Proceedings of the IASTED International Conference
Signal and Image Processing
October 27-31, 1998, Las Vegas, Nevada – USA

A Number Representation Scheme with Carry-Free Rounding for Floating-Point Signal Processing Applications

Behrooz Parhami
Department of Electrical and Computer Engineering
University of California
Santa Barbara, CA 93106-9560, USA
parhami@ece.ucsb.edu

Stefan Johansson
Department of Applied Electronics
Lund University
221 00 Lund, Sweden
sjn@tde.lth.se

Abstract *– In this paper, we present a new redundant number representation called "double-LSB". The representation is obtained by taking a normal binary unsigned or 2's-complement number and adding an extra LSB with the same weight as the normal LSB (least significant bit). The advantages and drawbacks of this method are discussed and simple arithmetic algorithms are shown. In particular, it is shown how floating-point computation results can be properly rounded with no carry propagation.*

1. Introduction

Since the choice of number representation affects the implementation cost and delay of all arithmetic operations, many novel representation systems have been studied and compared against conventional schemes with respect to advantages and drawbacks [1]. In this paper, we propose yet another such novel representation.

The double-LSB representation is obtained by adding an extra bit to an unsigned or 2's-complement number. This extra bit has the same weight as the LSB. For example, four bit number is represented as $[a_3\ a_2\ a_1\ a_0\ a_0']$. Throughout the paper the extra LSB will be distinguished by a prime; e.g., a_0'. This gives a redundant representation where all numbers, except the highest and lowest, have two different representations. These numbers will be referred to as k-bit double-LSB numbers.

A double-LSB number can be converted to an infinitely long binary number by extending the number to the right, with the extra LSB repeated infinitely many times. E.g. [01101'] = [0110.11111...] is justified by the identity $x = x/2 + x/4 + x/8 + \ldots$.

Using the double-LSB format with 2's-complement numbers has the advantage that the range becomes symmetrical. A standard k-bit 2's-complement number has the range $[-2^{k-1}, 2^{k-1}-1]$ but with double-LSB this becomes $[-2^{k-1}, 2^{k-1}]$. This means that all numbers in this range can be complemented without causing overflow. Other ways of representing numbers with a symmetrical range include one's complement and sign magnitude. These however have some drawbacks.

Rounding numbers sometimes requires the addition of one in the least significant position. This normally requires a full carry propagation but with double-LSB representation this can be avoided. Rounding without full carry propagation is one of the major advantages of the double-LSB representation.

Conversion between standard representation and double-LSB is very simple. To convert from standard representation to double-LSB a zero is appended to the number. To convert from double-LSB to standard representation a full carry propagate adder or incrementer is necessary.

2. Addition and subtraction

Changing the sign of a two's complement number requires a full carry propagation. When using double-LSB representation, complementing can be performed only by inverting all bits in the number and thus no carry propagation is necessary. This is important when the complement is not followed by an addition, e.g. in convergence algorithms for division [2].

Addition and subtraction with carry propagate adders is simple. Two k-bit double-LSB numbers, a and b, can be added to obtain their sum s, using one k-bit adder. This can be done by using a_0' as a carry-in and letting $s_0' = b_0'$, as seen in Figure 1. In the case of subtraction, the subtrahend is negated (via bitwise complementation) before performing addition. The double-LSB representation also makes it possible to perform the operation $y = -a-b$ with a standard adder.

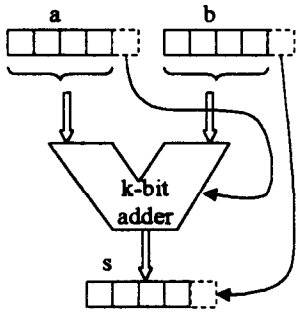

Figure 1. Addition of double-LSB integers.

Bit-serial adders can be used with the same principle as above, a_0' is used as carry-in in the first cycle and s_0' is set to b_0'. To produce a result where the extra carry is always zero however, the bit-serial adder must be redesigned because two carry bits need to be propagated.

3. Multiplication

Multiplication by two is done by shifting the number to the left, but instead of shifting in zeroes, the value of the extra LSB is shifted in. The extra LSB remains unchanged. This procedure can be understood by first converting the number to an infinitely long binary number, as discussed in Section 1, then shifting left, and finally converting back to a double-LSB number.

The add and shift method requires one more addition but no additional hardware if a carry propagate adder is used. The result can always be represented with $2k$ bits, given that $2^k * 2^k = 2^{2k}$.

Tree multiplier can also be used but it generally requires more hardware. Figure 2 shows a 4-bit standard 2's-complement multiplier that computes $p = a * b$, using Baugh and Wooley's method [3].

$$
\begin{array}{ccccccc}
 & & & a_3\bar{x}_0 & a_2x_0 & a_1x_0 & a_0x_0 \\
 & & a_3\bar{x}_1 & a_2x_1 & a_1x_1 & a_0x_1 & \\
 & a_3\bar{x}_2 & a_2x_2 & a_1x_2 & a_0x_2 & & \\
a_3x_3 & \bar{a}_2x_3 & \bar{a}_1x_3 & \bar{a}_0x_3 & & & \\
\bar{a}_3 & & & a_3 & & & \\
1 \quad \bar{x}_3 & & & x_3 & & &
\end{array}
$$

Figure 2. Multiplication of 4-bit 2's-complement numbers using Baugh and Wooley's method.

