Arithmetic with Binary-Encoded Balanced Ternary Numbers

Behrooz Parhami and **Michael McKeown**[*]

Department of Electrical and Computer Engineering, University of California
Santa Barbara, CA 93106-9560, USA
[*] Currently with Electrical Engineering Department, Princeton University

parhami@ece.ucsb.edu

Abstract

Ternary number representation and arithmetic, based on the radix-3 digit set {−1, 0, +1}, *has been studied at various times in the history of digital computing. Some such studies concluded that we should abandon ternary in favor of binary computation. Others, demonstrated promise and potential advantages, but, for various reasons, including inertia, did not lead to widespread use. By proposing an efficient binary encoding for balanced ternary numbers, along with the corresponding arithmetic circuits, we argue that a reexamination of the decision against using ternary arithmetic might be in order.*

Keywords—Arithmetic/logic unit, Computer arithmetic, Nonbinary systems, Radix-3 number representation.

Background and Motivation

Binary arithmetic is predominant in digital systems, so much so that few question its superiority or optimality. Decimal arithmetic, which until recently was mostly implemented by means of software where needed, has emerged as a candidate for hardware implementation, with a variety of representations, algorithms, and design methods proposed [Wang10]. Early in the history of electronic computers, the choice of number representation radix, the binary-versus-decimal question in particular, was given quite a bit of attention, with the binary system prevailing at the end [vonN45], [Burk47]. Radices other than 2^h and 10 have been mostly ignored.

In reality, ternary (radix-3) representation came quite close to being chosen over binary as the preferred method, eventually losing by a narrow margin. It was argued that under some fairly realistic assumptions about circuit cost and latency, radix 3 is closer to the theoretically optimal radix e (the base of the natural logarithm) than any other integer radix. However, practical engineering considerations, having to do with the ease of dealing with 2-state devices and binary logic, favored radix 2 over radix 3 [Haye01].

Despite the negative assessment above, the Setun computer, working with balanced ternary arithmetic, was actually built in 1958 at Moscow State University and found to be quite usable and competitive [Klim99]. Until 1965, some 50 units of this machine had been built. Advantages cited for this ternary machine included lower production cost and greater energy economy.

The TERNAC computer [Frie72], implemented at State University of New York, Buffalo, in 1973, did not use hardware specifically designed for dealing with ternary numbers, but emulated ternary arithmetic operations by representing each balanced ternary number as two binary numbers, one positive and the other negative. Arithmetic operations were then microprogrammed, using the hardware's binary arithmetic capabilities. TERNAC was a load/store machine, with 24-trit integers and a 48-trit floating-point (42 for mantissa, 6 for exponent) format.

The history of balanced ternary number system and arithmetic goes much further back than the Setun and TERNAC computers, however. In 1820, John Leslie presented methods for computing in any radix, with an arbitrary digit set [Lesl20]. Two decades later, Augustin Cauchy discussed signed-digit numbers in various bases and Leon Lalanne followed by enumerating the virtues of balanced ternary number representation. In 1840, Thomas Fowler (1777-1843), a contemporary of Charles Babbage, chose balanced ternary to build his calculating machine in England. A description of this machine by the mathematician Augustus DeMorgan led Pamela Vass, David Hogan, and Mark Glusker to design and build a replica in the year 2000 [Fowl13].

Finally, work on ternary number representation and arithmetic has not been limited to the Setun and TERNAC experiments [Halp68], [Eich86], [Gund06], [Gund06a], [Tern13]. Most proposals for ternary arithmetic envisage multivalued signals to encode the three digit values in balanced or standard ternary. Multivalued signaling and logic, extensively studied within a research community with conferences and a specialized journal [Mult13], and practically available for several decades, has proven uncompetitive in most instances.

The last observation above motivates us to investigate potential advantages of balanced ternary arithmetic, using binary encoding. Unlike TERNAC, however, we aim to design actual arithmetic function units that deal with ternary numbers directly, rather than via emulation with binary arithmetic instructions.

Notational Conventions

A balanced ternary integer $x = (x_{k-1}x_{k-2} \ldots x_1x_0)_{\text{three}}$, with each x_i being from the digit set {–1, 0, +1}, denotes the value $\Sigma_{0 \le i \le k-1}\, x_i 3^i$. Balanced ternary integers with k digits have values from $-(3^k - 1)/2$ to $(3^k - 1)/2$, a fully symmetric range. Negation, or change of sign, is done by digitwise complementation.

