The equivalence of twos-complement addition and the conversion of redundant-binary to twos-complement numbers

Gerard M Blair

The Department of Electrical Engineering
The University of Edinburgh
The King's Buildings
Edinburgh, EH9 3JL
Scotland, UK

ABSTRACT

The equivalence between redundant-binary to twos-complement number conversion and twos-complement addition is shown using a simple transform between the two number domains. As a consequence, all hardware architectures designed for either operation may be easily adapted to implement the other.

1. Introduction

The design of logic for high-speed arithmetic operation can be undertaken using a redundant binary (RB) representation[1, 2] which has the advantage that it avoids carry-propagation. In such systems, there is the need to convert the final result form RB number format to twos-complement number representation. Although it is possible to convert from one particular encoding of RB representation using a two-complement subtraction (as explained below), several designers have proposed logic architectures specifically for the conversion process[1-4]. While some of these authors note the similarity of their designs to twos-complement adders, it has not yet been suggested that *all* logic/circuit architectures for either twos-complement addition or RB to twos-complement conversion can be used with the same advantages in both applications.

This paper is motivated by previous work[5] concerning a paper by Srinivas and Parhi[4] which proposed a fast twos-complement adder containing an RB to twos-complement convertor. Their method was to convert the two twos-complement operands to an RB representation, perform a redundant addition, and then convert the result back to twos-complement representation. The key to their high-speed was in the final conversion circuitry. However, we have shown[5], for this specific example, that the final-conversion circuitry alone can be used to implement an even faster addition by transforming the original twos-complement operands directly to the input format of the conversion logic, and by inverting the conversion logic's outputs. This eliminates the redundant addition stage and produces a shorter critical path. In brief, we revealed that the Srinivas and Parhi[4] converter is the same architecture as a hybrid carry lookahead/carry select adder[6].

In this paper we extend the previous work to prove the fundamental equivalence between architectures for twos-complement addition and RB to twos-complement conversion. The proof results in a simple procedure to transform any architecture between these two functions and this is illustrated by a worked example.

2. Demonstration of equivalence

In the following proof, we present a notation for RB numbers and for a similar number system which expresses pairs of twos-complement numbers. In the former case we show the equation to convert the RB number into twos-complement representation; and in the latter, to convert the number pairs into their twos-complement sum. We then introduce a simple transform between the two number systems and demonstrate, through the equations, an equivalence between the two number transforms.

Let $S_x = \{\ 1, 0, \bar{1}\ \}$ where $\bar{1}$ represents -1, and let X_n be the set of n-digit numbers: $(x_{n-1}, \cdots, x_0)$ where $x_i \in S_x$ for $i = 0 \rightarrow n-1$. This is the *redundant-binary* number representation. Any n-digit RB number ($x \in X_n$) can be converted to twos-complement representation by the function:

$$T_x(x) = \sum_{i=0}^{n-1} x_i\ 2^i \tag{1}$$

where T_x: $X_n \rightarrow [-2^n + 1, 2^n - 1]$. This range can be represented in an (n+1) bit binary twos-complement number provided that the most-significant-bit (msb) is interpreted as the sign-bit.

Let $S_y = \{\ 0, 1, 2\ \}$, and Y_n be the set of n-digit numbers: ($y_{n-1}, \cdots, y_0$) where $y_i \in S_y$ for $i = 0 \rightarrow n-1$. The set S_y is motivated by twos-complement addition, and may be thought of as the initial summation of the corresponding bits of two n-bit (positive) operands in a twos-complement addition. Let us call this the *initial-sum* (IS) number representation. Any n-digit IS number ($y \in Y_n$) can be converted to twos-complement representation by the function:

$$T_y(x) = \sum_{i=0}^{n-1} x_i\ 2^i \tag{2}$$

where T_y: $Y_n \rightarrow [0, 2^{n+1} - 2]$. This range can be represented in an (n+1) bit binary twos-complement number provided that the value is interpreted as a positive integer (i.e. the msb is not a sign bit).

Both S_x and S_y have three elements and so (at least) two bits are required for encoding. Examples of common encodings for both sets are given in Table 1.

