Stored-transfer representations with weighted digit-set encodings for ultrahigh-speed arithmetic

G. Jaberipur and B. Parhami

Abstract: Redundant representations play an important role in high-speed computer arithmetic. One key reason is that such representations support carry-free addition, that is, addition in a small, constant time, independent of operand widths. The implications of stored-transfer representation of digit sets and the associated addition schemes, as an extension of the stored-carry concept to redundant number systems, on the speed and cost of arithmetic algorithms, are explored. Two's-complement digits as the main part and any two-valued digit (twit) in place of a stored carry are allowed, leading to further broadening of the generalised signed-digit representations. The characteristics of the digit sets, possibly not having zero as a member, that allow for most efficient carry-free addition, are investigated. Circuit speed is gained from storing or saving, instead of combining through addition, the interdigit transfers generated during the carry-free addition process. Encoding efficiency is gained from using a twit-transfer set encoded by one logical bit, where more bits would otherwise be needed to represent a transfer value.

1 Introduction

A positional radix-r number system is deemed redundant if the cardinality of its digit set is greater than r; for example, decimal digit set $\{0, 1, 2, \ldots, 9, 10, 11\}$, or binary digit set $\{-1, 0, 1\}$ [1]. In modern digital circuits, redundancy is commonly introduced in number representation with the aim of improving the speed or efficiency of arithmetic operations [2, 3].

One reason for speed improvement with redundancy is the possibility of carry-free addition; that is, addition in a small, constant time, independent of operand widths. This desirable property is routinely exploited in digital system designs where internal redundant representations (invisible to the casual user) are employed, although explicit use of redundant representations is also gaining in popularity [4–6]. In carry-free addition, a carry produced by one-digit position is always absorbed by the next higher position [1]. Because the term 'carry' often conjures the notion of a propagating signal, the information produced by one stage and absorbed by the next stage is referred to as a 'transfer' value or digit.

Another reason for the usefulness of redundant representations is that redundancy allows some imprecision in the decision processes associated with arithmetic algorithms (such as quotient or root digit selection [7]); this tolerance to imprecision removes enough complexity from the computation's critical path to yield significant performance improvement. As a specific example, quotient digits in high-radix division can be chosen by inspecting only a few bits of the divisor and of the redundant partial remainder, thus allowing a much shorter cycle time via hardwired or tabular implementation of the quotient digit selection process [8].

Contributions to redundant number representation are of two main types. In abstract studies (e.g. [1, 9, 10]), arithmetic algorithms are presented in terms of digit-level operations, specifying how each result digit is derived from operand digits and auxiliary quantities such as interdigit transfers. Implementation-oriented studies, on the other hand, are often based on specific encodings for digit sets encountered in the course of solving particular design problems; for example, implementation of a high-speed two's-complement full-tree multiplier [11]. Some contributions of this latter type have dealt with limited classes of digit-set encodings without directly associating them with a specific design problem. Falling into the latter category are hybrid-redundant number systems [12, 13] and representation paradigms of high-radix signed digit numbers [14].

Certain weighted encodings of redundant number systems, using two-valued digits (twits, or generalised bits) have been recently shown to fill the gap between the aforementioned contributions [15]. Twits are exemplified by a unibit that represents a value in $\{-1, 1\}$, with -1 encoded as logical 0 and 1 encoded as logical 1. Such encodings lead to efficient representations for redundant number systems and make it possible to realise arithmetic circuits based on widely available optimised full/half adders, compressors, and carry acceleration cells. Furthermore, they allow faithful representation of digit sets, including those not having zero as a member, leading to enhanced encoding efficiency in some cases. Here, we focus on stored-transfer representation of redundant digit sets [16], as an instance of weighted twit-set (WTS) encodings [15], study the implications of twit transfers, and adapt the conventional carry-free addition algorithm to stored-transfer representation of general digit sets that may exclude 0 (e.g. digit values extending from 3 to 12).

doi:10.1049/iet-cds:20050228

Paper first received 10th August 2005 and in revised form 23rd May 2006

G. Jaberipur is with the Department of Electrical and Computer Engineering, Shahid Beheshti University, Tehran 1983963113, Iran and also with the School of Computer Science, Institute for Studies in Theoretical Physics and Mathematics

B. Parhami is with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106-9560, USA

E-mail: parhami@ece.ucsb.edu

Conventional carry-free addition of radix-r operands x and y, whose digits x_i and y_i belong to the redundant digit set $\Delta = [\alpha, \beta]$, is described as follows (see Fig. 1*a*):

Algorithm 1 (Conventional carry-free addition to compute $s = x + y$): Perform the following digit operations for all positions i ($0 \le i < k$) concurrently:

1. Compute the position sum $p_i = x_i + y_i$.
2. Derive the interim sum digit w_i and transfer digit t_{i+1} satisfying $w_i = p_i - rt_{i+1}$.
3. Form the final sum digit $s_i = w_i + t_i$.

If step 3 of Algorithm 1 is to yield a valid digit s_i in Δ, without producing further transfers, the interim sum digit w_i must be restricted in $D = [a, b]$, satisfying for all possible values of t_i

$$\alpha - t_i \le a < b \le \beta - t_i$$

We derive bounds on possible values of t_i, and the necessary and sufficient number of these values for carry-free addition to be applicable, in Section 2. Note that the digit-size additions of steps 1 and 3, though quite fast compared with word-size additions required with nonredundant representations, are merely used for algorithm description and need not be explicitly performed in hardware. The addition in step 1 can be avoided, for example, by noting that w_i and t_{i+1} are directly computable in hardware as functions of x_i and y_i (see Fig. 1*a*), that is

$$w_i = \omega(x_i, y_i); \quad t_{i+1} = \tau(x_i, y_i)$$

This, in effect, fuses steps 1 and 2 and allows the designer to choose the best possible merged implementation. It may be the case, with certain digit sets and/or encodings, that some form of addition is still part of the best hardware implementation scheme for ω and τ, but this is not required. We are thus motivated to investigate methods for eliminating, or else simplifying, the addition in step 3. In Section 3 we discuss stored-transfer representation of redundant number systems [16] and adapt Algorithm 1 to such numbers. In Section 4 we study the implications of limiting the transfer set to one twit. Section 5 offers an efficient realisation of single-twit stored-transfer adder based on the SUT adder of [15] and compare its performance with other redundant adders. Conversion from/to two's-complement representation is taken up in Section 6.

