1.3. Design A Logic Circuit That Has Three Inputs, A, B, And C, And Whose Output Will Be HIGH Only When

1.3. Design A Logic Circuit That Has Three Inputs, A, B, And C, And Whose Output Will Be HIGH Only When

Introduction to Logic Circuit Design

Designing logic circuits is fundamental in digital electronics, enabling the automation of decision-making processes within electronic devices. A logic circuit with three inputs—A, B, and C—can perform complex logical functions, depending on how the inputs are combined. The goal of this design is to create a circuit whose output is HIGH (logic 1) only under specific conditions based on these inputs. Such circuits have applications in digital systems where certain conditions must be met before an action is triggered, such as in safety systems, control panels, and digital decision-making algorithms.

In this article, we will explore the principles behind designing a three-input logic circuit, understand the logic functions involved, and step-by-step guide to implementing such a circuit. We will also discuss how to analyze its truth table and optimize the design for real-world applications.

Understanding the Requirements

Before diving into circuit design, it is crucial to clearly define the conditions under which the output will be HIGH. For this specific scenario, the circuit outputs HIGH only when certain combinations of inputs are true. The exact logical conditions depend on the problem statement, which typically states:

- The output will be HIGH only when a specific combination or set of input conditions are met.
- For example, the output could be HIGH only when all three inputs are HIGH, or when exactly two inputs are HIGH, or when certain inputs are LOW.

For this article, let's assume the common scenario:
The output should be HIGH only when exactly two of the three inputs A, B, and C are HIGH.
This condition is often required in voting systems, majority logic circuits, or decision-making processes.

Summary of the condition:
- Output = HIGH when any two inputs are HIGH and the third is LOW.
- Output = LOW for all other combinations (all inputs LOW, all HIGH, or only one input HIGH).

This setup ensures the circuit acts as a "majority-two" detector.

Constructing the Truth Table

The truth table is a fundamental step in logic circuit design. It lists all possible input combinations and the corresponding output.

A	B	C	Output (Y)
0	0	0	0
0	0	1	0
0	1	0	0
1	0	0	0
1	1	0	1
1	0	1	1
0	1	1	1
1	1	1	0

From the truth table, the output is HIGH only when:
- A=1, B=1, C=0
- A=1, B=0, C=1
- A=0, B=1, C=1

This confirms the "exactly two inputs are HIGH" condition.

Deriving the Boolean Expression

To implement the circuit, we need a Boolean expression for the output based on the inputs. Using the truth table, we can derive the expression via Sum of Products (SOP) form:

Y = (A AND B AND NOT C) OR (A AND NOT B AND C) OR (NOT A AND B AND C)

Expressed in Boolean algebra:

Y = (A·B·¬C) + (A·¬B·C) + (¬A·B·C)

Where:
- “·” denotes AND
- “+” denotes OR
- “¬” denotes NOT

This expression captures all the input combinations where the output should be HIGH.

Implementing the Logic Circuit

To translate the Boolean expression into a physical circuit, follow these steps:

Components Needed:
- AND gates
- OR gates
- NOT gates (inverters)
- Inputs A, B, C
- Output Y

Step-by-step Construction:

1. Create NOT signals for each input:
- NOT A (¬A)
- NOT B (¬B)
- NOT C (¬C)

2. Generate the AND conditions:
- First AND gate: A AND B AND NOT C
- Second AND gate: A AND NOT B AND C
- Third AND gate: NOT A AND B AND C

3. Combine the AND outputs:
- Connect all three AND gate outputs into a single OR gate.
- The output of this OR gate gives the final output Y.

Circuit Diagram Overview:
- Inputs A, B, C
- Inverters for each input to generate ¬A, ¬B, ¬C
- Three AND gates configured as described
- One OR gate to combine the AND outputs
- Output Y

This configuration ensures that the circuit outputs HIGH only when exactly two inputs are HIGH.

Optimizing the Circuit

Optimization involves reducing the number of gates, minimizing propagation delay, and ensuring the circuit is cost-effective.

Simplification Techniques:
- Use NAND or NOR gates where possible, as they are often more economical.
- Combine logic expressions using Boolean algebra simplification rules.
- For this particular function, the original expression is already minimal.

Simplified Expression:
The original expression is already minimal, but it can also be written using XOR and AND gates:

$Y = (A \cdot B \cdot \neg C) + (A \cdot \neg B \cdot C) + (\neg A \cdot B \cdot C)$

Alternatively, it can be interpreted as:
- Y = (A AND B AND NOT C) OR (A AND NOT B AND C) OR (NOT A AND B AND C)

which is straightforward to implement with the described gates.

Designing the Circuit Using Logic Gate Symbols

Visualizing the circuit with standard logic gate symbols can aid in understanding and constructing the physical or simulated circuit.