	S_y		S_x	
(l,r)	code1	code2	code3	code4
(0,0)	0	0	0	0
(0,1)	1	1	1	$\bar{1}$
(1,0)	1	2	na	1
(1,1)	2	na	$\bar{1}$	0

Table 1: common encodings of S_y and S_x

If we consider S_y purely in the context of addition then *code1* can be seen as the sum of the left (l) and right (r) naturally encode S_y by forming the digits as pairs of their corresponding bits. On the other hand, the encoding *code2* is commonly used in the design of carry-lookahead and carry-select adders. In these architectures, the left bit of the encoding is know as the *generate* signal (the bit-wise AND of the two operands), and the right bit is the *propagate* signal (the bit-wise XOR). Note that in *code2* one of the possible values of the encoding is defined as *not applicable* (na); in some definitions this code value is assigned to one of the three elements.

Code3 illustrates the *sign-magnitude* encoding for RB digits: the left bit is the magnitude of the digit and the right bit is the sign (equal to "1' if the digit is negative). An alternative encoding for RB numbers is the *positive-negative* representation shown as *code4* in which the value of the digit is the right bit subtracted from the left bit. Clearly, digital logic can be used to transform between different encoding schemes both within the same, and between the different, sets.

We define the function $t_{xy}(z_i) = (1 - z_i)$ for $z_i \in S_x \cup S_y$. The subscript is chosen to indicate that t_{xy} can acts on both S_x and S_y. It can be seen that: t_{xy}: $S_x \rightarrow S_y$ and t_{xy}: $S_y \rightarrow S_x$; furthermore t_{xy} is self inverting in that $t_{xy}(t_{xy}(z_i)) = z_i$ for all $z_i \in S_x \cup S_y$. The mapping can be summarized as:

$$2 \leftrightarrow \bar{1},\ 1 \leftrightarrow 0,\ 0 \leftrightarrow 1$$

Finally we extend the definition of t_{xy} to mappings between elements of the the sets X_n and Y_n by applying t_{xy} to each digit in turn.

We now seek to establish the relationship between the results of the functions T_x and T_y on the corresponding elements in X_n and X_n related by the function t_{xy}. Let $y \in Y_n$ and define $x = t_{xy}(y)$

noting that therefore $y = t_{xy}(x)$. Since $x \in X_n$, we may apply the function T_x:

$$\begin{aligned} T_x(x) &= T_x(tr_{xy}(y)) \\ &= \sum_{i=0}^{n-1} tr_{xy}(y_i)\, 2^i \\ &= \sum_{i=0}^{n-1} (1 - y_i)\, 2^i \\ &= \sum_{i=0}^{n-1} 2^i - \sum_{i=0}^{n-1} y_i\, 2^i \\ &= 2^n - 1 - T_y(y) \end{aligned}$$

The values in this equation are valid in ordinary twos-complement arithmetic. Therefore we can use the well know relation that for any number a represented in two-complement binary notation: $-a = \bar{a} + 1$ i.e. the negative of a number is its inverse plus one. The previous equation thus becomes:

$$T_x(x) = 2^n + \overline{T_y(y)} \tag{3}$$

In practical terms, this means that to convert a RB number ($x \in X$) to twos-complement representation:

1) transform x by applying tr_{xy} digit by digit

2) convert the resulting IS word to twos-complement representation (i.e. perform the addition)

3) invert all the bits and add 2^n

The final addition of 2^n effectively cancels the inversion of the most-significant bit of $T_y(y)$ and removes the sign extension of negative results; this being consistent with the non-negative range of $T_x(x)$.

Due to symmetries in the development of equation (3), it also follows that twos-complement addition can be performed by transforming the IS number to a RB number using tr_{xy}, converting the RB number to twos-complement representation, and then inverting the result and adding 2^n.

In either case, if the result of the transformation is an (n+1) bit word (as would be the case in n-digit hardware design), then the final step can be summarized as "invert all the bits except the most-significant-bit".

3. Example of function transformation

In twos-complement adder design, a common technique involves transforming the bits of the input operands A and B into *generate* ($G_i = a_i \, . \, b_i$) and *propagate* ($P_i = a_i \oplus b_i$) signals so that the equations for *sum* and *carry* can be rewritten as

$$carry_i = G_i \;+\; P_i \;.\; carry_{(i-1)}$$

$$sum_i = P_i \oplus carry_{(i-1)}$$

In effect, the initial transformation corresponds to a conversion from *code1* to *code2* in table 1. Figure 1(a) shows logic based upon these equation to implement a simple ripple adder.

