**Fast Logarithm Converter for Fixed-point Numbers without Look-up Table**
http://www.winnyefanho.net

**Abstract**
A fast algorithm computes the logarithm value to base 2 of any fixed-point numbers. The input value must be positive while the output value is a signed fixed-point number. Unlike the commonly used methods, look-up table is not required to reduce ROM size. It is suitable for being implemented by hardware in integrated circuits, or, by software in DSP or different kinds of MCU in which the ROM size is limited.

**Background**
Logarithm converter is useful in various applications such as signal processing, digital communcations, control systems, etc. In most of the cases, logarithm converter is achieved by the method described by the block diagram as shown in Fig. 1. The input fixed-point value, x, is firstly divided into mantissa, m, and exponent, e, similar to floating-point representation as:

$$x = m\ 2^e$$

The logarithmic value of x can be expressed in terms of m and e as:

$$y = \log_2 x = e + \log_2 m$$

The exponent is a signed integer and it is exactly the integer part of y. Since the exponent is to base 2, m is a fixed-point value in the range of [1.0, 2.0). Then, its logarithmic value is in the range of [0.0, 1.0), i.e., the fractional part of y. The final logarithmic output is simply the sum of the integer part and the fractional part.

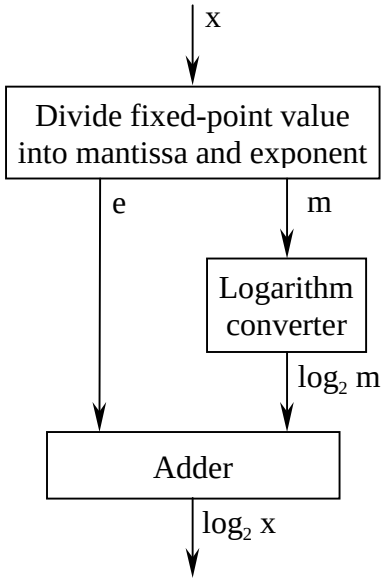

Fig. 1

The main problem is to compute the logarithmic value of m. There are many well-known algorithms to do it. Most of the commonly used methods require look-up tables that occupy so much ROM space. Those methods may not be feasible in the

systems that are lack of ROM. Some methods make use of smaller tables in conjunction with arithmetic operations. However, the results would be less accurate as the table size is reduced. The present invention is a fast and simple algorithm to solve the above problems. It generates the logarithmic value by iterations that involve multiplication, comparison and bit shifting only, but no look-up table is required. The accuracy of the result is unlimited as the number of iterations increases.

**Description**

The present invention is an algorithm to convert the mantissa, m, to its logarithmic value, $\log_2 m$. Since the logarithmic value of m is always in the range of [0.0, 1.0), $\log_2 m$ can be expressed in binary format as:

$$\log_2 m = 0.b_{-1}b_{-2}\ldots b_{-i}\ldots$$

where $b_{-i} = \{0, 1\}$ for all integer $i > 0$. The weighting of each bit $b_{-i}$ is $2^{-i}$.
Mathematically, the binary value can be expressed by the equation as follows:

$$\log_2 m = b_{-1}2^{-1} + b_{-2}2^{-2} + \ldots + b_{-i}2^{-i} + \ldots$$

Taking the anti-logarithm, we have

$$m = 2\text{^}(b_{-1}2^{-1})\ 2\text{^}(b_{-2}2^{-2})\ \ldots\ 2\text{^}(b_{-i}2^{-i})\ \ldots$$

The algorithm is to find out the bits, $b_{-i}$, by iterations starting from $b_{-1}$, then $b_{-2}$ and so on. In the first iteration, the bit $b_{-1}$ can be obtained by defining $m_1 = m$ and taking the square of $m_1$ as:

$$m_1^2 = 2\text{^}(b_{-1})\ 2\text{^}(b_{-2}2^{-1})\ \ldots\ 2\text{^}(b_{-i}2^{-i+1})\ \ldots$$

It can be seen that $m_1^2 \geq 2$ if and only if $b_{-1} = 1$ so that the value of $b_{-1}$ can be asserted. Then, the term containing $b_{-1}$ should be removed from $m_1^2$ in order to find out $b_{-2}$ in the second iteration. It can be done by dividing $m_1^2$ by $2\text{^}(b_{-1})$ so that

$$m_2 = m_1^2 / 2 \text{ if } b_{-1} = 1, \text{ or}$$
$$m_2 = m_1^2 \text{ if } b_{-1} = 0$$

In the second iteration, we start with

$$m_2 = 2\text{^}(b_{-2}2^{-1})\ 2\text{^}(b_{-3}2^{-2})\ \ldots\ 2\text{^}(b_{-i}2^{-i+1})\ \ldots$$

to obtain the value of $b_{-2}$. Similarly, the remaining bits can be found by the subsequent iterations until the required precision is obtained.

When the algorithm is implemented by hardware, the logarithm converter circuit can be illustrated by the block diagram in Fig. 2. The working register keeps the value $m_i$ for the i-th iteration. The square multiplier computes $m_i^2$ for the comparator to compare it with the constant 2.0. Then, the selector according to the comparator output selects $m_i^2$ or $m_i^2/2$ as $m_{i+1}$ for the next iteration. At the same time, the comparator output is the i-th bit, $b_{-i}$, of $\log_2 m$ so that it is serially left-shifted into the SIPO register. After sufficient

number of iterations, the value of $\log_2 m$ can readily be obtained from the SIPO register.

Working register – It is initialised with the input value m (i.e., $m_1$). After each iteration, this register is updated with $m_{i+1}$ for the next iteration.

