Edward Gonzalez

MTEC 3140

March 5th ,2024

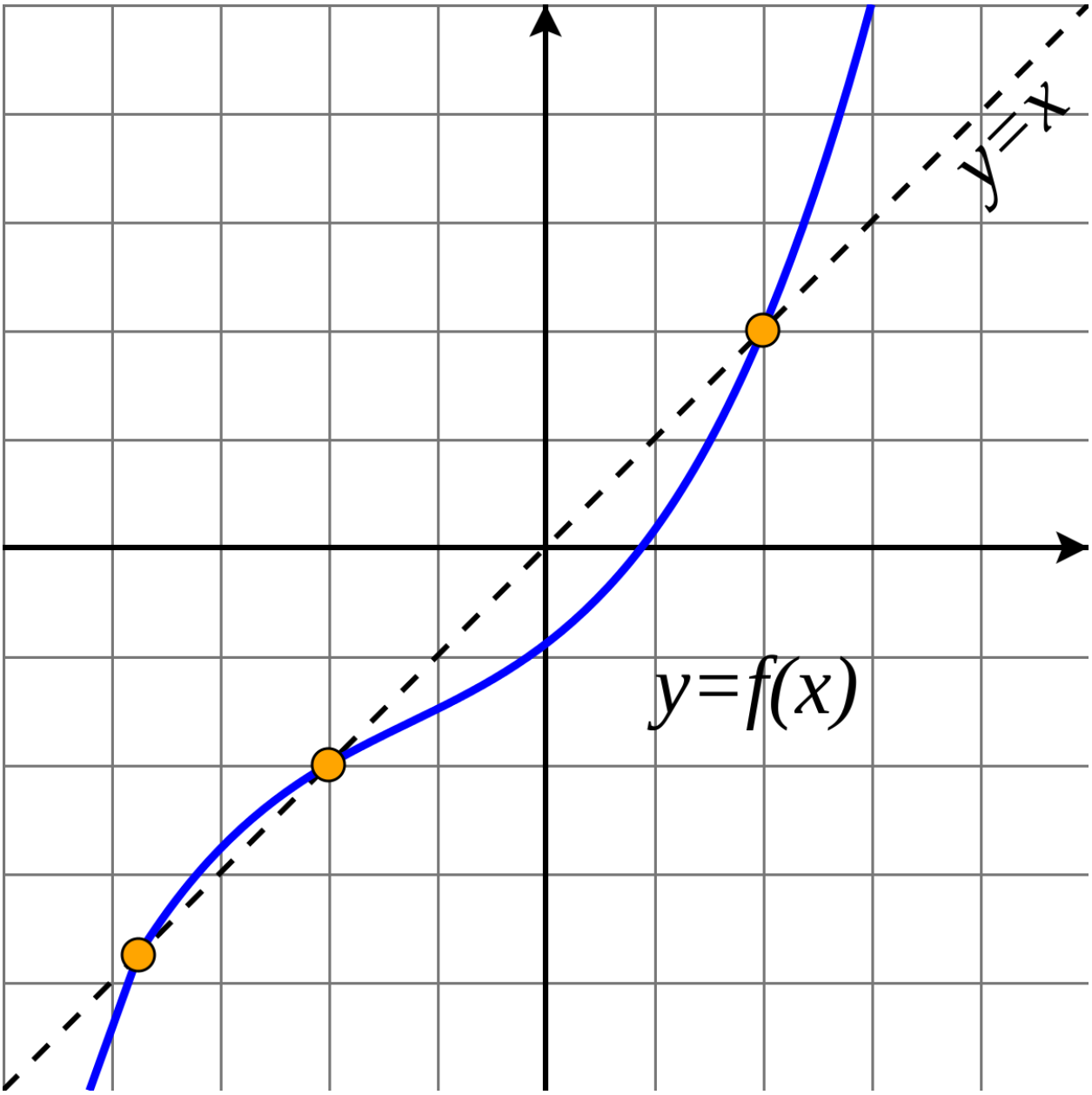
# Fixed Point Math in Mozzi

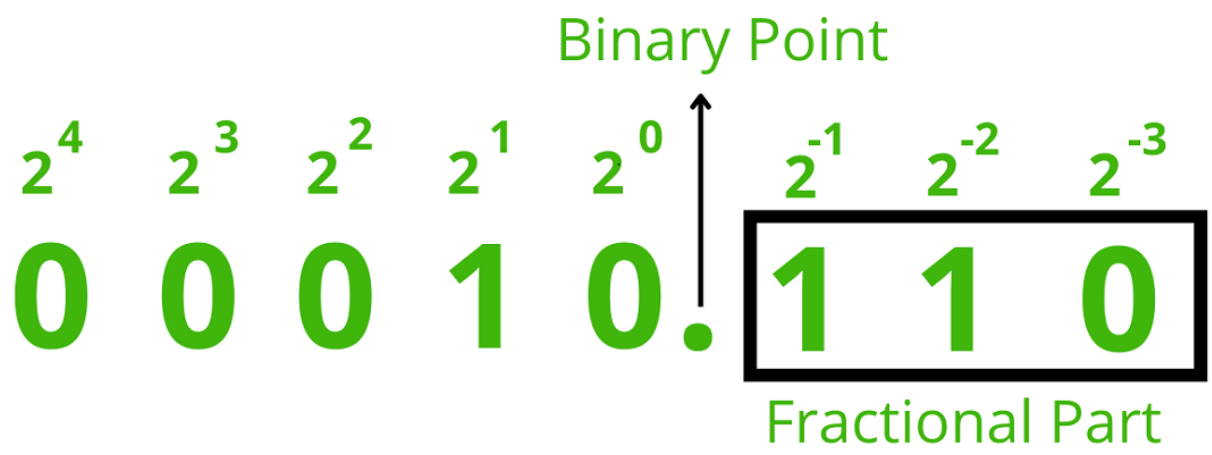
The fixed point is a given number value(s) that is preserved under any change or transformation. This arithmetic term applies to both mathematics and computer science, which functions similarly. The algorithm is computation is good at running numbers fast with less demanding hardware. Additionally, it works best when the range of values and its limitations of bits. So, we'll have to know our biggest and smallest values to get the best accuracy. The algorithm runs quickly with few resources being drawn, but one drawback is scalability isn't good. This is because the arithmetic has poor range leading to inaccurate values from overflow and/or underflow. However, if we scale our number values to be bigger than the errors will decrease. Fixed point is efficient based on size, complexity, and cost of a small device that isn't hardware intensive or demanding difficult computations. Also, they're cost–efficient since it acknowledges the numbers in take a number range as whole numbers. Instead of counting every individual number down to the most finite point we decide them in blocks. The reason for this finite and constant value is because it has a constant number of digits after the decimal point rather than a dynamic range.

In mathematics, fixed point is a value that does not change its formation under transformation because the data are mapped in the function. The mapping is set on X which is one fixed point that can be obtained with a limited amount of successive approximation. A function's fixed points are identical when inversed except negative value (if applied), or another numerical value. For example, if your fixed point is $f(2) = 2$ while on a graph $f(x) = (2,2)$ will have another fixed of $f(x) = (2, -2)$. This is because there is a $y = x$ line that'll intercept with each fixed point of that function. Ultimately, it represents the domain of an element in a function