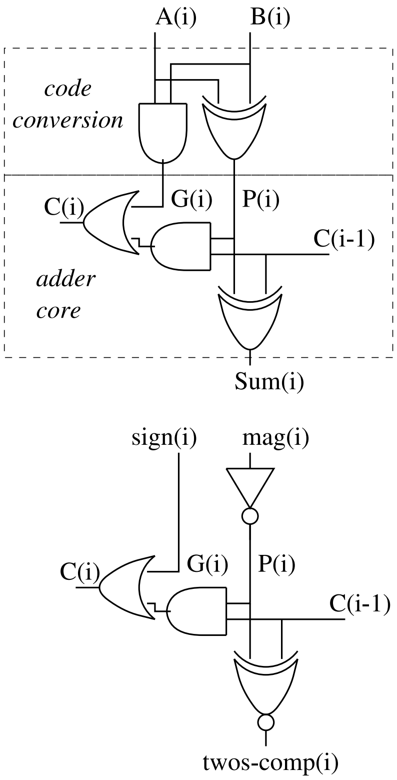

Fig. 1. The transformation of logic from (a) twos-complement addition, to (b) redundant-binary to twos-complement number conversion.

This logic may be used directly to implement a conversion from a redundant binary representation in the following manner. Let us assume that the RB number is encoded in sign-magnitude form (as in *code3)* The first step is to map the encoding of elements S_y onto an encoding for the corresponding elements in S_x (according to the function t_{xy}). We could map onto *code1* and use the full logic of figure 1a; however, it is more efficient to map onto *code2* (the *generate-propagate* code) and so eliminate the first layer of logic (which converts *code1* to *code2).* Thus we have the transformation:

	$code3$		t_{xy}		$code2$	
(0, 0)	≡	0	→	1	≡	(0, 1)
(0, 1)	≡	1	→	0	≡	(0, 0)
(1, 1)	≡	$\bar{1}$	→	2	≡	(1, 0)

which is effected by inverting the magnitude (right-hand) bit.

The second step is to use the remaining logic from figure 1a (after the G_i and P_i signals), and the final step is to invert the outputs (except for bit n+1 which is the final carry-out). Thus the full conversion from sign-magnitude RB number representation to binary twos-complement is implemented by the logic in Figure 1b.

This example used a simple carry-ripple technique; however, the transformation procedure can be applied to all twos-complement adder designs. For instance, a hierarchical carry lookahead technique such as the Brent-Kung adder[7] could be used in exactly the same way to form an RB to twos-complement convertor. Just as in the example above, the sign-magnitude format is converted to the corresponding *generate-propagate* format according to the tr_{xy} transform, and all the sum outputs are inverted (although not the final carry).

4. Conclusion

While the RB to twos-complement converter has often been viewed as a distinct logic-design problem, we have shown that it is essentially equivalent to twos-complement adder design. The important implications are that any advance in logic or circuit techniques in either application area can be immediately applied to the other; and that the design of RB to twos-complement converters may draw directly upon the extensive literature for high-performance twos-complement adders.

References

1. R F Woods and J V McCanny, "Design of a high-performance IIR digital filter chip", *IEE Proc-E*, vol. 139, no. 3, pp. 195-202, May 1992.

2. H Makino, Y Nakase, H Suzuki, H Morinaka, H Shinohara, and K Mashiko, "An 8.8-ns 54x54-bit multiplier with high speed redundant binary architecure", *IEEE J. Solid-State Circuits*, vol. 31, no. 6, pp. 773-782, June 1996.

3. S-M Yen, C-S Laih, C-H Chen, and J-Y Lee, "An efficient redundant-binary number to binary number converter", *IEEE J. Solid-State Circuits*, vol. 27, no. 1, pp. 109-112, Jan 1992.

4. H R Srinivas and Keshab K Parhi, "A Fast VLSI Adder Architecture", *IEEE Journal of Solid-State Circuits*, vol. 27, no. 5, pp. 761-767, May 1992.

5. J.M. Dobson and G.M. Blair, "Fast twos complement VLSI adder design", *IEE Electronics Letters*, vol. 31, no. 20, pp. 1721-1722, Sept 1995.

6. T Lynch and E E Swartzlander, "A spanning tree carry lookahead adder", *IEEE Trans Computers*, vol. 41, no. 8, pp. 931-939, Aug 1992.

7. R P Brent and H T Kung, "A regular layout for parallel adders", *IEEE Trans Computers*, vol. 31, no. 3, pp. 260-264, Mar 1982.

Gerard M Blair is a Senior Lecturer at the Department of Electrical Engineering, The University of Edinburgh, The King's Buildings, Edinburgh, EH9 3JL, Scotland, UK