For ease of representation, and to avoid confusion between logical signal values 0 and 1 and ternary digit values 0 and 1, we represent ternary digits as N (–1), Z (0), and P (1), and we omit the subscript "three" when there is no ambiguity. Thus, the 7-digit code $x_6x_5x_4x_3x_2x_1x_0$ = PNZZNNP is the balanced ternary representation of the integer $3^6 - 3^5 - 3^2 - 3^1 + 3^0 = 475$.

Any binary encoding of a ternary digit value requires at least 2 bits. There are 4! = 24 ways of assigning 3 of the 4 possible 2-bit codes to the three digit values N, Z, and P. For example, one can use a 2-bit 2's-complement code for the three digit values: N (11), Z (00), and P (01). Our prior experience [Jabe12] with the use of posibits (normal bits, assuming values in {0, 1}) and negabits (negatively weighted bits, assuming values in {–1, 0}) leads us to use the $\langle n, p\rangle$ encoding for balanced ternary digits: N (10), Z (00 or 11), and P (01). Note that allowing 11 to denote Z, rather than using it as a don't-care state, relieves us from the burden of watching out for this combination and setting it to 00 whenever encountered.

The code just discussed has a variant that uses inversely encoded negabits (IE-negabits), where N is encoded as 0 and Z as 1. This variant, which we may call the $\langle n^-, p\rangle$ code, often leads to simplifications, given that the logical bit value and the corresponding digit value move in the same direction (0 : N < Z : 1), with the logical bit essentially being a biased representation of the digit value, using a bias of +1. To distinguish this encoding, we attach a "–" superscript to the corresponding IE-negabit constant: N (0^-0), Z (0^-1 or 1^-0), and P (1^-1).

From this point on, 0 and 1 denote logical signal values, while N, Z, and P denote balanced ternary digit values; thus the 0/1 overlap will not lead to any confusion.

Summary of Results

Addition/Subtraction

The truth tables for the sum and carry outputs of a balanced ternary half-adder are depicted in Fig. 1. Interpreting the table is quite simple. For example, the half-addition P + P results in PN, that is, a sum digit N and a carry digit P (i.e., 1 + 1 = 3 – 1). Such a balanced ternary half-adder has a gate complexity about twice that of a binary HA. Similarly, a balanced ternary full-adder has a complexity comparable to two binary FAs. If negabits and posibits are written as uppercase and lowercase letters, respectively, and using d to denote carry-out (so as to avoid the use of subscripts "in" and "out" for the carry), we have:

$$\langle X, x\rangle + \langle Y, y\rangle + \langle C, c\rangle = \langle D, d\rangle\, \langle S, s\rangle$$

A ternary ripple-carry adder can be built by cascading a number of ternary FAs. The double complexity of a ternary FA compared with its binary counterpart does not imply a 100% increase in cost relative to binary arithmetic. For the same number representation range, a balanced ternary system needs a factor of $\log_2 3 = 1.585$ fewer digits compared with the 2's-complement format. The factor of 1.585 improvement in carry propagation time leads to a faster ripple-carry adder, thus potentially justifying the cost factor of 2/1.585 = 1.262, even if we do not consider other advantages of ternary arithmetic.

	N	Z	P
N	P	N	Z
Z	N	Z	P
P	Z	P	N

(a) Sum digit

	N	Z	P
N	N	Z	Z
Z	Z	Z	Z
P	Z	Z	P

(b) Carry digit

Fig. 1. Sum and carry digits in balanced ternary half-adder.

Designs of fast (logarithmic-time) adders of various kinds are currently under investigation. Using positive and negative carry-generate and carry-propagate signals, one can employ two copies of any fast carry network to find the intermediate carry signals. For a linear-cost network, such as the Brent-Kung design [Parh10], the same cost increase of 1.262 (discussed for ripple-carry adders) will apply, along with some speed improvement. For fastest possible networks such as the Kogge-Stone design, having $O(k \log k)$ or greater complexity, the cost factor would be smaller, thus rendering the improvement in speed even more cost-effective.

Multioperand Addition

In balanced ternary arithmetic, multioperand addition is nearly identical to the binary case. This is because, except for the {0, 1} digit set changing to {N, Z, P}, combining schemes, and associated block diagrams remain the same. For example, (3; 2)-counters, (4; 2)-counters, and various other parallel counter/compressor circuits can be used here with no modification in the high-level block diagrams; only the lowest-level circuit detail will change.

