Convert (109.875) 10 Into The Following Number Systems Using 8 Integer Bits And 4 Fraction Bits. A) Fixed

Convert (109.875) 10 Into The Following Number Systems Using 8 Integer Bits And 4 Fraction Bits. A) Fixed

Converting decimal numbers into different number systems is a fundamental task in digital electronics, computer architecture, and embedded systems. In this article, we will explore how to convert the decimal number 109.875 into various number representations using a fixed-point format with 8 bits allocated for the integer part and 4 bits for the fractional part. This specific configuration, often denoted as Q8.4, provides a balanced approach to represent fractional and integer values within a limited bit-width.

Understanding Fixed-Point Number Representation (Q8.4)

Before diving into the conversion process, it is essential to understand what fixed-point representation entails, especially the Q8.4 format.

What is Fixed-Point Representation?

Fixed-point representation is a method of encoding real numbers in binary form where the position of the radix point (decimal point in decimal numbers) is fixed. Unlike floating-point, which allows the decimal point to "float" to accommodate a wide range of values, fixed-point allocates a fixed number of bits to the integer and fractional parts, leading to simpler hardware and predictable precision.

Q8.4 Format Explained

- Total Bits: 12 bits (8 for integer, 4 for fractional)
- Integer Bits: 8 bits, allowing representation of values from -128 to 127 (assuming signed representation)
- Fraction Bits: 4 bits, providing a resolution of 1/2^4 = 1/16 = 0.0625

The value of a fixed-point number in Q8.4 format can be calculated as:

`Value = (Integer Part) + (Fractional Part) / 16`

Alternatively, the stored binary value represents:

`Stored Value = (Number in decimal) 2^Fraction Bits`

Given that, the range of representable numbers is from -128 to 127.9375, with a precision of 0.0625.

Step-by-Step Conversion of 109.875 to Fixed-Point (Q8.4)

Let's now proceed with the conversion process.

1. Express the Number in Decimal

The number we are converting is 109.875 in decimal.

2. Multiply by 2^Fraction Bits (16)

To find the fixed-point binary representation, multiply the decimal number by 16:

`109.875 × 16 = 1758`

This gives us an integer value that will be represented in binary form.

3. Convert the Result to Binary

Now, convert 1758 to binary:

- Divide 1758 repeatedly by 2:

Division Step	Quotient	Remainder	Binary Digit (from LSB to MSB)
1758 / 2	879	0	Least Significant Bit (LSB)
879 / 2	439	1	
439 / 2	219	1	
219 / 2	109	1	
109 / 2	54	1	
54 / 2	27	0	
27 / 2	13	1	
13 / 2	6	1	
6 / 2	3	0	
3 / 2	1	1	
1 / 2	0	1	Most Significant Bit (MSB)

Reading from MSB to LSB, the binary representation of 1758 is:

`11011010110`

This is an 11-bit binary number.

4. Pad the Binary Number to Fit 12 Bits

Since the total storage is 12 bits, pad with leading zeros if necessary:

`0011011010110`

But note, our total bits should be 12, so:

`00011011010110` (this is 14 bits, so we need to ensure the correct size)

Actually, the number is 1758 in decimal, which in binary is 11011010110 (11 bits). To store in 12 bits, pad with one leading zero:

`011011010110`

Now, this is the 12-bit binary representation.

5. Assigning the Bits to Fixed-Point Format

In Q8.4 format:

- Bits 11-4: Integer part
- Bits 3-0: Fractional part

Breakdown of the 12 bits:

Bit Position	11	10	9	8	7	6	5	4	3	2	1	0
Binary Digit	0	1	1	0	1	1	0	1	0	1	1	0

- Integer bits (Bits 11-4): 0 1 1 0 1 1 0 1 (from bit 11 to 4)
- Fraction bits (Bits 3-0): 0 1 1 0

Interpretation:

- Integer part: `01101101` (binary) = 0×128 + 1×64 + 1×32 + 0×16 + 1×8 + 1×4 + 0×2 + 1×1 = 64 + 32 + 8 + 4 + 1 = 109
- Fractional part: `0110` (binary) = 0×8 + 1×4 + 1×2 + 0×1 = 4 + 2 = 6

Since fractional bits are 4 bits, their decimal value is 6, representing:

`6 / 16 = 0.375`

Adding this to the integer part:

`109 + 0.375 = 109.375`

However, note that our original decimal number was 109.875, so the binary fractional component should be 0.875.

