3. (25 Pts) Design A Circuit That Converts Any 3-bit Number To Its Negative In Two's Complement System

3. (25 Pts) Design A Circuit That Converts Any 3-bit Number To Its Negative In Two's Complement System

Introduction

In digital systems and computer architecture, representing signed numbers efficiently is crucial for performing arithmetic operations such as addition, subtraction, and multiplication. The two's complement system is the most widely used method for representing signed integers in binary form because it simplifies the hardware design for arithmetic operations.

Designing a circuit capable of converting any 3-bit number into its negative in two's complement is a fundamental task in digital electronics. Such a circuit is not only essential for understanding number representations but also serves as a building block for more complex arithmetic logic units (ALUs) in microprocessors. Whether you are a student learning digital design or an engineer developing embedded systems, understanding how to implement this conversion efficiently is vital.

This article explores the detailed process of designing a 3-bit two's complement converter circuit. We will delve into the principles behind two's complement representation, step-by-step logic for the conversion, and the practical implementation of the circuit using basic logic gates. By the end of this guide, you will have a comprehensive understanding of how to design, analyze, and implement such a circuit.

Understanding Two's Complement Representation

Before designing the circuit, it is essential to understand how two's complement encoding works for 3-bit numbers.

What is Two's Complement?

Two's complement is a binary number system that allows for straightforward binary addition and subtraction of signed integers. It encodes both positive and negative numbers in a fixed number of bits.

Representing Numbers in 3 bits:

Decimal	Binary (unsigned)	Binary (2's complement)	Explanation

0	000	000	Zero
1	001	001	Positive one
2	010	010	Positive two
3	011	011	Positive three
-1	111	111	Negative one (two's complement)
-2	110	110	Negative two
-3	101	101	Negative three

How to Find the Two's Complement:

To find the negative of a number:

1. Take the binary representation of the number.
2. Invert all bits (1's complement).
3. Add 1 to the inverted bits.

For example, to find -3 in 3-bit:

- 3 in binary: 011
- Invert bits: 100
- Add 1: 100 + 001 = 101

Thus, 101 represents -3 in 3-bit two's complement.

Objectives of the Circuit Design

The primary goal is to design a digital circuit that:

- Accepts a 3-bit binary number as input.
- Outputs the 3-bit binary equivalent of its negative in two's complement.

Key Steps in the Design Process

1. Input Representation: 3-bit input, labeled as A2, A1, A0 (most significant bit to least significant).
2. Conversion Logic: Implement the two's complement conversion, which involves:
- Bitwise inversion (complement)
- Addition of 1
3. Output Representation: 3-bit output, labeled as Y2, Y1, Y0.

Logical Analysis of Two's Complement Conversion

To convert an input number $A = A_2A_1A_0$ to its negative:

$$
\text{Negative} = \text{Two's complement of } A = \overline{A} + 1

\]

Where:

- \(\overline{A}\) is the bitwise complement of A.
- The '+1' is added to the complemented bits.

Step 1: Bitwise Inversion

The first step is to invert all bits:

\[
\overline{A_2} = \text{NOT } A_2
\]
\[
\overline{A_1} = \text{NOT } A_1
\]
\[
\overline{A_0} = \text{NOT } A_0
\]

Step 2: Add 1 to the Inverted Bits

Adding 1 to the inverted bits involves a binary addition operation, which can be implemented using full adders.

The sum:

\[
Y = \overline{A} + 1
\]

can be broken down as:

\[
Y_0 = \overline{A_0} + 1
\]
\[
Y_1 = \overline{A_1} + \text{carry from previous addition}
\]
\[
Y_2 = \overline{A_2} + \text{carry from previous addition}
\]

This process can be simplified in hardware using XOR and AND gates.

Designing the Circuit

The circuit consists of two main parts:

1. Bitwise inversion of input bits
2. Adding 1 (binary incrementation)

Part 1: Inversion Logic

- Use three NOT gates, each inverting one input bit.
- Outputs: $\overline{A_2}$, $\overline{A_1}$, $\overline{A_0}$.