2 Transfer values in carry-free addition

In this section general characteristics of the interdigit transfer process are studied and bounds on transfer values within the framework of carry-free addition (Algorithm 1) are derived. The results obtained allow for the number of possible transfer values in steps 2 and 3 of Algorithm 1 to be kept to a minimum, leading to more efficient implementations. In this study, two practical restrictions imposed in [17] are relaxed; that is, we deal with digit sets that do not necessarily include zero as a member, and non-contiguous transfer sets are allowed. Proofs for formal results in this section are to be found in the Appendix.

Definition 1 (Characteristics of the digit set): Let r, $\Delta = [\alpha, \beta]$, and $\rho = \beta - \alpha + 1 - r$ denote the radix, digit set (interval of integer values, possibly excluding 0), and redundancy index of the number system, respectively. We postulate $r \ge 2$, $\alpha < \beta$, and $\rho > 0$. □

Definition 2 (Characteristics of the transfer set): Let $T = \{c_0, c_1, \ldots, c_{d-1}\}$ denote the set of d possible transfer values, listed in ascending order, that can be chosen in step 2 of Algorithm 1. The span of T is $\delta = c_{d-1} - c_0$ and its jth gap, $0 \le j < d - 1$, is $\delta_j = c_{j+1} - c_j \ge 1$. □

Lemma 1 (Bounds on c_0, c_{d-1}, and δ_j): For carry-free addition (Algorithm 1) to be applicable, we must have $c_0 \le \lfloor \alpha/(r-1) \rfloor$, $c_{d-1} \ge \lceil \beta/(r-1) \rceil$, and $\delta_j \le 1 + \lfloor (\rho - \delta)/r \rfloor$.

In Lemma 1, the focus is on the upper (lower) bound for c_0 (c_{d-1}) because these bounds determine the smallest span and consequently the minimum cardinality of transfer values. Note that, in general, the larger the number of choices for the transfer value, the greater the complexity of the circuit that must make the selection and the higher its latency.

Lemma 2 (Minimal span $\delta_{\min}$): The minimal span of transfer values in T is bounded from below by $1 + \lceil \rho/(r-1) \rceil$ and from above by $\min(\rho, 3 + \lfloor (\rho - 2)/(r-1) \rfloor)$. □

Lemma 2 indicates that the minimal span is in the following range:

$$1 + \lceil \rho/(r-1) \rceil \le \delta_{\min} \le \min(\rho, 3 + \lfloor (\rho - 2)/(r-1) \rfloor)$$

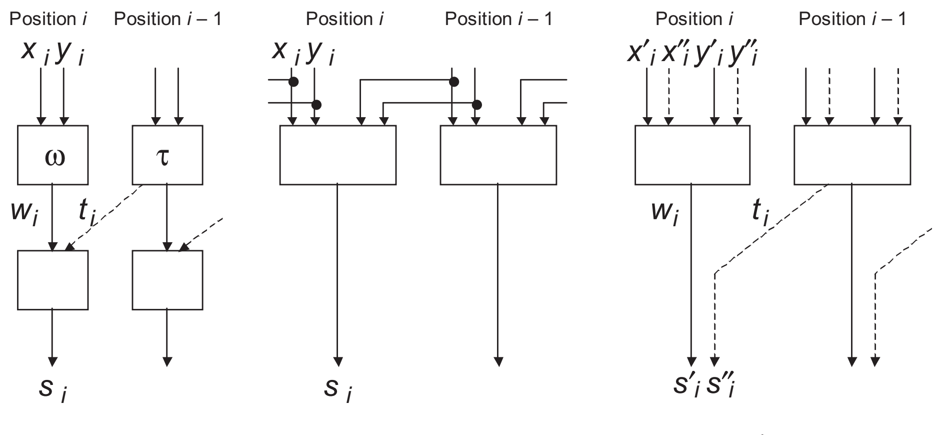

Fig. 1 *Carry-free addition schemes of Algorithms 1 and 2*
a Algorithm 1: Ordinary carry-free addition, two-stage and single-stage realisations
b Algorithm 2: Stored-transfer addition

Thus, whether ρ is small (so that the upper bound above is ρ) or relatively large (so that the upper bound is $3 + \lfloor(\rho - 2)/(r - 1)\rfloor$), the variation in $\delta_{\min}$ is quite limited.

Corollary 1 (Practical values for δ): For $r \geq 3$ and $\rho = 2$, $\rho \in [3, r-1]$, or $\rho = r$, the minimal span of transfer values is $\delta_{\min} = 2$, $\delta_{\min} \in [2, 3]$, or $\delta_{\min} = 3$, respectively. □

Theorem 1 (Applicability of carry-free addition): For the carry-free addition of Algorithm 1 to be applicable, it is necessary and sufficient to have $v_1 + v_2 < \min((r-2)(\rho-1), 2r-3)$, where $v_1 = \alpha \bmod (r-1)$ and $v_2 = (-\beta) \bmod (r-1)$. □

Note that the parameters v_1 and v_2 are also used in the proof of Lemma 2 in the Appendix. One consequence of Theorem 1 is the following generalisation of a previously known result regarding carry-free addition for generalised signed-digit (GSD) numbers.

