12 JULY 2018 BY PHILLIP JOHNSTON

Simple Fixed-Point Conversion in C

Operating on fixed-point numbers is a common embedded systems task. Our microcontrollers may not have floating-point support, our sensors may provide data in fixed-point formats, or we may want to use fixed-point mathematics control a value's range and precision.

There numerous fixed-point mathematics libraries around the internet, such as `fixed_point` or the Compositional Numeric Library for C++. If you are looking for a reliable solution to utilize long-term, spend some time to review these libraries to identify candidates for integration.

However, we don't always have the time required to select a library. Perhaps you just need to convert a fixed-point number for prototyping purposes, or you need to do a quick implementation for Friday's demo.

Below is a quick-and-dirty approach for converting between fixed-point and floating-point numbers. If you need to handle mathematical operations on fixed-point numbers, look for a library to integrate.

Lossy Conversion of Fixed-Point Numbers

First, we need to select our fixed-point type. For this example, we'll be using 16-bit fixed point numbers, in an `11.5` format (11 integral bits, 5 fractional bits):

```
/// Fixed-point Format: 11.5 (16-bit)
typedef uint16_t fixed_point_t;
```

We'll make a quick macro for the number of fractional bits:

```
#define FIXED_POINT_FRACTIONAL_BITS 5
```

Then we'll define two conversion functions:

```
/// Converts 11.5 format -> double
double fixed_to_float(fixed_point_t input);

/// Converts double to 11.5 format
fixed_point_t float_to_fixed(double input);
```

Now that we've gotten the groundwork out of the way, we'll write our fixed-point to floating-point conversion function. Converting from fixed-point to floating-point is straightforward. We take the input value and divide it by ($2^{\text{fractional\_bits}}$), putting the result into a `double`:

```
inline double fixed_to_float(fixed_point_t input)
{
    return ((double)input / (double)(1 << FIXED_POINT_FRACTIONAL_BITS))
}
```

To convert from floating-point to fixed-point, we follow this algorithm:

1. Calculate `x = floating_input * 2^(fractional_bits)`
2. Round `x` to the nearest whole number (e.g. `round(x)`)
3. Store the rounded `x` in an integer container

Using the algorithm above, we would implement our float-to-fixed conversion as follows:

```
inline fixed_point_t float_to_fixed(double input)
{
    return (fixed_point_t)(round(input * (1 << FIXED_POINT_FRACTIONAL_E
}
```

However, not all of our embedded systems utilize the standard library, and perhaps `round()` is not supplied. You can also just rely on truncation when converting to an integer. There will be some precision loss, but for a quick-and-dirty solution that may be acceptable:

```
inline fixed_point_t float_to_fixed(double input)
{
    return (fixed_point_t)(input * (1 << FIXED_POINT_FRACTIONAL_BITS));
}
```

If you need to support multiple fixed-point styles, you can provide interfaces for various integer widths and add the fractional bit count as an input argument:

```
// Convert 16-bit fixed-point to double
double fixed16_to_double(uint16_t input, uint8_t fractional_bits)
{
    return ((double)input / (double)(1 << fractional_bits));
}

// Equivalent of our 11.5 conversion function above
double r = fixed16_to_double(input, 5);
```

There you have it: quick-and-dirty fixed-point conversion methods.

Further Reading

- Wikipedia: Fixed-Point Arithmetic
- LINEAR11 Conversion Using a Sign Extension Bit Twiddling Hack
- C++11 Fixed Point Arithmetic Library
- Compositional Numeric Library for C++

Related Articles

C++11 Fixed Point Arithmetic Library

Demystifying ARM Floating Point Compiler Options

Enforce Correct Integer Arithmetic using the C++ Safe Numerics Library

UNCATEGORIZED

C, FIXED POINT, TIPS AND TRICKS