Open access • Journal Article • DOI:10.1109/TC.2004.14

Minimal weight digit set conversions — Source link

Braden J. Phillips, N. Burgess

Institutions: University of Adelaide

Published on: 01 Jun 2004 - IEEE Transactions on Computers (IEEE Computer Society)

Topics: Digit sum, Sliding window protocol and Binary number

Minimal Weight Digit Set Conversions

Braden Phillips, *Member*, *IEEE*, and Neil Burgess, *Member*, *IEEE*

Abstract—This paper considers the problem of recoding a number to minimize the number of nonzero digits in its representation, that is, to minimize the weight of the representation. A general sliding window scheme is described that extends minimal binary sliding window conversion to arbitrary radix and to encompass signed digit sets. This new conversion expresses a number of known recoding techniques as special cases. Proof that this scheme achieves minimal weight for a given digit set is provided and results concerning the theoretical average and worst-case weight are derived.

Index Terms—Digital arithmetic, redundant number systems, digit set conversion.

1 Introduction

Evaluation of arithmetic functions can be simplified by choosing an appropriate number representation. Radix or digit set can be selected to suit the characteristics of an algorithm or implementation technology. Such changes can achieve a number of benefits: The frequency of useful digits (such as zero) can be increased and the total number of digits required to represent a number can be reduced. The cardinality of the digit set can be reduced and this in turn may reduce the number of precomputed intermediate results to evaluate and store. Reduced cardinality also simplifies digit encoding for hardware implementation and increases the frequency of a given digit and, hence, the benefit available from precomputation of partial results.

It is usually necessary to trade these benefits one against the other. For example, increasing the radix usually reduces the number of digits required to represent a number at the cost of increased digit set cardinality.

Manipulation of number representation in this way is a fundamental technique in computer arithmetic. It provides an endless succession of publications as, for almost every change in implementation or algorithm, a different number representation becomes optimal. In many papers, the digit set conversion is implicit in the algorithm and not studied directly. Publications that deal with number representation in a general fashion are more rare.

The goal of this paper is to formalize a large class of number recoding techniques and provide general results concerning cardinality and average arithmetic weight. An algorithm for finding a representation of minimal arithmetic weight is presented and characterized. Even where a designer has no intention of implementing this algorithm directly, these results provide a useful upper bound to the benefit that can be expected from employing a digit set conversion.

- *B. Phillips is with the School of Electrical and Electronic Engineering, The University of Adelaide, Australia 5005. E-mail: Braden.Phillips@adelaide.edu.au.*
- *N. Burgess is with the Silicon Team, Icera Semiconductor, 2520 The Quadrant, Aztec West, Almondsburty, Bristol BS32 4AQ, UK. E-mail: neil.burgess@icerasemi.com.*

Manuscript received 8 Jan. 2003; revised 5 Aug. 2003; accepted 5 Nov. 2003. For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 118113.

A notation to describe the conversion of a number representation from one digit set to another is defined in Section 2. This notation and terminology provides the solid ground from which we proceed to survey existing digit set conversions in Section 3 and to examine in detail a general family of digit set conversions in Section 4.

1.1 Motivation

The focus in this paper is on digit set conversions that seek to minimize the number of nonzero digits in the representation of a number. Our aim is to provide theoretical results that are independent of a particular application or implementation technology. Nonetheless, the engineering audience will be concerned that these results are of more than theoretical interest. It is appropriate, therefore, that we consider possible applications before embarking on the general study.

Let us begin by considering the multiplication of two integers $A \times B$. A typical implementation forms a partial product $a_i B$ for each digit of the multiplier (a_i). The final result is formed from a sum of shifted partial products. Clearly, whenever a digit is equal to zero, there is one fewer partial product to generate and accumulate into the final result. Although thus varying the number of partial products is unlikely to be of any benefit for a general hardware multiplier, it is a useful technique for some software solutions [1] or for hardware when A is constant [2]. The latter case—a constant coefficient multiplier—occurs frequently in the implementation of digital filters. By reducing the number of nonzero digits in the representation of the constant coefficients, the hardware complexity is reduced and an improvement in clock speed may result.

The example of multiplication also exposes a trade off in choosing a number representation. One may consider precomputing a table of partial products $\{a_i B \,\forall\, a_i \in \mathcal{A}\}$. Techniques to reduce the number of nonzero digits a_i in the representation of A typically increase the cardinality of the digit set $\mathcal{A}$ and, hence, the size of the precomputed table and the effort required in precomputation.

Digit set conversion to reduce arithmetic weight has also been widely applied to exponentiation (for example, in [3], [4], [5], [6], [7], [8], [9]). To calculate A^B, most implementations require $\log_2 B$ squarings and then a multiplication for

every nonzero digit in B. Recoding the exponent B to reduce the number of nonzero digits can be used to reduce the number of multiplications.

The authors have also applied digit set conversion to reduce arithmetic weight for modular reduction [10] and for optimized squaring with precomputed partial products [11].

2 Formalization

In this section, we define a notation to express the operation of digit set conversion between positional number representations.

Let X be an n-digit positional *representation* of the integer $||X||$ using the digit set $\mathcal{X}$. We write:

$$X = (x_{n-1}, x_{n-2}, \ldots, x_0) \in \mathcal{X}^n$$

with

$$\mathcal{X} \subset \mathbb{Z} \text{ and } x_i \in \mathcal{X} \ \ \forall 0 \leq i < n$$

such that

$$||X|| = \sum_{i=0}^{n-1} x_i r_{\mathcal{X}}^i$$

for some radix $r_{\mathcal{X}} \in \mathbb{Z}$, $r_{\mathcal{X}} \geq 2$.

We may then define the following:

- A *digit set conversion* is a mapping $F : \mathcal{X}^n \rightarrow \mathcal{Y}^m$ such that, if $Y = F(X)$, then $||Y|| = ||X||$.
- A *fixed radix conversion* is a digit set conversion $F : \mathcal{X}^n \rightarrow \mathcal{Y}^m$ with $r_{\mathcal{X}} = r_{\mathcal{Y}} = r$.
- The *cardinality* of a digit set $\mathcal{X}$—the number of digits in the set—is denoted by $|\mathcal{X}|$.

Let us denote the set of representations of the integer i in set $\mathcal{X}^n$ as $\mathcal{V}_{\mathcal{X}^n}(i)$ according to:

$$\mathcal{V}_{\mathcal{X}^n}(i) = \{X : X \in \mathcal{X}^n \wedge ||X|| = i\}.$$

Thus, the number of representations of the integer i in the set $\mathcal{X}^n$ is $|\mathcal{V}_{\mathcal{X}^n}(i)|$. We also denote the set of all integers with representations in the set $\mathcal{X}^n$ as $\mathcal{D}_{\mathcal{X}^n} = \{||X|| : X \in \mathcal{X}^n\}$.

This notation allows us to make the following definitions:

- A digit set $\mathcal{X}$ is *complete* for n-digit representations of $\mathcal{D}$ if $\forall i \in \mathcal{D}\ \exists X \in \mathcal{X}^n$ such that $||X|| = i$.
- A digit set $\mathcal{X}$ is *redundant* for n-digit representations if $\exists i$ such that $|\mathcal{V}_{\mathcal{X}^n}(i)| > 1$.
- A digit set $\mathcal{X}$ is *nonredundant* for n-digit representations if $|\mathcal{V}_{\mathcal{X}^n}(i)| \leq 1\ \forall i \in \mathcal{D}$.

2.1 Examples

Some examples will help to clarify these definitions. Let us consider 3-digit representations ($n = 3$) in a digit set $\mathcal{X} = \{0, 1, 2, 3\}$ in radix 4 ($r_{\mathcal{X}} = 4$). One such representation is $X = (1, 0, 3)$ which has the integer value $||X|| = (1 \times 4^2) + (0 \times 4^1) + (3 \times 4^0) = 19$. It can be shown that this is the only representation of the value 19 in $\mathcal{X}^3$, that is, $\mathcal{V}_{\mathcal{X}^3}(19) = \{(1, 0, 3)\}$ and $|\mathcal{V}_{\mathcal{X}^3}(19)| = 1$.