Diagram Components:
- NOT gates for each input
- AND gates for each minterm
- A final OR gate to combine the minterms

Sample Layout:
```
A -----|>o-----| |-----
NOT AND ----|
B -----|>o-----| | OR ---- Y
NOT AND ----|
C --------|>o-----| |
AND
```

(Note: The diagram is simplified; actual wiring would need to connect inputs and gates accordingly.)

Testing the Circuit

Once constructed, testing the circuit with all input combinations is essential to confirm it works as intended.

A	B	C	Expected Y	Actual Y

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

Proper testing verifies the circuit's functionality and ensures it meets the specified conditions.

Applications of Such a Logic Circuit

This "exactly two inputs HIGH" logic circuit has practical applications in various fields:

- Majority Logic Circuits: Used in voting systems within digital processors to determine the majority vote.
- Error Detection: Detects specific error conditions where exactly two signals are active.
- Control Systems: Triggers actions only when specific combinations of sensors are activated.
- Security Systems: Activates alarms when two or more security sensors are triggered simultaneously.

Conclusion

Designing a logic circuit with three inputs where the output is HIGH only under specific conditions involves understanding Boolean algebra, constructing truth tables, deriving Boolean expressions, and implementing these expressions using logic gates. The approach demonstrated here—detecting when exactly two inputs are HIGH—is a common and practical application in digital electronics. By following systematic steps, engineers can create efficient and reliable circuits tailored to their specific needs, whether for simple decision-making or complex control systems.

Understanding these fundamental principles enhances your ability to design versatile digital circuits and lays the groundwork for more advanced digital system development.

Frequently Asked Questions

What is the basic requirement for the output to be HIGH in a logic circuit with inputs A, B, and C?

The output will be HIGH only when a specific logical condition, involving the states of A, B, and C, is met as defined by the circuit's design.

How do you determine the logic expression for a circuit that outputs HIGH only under certain input conditions?

You analyze the required input combinations and use logic gates (AND, OR, NOT) to create a Boolean expression that is true only for those conditions.

What common logic gate configuration can be used to ensure the output is HIGH only when all inputs A, B, and C are HIGH?

A 3-input AND gate can be used, which outputs HIGH only when A, B, and C are all HIGH.

Can you design a circuit where the output is HIGH only when exactly two inputs are HIGH?

Yes, by combining AND, OR, and NOT gates to check for cases where exactly two inputs are HIGH and the third is LOW.

How would you implement a circuit where the output is HIGH only when A and B are HIGH, regardless of C?

Use an AND gate with inputs A and B, and connect C through a NOT gate if needed, depending on the specific condition, or simply connect A and B directly to an AND gate.

What is the significance of truth tables in designing such logic circuits?

Truth tables help visualize all input combinations and their corresponding outputs, guiding the design of the correct logical expression.

How can Karnaugh maps assist in simplifying the logic circuit for this problem?

Karnaugh maps help identify minimal sum-of-products or product-of-sums expressions, leading to simpler circuit designs with fewer gates.

What are common real-world applications of logic circuits that output HIGH only under specific input conditions?

They are used in control systems, security access controls, alarm systems, and digital decision-making circuits where specific input conditions trigger an action.

How does the choice of logic gates affect the complexity and efficiency of the circuit?

Selecting the appropriate gates and minimal expressions reduces the number of components, power consumption, and improves circuit speed and reliability.

Additional Resources

Designing a Logic Circuit That Has Three Inputs, A, B, And C, And Whose Output Will Be HIGH Only When

Designing digital logic circuits requires a thorough understanding of how various logic gates interact to produce desired outputs based on multiple inputs. In particular, creating a circuit with three inputs—A, B, and C—that outputs a HIGH signal only under specific conditions is a fundamental task in digital electronics, underpinning complex decision-making processes in computing systems. This article provides a comprehensive overview of designing such a logic circuit, exploring the theoretical foundation, step-by-step design procedures, possible configurations, and practical considerations.

Understanding the Basic Concept

Before diving into the specifics of circuit design, it’s essential to clarify the problem statement and the underlying logic concepts. The goal is to develop a circuit with three binary inputs—A, B, and C—such that the output is HIGH (logic 1) only when certain conditions are met. For instance, the output might be HIGH only when:

- All three inputs are HIGH (A=1, B=1, C=1),
- Exactly two inputs are HIGH,
- Or some other specific combination of inputs.

The precise condition dictates the type of logic gates used and the overall configuration.

Key Definitions:
- Logic HIGH (1): Represents a true or ON state.
- Logic LOW (0): Represents a false or OFF state.
- Binary Inputs: Inputs that can be either 0 or 1.

Understanding the desired output condition is fundamental because it influences the selection of logic functions (AND, OR, NOT, XOR, NAND, NOR, XNOR) used in the circuit.

Step-by-Step Approach to Designing the Circuit

Designing a logic circuit with three inputs involves several methodical steps:

1. Define the Truth Table
The first step is to explicitly define the truth table that maps all input combinations to the desired output. For three inputs, there are 2^3 = 8 possible combinations:

A	B	C	Output
0	0	0	0
0	0	1	?
0	1	0	?
0	1	1	?
1	0	0	?
1	0	1	?
1	1	0	?
1	1	1	?