Let's verify:

- The fractional binary `0110` equals 6 in decimal.
- 6 / 16 = 0.375, which is less than 0.875.

This indicates an inconsistency, implying we need to refine the process.

Alternative approach:

Since the multiplication yielded 1758, and in binary, 1758 is:

`11011010110` (11 bits)

Pad to 12 bits:

`011011010110`

The fractional component is obtained by the last 4 bits: `0110`

Convert `0110` to decimal:

`0×8 + 1×4 + 1×2 + 0×1 = 6`

Fractional value:

`6 / 16 = 0.375`

But since the original number is 109.875, the fractional part in binary should correspond to 0.875, which equals 14/16 (since 14×1/16 = 0.875).

Thus, the fractional bits should be `1110` (binary for 14), not `0110`.

This discrepancy suggests that in the initial step, multiplying 109.875 by 16 gives 1758, which is correct, but the binary representation should correspond to 1758, with the fractional bits representing the fractional part.

Let's revisit the binary conversion:

1758 in decimal:

- 1758 / 2 = 879, remainder 0
- 879 / 2 = 439, remainder 1
- 439 / 2 = 219, remainder 1
- 219 / 2 = 109, remainder 1

- 109 / 2 = 54, remainder 1
- 54 / 2 = 27, remainder 0
- 27 / 2 = 13, remainder 1
- 13 / 2 = 6, remainder 1
- 6 / 2 = 3, remainder 0
- 3 / 2 = 1, remainder 1
- 1 / 2 = 0, remainder 1

Reading remainders from last to first gives:

`11011010110`

which is 11 bits.

To fill 12 bits, add a leading zero:

`011011010110`

Now, the fractional bits are the last 4 bits:

`0110` = 6 / 16 = 0.375

But the original fractional part is 0.875, which corresponds to 14 / 16 -> binary `1110`.

Therefore, to accurately represent 109.875, the scaled integer should be:

`109.875 × 16 = 1758`

and the fractional part is 0.875, which is 14/16.

Rearranged, the fractional bits should be `1110`.

So, the

Frequently Asked Questions

How do you convert the decimal number 109.875 into fixed-point format using 8 integer bits and 4 fractional bits?

First, multiply the decimal number by 2^4 (16): 109.875 × 16 = 1758.0. Then, represent 1758 in binary as 11011001110. Finally, fit this into 12 bits with 8 bits for the integer part and 4 bits for the fractional part, resulting in the fixed-point binary: 011011001110.

What is the fixed-point representation of 109.875 in binary with 8 integer bits and 4 fractional bits?

The fixed-point binary representation is 011011001110, where the first 8 bits (01101100) represent the integer part, and the last 4 bits (1110) represent the fractional part.

What is the decimal value of the fixed-point number 011011001110 with 8 integer bits and 4 fractional bits?

Converting 011011001110 back to decimal: integer part (01101100) = 108, fractional part (1110) = 14/16 = 0.875. Therefore, the total value is 108 + 0.875 = 109.875.

Are there any limitations or considerations when converting 109.875 to fixed-point with 8 integer bits and 4 fractional bits?

Yes, since 8 integer bits can represent values from -128 to 127 in signed fixed-point format, and 109.875 falls within this range, the conversion is valid. However, if the number exceeded this range, overflow would occur, leading to incorrect representation.

How does the choice of 8 integer bits and 4 fractional bits affect the precision and range of fixed-point representation for 109.875?

Using 8 integer bits allows representing numbers from -128 to 127, providing sufficient range for 109.875. The 4 fractional bits give a resolution of 1/16 (0.0625), allowing precise representation up to 0.875 fractional precision. This setup balances range and precision for values like 109.875.