It is well-known that the set of all integers with 3-digit representations in the set $\mathcal{X}$ is: $\mathcal{D}_{\mathcal{X}^3} = \{0, 1, 2, \ldots, 63\}$. We can, therefore, say that the digit set is *complete* for 3-digit representations of $\{0, 1, 2, \ldots, 63\}$. It is also possible to show that each of these integer values has only one representation in $\mathcal{X}^3$. Hence, we may say that $\mathcal{X}$ is *nonredundant* for 3-digit representations.

Let us take a second radix-4 digit set $\mathcal{Y} = \{0, 1, 2, 3, 4\}$ and imagine a digit set conversion F that maps representations in $\mathcal{X}^3$ onto representations in $\mathcal{Y}^3$. For example, it may be that, for $X = (1, 0, 3)$, we have $Y = F(X) = (0, 4, 3)$. Note that this conversion has preserved the arithmetic value of the representation: $||Y|| = ||X|| = 19$. Also note that $(1, 0, 3))$ and $(0, 4, 3)$ are both valid representations of the value 19 in $\mathcal{Y}^3$. In this case, we say that $\mathcal{Y}$ is redundant for 3-digit representations.

2.2 Comments on this Notation

Digit set conversion as stated above is sufficiently general to express a number of arithmetic representation schemes from existing literature. It does not exclude the possibility that the initial and final digit sets ($\mathcal{X}$ and $\mathcal{Y}$) are the same and that the digit set conversion is simply a recoding within that digit set. Similarly, it does not preclude conversions from digit sets in one radix to another. It is, however, assumed that a digit set is associated with a single radix—when we define the set $\mathcal{X}$, we must also state the radix $r_{\mathcal{X}}$. (Mixed radix conversions, for which this is not the case, are discussed briefly in Section 3).

2.3 Weight of a Representation

In a redundant number system, there may be more than one representation of a given algebraic value and those representations with the minimum number of nonzero digits are of particular interest. Subsequent sections place an emphasis on digit set conversions that decrease the frequency of nonzero digits. Let us now define this objective.

The number of nonzero digits in a number representation is variously called the Hamming weight, the arithmetic weight, or just the weight of that representation. Hamming weight is most often applied to the number of nonzero bits in a binary representation. Here, we will just use the term *weight* to emphasize that we may be dealing with the number of nonzero digits in a higher radix representation.

Let $C(Y)$ be the weight of the representation Y, thus:

$$C : \mathcal{Y}^m \rightarrow \mathbb{Z}$$

such that

$$C(Y) = \sum_{i=0}^{m-1} \delta(y_i),$$

where $\delta(y_i) = 1$ if $y_i \neq 0$ and $\delta(y_i) = 0$ otherwise.

We can then make the following definitions:

- The *minimum weight* of an integer $||Y||$ is:

$$M(||Y||) = min\{C(Y_i) : Y_i \in \mathcal{Y}^m \wedge ||Y_i|| = ||Y||\}.$$

- The *average weight* of a digit set conversion $F : \mathcal{X}^n \rightarrow \mathcal{Y}^m$ is:

$$\overline{C}(F) = \sum_{X \in \mathcal{X}^n} \frac{C(F(X))}{|\mathcal{X}^n|}.$$

TABLE 1
Booth Recoding

x_i	x_{i-1}	y_i
0	0	0
0	1	1
1	0	-1
1	1	0

- Call a representation Y a *minimal representation* if $C(Y) = M(||Y||))$.
- Call the digit set conversion $F : \mathcal{X}^n \rightarrow \mathcal{Y}^n$ a *minimal digit set conversion* if, for all $Y = F(X)$, Y is a minimal representation of $||X||$.

3 Existing Digit Set Conversions

This section presents a survey of digit set conversions from published literature, focusing on those that have been used to produce representations of reduced weight.

3.1 Booth Recoding

Booth recoding [12] converts a 2's complement binary number to a signed binary digit set $\{-1, 0, 1\}$. The recoded digits are selected from overlapping pairs of adjacent bits in the original multiplier according to Table 1. For example, the representation:

$$X = (1, 1, 0, 1, 1, 0, 1, 0, 1) \quad (1)$$

is converted to:

$$Y = (1, 0, -1, 1, 0, -1, 1, -1, 1, -1). \quad (2)$$

Booth conversion can reduce the weight of a representation [13]. This is due to the frequently cited observation that Booth conversion will replace strings of 1s by a string of zeros thus: $(0, 1, 1, 1, 1) = (1, 0, 0, 0, -1)$. However, Booth's is not a minimal recoding, as is evident from the example above in which the weight of the recoded form (2) is greater than that of the original (1).

If we consider the 2-bit scanning described by Table 1, it can be seen that, for large n, each row of the table is equally likely. Therefore, $y_i \neq 0$ is chosen half of the time. Half of the time an extra bit is required for $y_n = 1$. Hence, the average weight is $(n + 1)/2$ and, despite the elimination of strings of 1s, there is no improvement over nonredundant binary.

3.2 Modified Booth Recoding

Booth conversion proceeds by considering pairs of adjacent bits, with each pair overlapped by 1 bit. *Modified Booth recoding* [14] is the extension of this process to 3-bit groups overlapped by 1 bit. This is usually expressed as a conversion to the radix-4 digit set $\{-2, -1, 0, 1, 2\}$ according to the recoding rule in Table 2.

Overlapping groups of 4-bits to convert to a radix-8 digit set $\{-8, -6, -4, -2, 0, 2, 4, 6, 8\}$ also appears in [14] and there are many other examples of Booth conversion with various length groups. The authors of [15], [16] and others mention that the modified Booth technique (*s*-bit groups overlapped by 1 bit) can be extended to groups of any size. General treatments appear in [17] and [18] in which the criteria to be met by all correct *uniform overlapped multiple-bit scanning* techniques or *generalized multibit recoding* techniques are derived.

TABLE 2
Modified Booth Recoding

x_{2i+1}	x_{2i}	x_{2i-1}	y_i
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	2
1	0	0	-2
1	0	1	-1
1	1	0	-1
1	1	1	0

Although the original Booth algorithm does not improve the average weight, modified Booth conversion using 3-bit scanning converts 2 bits to a nonzero digit three-quarters of the time. For even n, an extra digit is required half the time (i.e., when $x_{n-1} = 1$). Thus, an average weight of $(3n + 4)/8$ is expected. To demonstrate that the conversion is not minimal, we may consider the conversion $X = (1, 0, 0, 1, 0, 0, 1, 0)$ to $Y = (1, -2, 1, 1, -2)$ and note that the alternative representation $Y = (0, 2, 1, 0, 2)$ has lower weight.

3.3 Further Modifications of Booth Recoding

Booth's original 2-bit scanning will convert the binary representation $(0, 1, 1, 1)$ to $(1, 0, 0, -1)$ but will also convert $(0, 1, 0, 1)$ to $(1, -1, 1, -1)$. Modified Booth with 3-bit scanning fails to be minimal when confronted with sequences of three bits $(0, 0, 1)$ as in the example above. It is possible to further improve the outcome by increasing the number of bits scanned. This is the idea behind the *recoded binary method* of [5] in which the signed bit y_i is determined by the four bits x_{i+1}, x_i, x_{i-1}, and x_{i-2}. The resultant binary representation Y has an average weight of $3n/8$ for large n. Unfortunately, this is no improvement over the 3-bit scanning above.

Having performed a binary conversion to improve weight, groups of adjacent bits can be combined to form higher radix digits. This leads to the *radix-4 string recoding* mentioned in [15] or the *recoded m-ary method* studied in [5].

The recoded *m*-ary method starts with the recoded binary method to improve weight. Then, *m*-bit digits are formed on regular *m*-bit boundaries. Following conversion with the recoded binary method, not all *m*-bit strings of signed bits can actually occur. This has the effect that the final representation contains digits in the set:

$$\{-2^{m-1}, \ldots, -2, -1, 0, 1, 2, \ldots, 2^{m-2} + 2^{m-1}\}.$$

For *n*-digit representations, the average weight following conversion is approximately $n(1 - 5/(2^{m+2}))$ [19].[1]

3.4 Minimal Binary Conversion

Redundant binary representation with the digit set $\{-1, 0, 1\}$, sometimes called *modified signed digit* representation (and often just called *signed digit* representation), does not exhibit a unique minimal form. Algorithms to generate a minimal representation are widely reported for both

1. Note that these results are incorrect in the original publication [5].

Binary Form	1	0	0	1	0	0	1	0	1	1	0	0	0	1	1	0
After Recoding	$y_3 = 9$ $z_3 = 0$				$y_2 = 1$ $z_2 = 1$				$y_1 = 3$ $z_1 = 2$				$y_0 = 3$ $z_0 = 1$			

Fig. 1. An example of the conversion from [26] in which groups of 4-bits form an odd digit y_i and offset z_i.