Corollary 2 (Lower bounds for r and ρ): The requirement $(r-2)(\rho-1) > v_1 + v_2$ of Theorem 1 cannot hold for $\rho < 2$ or for $r < 3$. Therefore both $\rho \geq 2$ and $r \geq 3$ are necessary conditions for carry-free addition using Algorithm 1. □

Theorem 2 (Insufficiency of $\rho = 2$ for carry-free addition): Algorithm 1 is inapplicable for a digit set $[\alpha, \beta]$ that satisfies $\rho = 2$ and $\alpha \bmod (r-1) = r-2$. □

Corollary 3 (Insufficiency of $\rho = 2$ in the special case of GSD): For ordinary generalised signed-digit number systems, where the digit set $[\alpha, \beta]$ includes zero as a member, $\rho = 2$ is insufficient for carry-free addition for two digit sets: (1) $\alpha = -1, \beta = r$; (2) $\alpha = -r, \beta = 1$.

Corollary 3 affirms the results first reported in [17]. The following two results are also generalisations of the corresponding results for generalised signed-digit representations.

Theorem 3 (Sufficiency of $\rho \geq 3$ for carry-free addition): The condition of Theorem 1 for carry-free addition is satisfied by all possible digit sets $[\alpha, \beta]$ having $\rho = \beta - \alpha + 1 - r \geq 3$. □

Theorem 4 (Minimal cardinality d of transfer set in carry-free addition): A minimal set T of possible transfer values in Algorithm 1 is at least three-valued; for $\rho \leq r$, it is at most four-valued.

The preceding results contain both bad news and good news. The bad news is that the transfer value cannot be represented by a single bit, thus forcing us to use two bits for its binary encoding. The good news is that, in virtually all practical cases, we do not need to go beyond two bits in representing the transfer values.

Corollary 4 (Contiguity of the transfer set for $\rho = 2$): In the case of $\rho = 2$, provided the conditions of Theorem 2 are met, the minimal transfer set always constitutes a three-valued interval of integers. □

Corollary 5 (Maximal encoding efficiency): Given $2 \leq \rho \leq r$ and $r = 2^h$, a redundant radix-r digit set with at most $2r$ members and the transfer values for carry-free addition of Algorithm 1 can be encoded by $h+1$ and two bits (i.e. the minimum possible for both), respectively. □

In the following sections, the practical consequences of the preceding results are explored, leading to the particularly efficient implementations in Section 5.

3 Stored-transfer representations

In a manner similar to stored-carry or carry-save representation of binary numbers [2], a study is carried out of the implications of stored-transfer or transfer-save representations of redundant digits, where the pair (w_i, t_i) in the carry-free addition of Algorithm 1 is viewed as an encoding of the sum digit s_i. This interpretation obviates the need for the final addition $s_i = w_i + t_i$ in step 3 (w_i is the main part and t_i the transfer part of a digit's stored-transfer encoding).

Definition 3: (Stored-transfer encoding): The stored-transfer encoding of a redundant digit set $\Delta = [\alpha, \beta]$ is based on a main digit set $D = [a, b]$ and a transfer set $G = \{c_0, c_1, \ldots, c_{d-1}\}$, such that any digit $z \in \Delta$ has a stored-transfer encoding $\langle z', z''\rangle$, with $z = z' + z''$, $z' \in D$, and $z'' \in G$. Primed and double-primed variables are used to designate the main and transfer parts of a digit. □

Example 1: A main part that is a 4-bit two's-complement number in $[-8, 7]$ and a four-valued stored transfer in $[-1, 2]$ constitute a 6-bit faithful encoding of the digit set $[-9, 9]$. Direct encoding of the digit set would require five bits, where the encoding is not faithful. □

The stored-transfer representation of definition 3 leads to a two-step formulation of carry-free addition, as described by Algorithm 2 and depicted in Fig. 1*b*.

Algorithm 2 (Stored-transfer addition to compute $s = x + y$): Perform the following digit operations for all positions i $(0 \leq i < k)$ concurrently:

1. Compute the position sum $p_i = x'_i + x''_i + y'_i + y''_i$.
2. Derive $s_i = \langle s'_i, s''_i\rangle$ satisfying $s'_i = p_i - rs''_{i+1}$.

Of course, steps 1 and 2 in this new two-step process can again be fused, in the manner previously outlined for Algorithm 1, leading to a merged or single-step implementation

$$s'_i = \sigma'(x'_i, x''_i, y'_i, y''_i); \quad s''_{i+1} = \sigma''(x'_i, x''_i, y'_i, y''_i)$$

Note that the transfer set $G = \{c_0, c_1, \ldots, c_{d-1}\}$, satisfying $c_0 < c_1 < \cdots < c_{d-1}$, is d-valued but does not necessarily contain a set of d consecutive integers. This more general view is taken in anticipation that it may provide added flexibility for optimisations. It can be seen later that even though such generalised transfer sets do not provide additional benefits directly, they can be used with minor modifications to the carry-free addition algorithm. On the other hand, the main part of a digit belongs to an interval $D = [a, b]$ of values. Whereas gaps in this set are also admissible, provided that the set contains one member from each of the r residue equivalence classes $j \bmod r$ $(0 \leq j \leq r-1)$, this generality has not been found to yield any speed or cost benefit.

An objection may be raised that Algorithm 2 simply shifts the complexity of the original step 3 in Algorithm 1 to the new step 1. The fact that this is not the case will become apparent when the methods employed in this paper are explained in more detail. Here, it is argued that the new scheme can, in principle, be faster than that of Algorithm 1. For instance, a four-operand addition, where two of the operands (transfer parts) are fairly small, can indeed be faster and less complex than two separate additions [18]. For another, the stored-transfer representation $\langle z', z'' \rangle$ may well contain the same total number of bits as the binary encoding of z [15]. In such a case, the function pairs (ω, τ) of Section 1 and (σ', σ'') of this section have comparable bit-level complexities.

