Parallel Canonical Recoding

Ç. K. Koç

Electrical & Computer Engineering
Oregon State University, ECE 220
Corvallis, Oregon 97331, USA

Abstract

We introduce a parallel algorithm for generating the canonical signed-digit expansion of an n-bit number in $O(\log n)$ time using $O(n)$ gates. The algorithm is similar to the computation of the carries in a carry look-ahead circuit. We also prove that if the binary number $x + \lfloor x/2 \rfloor$ is given, then the canonical signed-digit recoding of x can be computed in $O(1)$ time using $O(n)$ gates.

1   Introduction

Recoding techniques (Booth recoding, bit-pair recoding, etc.) for sparse signed-digit representations of binary numbers have been effectively used in multiplication [3, 4] and exponentiation algorithms [2]. For example, the original Booth recoding technique [3, 4] scans the bits of the multiplier one bit at a time, and adds or subtracts the multiplicand to or from the partial product, depending on the value of the current bit and the previous bit. The modified versions of the Booth algorithm scan the bits of the multiplier two bits or three bits at a time [4]. These techniques are equivalent in the sense that a sparse radix-2 signed-digit representation of the multiplier is obtained, in which, three symbols $\{\bar{1}, 0, 1\}$ are allowed for the digit set.

The signed-digit representation is named canonical if it contains no adjacent nonzero digits [6, 3, 4]. The canonical signed-digit vector can be constructed using an algorithm of Reitwiesner [6] who has shown the canonical signed-digit vector for x is unique if the binary expansion of x is viewed as padded with an initial zero. The algorithm computes the signed-digit representation y starting from the least significant digit and proceeding to the left. First the auxiliary carry variable c_0 is set to 0 and subsequently the binary expansion of x is scanned. The canonically recoded digit y_i and the next value of the auxiliary binary variable c_{i+1} for $i = 0, 1, 2, \ldots, n-1$ are calculated using the values of x_i, x_{i+1}, and c_i according to Table 1.

For example, we compute the canonical recoding of $x = 478 = (0111011110)$ by starting with $c_0 = 0$, and then computing y_i and c_{i+1} using x_{i+1}, x_i, and c_i for $i = 0, 1, \ldots, 9$. The resulting vector is $y = 1000\bar{1}000\bar{1}0$. In this example the number x contains 7 nonzero bits while its canonically recoded version contains only 3 nonzero digits. Meanwhile, the original Booth's algorithm calculates the following signed-digit representation: $y' = (100\bar{1}1000\bar{1}0)$, which contains 4 nonzero digits. The canonical signed-digit vector y is optimal in the sense that it has the minimum number of nonzero digits among all signed-digit vectors representing the same number. Unfortunately, the canonical signed-digit recoding technique seems to have a deficiency. The computation, as suggested by Table 1, is not parallel. It has been noted in [4, page 104] that "the bits of the multiplier are generated sequentially", while in the original and modified Booth's algorithms we may generate the bits simultaneously, i.e., there is no carry propagation. However, we show here that the generation of the canonically recoded digits can be obtained in $O(\log n)$ time using $O(n)$ gates.

2 Parallel Canonical Recoding

The most time-consuming process in the computation of the canonical signed-digit representation of a given binary number is the computation of the 'carry' vector. If the carry vector c for a given x is already available, then the digits of y can be computed in $O(1)$ time using $O(n)$ gates. As can be seen from Table 1, $y_i = 1$ when $x'_{i+1}(x_i \oplus c_i)$ is true and $y_i = \overline{1}$ when $x_{i+1}(x_i \oplus c_i)$ is true. Since y_i takes one of the three values $\{0, 1, \overline{1}\}$, we need to use 2 bits to encode it. Let $u_i v_i$ denote these two bits encoding y_i such that $u_i v_i = 00$, $u_i v_i = 01$, and $u_i v_i = 10$ encode the values of y_i as 0, 1, and $\overline{1}$, respectively. It follows that $u_i = x'_{i+1}(x_i \oplus c_i)$ and $v_i = x'_{i+1}(x_i \oplus c_i)$. Therefore, the computation of the entire y vector requires $2n$ EXOR and $2n$ AND gates, and is accomplished in $O(1)$ gate delays.