Multiplication

Owing to the fact that the product of two ternary digits in {N, Z, P} also belongs to the same set, balanced ternary multiplication is simpler than multiplication in other radices higher than 2. In fact, we can readily prove that radices 2 and 3, the latter only with the balanced digit set {N, Z, P}, are the only two radices with this property. Thus, just as in multioperand addition discussed earlier, all binary multiplication schemes remain valid for balanced ternary. These include the slow digit-at-a-time iterative schemes, fast logarithmic-time tree multipliers, and high-throughput array designs [Parh10].

The balanced ternary multiplication table is depicted in Fig. 2. Denoting the two digits being multiplied by (X, x) and (Y, y), the product digit will have the representation $(Xy \vee xY, XY \vee xy)$. Hardware realization entails the use of two 2-to-1 muxes, with select and enable controls.

	N	Z	P
N	P	Z	N
Z	Z	Z	Z
P	N	Z	P

Fig. 2. Truth table for balanced ternary multiplication.

Division and Square-Rooting

Other than the need for digit selection, division and square-rooting algorithms resemble multiplication. Given the use of signed-digit values, the distinction between restoring and nonrestoring division disappears in balanced ternary arithmetic. Here, we consider only digit-at-a-time division. Details of other division and square-rooting schemes will be reported in future. Our exposition is in terms of dividing a $2k$-digit balanced ternary dividend by a k-digit divisor, producing a quotient and a remainder, both having k digits. Overflow and divide-by-zero tests are identical to the same tests in binary division.

Figure 3 depicts an 8-by-4 division example, with the dividend ZPNZZPNZ (492 in decimal) and divisor NZPN (–25 in decimal). Using a quotient digit selection rule identical to that of nonrestoring binary division, based on the signs of the partial remainder and the divisor, produces the quotient NPNP (–20 in decimal) and the remainder ZNZP (–8). Because the standard definition of division requires that the remainder have the same sign as the dividend, a correction step is needed at the end, which, for this example, involves subtracting the divisor from the final remainder, producing the true remainder of PNZN (17 in decimal), and adding 1 to the quotient, yielding the true quotient NPZN (–19).

Note that quotient digit selection requires that the sign of the partial remainder be detected in every iteration; divisor sign is determined just once at the outset. Because, similar to the case of binary signed-digit numbers, the sign of a balanced ternary number matches the sign of its most-significant nonzero digit, determining the sign is nontrivial. One option is to use fast, logarithmic-time sign detection logic based on lookahead principles [Srik04]. Another option is to avoid the sign detection latency by using a redundant representation of the quotient, along with approximate digit selection [Parh10]. These and other options are being explored.

```
                  N P N P
          ---------------
N Z P N / Z P N Z Z P N Z
        +   N Z P N
            -------
            Z N P N P N Z
          +   P Z N P
              -------
              Z P N N N Z
            +   N Z P N
                -------
                Z N N P Z
              +   P Z N P
                  -------
                  Z N Z P
```

Fig. 3. Division example: 492 / (–25) = –20, remainder –8.

Input/Output Conversions

To interface a balanced ternary arithmetic unit with the external world, we must devise procedures for conversion of numbers to/from balanced ternary from/to decimal or binary format. Conversion from/to binary may be unnecessary if an entire system is designed to operate with balanced ternary arithmetic, as primary inputs are almost always in decimal format. At any rate, since the procedures are the same, regardless of decimal/binary I/O, we focus only on conversion from/to decimal.

Conversion from decimal to balanced ternary is no more difficult than conversion to binary. Instead of repeatedly dividing by 2, we need to implement a division-by-3 circuit and use it in *k* cycles to determine all the digits of the converted operand. Going in the other direction, we can use Horner's method [Parh10], which requires repeated multiplication by 3 (in decimal format), followed by the addition of a balanced ternary digit. Again, this process is of the same order of time/circuit complexity as binary-to-decimal conversion. Design details will be supplied in future.

Floating-Point Arithmetic

A key advantage of balanced ternary arithmetic materializes in floating-point arithmetic. A cause of added complexity and delay in binary and decimal floating-point arithmetic is the requirement for rounding. Both simple rounding and symmetric rounding, with midway values rounded up or down with equal probabilities for more balanced error (e.g., as in round-to-nearest-even), require full carry propagation in the worst case, thus lengthening the critical signal path in a synchronous design. Balanced ternary numbers, on the other hand, provide the same error characteristics with simple chopping.