Example 2: Stored-transfer representations of some redundant number systems appear in Table 1. The hybrid signed-digit entries (lines 6 and 7) use the radix $r = 2^h$. Note that even though not all entries in Table 1 are practically useful, they have been included in the list to demonstrate the generality of the results. It is the belief of the authors that such generality is desirable and must be pursued whenever it does not interfere with the clarity of presentation for more practical cases. One important reason for this viewpoint is the fact that general results ensure that no important special case has been overlooked by imposing arbitrary restrictions based on current practice or implementation technologies. □

The transfer sets of entries 6, 10, and 12 in Table 1 are two-valued and thus representable by a single bit. However, by Theorem 4, the cardinality d of the transfer set must be at least three for carry-free addition to be possible. Moreover, the noncontiguous two-valued transfer sets $\{-1, 1\}$ and $\{2, 4\}$ of entries 4, 9, and 12 do not satisfy the result of corollary 4, stating that for carry-free addition with $\rho = 2$, the transfer set must contain a three-valued interval of integers; that is, it must consist of three consecutive integers. In Section 4, where an implementation of stored-transfer addition is presented, a simple design modification is used to deal with these two problems. Note that in designing a stored-transfer encoding, the transfer set G used should preferably be of the minimum size prescribed by Theorem 4. Extra values in G, though they offer small advantages in extending the range of the digit set, degrade the encoding efficiency and increase implementation costs (i.e. latency, area and power). Furthermore, the wider digit set may not be preserved under carry-free addition, thus nullifying any accrued benefits.

Because a four-valued G is always sufficient by Theorem 4, our stored-transfer representations need at most two bits of redundancy per digit compared with binary encoding of the nonredundant digit set $[0, r-1]$. Virtually all practical redundant representations use power-of-two radices and thus imply at least one bit of redundancy. Therefore the incremental cost of the proposed scheme, in its initial form, and without the enhancement to be covered in Section 4, is one bit of redundancy per digit. The added cost of one bit per digit position enables significant latency improvement in the basic operation of carry-free addition and all other arithmetic operations that use addition as a building block. In multioperand addition, and thus in multiplication, as well as in subtractive and multiplicative division, the per-add savings are compounded over many addition levels.

Because the main part of digits in a stored-transfer representation can be in nonredundant two's-complement format, much of the digit-level addition circuits can be based on readily available, and well optimised, binary adder cells. For example, a digit adder can be built from an h-bit binary adder, computing the $(h+1)$-bit sum $x'_i + y'_i$, followed by a special $(h+5)$-input, $(h+2)$-output circuit; the inputs are the aforementioned $(h+1)$-bit sum and two two-bit stored transfers x''_i and y''_i, while the outputs are the h-bit sum digit s'_i and a two-bit generated transfer s''_{i+1}. Except for an $O(h)$-time digit addition, the rest of the computation may be performed in a small constant time, independent of the radix (see Section 4).

4 Two-valued stored transfers

The representational efficiency of the proposed stored-transfer scheme can be improved by using a design trick involving coupled encoding of the two components x' and x'' of a digit x. Consider a three-valued stored transfer $x'' \in \{-1, 0, 1\}$ attached to a main digit $x' = 2u' + v'$, where $u' = \lfloor x'/2 \rfloor$ and $v' = x' \bmod 2$. It is assumed that x' is encoded in two parts: a single bit denoting v' and an arbitrary encoding for u'. A stored-transfer digit $\langle 2u' + 0, 0 \rangle$ can be recoded as $\langle 2u' + 1, -1 \rangle$, and $\langle 2u' + 1, 0 \rangle$ as $\langle 2u' + 0, 1 \rangle$, thus making it unnecessary to store the transfer value 0. The resulting two-valued stored transfer renders the representational efficiency of the proposed scheme competitive with the most efficient redundant representations. The delay and circuit costs of this recoding are small, given that only a single bit v' in the encoding of x' is affected. The more general case of a three-valued transfer $x'' \in \{\lambda, \lambda+1, \lambda+2\}$ is handled with equal ease: recode $\langle 2u' + 0, \lambda+1 \rangle$ as $\langle 2u' + 1, \lambda \rangle$ and $\langle 2u' + 1, \lambda+1 \rangle$ as $\langle 2u' + 0, \lambda+2 \rangle$.

The modification of the preceding paragraph, which may be viewed as reintroducing step 3 of the carry-free addition process, but in much simpler form involving single-bit logical operations, can be applied after each carry-free

Table 1: Stored-transfer representations of some redundant number systems

	Number system	Δ	$\delta = \rho$	D	G
1	Stored-carry (SC)	$[0, r]$	1	$[0, r-1]$	$\{0, 1\}$
2	Stored-borrow (SB)	$[-1, r-1]$	1	$[0, r-1]$	$\{-1, 0\}$
3	Stored-carry-or-borrow (SCB)	$[-1, r]$	2	$[0, r-1]$	$\{-1, 0, 1\}$
4	Stored-carry-or-borrow (SCB)	$[-1, r]$	2	$[0, r-1]$	$\{-1, 1\}$
5	Stored-double-carry (SDC)	$[0, r+1]$	2	$[0, r-1]$	$\{0, 1, 2\}$
6	Hybrid S-D ($h-1$ B, 1 BSD)	$[-1, 2^h-1]$	1	$[0, r-1]$	$\{-1, 0\}$
7	Hybrid S-D (1 BSD, $h-1$ B)	$[-2^{h-1}, 2^h-1]$	2^{h-1}	$[0, r-1]$	$\{-2^{h-1}, 0\}$
8	Minimally asymmetric	$[-r/2-1, r/2]$	2	$[-r/2, r/2-1]$	$\{-1, 0, 1\}$
9	Minimally asymmetric	$[-r/2-1, r/2]$	2	$[-r/2, r/2-1]$	$\{-1, 1\}$
10	Excess-1 stored-carry	$[1, r+1]$	1	$[0, r-1]$	$\{1, 2\}$
11	Excess-3 stored double carry	$[3, r+4]$	2	$[1, r]$	$\{2, 3, 4\}$
12	Excess-3 stored double carry	$[3, r+4]$	2	$[1, r]$	$\{2, 4\}$