We need to concentrate on fast computation of the carry vector in order to parallelize the canonical recoding algorithm. The carry c_{i+1} is computed using the values of x_{i+1}, x_i, and c_i. We write the simplified boolean equation for computing c_{i+1} from Table 1 as

$$c_{i+1} = x_i x_{i+1} + c_i x_i + c_i x_{i+1} = x_i x_{i+1} + c_i(x_i + x_{i+1}) \ . \tag{1}$$

Let $g_i = x_i x_{i+1}$ and $p_i = x_i + x_{i+1}$, then the above equation can be written as

$$c_{i+1} = g_i + c_i p_i \ . \tag{2}$$

This formulation immediately suggests that the elements of the carry vector can be computed in the same way as the computation of the carries in a carry look-ahead circuit. The main difference is in the definitions of the generate g_i and propagate p_i functions, and that $c_0 = 0$ in the beginning, which simplifies the circuit even further. For example, if we write the equations giving c_i for $i = 1, 2, 3, 4$, we notice that they are exactly in the same format as those for the carries in a carry look-ahead circuit:

$$
\begin{aligned}
c_1 &= g_0 + c_0 p_0 &&= g_0 \ , \\
c_2 &= g_1 + c_1 p_1 &&= g_1 + g_0 p_1 \ , \\
c_3 &= g_2 + c_2 p_2 &&= g_2 + g_1 p_2 + g_0 p_1 p_2 \ , \\
c_4 &= g_3 + c_3 p_3 &&= g_3 + g_2 p_3 + g_1 p_2 p_3 + g_0 p_1 p_2 p_3 \ .
\end{aligned}
$$

In order to compute the carry vector $(c_n, c_{n-1}, \ldots, c_1)$, we follow the formulation of Brent and Kung [1]. Let (α, β) denote an ordered pair of binary numbers and '$\bullet$' denote the operation on such pairs defined as

$$(\alpha_1, \beta_1) \bullet (\alpha_2, \beta_2) = (\alpha_1 + \beta_1 \alpha_2, \beta_1 \beta_2) \ . \tag{3}$$

It is easy to prove that this operation is associative [1]. We will use this operation on pairs $Q_i = (g_i, p_i)$, and prove that the computation of the elements of the carry vector can be reduced to a prefix computation on the vector of pairs $(Q_{n-1}, Q_{n-2}, \ldots, Q_0)$. Since $c_1 = g_0$, the carry c_1 is simply equal to the first element of the pair Q_0. Furthermore, we notice that

$$Q_1 \bullet Q_0 = (g_1, p_1) \bullet (g_0, p_0) = (g_1 + g_0 p_1, p_0 p_1) \ ,$$

i.e., the carry c_2 is found by calculating $Q_1 \bullet Q_0$, and then taking the first element from the resulting pair. Similarly, the carry c_3 is the first element of the resulting pair obtained from $Q_2 \bullet Q_1 \bullet Q_0$. Since $c_{i+1} = g_i + c_i p_i$ is the first element of the pair obtained from the computation

$$(g_i, p_i) \bullet (c_i, p_0 p_1 \cdots p_{i-1}) = (g_i + c_i p_i, p_0 p_1 \cdots p_{i-1} p_i) \ ,$$

2

it follows by induction that c_{i+1} is obtained as the first element of the resulting pair from the computation $R_i = Q_i \bullet Q_{i-1} \bullet \cdots \bullet Q_0$. Furthermore, the operation '$\bullet$' is an associative operation, and thus, we can use a parallel prefix (suffix) computation algorithm to obtain the prefix (suffix) products $R_0, R_1, \ldots, R_{n-1}$.

It is well known that the prefix product of n quantities can be performed in $\log(n)$ time using $O(n)$ nodes [5, 1], where the unit of time is determined by how long it takes to execute a '$\bullet$' operation. In our case, the '$\bullet$' operation uses 2 AND gates and 1 OR gate, and requires only 2 gate delays. Therefore, the computation of R_i for $i = 0, 1, \ldots, n-1$ requires only $O(n)$ gates and $2\log(n)$ gate delays. Once we compute R_i, we extract c_{i+1} from it by taking the first element. The canonical signed-digit vector y is then computed using the elements of the input x and the carry c vectors. As we have explained earlier, this computation requires $2n$ EXOR and $2n$ AND gates, and is accomplished in $O(1)$ gate delays. Therefore, we conclude that the canonical signed-digit representation of an n-bit binary number can be computed in $O(\log n)$ gates using $O(n)$ gates.