Binary Form X	0	1	1	0	0	0	1	1	1	1	0	0	1	0	1
After Recoding Y	0	0	3	0	0	0	1	0	0	7	0	0	0	0	5

Fig. 2. An example of sliding window conversion. The binary form is converted by grouping 3-bit windows, starting from the right and progressing to the left.

multiplication (see, for instance, [14], [15], [20]) and exponentiation (including [3], [6], [21]).

In [22], it was proven that the representation with no two adjacent digits being both nonzero was both minimal and unique for a given algebraic value. This representation is variously called the *canonic*, *sparse*, or *nonadjacent* form. For large n, an average weight of $n/3$ is expected [23]. The exact distribution of weights of recoded n-bit representations is derived in [24].

In the previous section, Booth recoded binary representations were converted to higher radix by grouping bits. The same technique can be applied to a canonic binary representation: a number in canonic binary form is converted to radix $r = 2^m$ by forming m-bit groups from adjacent bits. A specific instance of this appears in [15] in which the canonic binary form is converted to radix-4 by forming digits from adjacent pairs of bits. The resultant representation has digits from the set $\{-2, -1, 0, 1, 2\}$. The more general case in which digits are formed on regular m-bit boundaries has a target digit set:

$$\{-(2^{m+2} - (-1)^m - 3)/6, \ldots, -1, 0, 1, \ldots, (2^{m+2} - (-1)^m - 3)/6\}.$$

For n radix r digits and n large, the average weight is approximately $n(1 - 2^{2-m}/3)$ [19].

3.5 Minimal Higher Radix Conversion

In [25], the authors seek to extend the concept of the canonic binary form to higher radices. They define a minimal form called the *generalized nonadjacent form* (GNAF) using the digit set $\{-r + 1, \ldots, r - 1\}$. The minimal representation of a number $||Y||$ using this digit set is not unique, but there is only one representation, Y, which satisfies the extra conditions:

$$\text{if } y_i y_{i+1} < 0 \text{ then } |y_i| < |y_{i+1}|$$
$$|y_i + y_{i+1}| < r.$$

The authors provide an algorithm to convert from a representation using digits $\{-r + 1, \ldots, r - 1\}$ to the corresponding GNAF. This conversion involves propagation of information from the rightmost digit to the left.

The authors of [23] seek their own canonic form for the digit set $\{-r + 1, \ldots, r - 1\}$. They find another conversion algorithm (again propagating information to the left) and use Markov chain analysis to show that the expected value of the minimum weight is $n(r - 1)/(r + 1)$ for large n. Combinatorial techniques are used to find the probability distribution of minimum weights.

3.6 Sliding Window Algorithms

A conversion to the set of odd digits $Y = \{0, 1, 3, \ldots, 2^m - 1\}$ is implicit in the exponentiation scheme of [26]. To convert from binary, nonoverlapping groups of m-bits are considered. Each group forms a digit y_i and an offset z_i such that, if the original group had a value x_i, then $x_i = y_i 2^{z_i}$. Fig. 1 demonstrates the process.

It can be seen that this method forms digits from groups of adjacent bits or *windows*. The digits are separated by strings of consecutive zeros. This method does not take advantage of strings of zeros that do not appear on m-bit boundaries. A more flexible window method is demonstrated in [27]. Similar conversions are presented in [4], [7], [20], [28].

The process to convert from the digit set $\mathcal{X} = \{0, 1\}$ to $\mathcal{Y} = \{0, 1, 3, \ldots, 2^m - 1\}$ can be simply expressed. Starting with the least significant bit, x_0, skip over all bits equal to 0 until a bit equal to 1 is found. This bit and the following $(m - 1)$ bits form the odd digit y_0. The process then returns to skipping zeros until another digit, y_1, is found and so on. Fig. 2 shows an example.

The conversion to odd m-bit digits in [20] is dubbed an *adaptive m-ary segmentation*. In [7], it is called SS(m) and, in [29], it is shown to be a minimal conversion. The average weight is found to be approximately $n/(m + 1)$ with the approximation getting better for large n.

The *string replacement algorithm k-SR* in which binary numbers are converted to a representation using the odd digits less than or equal to k is studied in [28]. A *canonical k-SR* form is defined and the average weight for this conversion is derived, observing, however, that the canonical form is not always minimal. Note that SS(m) is a special case of *k-SR* for $k = 2^{m-1}$ and will always generate a minimal representation. The probability distribution of *k-SR* recoded representations is derived in [30].

A combination of sliding windows and canonical binary recoding is called *adaptive m-ary segmentation canonical recoding* in [20] and *width-m NAF* representation in [21]. A representation is first converted to binary canonic form and a sliding window is then used to group adjacent nonzero digits into odd digits. Hence, the target digit set contains 0 and the $(2/3)(2^m + (-1)^{m+1})$ odd digits with an m-bit canonical representation. For $m \geq 3$, this is $\pm\{0, 1, 3, \ldots, (2/3)(2^m + (-1)^{m+1}) - 1\}$. The recoding achieves an average weight for large n of $3n/(3m + 4)$.

3.7 Mixed Radix Algorithms

There are similarities between the sliding window algorithms above and the hybrid number systems of [8] and [9].

TABLE 3
Comparison between Some Mixed Radix and Sliding Window Conversions

Name	General sliding window	Target digit set	Digit set cardinality	Average weight	Worst case weight	Ref.
Hybrid binary-ternary number system		$\{0,1\}$	2	$0.338n$	n	[8]
Hybrid ternary-quinary number system		$\{0,1,2\}$	3	$0.324n$	$n\log_3 2$	[9]
Canonic binary signed digit	$SW_{2,1,-1,1}$	$\{-1,0,1\}$	3	$0.333n$	$n/2$	
Binary unsigned sliding windows $SS(2)$	$SW_{2,2,0,3}$	$\{0,1,3\}$	3	$0.333n$	$n/2$	

However, the former represent a number with redundant digits from a single radix, whereas the hybrid methods use two radices, selecting zero digits from the higher radix whenever possible and nonzero digits from the lower radix otherwise. A drawback of this approach is that conversion requires repeated divisions by the numbers 3 or 5.

Table 3 shows a comparison between the two mixed radix conversions and two other conversions (both of which are specific instances of the general sliding window algorithm described in Section 4). From this table, it can be seen that, for similar digit sets, the sliding window conversions produce comparable average weight and have the advantage of trivial conversion from binary.

3.8 Search, Compression, and Other Algorithms

Given a particular arithmetic value, one might fix a digit set and seek the minimum weight representation in that digit set; alternatively, one may set about to find a digit set that yields a low weight representation. This paper considers the former approach; the latter is studied in, for example, [4] and [27].

The conversion from [4] uses ideas from data compression to find patterns of bits in the binary representation of a value and groups repeated patterns into higher radix digits. A quick comparison between this and sliding window conversion can be made by considering their application to exponentiation. A typical exponentiation algorithm precomputes a table of digit powers A^{b_i}. During evaluation of A^B, a multiplication is required for every nonzero digit in the recoded exponent. Taking 1,024-bit exponentiation as an example, we find that the scheme in [4] requires an average of 68 multiplications and 8.5 squares in precomputation and 137 multiplications in evaluation. The binary unsigned sliding window conversion SS(6) (from Section 3.6—a specific instance of the general sliding window algorithm of Section 4) would require 31 multiplications and 1 square in precomputation and an average of 146.3 multiplications in evaluation.

More general comparisons between sliding windows and search or compression conversions are difficult to make. The former use a digit set comprising adjacent odd digits; the latter use a sparse digit set with very long digits built-up as patterns extended from shorter digits. The former use an $O(\log||X||)$ conversion procedure, whereas the latter may require complex searches to find a good conversion. The best conversion will depend on the target application.