Figure 3 shows the same multiplier using double-LSB representation for both a and x.

$$
\begin{array}{cccccccc}
 & & & & a_3\bar{x}_0 & a_2x_0 & a_1x_0 & a_0x_0 \\
 & & & & a_3\bar{x}_0' & a_2x_0' & a_1x_0' & a_0'x_0 \\
 & & & a_3\bar{x}_1 & a_2x_1 & a_1x_1 & a_0x_1 & a_0x_0' \\
 & & a_3\bar{x}_2 & a_2x_2 & a_1x_2 & a_0x_2 & a_0'x_1 & a_0'x_0' \\
1 & a_3x_3 & \bar{a}_2x_0 & \bar{a}_1x_3 & \bar{a}_0x_3 & a_0'x_2 & & \\
 & \bar{a}_3 & & & \bar{a}_0'x_3 & & & \\
 & \bar{x}_3 & & & & & &
\end{array}
$$

Figure 3. Multiplication of 4-bit double-LSB numbers.

Comparing Figures 2 and 3, we note that the height of the highest column has not changed which means no added delay. The matrix size for this particular example is approximately 50% larger which means that 50% more hardware is required. For a k-bit multiplier the hardware increase is approximately $2k/k^2 = 2/k$. This implies that for a wide multiplier, the cost increase is minimal.

Multiplication using Booth's recoding [4] is very simple. Adding an extra LSB of 1 is the same thing as extending the number to the right with an infinite string of 1's. Therefore, when doing the recoding, instead of extending the number with zeros to the right, it is extended with the value of the extra LSB.

$[a_3\ a_2\ a_1\ a_0\ a_0'] = [a_3\ a_2\ a_1\ a_0.a_0'\ a_0'\ a_0' \ldots.]$

Recoding replaces all the repeated fractional digits with 0s and adjusts the first recoded digit accordingly. With this provision, both recoding and multiplication are performed in the normal way.

It is possible to design multipliers for double-LSB number using AMMs (additive multiply modules) if they are slightly modified. Multiplying two 4-bit double-LSB numbers, A and B, can be written as:

$AB = \{a_3\ a_2\ a_1\ a_0\} * \{b_3\ b_2\ b_1\ b_0\} + a_0'\{b_3\ b_2\ b_1\ b_0\} + b_0'\{a_3\ a_2\ a_1\ a_0\} + a_0'b_0'$.

This can be seen as a normal 2's-complement multiplication but with the addition of two extra 4-bit numbers and a one bit number. Similarly, multiplying two 8-bit numbers requires the addition of two 8-bit numbers due two the two extra LSB's. An 8-bit multiplier built with AMMs has 16 unused inputs. With some modifications to the AMMs these can be used to add the two extra numbers. The remaining bit $a_0'b_0'$ is used as the extra LSB of the result. The extra numbers are formed by a set of AND gates. These AND gates can be built into the AMMs. When the AND function is not needed, the control input c_x or c_y is set to one.

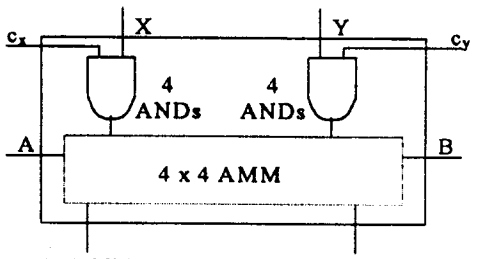

Figure 4. Additive multiply module for double-LSB numbers.

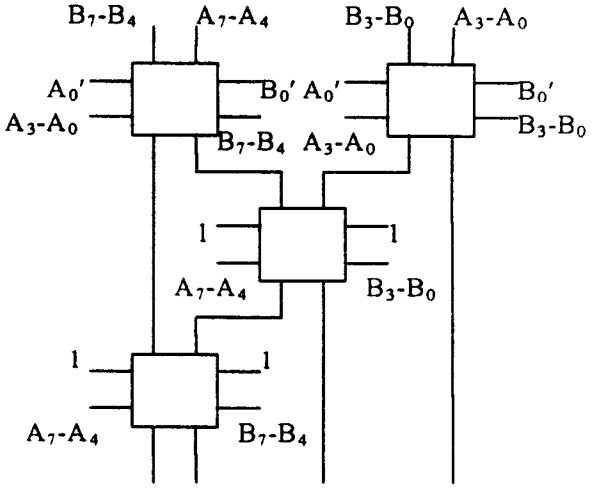

Figure 5. Double-length multiplication of double-LSB numbers using single length additive multiply modules.