Part 2: Addition of 1

- Implemented using full adders for each bit.
- The least significant bit (LSB) addition involves adding $\overline{A_0}$ and 1.
- Carry outputs from each adder are fed into the next adder for higher bits.

Full Circuit Diagram Concept

The complete circuit can be visualized as:

- Three NOT gates for inversion.
- Three full adders (or a combination of XOR, AND, OR gates) for addition.
- The initial carry-in for the least significant bit's adder is logic '1' to perform +1 addition.

Step-by-Step Implementation

1. Inversion Stage:

- Inputs: A2, A1, A0
- Outputs: $\overline{A_2}$, $\overline{A_1}$, $\overline{A_0}$

2. Addition Stage:

- For Y_0: Add $\overline{A_0}$ and 1 (carry-in = 1)
- For Y_1: Add $\overline{A_1}$ and the carry from previous addition
- For Y_2: Add $\overline{A_2}$ and the carry from previous addition

3. Output:

- Y_2, Y_1, Y_0 representing the negative number in two's complement.

Practical Implementation Using Logic Gates

Components Needed:

- 3 NOT gates
- 3 XOR gates (for sum calculation)
- 3 AND gates (for carry calculation)
- 1 OR gate (for carry propagation)
- Additional logic to generate the initial carry-in (which is logic 1)

Implementation Steps:

1. Generate inverted bits:
- $\overline{A_2} = \text{NOT } A_2$
- $\overline{A_1} = \text{NOT } A_1$
- $\overline{A_0} = \text{NOT } A_0$

2. Add 1 to $\overline{A_0}$:
- Sum: $Y_0 = \overline{A_0} \oplus 1$
- Carry out: $C_0 = \overline{A_0} \& 1$

3. Add $\overline{A_1}$ and carry C_0:
- Sum: $Y_1 = \overline{A_1} \oplus C_0$
- Carry out: $C_1 = \overline{A_1} \& C_0$

4. Add $\overline{A_2}$ and carry C_1:
- Sum: $Y_2 = \overline{A_2} \oplus C_1$
- Carry out can be ignored for 3-bit representation, as it indicates overflow.

Simplified Boolean Expressions

The final output bits are:

$$
Y_0 = \overline{A_0} \oplus 1
$$
$$
Y_1 = \overline{A_1} \oplus C_0
$$
$$
Y_2 = \overline{A_2} \oplus C_1
$$

Where:

$$
C_0 = \overline{A_0} \& 1 = \overline{A_0}
$$
$$
C_1 = \overline{A_1} \& C_0
$$

Since adding 1 is equivalent to XOR with 1 for the least significant bit, the circuit simplifies to:

$$
Y_0 = \overline{A_0} \oplus 1 = \text{NOT } \overline{A_0} = A_0
$$

Similarly, the rest can be

Frequently Asked Questions

What is the main objective of designing a circuit that converts any 3-bit number to its negative in the two's complement system?

The main objective is to create a circuit that takes a 3-bit binary number as input and outputs its negative equivalent in two's complement form, enabling simple and efficient negation of binary numbers in digital systems.

Which digital components are essential for designing a circuit that performs two's complement negation of a 3-bit number?

Essential components include XOR gates for bit inversion, AND gates for handling the carry, and possibly a full adder circuit to add 1 to the inverted bits, implementing the two's complement negation process.

How does the two's complement system represent negative numbers in a 3-bit binary system?

In a 3-bit two's complement system, negative numbers are represented by inverting all bits of the positive number and adding 1, with the most significant bit (MSB) serving as the sign bit (0 for positive, 1 for negative).

What is the step-by-step process to design the circuit that converts a 3-bit number to its negative in two's complement?

The process involves: 1) Invert all three bits of the input number using XOR gates with a logic '1' as the key; 2) Add 1 to the inverted bits using a 3-bit full adder; 3) The output of this addition is the negative of the original number in two's complement form.