Table 2: Stored unibit transfer (SUT)-like redundant digit sets

Number system	α	$\beta = \alpha + 2^h + 1$	$\lambda = \alpha + 2^{h-1}$	$c_0 = \lfloor \alpha/(2^h - 1) \rfloor$	$c_{d-1} = \lceil \beta/(2^h - 1) \rceil$
SUT	$-(2^{h-1} + 1)$	2^{h-1}	-1	-1	1
SDC	0	$2^h + 1$	2^{h-1}	0	2
Excess-3 SDC	3	$2^h + 4$	$2^{h-1} + 3$	0	2
Excess-2^h SCB	$2^h - 1$	2^{h+1}	$2^h + 2^{h-1} - 1$	1	3
Short-2^h SUT	$-(2^h + 2^{h-1} + 1)$	-2^{h-1}	$-(2^h + 1)$	-2	0

addition operation to keep representations efficient in the arithmetic circuits and their associated registers or it can be applied only at the interface between the arithmetic unit and the storage system.

Other *ad hoc* simplifications and efficient implementations for special cases may be derived. For example, the following algorithm (basically as a detailed description of Algorithm 2) is applicable for addition of two k-digit radix-2^h stored-transfer numbers x and y, where the main part of each digit is an h-bit two's-complement number and the transfer is a unibit in $G = \{-1, 1\}$.

Algorithm 3 (Radix-2^h stored-unibit-transfer addition to compute $s = x + y$): Perform the following digit operations for all positions i ($0 \le i < k$) concurrently:

1. Form the h-bit two's-complement value $z_i = x''_i + y''_i$.
2. Derive the carry-save sum $(u_{i+1}, v_i) = z_i + x'_i + y'_i$.
3. Add u_i to v_i, forming the binary position sum p_i.
4. Derive s'_i and s''_{i+1} satisfying $s'_i = p_i - rs''_{i+1}$.
5. Adjust s''_i and the least significant bit of s'_i.

Consider $G = \{-1, 1\}$, with its two members encoded as $\{0, 1\}$. The rightmost bit of z_i is always 0, the next bit is derived by an XNOR operation on the unibits, and the identical leftmost $h - 2$ bits by a NOR operation (XNOR and NOR are complements of logical XOR and OR functions). Step 2 can be realised by standard full-adders. Step 3 requires an adder that is h bits wide ($h - 1$ if an extra half-adder is used in step 2); this adder can be of any suitable design. In step 4, s'_i and s''_{i+1} are directly derived in constant time from p_i and its two most significant bits, respectively. Step 5 involves one gate delay, as previously discussed. Only step 3 has a latency that depends on h. Moreover, steps 1, 2 and 3, 4 may be partially overlapped to further reduce the constant-time component of the addition latency. A high-level circuit design, based on standard full/half-adders and carry acceleration cells for stored-unibit-transfer addition/subtraction (i.e. with $G = \{-1, 1\}$) is offered in [15].

Unfortunately, the two-valued transfer scheme just discussed is not applicable to arbitrary digit sets. Corollary 4 suggests the suitability of this scheme for any digit set with $\rho = 2$. However, there are other digit sets with $\rho > 2$ and minimal δ that lead to consecutive three-valued transfer sets (i.e. $\delta = 2$). By Lemma 2, such desirable digit sets are limited to $2 \le \rho \le r - 1$. For example, it is easy to verify that possible transfer sets for radix-16 digit sets $[-9, 9]$ (with $\rho = 3$) and [20, 45] (with $\rho = 10$) are $\{-1, 0, 1\}$ and $\{1, 2, 3\}$, respectively. But the main part of the two-valued stored-transfer representation of these digit sets is necessarily redundant. This complicates the addition scheme and renders the simple adder design, as previously discussed and implemented in [15], inapplicable. Therefore in the following section we focus exclusively on minimally redundant digit sets with $\rho = 2$.

5 Encoding of stored transfers

The results derived in Section 2 suggest the applicability of Algorithm 2 (as well as Algorithm 1) for arbitrary digit sets $[\alpha, \beta]$, possibly excluding zero as a member and with $\alpha < \beta$. As discussed earlier, to obtain an efficient addition scheme, the focus is on digit sets with $\rho = 2$, using stored transfer encoding of the digit set with a nonredundant two's-complement main part and a twit transfer part. Thus the digit set $[\alpha, \beta]$ in a radix-2^h number system is decomposed into $D = [-2^{h-1}, 2^{h-1} - 1]$ as the two's-complement main part and $G = \{\lambda, \lambda + 2\}$ as the twit transfer part. With these assumptions, given a value α as the lower bound of the digit set, the upper bound β and the twit parameter λ are derived as

$$\beta = \alpha + 2^h + 1; \quad \lambda = \alpha + 2^{h-1}$$

Example 3 (SUT-like digit sets): Table 2 shows the characteristic parameters of some stored-transfer digit sets, using twit transfers in $\{\lambda, \lambda + 2\}$, that satisfy the conditions of corollary 4. It is shown that for such digit sets, the SUT addition scheme (Algorithm 3) is applicable. □