Suppose the desired output is HIGH only when at least two inputs are HIGH. The corresponding output column would be:

A	B	C	Output
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

2. Derive the Boolean Expression
Using the truth table, derive the Boolean expression for the output. For the above case, the output is HIGH when at least two inputs are HIGH:

- When (A=1, B=1, C=0)
- When (A=1, B=0, C=1)
- When (A=0, B=1, C=1)
- When (A=1, B=1, C=1)

Expressed as a sum of minterms:

Y = (A AND B AND NOT C) OR (A AND NOT B AND C) OR (NOT A AND B AND C) OR (A AND B AND C)

Simplify the expression:

Y = (A AND B) OR (B AND C) OR (A AND C)

This simplified Boolean expression is crucial for efficient circuit design.

Designing the Circuit Using Logic Gates

Once the Boolean expression is established, the next step is to implement it using physical logic gates.

1. Basic Logic Gate Implementation
The expression:

Y = (A AND B) OR (B AND C) OR (A AND C)

can be directly mapped onto a combination of AND and OR gates:

- Three AND gates to compute (A AND B), (B AND C), and (A AND C).
- A three-input OR gate to combine these three outputs.

2. Circuit Diagram Overview
- Inputs A, B, and C feed into three AND gates.
- The outputs of these AND gates feed into a three-input OR gate.
- The OR gate produces the final output, HIGH only when any two (or all three) inputs are HIGH.

3. Alternative Implementations
Depending on the available components, alternative implementations are possible:

- Using NAND or NOR gates for cost or power efficiency.
- Employing complex gates like XOR/XNOR if the desired logic can be expressed differently.
- Combining multiple gates to optimize speed or minimize the number of components.

Practical Considerations and Optimization

Designing a logic circuit is not only about correctness but also about efficiency, reliability, and cost-effectiveness.

1. Power Consumption
- Using fewer gates or gates with low power ratings reduces overall power consumption.
- Simplified Boolean expressions help minimize the gate count.

2. Speed and Propagation Delay
- The number of gates affects the circuit's speed.
- Minimizing the number of gate levels reduces delay.

3. Cost and Complexity
- Simplified expressions and gate configurations lower manufacturing and maintenance costs.
- Choosing standard gates ensures compatibility and ease of troubleshooting.

4. Noise Immunity and Signal Integrity
- Proper buffering and shielding are essential in practical applications.
- Logic gate choices impact the circuit's noise immunity.

Features and Pros/Cons of the Design

Features:
- Implements a specific logical condition (e.g., at least two inputs HIGH).
- Scalable to more inputs with similar principles.
- Modular design with basic logic gates.

Pros:
- Clear Boolean expression simplifies implementation.
- Flexible; can adapt to different output conditions by modifying the Boolean expression.
- Easy to troubleshoot due to modular gate-based structure.

Cons:
- May require multiple gates, increasing complexity.
- Propagation delay accumulates with more gate levels.
- Physical limitations in compact designs, especially for high-speed applications.

Extensions and Practical Applications

This basic three-input circuit can serve as a building block in various digital systems:

- Voting circuits: Determining majority in fault-tolerant systems.
- Control systems: Activating outputs only under specific conditions.
- Error detection: Combining multiple signals to detect inconsistencies.
- Security systems: Triggering alarms when multiple sensors are activated.

Moreover, the principles used here extend to more complex logic functions, enabling the design of sophisticated digital devices.

Conclusion

Designing a logic circuit with three inputs A, B, and C, that outputs HIGH only under specific conditions, involves a systematic approach grounded in Boolean algebra and logic gate implementation. Starting from a clear truth table, deriving the Boolean expression, and translating it into a physical circuit ensures accuracy and efficiency. The method outlined—identifying the desired logic condition, simplifying the Boolean expression, and implementing it with standard gates—is fundamental in digital electronics. By understanding these principles, engineers and students can create versatile circuits tailored to a wide range of applications, from simple decision-making to complex computational logic.

Through careful design considerations—balancing speed, power, cost, and complexity—such circuits can be optimized for practical use, forming the backbone of modern digital systems. Whether for educational purposes, prototyping, or final product development, mastering these foundational concepts is essential for advancing in the field of digital electronics.

1 3 Design A Logic Circuit That Has Three Inputs A B And C And Whose Output Will Be High Only When

1 3 Design A Logic Circuit That Has Three Inputs A B And C And Whose Output Will Be High Only When

Related Articles

- A 12,000-N Car Is Raised Using A Hydraulic Lift, Which Consists Of A U-tube With Arms Of Unequal Areas,
- 1. Dynamic Allocation Of Depletable Resources. (1) The N-Period, Constant-Cost, No-Substitute Case: The
- You Are Tasked With Improving The Performance Of A Functional Unit. The Computation For The Functional

Back to Home