What are the advantages of using combinational logic to implement the two's complement negation circuit for a 3-bit number?

Using combinational logic ensures the circuit is fast, simple, and does not require memory elements, providing immediate output based solely on current inputs, which is ideal for real-time digital systems.

Can this circuit be extended to handle larger bit-widths, such as 8-bit or 16-bit numbers?

Yes, the same principles apply; the circuit can be scaled by increasing the number of XOR gates and full adders accordingly, although the complexity and size will increase proportionally with the number of bits.

Additional Resources

Designing a Circuit to Convert Any 3-bit Number to Its Negative in Two’s Complement System

---

Introduction

In digital electronics and computer architecture, the two’s complement system is a widely adopted method for representing signed integers. It simplifies the design of arithmetic circuits by enabling addition and subtraction operations to be performed uniformly, regardless of the sign of the numbers involved. A fundamental task within this realm is designing a circuit that can efficiently convert any given 3-bit number into its negative equivalent in two’s complement form. This process is essential in various applications, including arithmetic logic units (ALUs), sign extension units, and computational processors that handle signed integers.

This detailed review delves into the design principles, logical implementation, and practical considerations for constructing a circuit capable of transforming any 3-bit number into its negative counterpart within the two’s complement system. It covers the theoretical underpinnings, step-by-step logic, hardware realization, and optimization strategies, providing a comprehensive understanding suitable for students, engineers, and enthusiasts.

Understanding the 3-bit Two’s Complement Number System

Representation of 3-bit Signed Numbers

- The 3-bit two’s complement number system can represent integers from -4 to +3.
- The bit pattern structure:
- The most significant bit (MSB) is the sign bit:
- `0` indicates a non-negative number.
- `1` indicates a negative number.
- The remaining bits represent the magnitude, with the negative numbers stored in two’s complement form.

Binary	Decimal	Sign	Explanation
000	0	+	Zero
001	1	+	+1
010	2	+	+2
011	3	+	+3
100	-4	-	Two’s complement of 4
101	-3	-	Two’s complement of 3
110	-2	-	Two’s complement of 2
111	-1	-	Two’s complement of 1

Two’s Complement Conversion Basics

- To find the negative of a number `N`:
1. Take the bitwise complement (invert all bits).
2. Add 1 to the complemented bits.
- Mathematically:

$$
\text{Negative}(N) = \sim N + 1
$$

- For example, to negate `010` (+2):
- Complement: `101`
- Add 1: `101 + 001 = 110` (which is -2 in two’s complement).

Goals of the Circuit Design

- Input: Any 3-bit number $N = N_2 N_1 N_0$.
- Output: The negative of this number $-N$ in two’s complement form.
- Constraints:
- The circuit should work for all 3-bit inputs.
- The operation should be efficient, ideally in a single or minimal number of logic stages.
- The design should be scalable for similar systems, possibly extendable to larger bit-widths.

Fundamental Approach to Negation in Hardware

The core idea is to implement the two’s complement negation operation:

$$
-N = \sim N + 1
$$

This involves two primary steps:

1. Bitwise Complementation: Apply NOT operation to each input bit.
2. Addition of 1: Add binary 1 to the complemented number.

Thus, the circuit essentially combines a bitwise NOT operation with a binary adder.

Detailed Circuit Design

Step 1: Implementing the Bitwise Complement

- For each input bit N_i, generate its complement $\sim N_i$.
- This is straightforward using NOT gates:
- $\sim N_2$
- $\sim N_1$
- $\sim N_0$

Step 2: Adding 1 to the Complemented Number

- The addition of 1 can be performed using a 1-bit full adder.
- Since the complement is a 3-bit number, the addition involves a 3-bit adder:
- Inputs: $\sim N_2, \sim N_1, \sim N_0$
- Carry-in: 1 (to add the +1 in two’s complement negation)

Hardware Implementation

The overall hardware structure involves:

- Three NOT gates for complementing each bit.
- A 3-bit binary adder (comprising three full adders) to add the carry-in (logic high, 1) to the complemented bits.

