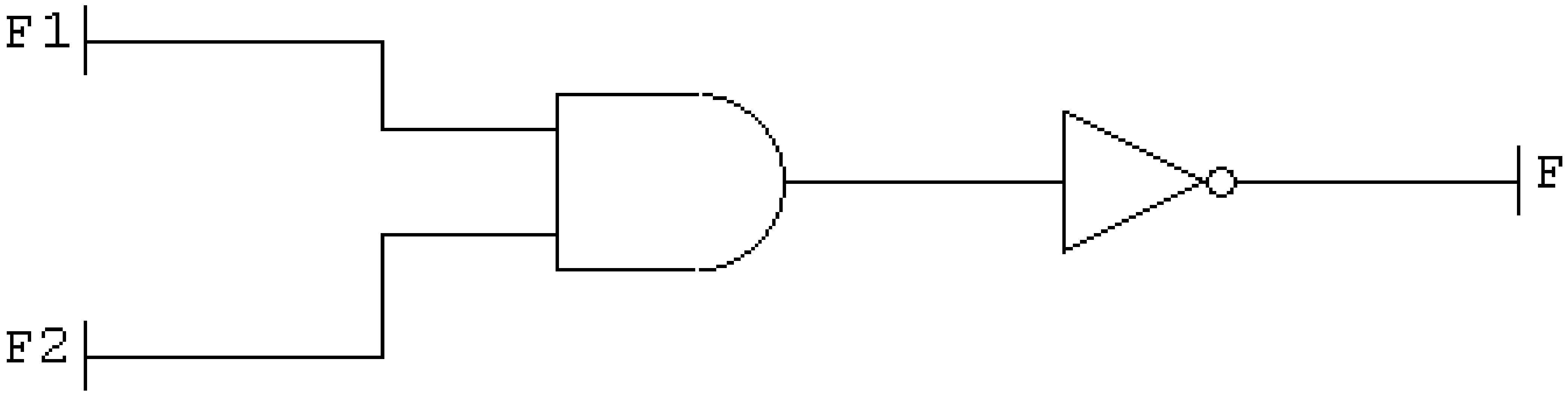
CIRCUIT ANALYSIS

2. Lower path:



F_2 = (A ∧ B) ∧ ~C

04 CIRCUIT ANALYSIS



$F = \sim(F_1 \wedge F_2)$

$= \sim(\ ((A \wedge B) \wedge C) \wedge ((A \wedge B) \wedge \sim C)\ )$

$= \sim(\ (A \wedge B) \wedge C \ \wedge\ (A \wedge B) \wedge \sim C\ )$

$= \sim(((A \wedge B) \wedge (A \wedge B)) \wedge (C \wedge \sim C))$

$\because\ (A \wedge B) \wedge (A \wedge B) = A \wedge B$

$\therefore\ F = \sim((A \wedge B) \wedge (C \wedge \sim C))$

$\because\ C \wedge \sim C = c$ “Contradiction” → Negation low

$\therefore\ F = \sim((A \wedge B) \wedge c)$

$\because\ (A \wedge B) \wedge c = c$ “Contradiction”

→ Universal bound low

$\therefore\ F = \sim c = t$ “Tautology”

→ Negation of t and c

The simplest form = t “Tautology”

05

TRUTH TABLE

$F = \sim(\ ((A \wedge B) \wedge C) \wedge ((A \wedge B) \wedge \sim C)\ )$

A	B	C	~C	F_1 ((A ∧ B) ∧ C)	F__2 (A ∧ B) ∧ ~C)	F_1 ∧ F_2	F ~(F_1 ∧ F_2)
T	T	T	F	T	F	F	T
T	T	F	T	F	T	F	T
T	F	T	F	F	F	F	T
T	F	F	T	F	F	F	T
F	T	T	F	F	F	F	T
F	T	F	T	F	F	F	T
F	F	T	F	F	F	F	T
F	F	F	T	F	F	F	T

$F = t$ "Tautology"

A	B	C	F
T	T	T	T
T	T	F	T
T	F	T	T
T	F	F	T
F	T	T	T
F	T	F	T
F	F	T	T
F	F	F	T

The circuit is a tautology since the output is true for all possible inputs.

