**Fixed Point**

Fixed point and floating point refer to the format used to store and manipulate numeric representations of data.

Floating-point represent and manipulate rational numbers via a minimum of 32 bits in a manner similar to scientific notation, where a number is represented with a mantissa and an exponent (e.g., A x 2B, where 'A' is the mantissa and ‘B’ is the exponent), yielding up to 4,294,967,296 possible bit patterns (232). Floating point numbers offer a vast range and high precision, perfect for scientific calculations.

Fixed-point numbers, however, take a different approach. Here, the decimal point is **fixed** at a specific location, typically at the binary equivalent (think 0s and 1s) of the number. Imaging a regular with pre-defined decimal point. One can measure the distance within the ticks on the ruler, but not beyond it.

This fixed location divides the bits in a fixed-point number into two sections:

* Integer bits: Represent the whole numbers.
* Fractional bits: Represent the decimal part.

The number of bits allocated to each section determines the range and precision of the fixed-point number.

For example, the number 11010.12 represents the value of:

***11010.12***

*= 1 \* 24 + 1 \* 23 + 0 \* 22 + 1 \* 21 + 0\* 20 + 1 \* 2-1*

*= 16 + 8 + 2 + 0.5*

*= 26.5*

In Julia, there is a package called *FIxedPointNumbers.jl* for working with fixed-point numbers. An example of it is:

using FixedPointNumbers

*# Define a 16-bit fixed-point type with 4 fractional bits*

*Fx16\_4 = FixedPointNumbers.N(16, 4)*

*# Convert a floating-point number to fixed-point*

*fixed\_point\_value = convert(Fx16\_4, 3.14159)*

*# Perform fixed-point arithmetic*

*half\_value = fixed\_point\_value / 2*

*println("Fixed-point representation: $(fixed\_point\_value)")*

This code defines a fixed point with 16 integer bits and 4 fractional bits. It then converts a floating point to a fixed-point.

Fixed-point operations have various advantages:

* Efficiency: Fixed-point operations are typically implemented using integer operations, making them faster and requiring less hardware overhead compared to floating-point calculations. This is crucial for resource-constrained embedded systems with limited processing power.
* Deterministic: Fixed-point calculations are predictable as the decimal point location is fixed. This ensures consistent behavior and avoids rounding errors that can occur with floating-point numbers.
* Control: You have complete control over the range and precision of your calculations by defining the number of bits allocated to the integer and fractional parts.

Despite the advantages of, it has its shortcomings as well:

* Limited Range: Fixed-point numbers cannot represent a vast range of values like floating-point numbers.
* Scaling Complexity: Careful scaling is required to prevent overflow or loss of precision.
* Less Intuitive: Working with fixed-point numbers can be less intuitive compared to floating-point, especially when dealing with scaling and conversions.

Fixed-point makes it ideal for various embedded system applications:

* Digital Signal Processing: fixed-point’s efficient implementation of filters and other mathematical operations make it excel at audio and image processing tasks.
* Motor Control: Fixed-point calculation for speed, position help give precise control of motors
* Sensor Data Processing: Sensor readings like temperature, pressure, and acceleration often require high-speed signal conditioning and filtering, perfectly suited for fixed-point arithmetic.
* Internet of Things: Fixed-point’s power efficiency and real-time performance is critical for Internet of Things devices

Aditya Patel – Review:

Pros:

* I think the blog does a good job at introducing the concept of fixed-point numbers.
* The blog gives a clear method of calculating and reading fixed-point numbers

Improvements:

* I think that you could go a little more in-depth with the real-world application of fixed point numbers. Instead of giving more numbers of examples, focus on explaining what fixed-point numbers do in the example.
* Similarly, I think the benefits and drawbacks sections need additional detail and supporting examples to ensure clarity to a non-technical crowd.