In the addition scheme for radix-2^h stored-unibit-transfer operands (first entry of Table 2), an h-bit two's-complement sum of the transfers is derived (step 1 of Algorithm 3), and a standard three-operand addition is performed (fused steps 2–4 of Algorithm 3). Both operations are performed in parallel for all radix-2^h digits. The required high-level

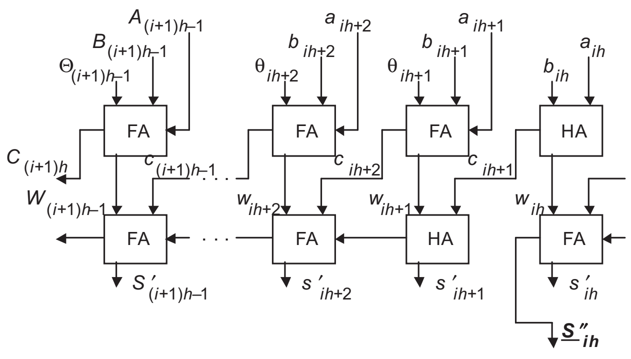

Fig. 2 *SUT adder*

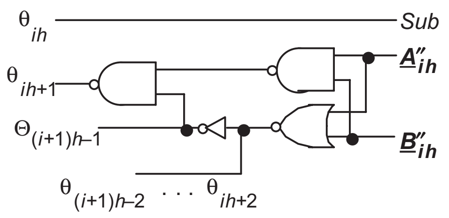

Fig. 3 *SUT transfer addition cell*

Table 3: Comparison of SUT adder with other redundant adders

Adder type	Asymmetry	Cost	Latency with carry ripple	Direct carry acceleration	Subtraction penalty
Maximally redundant [6]	None	$>42h$	$(h+1)\,\Delta_{FA}$ + a few logic levels	Applicable	Minimal
HSD [12]	≈50%	$32h+10$	$>h\,\Delta_{FA}$	Not applicable	Substantial
SUT	Minimal	$28h$	$(h+1)\,\Delta_{FA}$	Applicable	Minimal

circuit design is reproduced in Fig. 2 from [15], where the transfer addition cell of step 1, generating the θ inputs, appears in Fig. 3. Note that step 5 of Algorithm 3 is taken care of by in-place reduction [15] in the lower full-adder in position ih of Fig. 2.

Table 3 presents some performance measures for the proposed SUT adder, along with those of two other high-radix redundant representations. The addition scheme of Fahmy and Flynn [6], proposed in the context of implementing a floating-point adder, is based on maximally redundant radix-16 ($h=4$) symmetric signed digits in $[-15, 15]$, where a digit is represented by a 5-bit two's-complement number, excluding the value -16. The compound, maximally redundant signed-digit adder (first row of Table 3) computes three values simultaneously: the actual sum, and the incremented and decremented sums (sum $\pm$ 1). This is done for reduced latency, at the cost of tripling the layout area and power consumption relative to a standard 5-bit two's-complement adder. Despite the use of the aforementioned speed-up technique, the extra control logic needed increases the overall addition latency by several logic levels [6], the extent of which has not been specified.

The hybrid redundancy scheme of Phatak and Koren [12] (second row in Table 3), suffers from highly asymmetric digit sets (see row 7 of Table 1), which reduces its appeal for general-purpose applications and complicates the use of addition circuitry for the subtraction operation. Moreover, the inapplicability of carry acceleration techniques leads to a best-case O(h) addition latency for radix-2^h operands. The design is based on the use of nonstandard adder cells for redundant and nonredundant positions, realised by 42 and 32 transistors, respectively, which result in the implementation cost of $32h+10$ transistors for each radix-2^h digit. The average latency per radix-2 position is roughly equal to that of one full adder (Δ_{FA}).

The proposed SUT adder, represented in the third row of Table 3, requires two rows of full adders. This corresponds to an active hardware redundancy factor of two, compared to a redundancy factor of at least three for the addition scheme in Fahmy and Flynn [6]. The latency of the proposed adder is no greater than that of a $(h+1)$-bit two's-complement adder, which can be reduced drastically via standard carry acceleration. A radix-2 slice in the proposed adder is realisable by roughly 28 transistors, given the use of 14-transistor full adders [19].

An attractive solution for the addition of stored-twit-transfer operands, in cases where the stored transfers are twits other than unibit, is to design a special transfer adder for twit transfers and use the same three-operand adder of Fig. 1. Table 4 shows the addition summary for two twits $\underline{\boldsymbol{A}}''$ and $\underline{\boldsymbol{B}}''$ in $\{\lambda, \lambda+2\}$ and a special binary representation of the sum. Following the convention in Jaberipur *et al.* [15], bold-italic underlined uppercase type is used for twits, while uppercase (lowercase) regular type corresponds to negabits (posibits).

Based on Lemma 1, for $r=2^h$, the minimum value for generated transfers in carry-free addition based on Algorithms 1 and 2 is $t_{\min}=\lfloor\alpha/(2^h-1)\rfloor$, leading to $\alpha=t_{\min}(2^h-1)+v$, where $0\le v\le 2^h-2$. Substituting for α in the equation for λ yields

$$\begin{aligned}\lambda &= \alpha+2^{h-1}\\ &= t_{\min}(2^h-1)+v+2^{h-1}\\ &= 2^h(t_{\min}+1)+v-2^{h-1}-t_{\min}\end{aligned}$$

Using this equation for λ the possible transfer sum values 2λ, $2\lambda+2$, and $2\lambda+4$ can be expressed as follows, where $i=0$, 2, and 4, respectively:

$$\begin{aligned}2\lambda+i &= \lambda+t_{\min}+\lambda-t_{\min}+i\\ &= 2^h(t_{\min}+1)+(v-2^{h-1}+i)+(\lambda-t_{\min})\end{aligned}$$

The first, middle, and last terms on the right-hand side of the equation are represented in Table 4 as τ_h, $\Theta_{h-1}\theta_{h-2}\ldots\theta_2\theta_1$, and τ_0, respectively, where the middle term represents an h-bit two's-complement number in the range $-2^{h-1}\le v-2^{h-1}\le v-2^{h-1}+4\le 2^{h-1}-1$. This conformance is necessary for using the adder of Fig. 1, but it requires the constraint $v\le 2^h-5$. This restriction, given the looser bound $v\le 2^h-2$, means that our addition scheme does not work for $v=2^h-3$ and $v=2^h-4$; note that the case $v=2^h-2$ is already excluded by Theorem 2.