Finally, it is worth noting that another approach is possible: One may abandon digit set conversion to a positional representation and yet still seek to decompose a number into a representation that facilitates efficient computation. Thus, in [31], an exponent is represented by an addition chain for efficient exponentiation and, in [2], a multiplier is factorized for efficient multiplication.

4 Generalized Sliding Windows

Let us consider, in very general terms, the complexity of minimal digit set conversion. According to [32], fixed radix conversion from any digit set with radix greater than 2, to a complete, redundant, contiguous digit set can take place in constant time. Does this still hold if the conversion must also be minimal?

Fig. 3 examines a fixed radix conversion to a complete, redundant, but noncontiguous digit set. Consider conversion of the least significant digit. For a correct conversion, $||Y|| \bmod r = ||X|| \bmod r$ and, thus, we must have $y_0 \bmod r = x_0 \bmod r$. This means that there are only two possible choices in $\mathcal{Y}$ for y_0 and Fig. 3 traces the implications of each decision. Note that the optimal choice of y_0 depends upon the values of an arbitrary number of digits to the left. Similarly, an optimal choice of y_m cannot be made without examination of all of the digits to the right.

The problem in Fig. 3 arises because of carry propagation due to the introduction of a negative digit. A similar situation can be constructed using positive digits and borrow propagation, as in Fig. 4.

An upper bound on the complexity of finding a minimal representation in these digit sets can be determined by considering a brute force approach in which the complete set of possible representations is enumerated. Beginning at the least significant digit, x_0, one can record each of the possible values for y_0. For each value of y_0, there will be a set of possible values for y_1 and, proceeding in this manner, one can find the set of representations $\mathcal{V}_{\mathcal{Y}^n}(||X||)$. If there are at most d possible values for each digit, then enumerating all representations is a process of complexity $O(d^n)$.

$r = 8$, $\mathcal{X} = \{0, 1, 2, 3, 4, 5, 6, 7\}$, $\mathcal{Y} = \{-6, 0, 1, 2, 3, 4, 5, 6, 7\}$

Consider $X = (2, 7, 7, 7, \ldots, 7, 2, 7, 7, 7, \ldots, 7, 2)$.

We can select $y_0 = -6$ in which case we must have $Y = (2, 7, 7, 7, \ldots, 7, 3, 0, 0, 0, \ldots, 0, -6)$

or we can select $y_0 = 2$ in which case we have $Y = (3, 0, 0, 0, \ldots, 0, -6, 7, 7, 7, \ldots, 7, 2)$

Fig. 3. A difficult digit set conversion. The optimal choice of the least significant digit depends on which string of 7s is longer. This decision can only be based on knowledge of all the digits rather than just a finite subset.

$r = 8$, $\mathcal{X} = \{0, 1, 2, 3, 4, 5, 6, 7\}$, $\mathcal{Y} = \{0, 1, 2, 3, 4, 5, 6, 7, 77\}$

where all values are shown in radix 10.

Consider $X = (1, 6, 1, 6, 1, 6, 0, \ldots, 0, 1, 5)$.

We can select $Y = X$,

or we can select $y_0 = 77$ in which case we must have $Y = (1, 6, 1, 6, 1, 5, 7, \ldots, 7, 0, 77)$

and hence can generate $Y = (\ldots, 0, 77, 0, 77, 0, 77, 7, \ldots, 7, 0, 77)$

Fig. 4. Another difficult digit set conversion. The optimal choice of the least significant digit depends on which is longer: the initial string of 0s or the recurring string of 1s and 6s.

The awkwardness of these two examples is due to their noncontiguous digit sets. However, as discussed in the following sections, there are some useful noncontiguous digit sets for which less computationally complex minimal conversions can be found.

4.1 A Generalized Sliding Window Algorithm

The sliding window algorithms of Section 3 can be seen as specific instances of a more general family of sliding window digit set conversions. Let us define a family of digit set conversions $SW_{r,m,l,u}$.

The digit set conversion $SW_{r,m,l,u} : \mathcal{X}^n \to \mathcal{Y}^{n+1}$ is a fixed radix-r conversion from digit set $\mathcal{X} = \{0, 1, 2, \ldots, r-1\}$ to the set of digits $\mathcal{Y} = \{y : l \le y \le u, y \neq 0 \bmod r\} \cup \{0\}$. The parameter m is the width of the sliding window and must be an integer greater than or equal to 1. This results in a target digit set of cardinality given by (3).

$$|\mathcal{Y}| = u - l + 1 - \lfloor u/r \rfloor - \lfloor (-l)/r \rfloor. \tag{3}$$

The lower and upper bounds on the target digit set, l and u, respectively, are subject to a number of conditions. The digit set must contain 0 and, for the representation of positive integers, it must contain 1. Also, for every window of m digits $(x_{i+m-1}, \ldots, x_{i+1}, x_i)$ and a carry bit $c \in \{0, 1\}$, the target digit set must contain either of the digits:

$$y_i = c + \sum_{j=0}^{m-1} x_{i+j} r^j \text{ or } y_i = c - r^m + \sum_{j=0}^{m-1} x_{i+j} r^j. \tag{4}$$

These considerations lead to the following conditions:

1. $1 - r^m \le l \le 0$,
2. $1 \le u \le r^m - 1$,
3. $|\mathcal{Y}| \ge r^m - r^{m-1} + 1$.

The proof of the minimality of SW given in the Appendix also requires:

4. $u \bmod r = -1$,
5. if $l < 0$, then $l \bmod r = 1$.

An algorithm for the SW conversion is given as pseudocode in Fig. 5. This begins by converting the least significant digit and proceeds to convert digits to the left. Zeros are skipped until a nonzero digit, x_i, is found: This and the following $m - 1$ digits form the digit y_i. At this point, SW checks if the digit set $\mathcal{Y}$ contains the digit y_i. If not, according to (4), $\mathcal{Y}$ must contain $y_i - r^m$ and this digit is used instead. The algorithm also checks x_{i+m}. If $x_{i+m} = r - 1$ and it is possible to set y_i negative, a carry is generated that will set y_{i+m} and, possibly, subsequent digits to zero. Table 4 shows some examples. Note that SW may generate a carry out that will require one extra digit to store.

The following sections derive some general results concerning the SW conversion. One result, the complexity, can be obtained immediately from the algorithm. We note that there is one iteration of the SW algorithm for each digit of X. From this, it is clear that the computational complexity of SW is $O(\log ||X||)$.

4.2 Minimality of SW

Theorem 1 below states the minimality of the SW conversion. Proof of this result is provided in the Appendix.

Theorem 1. *For all $X \in \mathcal{X}^n$, we have*

$$C(SW_{r,m,l,u}(X)) = M(||X||).$$

4.3 Average Weight of SW

The average weight of $SW_{r,m,l,u}$ can be determined by Markov analysis of the state diagram in Fig. 6 (following the procedure employed in [33]). Each state represents the selection of a single digit.

In Fig. 6, the value p corresponds to the probability of choosing $y_{i+m} = 0$ following the selection of some $y_i \neq 0$. To determine this value, we need to consider the selection of the nonzero digit that occurs on the transition from state 0 to state 1. Let us start by imagining the system is in state 0 and that the previous nonzero digit selected was greater than 0. In the algorithm of Fig. 5, this corresponds to the *carry* variable being zero. We will choose a nonzero digit for y_i if $x_i \neq 0$, i.e., in $r^{m-1}(r-1)$ possible cases. Of these, we will choose a positive digit such that $y_m = 0$ when $x = \sum_{j=0}^{m-1} r^j x_{i+j} \le u$ and $x_{i+m} = 0$. The number of cases with

```
/* Perform the conversion Y = SW_{r,m,l,u}(X) */
skip = 0, carry = 0
x_n = 0, x_{n+1} = 0, ..., x_{n+m} = 0
for i = 0 to n − 1
    if (skip = 0) then
        if (x_i + carry mod r = 0) then
            /* Skip over zero digits */
            y_i = 0
        else
            x = Σ_{j=0}^{m−1} x_{i+j} × r^j
            if (x + carry > u) then
                /* Must choose a negative digit */
                y_i = x + carry − r^m
                carry = 1
            elseif ((x_{i+m} = r − 1) and (x + carry ≥ l + r^m)) then
                /* Choosing a negative digit may reduce the weight */
                y_i = x + carry − r^m
                carry = 1
            else
                /* Choose a positive digit */
                y_i = x + carry
                carry = 0
            end if
            skip = m − 1
        end if
    else
        y_i = 0
        skip = skip − 1
    end if
next i
y_n = carry
```

Fig. 5. Pseudocode for the general sliding window digit set conversion $SW_{r,m,l,u}$.