Conclusion

We have argued that the use of balanced ternary number representation and arithmetic, by means of binary-encoded digits, can be competitive, and leads to higher speeds for certain applications. Advantages of balanced ternary include fully symmetric representation, ease of negation or sign change (by switching Ps and Ns), use of fewer digits for the same range (albeit, with 2 bits used to represent each digit), and rounding to nearest value via truncation. The benefits result in part from our efficient encoding using posibits and negabits.

References

[Burk47] Burks, A. W., H. H. Goldstine, and J. von Neumann, "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument," Institute for Advanced Study Report, 2nd ed., 1947.

[Eich86] Eichmann, G., Y. Li, and R. R. Alfano, "Optical Binary Coded Ternary Arithmetic and Logic," *Applied Optics*, Vol. 25, No. 18, pp. 3113-3121, 1986.

[Fowl13] Web site for "The Ternary Calculating Machine of Thomas Fowler" (Accessed on July 17, 2013): http://www.mortati.com/glusker/fowler/index.htm

[Frie72] Frieder, G., "Ternary Computers — Part 1: Motivation & Part 2: Emulation," *Proc. 5th Workshop Microprogramming*, 1972, pp. 83-89.

[Gund06] Gundersen, H. and Y. Berg, "A Novel Balanced Ternary Adder Using Recharged Semi-Floating Gate Devices," *Proc. IEEE Int'l Symp. Multivalued Logic*, 2006, pp. 18-21.

[Gund06a] Gundersen, H. and Y. Berg, "A Balanced Ternary Multiplication Circuit Using Recharged Semi-Floating Gate Devices," *Proc. 24th Norchip Conf.*, 2006, pp. 205-208.

[Halp68] Halpern, I. and M. Yoeli, "Ternary Arithmetic Unit," *Proc. IEE*, Vol. 115, No. 10, pp. 1385-1388, October 1968.

[Haye01] Hayes, B., "Third Base," *American Scientist*, Vol. 89, No. 6, pp. 490-494, November-December 2001.

[Jabe12] Jaberipur, G. and B. Parhami, "Efficient Realisation of Arithmetic Algorithms with Weighted Collections of Posibits and Negabits," *IET Computers & Digital Techniques*, Vol. 6, No. 5, pp. 259-268, September 2012.

[Klim99] Klimenko, S. V., "Computer Science in Russia: A Personal View," *IEEE Annals of the History of Computing*, Vol. 29, No. 3, pp. 16-30, 1999.

[Knut81] Knuth, D., *The Art of Computer Programming: Seminumerical Algorithms*, Addison-Wesley, 2nd ed., 1981, pp. 190-192.

[Lesl20] Leslie, J., *The Philosophy of Arithmetic: Exhibiting a Progressive View of the Theory and Practice of Calculation*, 2nd ed., 1820.

[Mult13] Multivalued logic home page, sponsored by IEEE Computer Society (accessed on July 17, 2013): http://web.cecs.pdx.edu/~mperkows/ISMVL/=index.html

[Parh10] Parhami, B., *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford, 2nd ed., 2010.

[Srik04] Srikanthan, T., S. K. Lam, and M. Suman, "Area-Time Efficient Sign Detection Technique for Binary Signed-Digit Number System," *IEEE Trans. Computers*, Vol. 53, No. 1, pp. 69-72, January 2004.

[Stak02] Stakhov, A., "Brousentsov's Ternary Principle, Bergman's Number System and Ternary Mirror-Symmetrical Arithmetic," *The Computer J.*, Vol. 45, pp. 221-236, 2002.

[Tern13] Ternary.info, Web site of the Special Interest Group on Balanced Ternary Numeral System and Trinary Logic (Accessed on July 17, 2013): http://www.ternary.info/modules/newbb/index.php

[vonN45] von Neumann, J., "First Draft of a Report on the EDVAC," *IEEE Annals of the History of Computing*, Vol. 15, No. 4, pp. 27-75, 1993. [A reproduction of the 1945 report.]

[Wang10] Wang, L. K., M. A. Erle, C. Tsen, E. Schwarz, and M. J. Schulte, "A Survey of Hardware Designs for Decimal Arithmetic," *IBM J. Research and Development*, Vol. 54, No. 2, pp. 8:1-8:15, March-April 2010.