06 CODE

01 Circuit Input Methods

•Two modes for entering the circuit:

–**Equation Mode (E)**: User enters Boolean expression directly

–Supports ∧ (AND), ∨ (OR), ~ (NOT)

–**Question Mode (Q)**: Step-by-step circuit builder

–User selects inputs, gates, levels, and operators

06 CODE

02 Truth Table Generation

•Automatically calculates number of rows = 2^n

–Uses binary decrement function to generate combinations

–Displays inputs and final output column

–Stores results in 2D vector structure

06 CODE

03 Logical Analysis

•Checks if expression is:

–Tautology (all outputs = 1)

–Contradiction (all outputs = 0)

–Satisfiable (at least one output = 1)

–Displays satisfying input combinations

04 Circuit Equivalence

•Compares output columns of two truth tables

–If all rows match → Equivalent

–If any row differs → Not Equivalent

–Ensures correctness of simplification

06 CODE

05 Boolean Simplification

- •Extracts minterms from truth table
- –Merges terms differing in one bit
- –Uses '-' to represent Don't Care conditions
- –Implements Quine–McCluskey algorithm idea
- –Generates minimized Boolean expression

06 CODE

06 Quine–McCluskey

–A systematic method for minimizing Boolean functions

–Alternative to Karnaugh Map (suitable for computer implementation)

Main Steps:

1) List all minterms (rows where output = 1)
2) Group terms by number of 1s
3) Merge terms that differ in only one bit
4) Replace differing bit with '-' (Don't Care)
5) Extract Prime Implicants
6) Build minimized Boolean expression

06 CODE

07 Test (1)

F = ~(((A ∧ B) ∧ C) ∧ ((A ∧ B) ∧ ~C))

```
Do you want to enter the circuit as an equation or answer questions? (E/Q): E
Enter Logic Equation (OR : ∨ , AND : ∧): ~( ((A ∧ B) ∧ C) ∧ ((A ∧ B) ∧ ~C) )

Circuit: ~( ((A ∧ B) ∧ C) ∧ ((A ∧ B) ∧ ~C) )
Truth Table:
============================================================
    A       |       B       |       C       |     Output
============================================================
    T       |       T       |       T       |       T
    T       |       T       |       F       |       T
    T       |       F       |       T       |       T
    T       |       F       |       F       |       T
    F       |       T       |       T       |       T
    F       |       T       |       F       |       T
    F       |       F       |       T       |       T
    F       |       F       |       F       |       T
============================================================
```

06 CODE

```
Do you want to check whether your simplified equation is equivalent to the circuit? (0/1)
0 -NO
1 -Yes
0

Circuit after simplification: (1)
Truth Table:
===========================================================
    A       |       B       |       C       |    Output
===========================================================
    T       |       T       |       T       |       T
    T       |       T       |       F       |       T
    T       |       F       |       T       |       T
    T       |       F       |       F       |       T
    F       |       T       |       T       |       T
    F       |       T       |       F       |       T
    F       |       F       |       T       |       T
    F       |       F       |       F       |       T
===========================================================
The two logical expressions are equivalent.
The expression is a Tautology!
Tip: You can try changing one gate (AND/OR/NOT) to see how it affects the output.
```

06 CODE

08 Test (2)

$F = \sim(((A \land B) \land C) \lor ((A \land B) \land \sim C))$

```
Do you want to enter the circuit as an equation or answer questions? (E/Q): E
Enter Logic Equation (OR : ∨ , AND : ∧): ~( ((A ∧ B) ∧ C) ∨ ((A ∧ B) ∧ ~C) )

Circuit: ~( ((A ∧ B) ∧ C) ∨ ((A ∧ B) ∧ ~C) )
Truth Table:
=============================================================
    A       |       B       |       C       |     Output
=============================================================
    T       |       T       |       T       |       F
    T       |       T       |       F       |       F
    T       |       F       |       T       |       T
    T       |       F       |       F       |       T
    F       |       T       |       T       |       T
    F       |       T       |       F       |       T
    F       |       F       |       T       |       T
    F       |       F       |       F       |       T
=============================================================
```

06 CODE

```
Do you want to check whether your simplified equation is equivalent to the circuit? (0/1)
0 -NO
1 -Yes
0

Circuit after simplification: (~B) v (~A)
Truth Table:
============================================================
    A       |       B       |       C       |     Output
============================================================
    T       |       T       |       T       |       F
    T       |       T       |       F       |       F
    T       |       F       |       T       |       T
    T       |       F       |       F       |       T
    F       |       T       |       T       |       T
    F       |       T       |       F       |       T
    F       |       F       |       T       |       T
    F       |       F       |       F       |       T
============================================================
```

06 CODE

```
The two logical expressions are equivalent.
Satisfiable with values:
    A: T    B: F    C: T
    A: T    B: F    C: F
    A: F    B: T    C: T
    A: F    B: T    C: F
    A: F    B: F    C: T
    A: F    B: F    C: F
```

Thank You