TABLE 4
Examples of SW Conversion

A ($r_{\mathcal{A}} = 2$)		(1,	0,	1,	1,	1,	1,	1,	0,	1,	0,	0,	0,	1,	0,	1,	0)
$SW_{2,3,0,7}(A)$	(0,	0,	0,	5,	0,	0,	7,	0,	0,	5,	0,	0,	0,	0,	0,	5,	0)
$SW_{2,3,-7,7}(A)$	(0,	0,	3,	0,	0,	0,	0,	0,	0,	-3,	0,	0,	0,	0,	0,	5,	0)
$SW_{2,3,-1,5}(A)$	(0,	0,	3,	0,	0,	0,	-1,	0,	0,	5,	0,	0,	0,	0,	0,	5,	0)
B ($r_{\mathcal{B}} = 4$)		(1,	0,	0,	1,	2,	2,	0,	3,	3,	1,	1,	0,	3,	1,	1,	0)
$SW_{4,2,-15,15}(B)$	(0,	1,	0,	0,	0,	6,	0,	9,	0,	0,	-3,	0,	5,	0,	0,	-11,	0)
$SW_{4,2,-7,7}(B)$	(0,	1,	0,	0,	0,	7,	0,	-7,	0,	-1,	0,	5,	0,	3,	0,	5,	0)
C ($r_{\mathcal{C}} = 2$)		(1,	0,	0,	1,	0,	0,	1,	0,	0,	1,	0,	0,	1,	0,	0,	1)
$SW_{2,3,-5,5}(C)$	(0,	1,	0,	0,	1,	0,	0,	1,	0,	0,	1,	0,	0,	1,	0,	0,	1)
D ($r_{\mathcal{D}} = 2$)		(1,	0,	0,	0,	1,	0,	0,	0,	1,	0,	0,	0,	1,	0,	0,	1)
$SW_{2,3,-7,7}(D)$	(1,	0,	0,	0,	-7,	0,	0,	0,	-7,	0,	0,	0,	-7,	0,	0,	0,	-7)

$x \leq u$ given that $x_i \neq 0$ is $u - \lfloor u/r \rfloor$ and the probability of $x_{i+m} = 0$ is $1/r$. Therefore, the probability of choosing a positive digit $y_i > 0$ such that $y_{i+m} = 0$ is:

$$\Pr(y_i > 0 \cap y_{i+m} = 0 \mid x_i \neq 0 \cap carry = 0) = \frac{u - \lfloor u/r \rfloor}{r^m(r-1)}. \quad (5)$$

Similar considerations lead to the following probabilities:

$$\Pr(y_i < 0 \cap y_{i+m} = 0 \mid x_i \neq 0 \cap carry = 0) = \frac{-l - \lfloor -l/r \rfloor}{r^m(r-1)}, \quad (6)$$

$$\Pr(y_i > 0 \cap y_{i+m} = 0 \mid x_i \neq 0 \cap carry = 1) = \frac{u - \lfloor u/r \rfloor}{r^m(r-1)}, \quad (7)$$

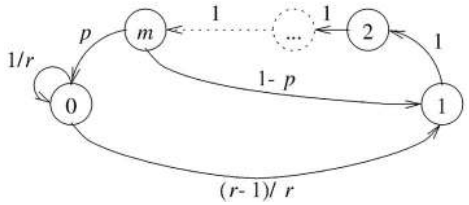

Fig. 6. State diagram for $SW_{r,m,l,u}$. State 0 corresponds to the selection of a zero such that the next digit may be nonzero. State 1 selects a nonzero digit. States 2 to m represent the selection of the $m-1$ zeros that must follow a nonzero digit.

$$\Pr(y_i < 0 \cap y_{i+m} = 0 \mid x_i \neq 0 \cap carry = 1) = \frac{-l - \lfloor -l/r \rfloor}{r^m(r-1)}. \tag{8}$$

Taking the sum of (5) and (6) or (7) and (8) yields the same result and, hence, the value of p is independent of the $carry$ variable:

$$p = \Pr(y_i \neq 0 \cap y_{i+m} = 0 \mid x_i \neq 0) = \frac{u - l - \lfloor -l/r \rfloor - \lfloor u/r \rfloor}{r^m(r-1)}. \tag{9}$$

Now, define $p_i(k)$ as the probability of being in state i after k digits. From Fig. 6, we can write:

$$\begin{aligned} p_0(k+1) &= \frac{1}{r}p_0(k) + p\, p_m(k), \\ p_1(k+1) &= \frac{r-1}{r}p_0(k) + (1-p)p_m(k), \\ p_i(k+1) &= p_{i-1}(k) + \frac{1}{r}p_m(k) \text{ for } 2 \leq i \leq m. \end{aligned}$$

These equations can be combined into a single matrix equation:

$$P(k+1) = QP(k), \tag{10}$$

where

$$Q = \begin{bmatrix} \frac{1}{r} & 0 & 0 & 0 & \dots & 0 & p \\ \frac{r-1}{r} & 0 & 0 & 0 & \dots & 0 & 1-p \\ 0 & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & 1 & 0 \end{bmatrix}$$

Taking the Z-transform of (10) and rearranging yields:

$$\mathbf{P}(z) = [zI - Q]^{-1} zP(0), \tag{11}$$

where $\mathbf{P}(z)$ is the Z-transform of $P(k)$. Equation (11) can be solved using row-reduction to perform the matrix inversion. As $k \to \infty$, the frequency of state i approaches $kp_i(k)$ (from the strong law of large numbers for Markov chains [23]) and the final value theorem for Z-transforms asserts that $\lim_{k\to\infty} p(k) = \lim_{z\to 1}(z-1)\mathbf{P}(z)$. Thus, we find that:

$$\lim_{k\to\infty} p_1(k) = \frac{1}{m + (pr)/(r-1)}. \tag{12}$$

There is a state transition for each of the n digits and the frequency of state 1 corresponds to the frequency of nonzero digits. So, the average weight for large n is:

$$\overline{C}(SW_{r,m,l,u}) \sim \frac{n}{m + (pr)/(r-1)}. \tag{13}$$

4.4 Worst-Case Weight

The worst-case weight for $SW_{r,m,l,u}$ occurs when every mth digit in the final representation is nonzero. The worst-case weight is therefore $\lceil n/m \rceil$. The conversion of C in Table 4 shows an example.

There is an exception for the case $r = 2$, $l = 1 - 2^m$, and $u = 2^m - 1$. In this instance, it is always possible to set the bit following a window to zero. The worst-case weight occurs when a window is formed every $m+1$ digits. The weight is then $\lfloor n/(m+1) \rfloor + 1$. The conversion of D in Table 4 shows an example.

5 Evaluation and Conclusions

5.1 Evaluation

Table 5 summarizes the features of many of the digit set conversions discussed in this paper and shows those that may be considered equivalent to a specific instance of the SW conversion (equivalent in that both are minimal conversions to the same digit set). Note that SW conversion expresses as special cases a number of widely used conversions (such as binary canonic form and unsigned sliding windows) as well as some more exotic extreme cases.

That SW conversion is minimal does not necessarily mean it will be the best choice for a given application. For example, in many circumstances, it would be better to choose a constant-time nonminimal conversion such as modified Booth than a logarithmic-time minimal SW conversion. Nevertheless, the results concerning SW conversion do provide a set of bounds according to which such design decisions can be made.

Where an SW conversion is used, the general results allow a designer to explore the trade off between digit set and weight. Such comparisons are considered in [1] and [10] which describe an implementation of the 1,024-bit RSA public key cryptosystem that makes extensive use of sliding window conversion.

The critical function of this system was to evaluate 512-bit modular powers and to do so on a RAM-constrained 32-bit microprocessor without a hardware multiplier or a long-wordlength hardware coprocessor. Sliding window digit set conversion was used for modular exponentiation, multiplication, modular reduction, and optimized squaring. The case of multiplication provides a useful example for the current discussion.