Additional Resources

Convert $(109.875)_{10}$ into Fixed-Point Representation Using 8 Integer Bits and 4 Fraction Bits

---

Introduction

In digital systems, fixed-point representation is a common method for encoding real numbers, especially in embedded systems, digital signal processing, and hardware design where resource constraints make floating-point operations costly or impractical. Ensuring accurate and efficient

conversions from decimal to fixed-point formats requires a clear understanding of number systems, bit allocation, and the underlying mechanics of fixed-point encoding.

This detailed review focuses on converting the decimal number 109.875 into fixed-point notation using 8 integer bits and 4 fractional bits. We will analyze the process step-by-step, covering the theoretical foundations, practical conversion methods, potential pitfalls, and implications of the resulting representation.

Understanding Fixed-Point Representation

What is Fixed-Point Representation?

Fixed-point representation encodes real numbers as integers scaled by a fixed factor determined by the number of fractional bits. The general form:

$$
\text{Number} = \text{Integer Value} \times 2^{-\text{Number of Fraction Bits}}
$$

The total number of bits (here, 12 bits in total) is split into:

- Integer bits: Used for the whole part of the number, including sign bits if signed.
- Fractional bits: Used for the fractional part, providing the precision.

In our case:

- Total bits: 12 (8 integer + 4 fractional)
- Bit allocation:
- Integer bits: 8
- Fraction bits: 4

Assuming signed representation, the range of representable values depends on whether the encoding is signed or unsigned. Typically, fixed-point representations for real numbers are signed, allowing both positive and negative values.

Step 1: Determining the Range and Sign

Signed vs. Unsigned Fixed-Point

- Signed fixed-point: uses two's complement notation, allowing negative and positive values.
- Unsigned fixed-point: only positive numbers; not suitable here since

109.875 exceeds unsigned 8-bit maximum (255), but in the fixed-point context, the total bits are 12, so the maximum unsigned value is $2^{12} - 1 = 4095$.

Given that 109.875 is positive, and the total number of bits is 12, the signed fixed-point representation is more flexible, especially if negative numbers are considered later.

Step 2: Calculating the Scaling Factor

The number of fractional bits (4) means that:

$$
\text{Scaling factor} = 2^{4} = 16
$$

This scaling factor determines how the real number maps to an integer:

$$
\text{Integer representation} = \text{Real number} \times 16
$$

Step 3: Converting the Number to an Integer

Multiply the decimal number by the scaling factor:

$$
109.875 \times 16 = 109.875 \times 16
$$

Calculating:

$$
109.875 \times 16 = (109 \times 16) + (0.875 \times 16) = 1744 + 14 = 1758
$$

(Precisely, $0.875 \times 16 = 14$, since $0.875 = 7/8$, and $7/8 \times 16 = 14$.)

Thus, the fixed-point integer value:

$$
\boxed{1758}
$$

This is the integer value that, when scaled back by dividing by 16, yields the original number.

Step 4: Representing the Number in Binary

Next, convert 1758 into binary using 12 bits (since total bits are 8 integer + 4 fractional bits).

Binary Conversion of 1758

Divide 1758 repeatedly by 2:

Division Step	Quotient	Remainder (bit)
1758 / 2	879	0
879 / 2	439	1
439 / 2	219	1
219 / 2	109	1
109 / 2	54	1
54 / 2	27	0
27 / 2	13	1
13 / 2	6	1
6 / 2	3	0
3 / 2	1	1
1 / 2	0	1

Reading remainders from bottom to top:

Binary: 11011011110

Total bits: 11 bits.

Since we have 12 bits total, we pad with a leading zero:

$$
\text{Binary (12 bits)}: 0\,11011011110
$$

But note that the binary number is 11 bits; adding a leading zero:

$$
\boxed{011011011110}
$$

Alternatively, write as:

$$
\textbf{0}11011011110
$$

which is 12 bits.