Table 4: Addition of two twit transfers

$\underline{\boldsymbol{A}}''$	$\underline{\boldsymbol{B}}''$	$\underline{\boldsymbol{A}}''+\underline{\boldsymbol{B}}''$	τ_h	$\Theta_{h-1}\theta_{h-2}\ldots\theta_2\theta_1\theta_0$	τ_0
0	0	2λ	$t_{\min}+1$	$v-2^{h-1}$	$\lambda-t_{\min}$
0	1	$2\lambda+2$	$t_{\min}+1$	$v-2^{h-1}+2$	$\lambda-t_{\min}$
1	0	$2\lambda+2$	$t_{\min}+1$	$v-2^{h-1}+2$	$\lambda-t_{\min}$
1	1	$2\lambda+4$	$t_{\min}+1$	$v-2^{h-1}+4$	$\lambda-t_{\min}$

Table 5: Sample digit sets for stored-twit-transfer addition

Range of $[\alpha, \beta]$	$\lambda=\alpha+8$	$c_0=\lfloor\alpha/15\rfloor$	$c_{d-1}=\lceil\beta/15\rceil$	Digit sets that are excluded by:	
				Theorem 2	$v>11$
$[-75, -58]\to[-64, -47]$	$-67\to-56$	-5	-3	$[-61, -44]$	$[-63, -46]$, $[-62, -45]$
$[-15, 2]\to[-4, 13]$	$-7\to4$	-1	1	$[-1, 16]$	$[-3, 14]$, $[-2, 15]$
$[0, 17]\to[11, 28]$	$8\to19$	0	2	$[14, 31]$	$[13, 30]$, $[12, 29]$
$[210, 227]\to[221, 238]$	$218\to229$	14	16	$[224, 241]$	$[223, 240]$, $[222, 239]$

The SUT adder of Fig. 2 generates a transfer value in $\{-1, 1\}$ in position h, where adding τ_h produces a transfer in $\{t_{\min}, t_{\min}+2\}$. Finally, τ_0 of the next higher radix-2^h digit is also added to the latter transfer value to yield a value in $\{\lambda, \lambda+2\}$. The two additions just described do not actually take place, and τ_h and τ_0 are not physically stored. They are merely used to show the relevant interpretation of the transfers produced. Note that for the SUT adder with $\lambda = t_{\min} = -1$ and $v = 2^{h-1} - 2$, we have $\tau_h = \tau_0 = 0$ and the transfer sum values associated with the entries in Table 4 are -2, 0, 0, and 2.

Example 4 (Radix-16 stored-twit-transfer addition): Table 5 lists some α, β, and λ values, for which the twit-transfer addition scheme works, along with their associated transfer range represented by c_0 and c_{d-1}. Here, $h = 4$. □

Example 5 (Logic equations for transfer addition cell): The transfer addition cells for different values of λ (and correspondingly v), though not exactly the same, have comparable complexities. Table 6 lists the logic equations defining the bits of the transfer sum for $h = 4$ and all 12 possible values of v. □

6 Conversion to/from two's complement

The following algorithm provides a general method to convert a two's-complement integer x to a stored-twit-transfer representation in radix 2^h, where the twit transfers are in $\{\lambda, \lambda+2\}$ and the two possible values are encoded logically as $\{0, 1\}$.

Algorithm 4 (Conversion from two's complement to stored-twit-transfer):

1. Sign-extend x until its width is a multiple of h, say kh
2. Compute $y = x - \lambda(2^{kh}-1)/(2^h-1)$, where the subtrahend is composed of λ-valued digits in positions 0, h, $2h, \ldots, (k-1)h$, weighted 1, 2^h, $2^{2h}, \ldots, 2^{(k-1)h}$
3. Transform the bits of y to the target bits as follows (see Table 7 for justification):

   a. $X'_{ih-1} = \neg y_{ih-1}$, for $1 \le i \le k-1$.
   b. $X''_{ih} = y_{ih}\, y_{ih-1}$, for $1 \le i \le k-1$; $X''_0 = 0$.
   c. $x'_{ih} = y_{ih} \oplus y_{ih-1}$, for $1 \le i \le k-1$.
   d. $x'_{ih-j} = y_{ih-j}$, for $1 \le i \le k-1$ and $2 \le j \le h$.