For a multiplication $B \times A$, a table of partial products $b_i A$ was precomputed. Then, to evaluate the product, an accumulation was required for every nonzero digit in the multiplier. Binary signed sliding windows $SW_{2,m,-2^m+1,2^m-1}$ were used to recode the multiplier. Note, however, that only the positive partial products $\{A, 3A, \dots, (2^m-1)A\}$ were precomputed as negative partial products can be handled by subtracting the corresponding positive partial product. That negative partial products can be handled in this way gives the signed conversion $SW_{2,m,-2^m+1,2^m-1}$ an advantage over the unsigned $SW_{2,m+1,0,2^{m+1}-1}$ which otherwise achieves the same average weight for a digit set of the same cardinality.

TABLE 5
Summary of Digit Set Conversions

Name	SW case	Target digit set	Average weight (for large n)	Ref.
Non-redundant radix r	$SW_{r,1,0,r-1}$	$\{0,1,\ldots,r-1\}$	$\frac{n(r-1)}{r}$	
Canonic binary signed digit	$SW_{2,1,-1,1}$	$\{-1,0,1\}$	$\frac{n}{3}$	[22]
Minimal signed digit	$SW_{r,1,-r+1,r-1}$	$\{-r+1,\ldots,r-1\}$	$\frac{n(r-1)}{r+1}$	[23], [25]
Binary unsigned sliding windows	$SW_{2,m,0,2^m-1}$	$\{0,1,3,\ldots,2^{m-1}+1$	$\frac{n}{m+1}$	[4], [7], [20]
Binary signed sliding windows	$SW_{2,m,-2^m+1,2^m-1}$	$\{1-2^m,\ldots,2^m-1\}$	$\frac{n}{m+1}$	[10]
Adaptive m-ary segmentation canonical	$SW_{2,m,-z,z}$ $z=(\frac{2}{3})(2^m+(-1)^{m+1})-1$ $m\geq 3$	$\pm\{0,1,3,\ldots,z\}$	$\frac{3n}{3m+4}$	[20], [21]
Generalised sliding windows	$SW_{r,m,l,u}$	$\{y\|y\in[l,u]\cap y\neq 0 \bmod r\}$ $\cup\{0\}$	$\frac{n}{m+(pr)/(r-1)}$ $p=\frac{u-l-\lfloor\frac{-l}{r}\rfloor-\lfloor\frac{u}{r}\rfloor}{r^m(r-1)}$	
Booth recoding		$\{-1,0,1\}$	$\frac{n}{2}$	[12]
Modified Booth		$\{-2,-1,0,1,2\}$	$\frac{3n}{2}$	[14]
Recoded m-ary method		$\{-2^{m-1},\ldots,2^{m-1}+2^{m-1}\}$	$n-\frac{5n}{2^{m+2}}$	[5]
Radix-r canonical		$\pm\{0,1,\ldots,\frac{2^{m+1}-(-1)^m-3}{6}\}$	$n(1-\frac{1-2^{2-m}}{3})$	[15], [19]
Canonical k-SR		$\{0,1,3,\ldots,k\}$	$\frac{n2^{k-2}}{2^k-1}+\frac{(2^k-k-1)2^{k-2}}{(2^k-1)^2}$	[28], [30]
Hybrid binary-ternary number system		$\{0,1\}$	$0.338n$	[8]
Hybrid ternary-quinary number system		$\{0,1,2\}$	$0.324n$	[9]

Fig. 7 shows the trade off between window length m and the total number of accumulations required for a 512-bit multiplication (including those required for precomputation). It is worth observing that a slightly suboptimal choice of m does not increase the number of operations dramatically but does reduce the size of the tables of precomputed results. (In the example, choosing $m=4$ rather than $m=5$ increases the average number of accumulations from 88.14 to 99.33, but reduces the number of precomputed partial products to store from 16 to 8.)

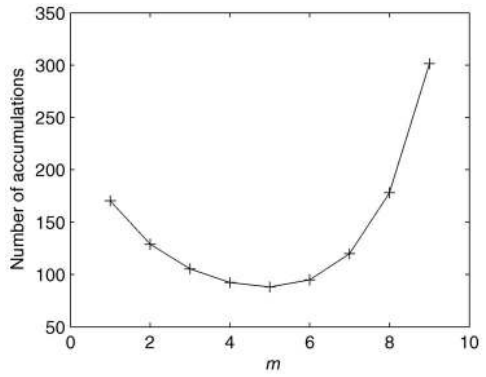

Fig. 7. Number of accumulation operations for 512-bit multiplication using $SW_{2,m,-2^m+1,2^m-1}$.

5.2 Conclusions

Although conversion to a redundant digit set can be accomplished in constant time, the examples at the beginning of Section 4 demonstrate that the conversion to a representation of minimal weight may be of exponential complexity. However, there is a large class of digit sets for which a minimal representation may be found using a sliding window conversion of logarithmic complexity. The SW conversion scheme described in this paper encompasses signed and unsigned digit sets at arbitrary radix and expresses a number of previously published sliding window conversions as special cases. Proof of the minimality of SW as well as general results concerning digit set cardinality and average weight have been provided.

Exploration of the SW parameters (radix, window length, lower and upper digit set bounds) exposes trade offs in the design of a system and allows designers to choose an optimal conversion for their particular application and implementation platform.

APPENDIX A

PROOF OF THE MINIMALITY OF SW

A.1 The Effect of Carry Propagation

To prove the minimality of $SW_{r,m,l,u}$, it will help to make some preliminary observations concerning the effect of injecting carry (or borrow) digits into a representation.

Lemma 1. *Given a representation A in the $SW_{r,m,l,u}$ digit set $\mathcal{A}^n$ and values $i \in [0, n)$ and $c \in [l/(r-1), u/(r-1)]$, it is possible to find a representation A' from the same digit set such that $||A'|| = ||A|| + cr^i$ and $C(A') \leq C(A) + 1$.*

Proof. The Algorithm given in Fig. 8 can be used to construct A' from A. Within the loop, the new digit $a'_i = a_i + c$ must satisfy one of the following four cases:

1. $a'_i \equiv 0 \bmod r$. In this instance, we set $a'_i = 0$, thereby reducing the weight of the representation by 1. To compensate for this change, a carry is propagated to the left. From $l \leq a_i \leq u$ and $l/(r-1) \leq c \leq u/(r-1)$, we have $l/(r-1) \leq a'_i/r \leq u/(r-1)$. So, the carry out is subject to the same bounds as the carry in and we may proceed to consider the effect of the carry out on the weight of the new representation.

```
/* Find A' = A + cr^i such that C(A') ≤ C(A) + 1 */
A' = A
while c ≠ 0
        a'_i = a'_i + c
        if a'_i mod r = 0 then          /* Case 1 */
                c = a'_i / r
                a'_i = 0
                i = i + 1
        else if a'_i > u then           /* Case 2 */
                c = 1
                a'_i = a'_i - r^m
                i = i + m
        else if a'_i < l then           / *Case 3 */
                c = -1
                a'_i = a'_i + r^m
                i = i + m
        else                            /* Case 4 */
                c = 0
        end if
end while
```

Fig. 8. Pseudocode for carry propagation in a representation using the SW digit set.

2. $a'_i > u$ and $a'_i \bmod r \neq 0$. We have $u < a'_i \leq u + u/(r-1)$ from which we may derive that $u > a'_i - r^m > l$. Also note that for, $a'_i > u$, we must have $a_i > 0$. We can therefore set a'_i to the valid digit $a'_i - r^m$. This change does not immediately effect the weight of the representation; however, to correct for this change, a carry of 1 is propagated into the digit a'_{i+m}. This carry propagation may have a subsequent effect on the weight.
3. $a'_i < l$ and $a'_i \bmod r \neq 0$. We have $l > a'_i \geq l + l/(r-1)$ from which we may derive that $u > a'_i + r^m > l$. Also note that, for $a'_i < l$, we must have $a_i < 0$. We can therefore set a'_i to the valid digit $a'_i + r^m$. This change does not immediately effect the weight of the representation; however, to correct for this change, a carry of -1 is propagated into the digit a'_{i+m}. This carry propagation may have a subsequent effect on the weight.
4. $a'_i \in \mathcal{A}$, in which case A' is a valid representation and the carry propagation will terminate. If $a'_i = 0$, then the weight has decreased by one; if $a'_i \neq 0$ and $a_i \neq 0$, then the weight is unchanged; or, if $a'_i \neq 0$ and $a_i = 0$, then the weight has increased by 1.