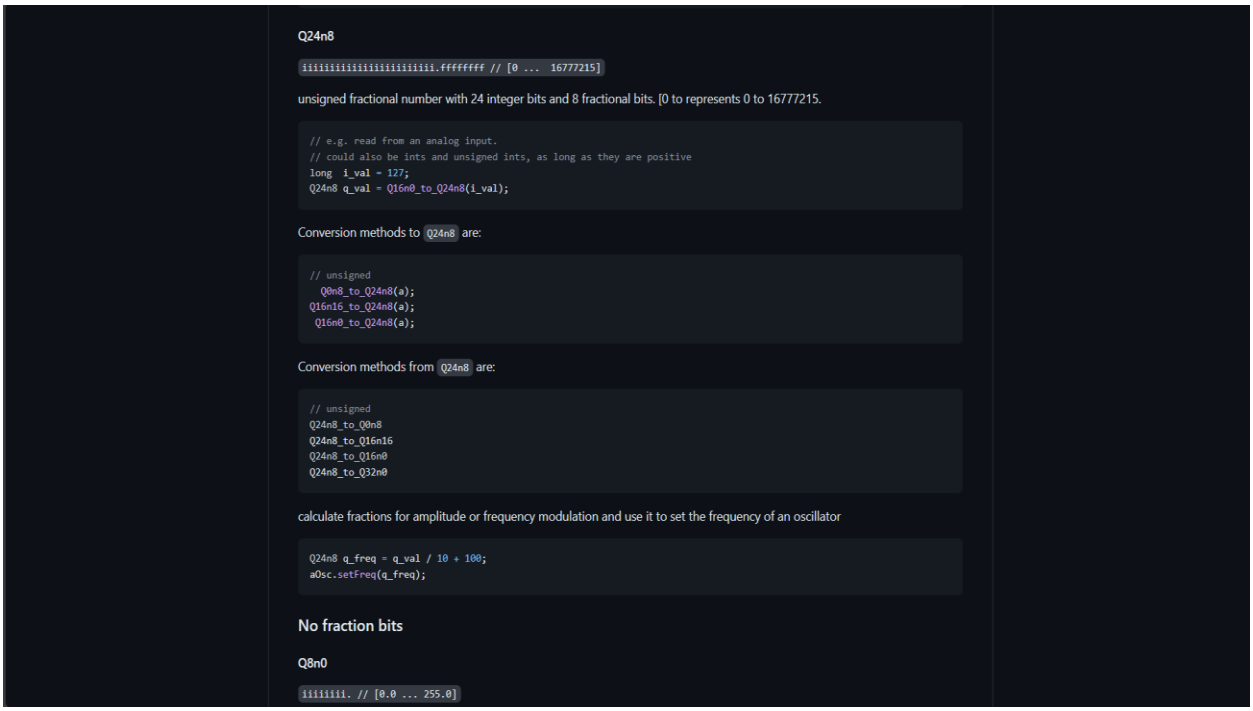
In computer science, fixed points represent the fixed number of integers (before) and fractional (after) of a decimal point. However, it's mainly a consistent position with a fixed number after the decimal point. They're in the binary point position which just alludes to bits either *0* or *1* (data/info) in the decimal. There's a specific number of bits reversed after the decimal point so let's say your maximum number is *9999.9999* then your smallest number will be *0000.0001*. As a result, the algorithm or code will know what numbers are available within the decimal point's range. Also, if we're using a 32 – bit then our integer part will have 16 bits while our fractional part will have 16 bits. Those will be our highest and lowest values to store bits if we're maximizing the Arduino.

In Mozzi, this can be integrated by the following use cases/examples below:


y=x
y=f(x)




From [Fixed-point arithmetics in mozzi · GitHub]



Q24n8

`iiiiiiiiiiiiiiiiiiiiiiii.ffffffff // [0 ... 16777215]`

unsigned fractional number with 24 integer bits and 8 fractional bits. [0 to represents 0 to 16777215.

```
// e.g. read from an analog input.
// could also be ints and unsigned ints, as long as they are positive
long  i_val = 127;
Q24n8 q_val = Q16n0_to_Q24n8(i_val);
```

Conversion methods to `Q24n8` are:

```
// unsigned
  Q0n8_to_Q24n8(a);
Q16n16_to_Q24n8(a);
 Q16n0_to_Q24n8(a);
```

Conversion methods from `Q24n8` are:

```
// unsigned
Q24n8_to_Q0n8
Q24n8_to_Q16n16
Q24n8_to_Q16n0
Q24n8_to_Q32n0
```

calculate fractions for amplitude or frequency modulation and use it to set the frequency of an oscillator

```
Q24n8 q_freq = q_val / 10 + 100;
aOsc.setFreq(q_freq);
```

**No fraction bits**

Q8n0

`iiiiiiii. // [0.0 ... 255.0]`

### Q7n0

`[+|-] . // [-128 ... 127]`

Same as `char`. Signed fractional number with 7 integer bits 0 fractional bits.

```
// unsigned
Q16n0_to_Q15n16

// signed
Q7n0_to_Q7n8
Q7n0_to_Q15n16
```

### Q15n0

`[+|-] fffffffffffffff. // [-2147483648 ... 2147483647]`

Signed number with 15 integer bits and 0 fractional bits.

Conversion methods to

```
Q15n0_to_Q15n16
```

Conversion methods from

```
Q15n16_to_Q15n0
```

### Q31n0

`[+|-] iiiiiiiiiiiiiiiiiiiiiiiiiiiiiii. // [-2147483648 ... 2147483647]`

Signed (normal long int) number with 31 integer bits and 0 fractional bits.

Conversion methods to

```
Q23n8_to_Q31n0
```