Figure 4 shows the modified AMM. The dotted box contains the original AMM. The modified AMM now performs the operation $AB + c_xX + c_yY$. By setting c_x and c_y to one it can be used as a normal AMM. Figure 5 shows an example of an 8*8 multiplier built of four 4*4 AMMs.

4. Division

Dividing by 2 is done by a right shift. For correct rounding, it can be shown that the extra LSB should be set to the AND of itself and all the bits that are shifted out.

Double-LSB representation can be used together with all the standard division algorithms but sometimes this might require extra considerations. When doing division with standard restoring or non-restoring algorithms, the partial reminder is kept in double-LSB format. Since the quotient digit is dependent on the sign of the partial reminder, the sign detector must be designed to correctly detect the sign of a double-LSB number.

This problem does not arise when using binary [5] or high-radix [6] SRT division. In this case, only a few of the most significant bits or digits are used to determine the next quotient digit, and having the extra LSB will not make a difference.

5. Floating point arithmetic

In floating point arithmetic the resulting mantissa often contains more bits than necessary and has to be shortened to the appropriate length for the ALU output. This is done by removing the extra bits beyond the desired LSB and rounding. This rounding can in a worst-case scenario result in full carry propagation if rounding up takes place. This is where the double-LSB number representation shows it's biggest advantage. Assume the number $(x_{k-1}x_{k-2}\ldots x_1x_0.x_{-1}x_{-2}\ldots x_{-n}x_{-n}')$ is to be rounded to $(x_{k-1}x_{k-2}\ldots x_1x_0x_0')$. In case of rounding up, instead of adding a one, which may result in a carry propagation, x_0' is set to one. The size of the fractional part $f = (.x_{-1}x_{-2}\ldots x_{-n}x_{-n}')$ determines whether to round up or down. The following rule gives the size of f.

$f < ½$ iff $x_{-1} = 0$ and any of x_{-2} to $x_{-n}' = 0$
$f > ½$ iff $x_{-1} = 1$ and any of x_{-2} to $x_{-n}' = 1$
$f = ½$ otherwise

Rounding to nearest even [7] can be done by setting:

$x_0' = 1$ if $f > ½$ or if $f = ½$ and $x_0 = 1$
$x_0' = 0$ if $f < ½$ or if $f = ½$ and $x_0 = 0$

Other rounding schemes, notably downward-directed and upward directed rounding [7] that are useful for interval arithmetic, can be similarly accommodated.

6. Conclusions

We have presented a new redundant number representation called double-LSB obtained by adding an extra LSB to standard unsigned or 2's-complement numbers. This requires the use of one additional bit for storage as well as extra cost of arithmetic in the form of increased hardware or delay.

All numbers, including zero, have two different representations. Although both representations can be used in arithmetic operations, special considerations are required for zero and sign detection.

The major advantage is that rounding, which often requires an addition of one in the least significant position, can be done without carry propagation. If double-LSB is used with two's complement numbers it gives the added benefit of a symmetric range.

Even though the numbers contain $k+1$ bits, many arithmetic operation can be performed with little increase in hardware or delay. Addition and subtraction require no extra cost and complementation is even simpler since it can be done by bitwise inversion.

Shift-and-add multiplication requires one extra cycle. Booth's recoding can be used with no extra cost at all. Tree multipliers however require some extra hardware (but there is no increase in delay). Restoring and non-restoring as well as SRT division algorithms can be used with double-LSB representation with only minor modifications.

Each k-bit double-LSB number can be viewed as a redundant radix-2^k digit, with its value in $[0, 2^k]$ or $[-2^{k-1}, 2^{k-1}]$. This viewpoint allows multi-precision arithmetic to be performed in a carry-free manner using the rules of generalized signed-digit [8] arithmetic.

References

[1] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford University Press, 1998 (to appear).

[2] D. Ferrari, "A division method using a parallel multiplier," *IEEE Trans. Computers*, vol. 16, pp. 224-226, Apr. 1967.

[3] C. R. Baugh and B. A. Wooley, "A two's complement parallel array multiplication algorithm," *IEEE Trans. Computers*, vol. 22, pp. 1045-1047, Dec. 1973.

[4] A. D. Booth. "A signed binary multiplication technique," *Quarterly J. Mech. Appl. Math.*, vol. 4, part 2, pp. 236-240, 1951.

[5] I. Koren, *Computer Arithmetic Algorithms*, Prentice Hall, 1993, pp. 127-133.

[6] J. E. Robertson, "A new class of digital division methods," *IRE Trans. Electronic Computers*, vol. 7, pp. 218-222, Sep. 1958.

[7] *Standard for Binary Floating-Point Arithmetic*, ANSI/IEEE 754, 1985

[8] B. Parhami, "Generalized signed-digit number systems: A unifying framework for redundant number representations," *IEEE Trans. Computers*, vol. 39, pp. 89-98, Jan 1990.