3   An Example

Here we show how to compute the canonical recoding of a 9-bit binary number using the Ladner-Fischer parallel prefix circuit [5]. The Ladner-Fischer parallel prefix circuit computes the prefix product of n quantities in d time using $4n - F(5 + d) + 1$ nodes, where $n = 2^d$ and $F(i)$ is the ith Fibonacci number defined by the recursion $F(i) = F(i-1) + F(i-2)$ with the initial conditions $F(0) = 0$ and $F(1) = 1$. It follows that for $n = 8$, we have $d = 3$, and thus the prefix circuit requires 3 units of time, and has 12 nodes. Given $(x_8, x_7, \ldots, x_0)$, we first compute the generate and propagate functions $g_i = x_i x_{i+1}$ and $p_i = x_i + x_{i+1}$ for $i = 0, 1, \ldots, 7$. This computation is accomplished using the circuit elements denoted by the hollow circles on the top of Figure 1. The pairs $Q_i = (g_i, p_i)$ are then applied to the 8-input Ladner-Fischer circuit. The parallel prefix circuit computes R_i for $i = 0, 1, \ldots, 7$ using the prefix nodes denoted by the filled circles in Figure 1. We then extract c_i for $i = 1, 2, \ldots, 8$ from R_i. The value of $c_0 = 0$ by definition. Finally, we compute y_i for $i = 0, 1, \ldots, 8$ using x_i, x_{i+1}, and c_i in the last level of the circuit using the hollow square boxes. The output y_i is a 2-bit number, and thus, the hollow square box is a 3-input 2-output combinational circuit. The darker lines in Figure 1 denote pairs of binary numbers.

4   A Property of Parallel Canonical Recoding

Suppose that we are interested in computing $a + b$ using a carry look-ahead adder, where $a = x$ and $b = \lfloor x/2 \rfloor$. In order to compute the sum vector s, the carry look-ahead adder computes the carries in advance using the carry look-ahead circuit. The generate and propagate functions of this addition operation are $g_i = a_i b_i = x_i x_{i+1}$ and $p_i = a_i + b_i = x_i + x_{i+1}$, respectively. Therefore, the generate and propagate functions obtained from this addition are exactly the same generate and propagate functions which would be computed by the parallel canonical recoding algorithm. Thus, we can use a carry look-ahead adder to compute parallel canonical recoding of a binary number. Once the carries are available, the computation of the canonical signed-digits requires only $O(1)$ time using $2n$ EXOR and $2n$ AND gates.

3

References

[1] R. P. Brent and H. T. Kung. A regular layout for parallel adders. *IEEE Transactions on Computers*, 31(3):260–264, March 1982.

[2] Ö. Eğecioğlu and Ç. K. Koç. Exponentiation using canonical recoding. *Theoretical Computer Science*, 129(2):407–417, 1994.

[3] K. Hwang. *Computer Arithmetic, Principles, Architecture, and Design.* New York, NY: John Wiley & Sons, 1979.

[4] I. Koren. *Computer Arithmetic Algorithms.* Englewood Cliffs, NJ: Prentice-Hall, 1993.

[5] R. Ladner and M. Fischer. Parallel prefix computation. *Journal of the ACM*, 27(4):831–838, October 1980.

[6] G. W. Reitwiesner. Binary arithmetic. *Advances in Computers*, 1:231–308, 1960.

Table 1: The canonical recoding.

x_{i+1}	x_i	c_i	y_i	c_{i+1}	Comments
0	0	0	0	0	string of 0s
0	0	1	1	0	end of 1s
0	1	0	1	0	a single 1
0	1	1	0	1	string of 1s
1	0	0	0	0	string of 0s
1	0	1	$\overline{1}$	1	a single 0
1	1	0	$\overline{1}$	1	beginning of 1s
1	1	1	0	1	string of 1s

Figure 1: An example of the parallel canonical recoding.