Block diagram components:

1. Input lines: N_2, N_1, N_0
2. Complement block: NOT gates producing $\sim N_2, \sim N_1, \sim N_0$
3. Adder block: Three full adders connected in cascade:
- First adder: inputs $\sim N_0$ and carry-in = 1.
- Subsequent adders: sum outputs from previous stage and complemented bits.

4. Outputs: $-N_2, -N_1, -N_0$ (the negative number)

Logic Equations for Each Bit

Let’s derive the logic equations for each output bit:

- Input bits: N_2, N_1, N_0
- Complemented bits: $C_2 = \sim N_2$, $C_1 = \sim N_1$, $C_0 = \sim N_0$
- Carry-in: $C_{in} = 1$

Outputs:

$$
\begin{aligned}
\text{Sum}_0 &= C_0 \oplus C_{in} \\
C_1 &= C_0 \land C_{in} \quad \text{(carry from bit 0)} \\
\text{Sum}_1 &= C_1 \oplus C_1 \\
C_2 &= C_1 \land C_1 \quad \text{(carry from bit 1)} \\
\text{Sum}_2 &= C_2 \oplus N_2 \\
\end{aligned}
$$

But more precisely, the full adder equations for each bit are:

$$
\text{Sum}_i = A_i \oplus B_i \oplus C_{i}
$$
$$
C_{i+1} = (A_i \land B_i) \lor (C_i \land (A_i \oplus B_i))
$$

Where:

- A_i is the complemented input bit.
- B_i is 0 for bits 1 and 2, since only the least significant bit has the added 1.
- C_i is the carry-in for bit i.

For our specific case:

- $A_0 = \sim N_0$,
- $A_1 = \sim N_1$,
- $A_2 = \sim N_2$,
- $B_0 = 1$,
- $B_1 = 0$,
- $B_2 = 0$,
- $C_{in} = 1$.

Calculations:

- Bit 0:

$$
\text{Sum}_0 = \sim N_0 \oplus 1 = \sim N_0 \oplus 1
$$

$$
C_1 = (\sim N_0 \land 1) = \sim N_0
$$

- Bit 1:

$$
\text{Sum}_1 = \sim N_1 \oplus 0 \oplus C_1 = \sim N_1 \oplus C_1
$$

$$
C_2 = (\sim N_1 \land 0) \lor (C_1 \land (\sim N_1 \oplus 0)) = C_1 \land \sim N_1
$$

- Bit 2:

$$
\text{Sum}_2 = \sim N_2 \oplus 0 \oplus C_2 = \sim N_2 \oplus C_2
$$

Final Implementation: Step-by-Step Summary

1. Complement Input Bits:
- Use NOT gates to generate $\sim N_2, \sim N_1, \sim N_0$.

2. Add 1 Using Full Adders:
- Use the complemented bits as inputs to a 3-bit ripple-carry adder.
- The least significant bit adder receives the complemented bit and carry-in = 1.
- Carry propagates through subsequent adders, finally producing the negative number in two’s complement.

3. Outputs:
- The sum bits from the adder give $-N$.

Practical Considerations and Optimization

- Hardware Efficiency:

- Using shared logic, such as combining

3 25 Pts Design A Circuit That Converts Any 3 Bit Number To Its Negative In Two S Complement System

3 25 Pts Design A Circuit That Converts Any 3 Bit Number To Its Negative In Two S Complement System

Related Articles

- A Textbook Store Sold A Combined Total Of 368 Chemistry And History Textbooks In A Week. The Number Of
- A Marketing Firm Is Considering Making Up To Three New Hires. Given Its Specific Needs, The Firm Feels
- A Company Makes Two Products, X1 And X2. They Require At Least 20 Of Each To Be Produced. Which Of The

Back to Home