The weight of the representation is only increased when the carry terminates. Therefore, the injection of the carry can increase the weight of the representation by at most 1. □

Lemma 2. *Given a representation A in the $SW_{r,m,l,u}$ digit set $\mathcal{A}^n$ with $a_i \neq 0$ for some $i \in [0, n)$ and a value $c = \pm 1$, it is possible to find a representation A' from the same digit set such that $||A'|| = ||A|| + cr^i$ and $C(A') \leq C(A)$.*

The proof of Lemma 2 follows that of Lemma 1 with the additional observation that either the carry does not propagate (and the weight is unchanged) or, due to

conditions 4 and 5 in Section 4.1, it is possible to set $a'_i = 0$ (in which case the worst carry propagation can do is restore the weight to its original value).

A.2 The Minimality of SW

We are now ready to provide the proof for the minimality of $SW_{r,m,l,u}$. Recall that we wish to prove Theorem 1 which states that, for all $X \in \mathcal{X}^n$, we have

$$C(SW_{r,m,l,u}(X)) = M(||X||).$$

Proof. Let us assume the contrary and seek a contradiction. Assume that, for some X with $Y = SW_{r,m,l,u}(X)$, there exists another representation $Z \in \mathcal{Y}^n$ such that $||Z|| = ||Y||$ and $C(Z) < C(Y)$. Consider the following cases:

1. $y_0 = z_0 = 0$. Let $Y' = (0, y_{n-1}, \ldots, y_2, y_1) = R(Y)$, that is, Y, shifted right by 1 digit. Let $Z' = R(Z)$. Now, $||Y'|| = ||Z'||$, $C(Y') = C(Y)$, and $C(Z') = C(Z)$, so, without loss of generality, let us test the case of Y' and Z' instead.
2. $y_0 = z_0 \neq 0$. Let $Y' = (0, y_{n-1}, \ldots, y_2, y_1) = R(Y)$, that is, Y, shifted right by one digit. Let $Z' = R(Z)$. Now, $||Y'|| = ||Z'||$, $C(Y') = C(Y) - 1$, and $C(Z') = C(Z) - 1$, so, without loss of generality, let us test the case of Y' and Z' instead.
3. $z_0 = 0$, $y_0 \neq 0$. From $||Y|| = ||Z||$, we must have $||Y|| \bmod r = ||Z|| \bmod r$. Given $||Y|| \bmod r = y_0 \bmod r \neq 0$ for $y_0 \neq 0$ and $y_0 \in \mathcal{Y}$ and $||Z|| \bmod r = z_0 \bmod r = 0$, we conclude $||Y|| \bmod r \neq ||Z|| \bmod r$ and this contradicts the requirement that $||Y|| = ||Z||$.
4. $z_0 = y_0 - r^m$ and $(z_1, \ldots, z_{m-1}) = (0, \ldots, 0)$. From $||Y|| \bmod r^{m+1} = ||Z|| \bmod r^{m+1}$, we have $z_m r^m + z_0 = y_m r^m + y_0 + a r^{m+1}$ for some integer a. Substituting $z_0 = y_0 - r^m$ gives:

$$z_m = y_m + ar + 1. \tag{14}$$

Now, for $z_0 \in \mathcal{Y}$ condition 1 in Section 4.1 implies that $-r^m + 1 \leq z_0 = y_0 - r^m$ and, hence, $y_0 \geq 0$. In converting x_0 to y_0 with SW, y_0 has not been set negative and we can therefore conclude that $x_m \neq r - 1$. Proceeding with SW, we generate y_m from x_m by adding multiples of r. We must have $y_m \neq (-1) \bmod r$. We can now return to (14) and conclude that $z_m \neq 0$. This means that it is possible to set $z'_0 = z_0 + r^m = y_0$ and compensate for this change by subtracting 1 from z'_m and propagating any borrow to the left. According to Lemma 2 above, this process can only reduce the weight of Z' or leave it unchanged. Finally, $||Z'|| = ||Y||$, $C(Z') \leq C(Z)$, and $z'_0 = y_0$, so case 2 provides a contradiction. (Note that this case can only occur for $l < 0$. Therefore, the requirement that $l \bmod r = 1$ can be relaxed for $l = 0$.)
5. $z_0 = y_0 + r^m$ and $(z_1, \ldots, z_{m-1}) = (0, \ldots, 0)$. From $||Y|| \bmod r^{m+1} = ||Z|| \bmod r^{m+1}$, we have $z_m r^m + z_0 = y_m r^m + y_0 + a r^{m+1}$ for some integer a. Substituting $z_0 = y_0 + r^m$ gives:

$$z_m = y_m + ar - 1. \tag{15}$$

Now, for $z_0 \in \mathcal{Y}$, condition 2 in Section 4.1 implies that $r^m - 1 \geq z_0 = y_0 + r^m$ and, hence, $y_0 \leq 0$. In converting x_0 to y_0 with SW, y_0 has been set negative and we can therefore conclude that $x_m = r - 1$. Proceeding with SW, we will choose $y_m = 0$. We can now return to (15) and conclude that $z_m \neq 0$ for $r \geq 2$. This means that it is possible to set $z'_0 = z_0 - r^m = y_0$ and compensate for this change by adding 1 to z'_m and propagating any carry to the left. According to Lemma 2 above, this process can only reduce the weight of Z' or leave it unchanged. Finally, $||Z'|| = ||Y||$, $C(Z') \leq C(Z)$, and $z'_0 = y_0$, so case 2 provides a contradiction.
6. For all remaining possibilities with $z_0 \neq 0$ and $y_0 \neq z_0$, we will use a process that transforms Z into some Z' such that $||Z'|| = ||Z||$, $C(Z') = C(Z)$, and either $z_0 = y_0$ or $z_0 = y_0 \pm r^m$ and $(z_1, \ldots, z_{m-1}) = (0, \ldots, 0)$. We can then refer to one of the cases above for a contradiction.

We must have $||Y|| \bmod r^m = ||Z|| \bmod r^m$ and, hence,

$$\sum_{i=0}^{m-1} z_i r^i \bmod r^m = y_0 \bmod r^m$$

because, for SW, we have $y_i = 0$ for $0 < i < m$. This implies that, for some integer a,

$$z_0 - y_0 = ar^m - z_{m-1} r^{m-1} - z_{m-1} r^{m-2} \ldots - z_1 r.$$

To transform Z to Z', begin by setting $Z' = Z$ and proceed to examine each of the z'_i for $i = 1$ to $i = m - 1$. If z'_i is not zero, then $z_0 - y_0$ is a multiple of r^i. Set $z'_0 \leftarrow z'_0 + r^i z'_i$ and $z'_i \leftarrow 0$, thereby reducing the weight of Z' by 1. We have that $l + lr^i \leq z'_0 \leq u + ur^i$ and it can be shown that it is possible to choose an integer $\alpha \in [l/(r-1), u/(r-1)]$ such that $l \leq z'_0 - \alpha r^m \leq u$. We can therefore set z'_0 to a valid digit and correct for this change by propagating a carry into z'_m. According to Lemma 1, this carry may increase the weight of Z' by at most 1. Overall, the effect of setting $z'_i = 0$ either decreases the weight of Z' or leaves it unchanged.

When the transformation terminates, we have $z'_i = 0$ for $0 < i < m$ and, hence, $z'_0 - y_0 = ar^m$. Also $1 - r^m \leq z'_0 \leq r^m - 1$ and $1 - r^m \leq y_0 \leq r^m - 1$, so $-(2r^m - 2) \leq ar^m \leq 2r^m - 2$ and, hence, $z'_0 = y_0$ or $z'_0 = y_0 \pm r^m$ as required.

All six cases lead to a contradiction of the assumption that there exists a representation with a weight less than that produced by $SW_{r,m,l,u}$. Hence, the hypothesis that $SW_{r,m,lu}$ is minimal holds. The conversion $SW_{r,m,l,u}$ always produces a minimal representation for its target digit set. □

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers whose suggestions have made a significant contribution to the quality of this manuscript.