Square multiplier – It computes the square of the input value. The resultant value has doubled number of decimal places of the input value. The extra decimal places should be ignored to keep the output value $m_i^2$ having the same precision as the input $m_i$. It is recommended to round off the output value when dropping the extra bits in order to obtain a more accurate logarithmic value.

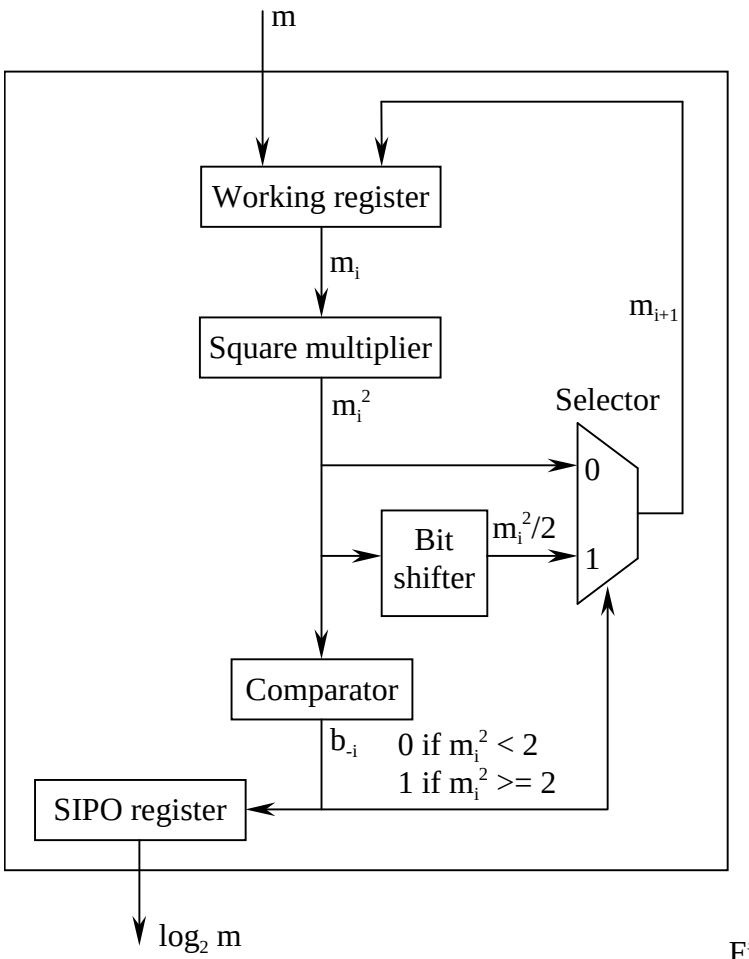


Fig. 2

Comparator – It always compares the input value with a constant 2.0. Its output bit is exactly bit 1 of the input value. It is logically a comparator but no logic gate is required.

Bit shifter and Selector – Both the bit shifter and selector can be combined together to be a shifter with selectable function. If the comparator output is 0, it directly output its input value as is. If the comparator output is 1, it right-shifts the input value by 1 bit.

SIPO register – It is a serial-in-parallel-out register. The comparator outputs of all iterations are serially left-shifted into this register one by one. Finally, the resultant logarithmic value can be read from its parallel output.

This algorithm can also be implemented by software as illustrated by the flow chart shown in Fig. 3. The variables m and y are used as the working register and the SIPO register respectively in Fig. 2. The two operations included in the same block have no data dependency. They can be done simultaneously in the parallel processing system.

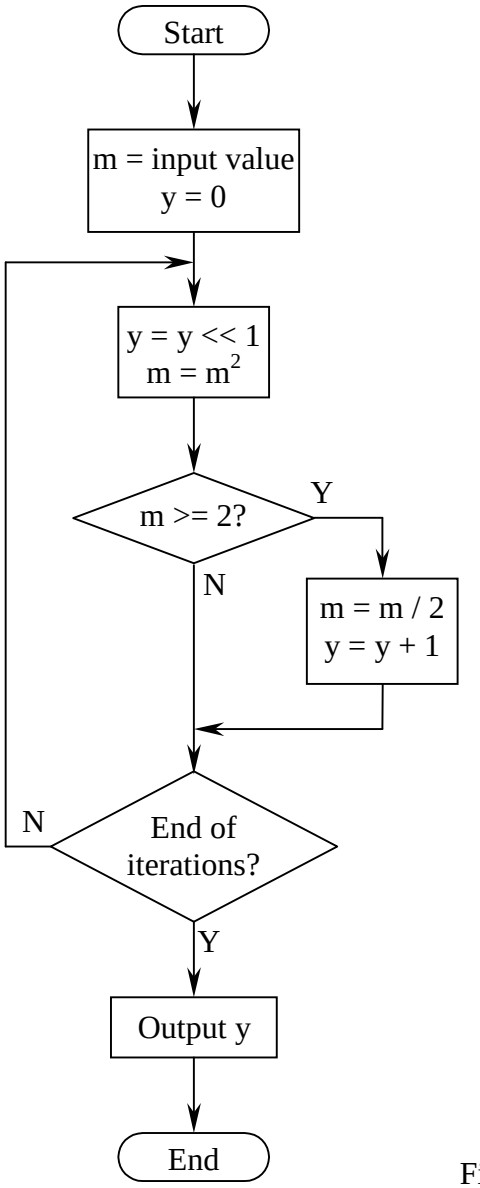


Fig. 3

**Reference patents**

5,365,465 Floating Point to Logarithm Converter, 26 Dec 1991.
5,524,089 Logarithm Computing Circuit for Fixed Point Numbers, 23 Nov 1993.
5,941,939 Logarithm/Inverse-Logarithm Converter and Method of Using Same, 25 Jun 1997