Step 5: Confirming the Fixed-Point Format

In fixed-point representation with 8 integer bits and 4 fractional bits, the binary layout is:

Sign bit	Integer bits (7 bits)	Fraction bits (4 bits)

Since the number is positive, the sign bit is 0.

Breaking down:

$$
\boxed{0\,11011011\,110}
$$

But to match the 12-bit binary:

$$
\text{Binary fixed-point} = 0\,11011011\,1100
$$

Note: Since the decimal value is positive and within the representable range, this binary corresponds directly to the fixed-point number.

Step 6: Verifying the Conversion

To verify:

$$
\text{Decimal} = \text{Binary integer} \times 2^{-4}
$$

Reversing the process:

- Convert binary back to decimal:

$$
\text{Binary} = 011011011110_2
$$

which in decimal is 1758, as calculated earlier.

- Divide by 16:

$$
\frac{1758}{16} = 109.875
$$

\]

matching the original number, confirming the correctness of the conversion.

Additional Considerations

Range and Overflow

- The maximum value for 8 integer bits (assuming signed) is:

\[
2^{7} - 1 = 127
\]

- The minimum value is:

\[
-2^{7} = -128
\]

- Our value, 109.875, comfortably fits within this range.

- The total fixed-point value (1758) is well within the total number of representable values (0 to 4095 for unsigned, or -2048 to 2047 for signed with 12 bits).

Precision and Quantization Error

- The fractional resolution is:

\[
\text{Resolution} = 2^{-4} = 0.0625
\]

- The fixed-point representation approximates the original number, but with quantization error:

\[
\text{Error} = |109.875 - \text{represented value}| \approx 0
\]

since the conversion is exact here.

Summary

- The decimal number 109.875 converts to a fixed-point binary representation with 8 integer bits and 4 fractional bits as:

$$
\boxed{\text{Binary: } 011011011110}
$$

- The internal integer value stored is 1758.

- When interpreted with 4 fractional bits, this corresponds exactly to 109.875.

- This process demonstrates how to perform fixed-point conversions, considering scaling, binary encoding, and range constraints.

Practical Applications and Implications

Hardware Implementation

In hardware design, fixed-point numbers are stored in registers or memory as binary integers. The conversion process involves:

- Scaling the real number by $2^{\text{Fraction Bits}}$
- Rounding to the nearest integer (if necessary)
- Storing the binary representation in hardware registers
- When retrieving the number, dividing by the scale factor

This fixed-point format allows for efficient arithmetic operations (addition, subtraction, multiplication) without floating-point hardware, reducing complexity and power consumption.

Limitations and Challenges

- Quantization errors: Fixed fractional bits limit the precision; small errors can accumulate.
- Range restrictions: Careful bit allocation is necessary to avoid overflow.
- Scaling issues: Proper scaling ensures the number fits within the format and maintains accuracy.

Conclusion

Converting decimal numbers like 109.875 into fixed-point representation with 8 integer bits and 4 fractional bits involves understanding the scaling factor, binary encoding, and range considerations. The process includes multiplying the number by $2^4 = 16$, converting to binary, and storing the integer value. This systematic approach ensures precise encoding suitable for hardware implementation, digital signal processing, and embedded system applications, where resource efficiency and deterministic behavior are paramount.

Through this detailed exploration, we've demonstrated the methodical steps involved in fixed-point conversion, highlighting the

Convert 109 875 10 Into The Following Number Systems Using 8 Integer Bits And 4 Fraction Bits A Fixed

Convert 109 875 10 Into The Following Number Systems Using 8 Integer Bits And 4 Fraction Bits A Fixed

Related Articles

- Assignment 12: Word Frequency Analysis Project Stem PLEASE HELP THE STRUGGLE IS REAL.For This Assignment,
- Fabian Is Organizing Textbooks On His Bookshelf. He Has A Spanish Textbook, A Biology Textbook, A Physics
- Complete The Following Tasks In Order On CorpDC:1. Create And Then Edit The LaptopGPO Group Policy Object

Back to Home