REFERENCES

[1] B.J. Phillips and N. Burgess, "Implementing 1,024-bit RSA Exponentiation on a 32-Bit Processor Core," *Proc. 2000 Int'l Conf. Application Specific Systems, Architectures, and Processors,* pp. 127-137, 2000.
[2] R. Bernstein, "Multiplication by Integer Constants," *Software Practice and Experience,* vol. 16, no. 7, pp. 641-652, 1986.
[3] J. Jedwab and C.J. Mitchell, "Minimum Weight Modified Signed-Digit Representations and Fast Exponentiation," *Electronics Letters,* vol. 25, no. 17, pp. 1171-1172, 1989.
[4] Y. Yacobi, "Exponentiating Faster with Addition Chains," *Proc. Advances in Cryptology (EUROCRYPT '90),* pp. 222-229, 1991.
[5] C.K. Koc, "High-Radix and Bbit Recoding Techniques for Modular Exponentiation," *Int'l J. Computer Math.,* vol. 40, nos. 3-4, pp. 139-156, 1991.
[6] C.N. Zhang, "An Improved Binary Algorithm for RSA," *Computers and Math. with Applications,* vol. 25, no. 6, pp. 15-24, 1993.
[7] L.C.K. Hui and K.Y. Lam, "Fast Square-and-Multiply Exponentiation for RSA," *Electronics Letters,* vol. 30, no. 17, pp. 1396-1397, 1994.
[8] V. Dimitrov and T. Cooklev, "Two Algorithms for Modular Exponentiation Using Nonstandard Arithmetics," *IEICE Trans. Fundamentals of Electronics, Comm., and Computer Sciences,* vol. E78-A, no. 1, pp. 82-87, 1995.
[9] C.Y. Chen, C.C. Chang, and W.P. Yang, "Hybrid Method for Modular Exponentiation with Precomputation," *Electronics Letters,* vol. 32, no. 6, pp. 540-541, 1996.
[10] B.J. Phillips and N. Burgess, "Signed Sliding Window Algorithms for Modulo Multiplication," *Electronics Letters,* vol. 36, no. 23, pp. 1925-1927, 2000.
[11] B. Phillips, "Optimised Squaring of Long Integers Using Precomputed Partial Products," *Proc. 15th IEEE Symp. Computer Arithmetic (ARITH15),* pp. 73-79, 2001.
[12] A.D. Booth, "A Signed Binary Multiplication Technique," *Quarterly J. Mechanics and Applied Math.,* vol. 4, pp. 236-240, 1951.
[13] C. Ghest, "Multiplication Made Easy for Digital Assemblies," *Electronics Letters,* vol. 44, pp. 56-61, 1971.
[14] O.L. MacSorley, "High-Speed Arithmetic in Binary Computers," *Proc. IRE,* vol. 49, pp. 91-103, 1961.
[15] K. Hwang, *Computer Arithmetic Principles.* Wiley, 1979.
[16] J.A. Starzyk and V.S.R. Dandu, "Overlapped Multi-Bit Scanning Multiplier," *Proc. IEEE Int'l Conf. Computer-Design (ICCD '85),* 1985.
[17] S. Vassiliadis, E.M. Schwarz, and D.J. Hanrahan, "A General Proof for Overlapped Multiple-Bit Scanning Multiplications," *IEEE Trans. Computers,* vol. 38, no. 2, pp. 172-183, Feb. 1989.
[18] H. Sam and A. Gupta, "A Generalized Multibit Recoding of Two's Complement Binary Numbers and Its Proof with Application in Multiplier Implementations," *IEEE Trans. Computers,* vol. 39, no. 8, pp. 1006-1015, Aug. 1990.
[19] B.J. Phillips, "An Optimised Implementation of Public-Key Cryptography for a Smart-Card Processor," PhD thesis, Univ. of Adelaide, 2000.
[20] C.K. Koc and C. Hung, "Adaptive m-Ary Segmentation and Canonical Recoding Algorithms for Multiplication of Large Binary Integers," *Computers and Math. with Applications,* vol. 24, no. 3, pp. 3-12, 1992.
[21] J.A. Solinas, "An Improved Algorithm for Arithmetic on a Family of Elliptic Curves," *Proc. Advances in Cryptology (CRYPTO '97),* pp. 3567-3571, 1997.
[22] G.W. Reitwiener, "Binary Arithmetic," *Advances in Computers,* vol. 1, pp. 261-265, 1960.
[23] S. Arno and F.S. Wheeler, "Signed Digit Representations of Minimal Hamming Weight," *IEEE Trans. Computers,* vol. 42, no. 8, pp. 1007-1010, Aug. 1993.
[24] L. O'Connor, "An Analysis of Exponentiation Based on Formal Languages," *Proc. Advances in Cryptology (EUROCRYPT '99),* pp. 375-388, 1999.
[25] W.E. Clark and J.J. Liang, "On Arithmetic Weight for a General Radix Representation of Integers," *IEEE Trans. Information Theory,* vol. 19, no. 6, pp. 823-826, 1973.
[26] H. Cohen and A.K. Lenstra, "Implementation of a New Primality Test," *Math. Computation,* vol. 48, no. 177, pp. 103-121, 1987.
[27] J. Bos and M. Coster, "Addition Chain Heuristics," *Proc. Advances in Cryptology (CRYPTO '89),* pp. 400-407, 1990.
[28] D. Gollmann, H. Yongfei, and C.J. Mitchell, "Redundant Integer Representations and Fast Exponentiation," *Designs, Codes, and Cryptography,* vol. 7, nos. 1-2, pp. 135-151, 1996.
[29] K.Y. Lam and L.C.K. Hui, "Efficiency of SS(I) Square-and-Multiply Exponentiation Algorithms," *Electronics Letters,* vol. 30, no. 25, pp. 2115-2116, 1994.
[30] L.J. O'Connor, "On String Replacement Exponentiation," *Designs, Codes, and Cryptography,* vol. 23, no. 2, pp. 173-183, 2001.
[31] D.E. Knuth, *The Art of Computer Programming, Volume 2, Seminumerical Algorithms.* Addison-Wesley, 1997.
[32] P. Kornerup, "Digit-Set Conversions: Generalizations and Applications," *IEEE Trans. Computers,* vol. 43, no. 5, pp. 622-629, May 1994.
[33] H. Freeman, "Calculation of Mean Shift for a Binary Multiplier Using 2, 3, or 4 Bits at a Time," *IEEE Trans. Electronic Computers,* vol. 16, no. 6, pp. 864-866, 1967.

Braden Phillips received the PhD degree from the University of Adelaide, Australia, in 2000 concerning the implementation of public-key cryptography on smart cards. Prior to the submission of his thesis, he worked as a process control engineer and was a founding partner in Current Dynamics, an embedded system design venture. In September 2000, he took up a lecturing position at Cardiff University in South Wales, a post he held for two years before returning to lecture at the University of Adelaide. His research interests include digital arithmetic, VLSI, and information security. He is a member of the IEEE and the IEEE Computer Society.

Neil Burgess received the PhD degree from Southampton University (United Kingdom) in 1986 for his work on graph theoretic techniques to model failure mechanisms in MOS VLSI circuits. He then spent two years designing DSP VLSI chips at GEC's Hirst Research Labs in Wembley, London, before moving to academia to lecture in microelectronic systems, first at Brunel University, London, then for six years at the University of Adelaide, Australia. In 1996, he was appointed director of the University of Adelaide's Centre for High-Performance Technology and Systems, whose major research activity was centered on the design of high-performance digital microelectronic circuits. In 1999, he returned to the United Kingdom to take up a Professorial Research Fellowship in the Division of Electronics at Cardiff University's School of Engineering. His research activities in these posts have revolved around digital VLSI design in both CMOS and GaAs and high-speed arithmetic processing, expanding this work to address applications including cryptography, image compression, and DSP. In October 2003, he left the academic sector to join Icera Semiconductor, a silicon start-up company whose design office is in Bristol, United Kingdom. He has published more than 80 papers and is an associate editor of the *IEEE Transactions of Computers*. He cochaired the 14th and 15th IEEE Symposia on Computer Arithmetic (in 1999 and 2001), the 13th IEEE International Conference on Application-Specific Systems Architectures and Processors (2002), and the 37th IEEE Asilomar Conference on Signals, Systems and Computers (2003). He is a member of the IEEE.