The delay of steps 1 and 3 of Algorithm 4 is constant and independent of h and k, but step 2 requires word-width carry propagation in general. However, there are special practical

Table 6: Transfer sum equations (h = 4 and 0 ≤ v ≤ 11)

v	Θ_3	θ_2	θ_1	θ_0
0	0	$\underline{\boldsymbol{X}}'' \wedge \underline{\boldsymbol{Y}}''$	$\underline{\boldsymbol{X}}'' \oplus \underline{\boldsymbol{Y}}''$	0
1	0	$\underline{\boldsymbol{X}}'' \wedge \underline{\boldsymbol{Y}}''$	$\underline{\boldsymbol{X}}'' \oplus \underline{\boldsymbol{Y}}''$	1
2	0	$\underline{\boldsymbol{X}}'' \vee \underline{\boldsymbol{Y}}''$	$\neg(\underline{\boldsymbol{X}}'' \oplus \underline{\boldsymbol{Y}}'')$	0
3	0	$\underline{\boldsymbol{X}}'' \vee \underline{\boldsymbol{Y}}''$	$\neg(\underline{\boldsymbol{X}}'' \oplus \underline{\boldsymbol{Y}}'')$	1
4	$\underline{\boldsymbol{X}}'' \wedge \underline{\boldsymbol{Y}}''$	$\neg(\underline{\boldsymbol{X}}'' \wedge \underline{\boldsymbol{Y}}'')$	$\underline{\boldsymbol{X}}'' \oplus \underline{\boldsymbol{Y}}''$	0
5	$\underline{\boldsymbol{X}}'' \wedge \underline{\boldsymbol{Y}}''$	$\neg(\underline{\boldsymbol{X}}'' \wedge \underline{\boldsymbol{Y}}'')$	$\underline{\boldsymbol{X}}'' \oplus \underline{\boldsymbol{Y}}''$	1
6	$\underline{\boldsymbol{X}}'' \vee \underline{\boldsymbol{Y}}''$	$\neg(\underline{\boldsymbol{X}}'' \vee \underline{\boldsymbol{Y}}'')$	$\neg(\underline{\boldsymbol{X}}'') \oplus (\underline{\boldsymbol{Y}}'')$	0
7	$\underline{\boldsymbol{X}}'' \vee \underline{\boldsymbol{Y}}''$	$\neg(\underline{\boldsymbol{X}}'' \vee \underline{\boldsymbol{Y}}'')$	$\neg(\underline{\boldsymbol{X}}'') \oplus (\underline{\boldsymbol{Y}}'')$	1
8	1	$\underline{\boldsymbol{X}}'' \wedge \underline{\boldsymbol{Y}}''$	$\underline{\boldsymbol{X}}'' \oplus \underline{\boldsymbol{Y}}''$	0
9	1	$\underline{\boldsymbol{X}}'' \wedge \underline{\boldsymbol{Y}}'')$	$\underline{\boldsymbol{X}}'' \oplus \underline{\boldsymbol{Y}}''$	1
10	1	$\underline{\boldsymbol{X}}'' \vee \underline{\boldsymbol{Y}}'')$	$\neg(\underline{\boldsymbol{X}}'' \oplus \underline{\boldsymbol{Y}}'')$	0
11	1	$\underline{\boldsymbol{X}}'' \vee \underline{\boldsymbol{Y}}''$	$\neg(\underline{\boldsymbol{X}}'' \oplus \underline{\boldsymbol{Y}}'')$	1

Table 7: Justification for transformations of Algorithm 4

y_{ih}	y_{ih-1}	Arithmetic value $2y_{ih} + y_{ih-1} = 2(x'_{ih} + 2X''_{ih}) + X'_{ih-1} - 1$	x'_{ih}	X''_{ih}	X'_{ih-1}
0	0	$0+0 = 2\,(0+0)+1-1$	0	0	1
0	1	$0+1 = 2\,(1+0)+0-1$	1	0	0
1	0	$2+0 = 2\,(1+0)+1-1$	1	0	1
1	1	$2+1 = 2\,(0+2)+0-1$	0	1	0

cases where this step may be eliminated (e.g. for $\lambda = 0$ and in the SUT case of [15] with $\lambda = -1$).

For the reverse conversion, the main parts are added, treating their corresponding transfers as doublebits (bits with double the normal weight), all in parallel. This yields a redundant number of the same value with two's-complement radix-2^h digits. The rest of the process follows conventional redundant-to-binary conversion techniques [1], except that addition of a constant $\lambda(2^{kh}-1)/(2^h-1)$ should be fused into the process. Therefore the reverse conversion, as is expected for any redundant representation, involves word-width carry propagation.

7 Conclusions

We have shown that the stored-transfer representation of redundant numbers offers speed and cost benefits in the carry-free addition process. We have proved the necessity of at least three transfer digit values, and sufficiency of four values, to allow carry-free addition in all cases of practical interest. We have further shown that by a simple adjustment in the final stage of the carry-free addition algorithm, the number of stored transfers can be reduced to two values, thus requiring a single bit for storage. The proposed stored-transfer scheme is thus competitive with other practical redundant representations with regard to storage cost.

In the course of establishing the theoretical basis for the proposed method, two practical restrictions imposed in the general signed-digit representations of Parhami [17] were relaxed, thereby providing the ability to deal with more general digit sets that do not necessarily include zero as a member and with noncontiguous transfer sets. This is an important aspect of the work reported in this paper because the generalisation comes with no inherent latency or cost penalty, but opens up valuable alternatives in exploring the implementation options in the design space associated with different redundant number representations.

We also demonstrated that converting a two's-complement number to stored-transfer form implies minimal constant cost and latency for many important practical cases, while the reverse conversion needs the obligatory carry propagation. This affinity with two's-complement numbers in representation and circuit implementation (i.e. use of standard full/half adders and compressors) is a key strength of the stored-transfer scheme.

Derivation of algorithms for stored-transfer multiplication and division is quite feasible. Very-high-radix SRT division algorithms with signed-digit partial remainders and signed-digit quotient [20] can be modified to accept stored-transfer operands. A series of arithmetic operations can thus be performed without carry propagation by representing the inputs, intermediate results, and outputs in stored-transfer format. Results on other arithmetic operations, particularly for floating-point operands, and a number of useful arithmetic support functions (such as shifting) will be reported in the near future.

8 Acknowledgments

G. Jaberipur's research was supported, in part, by IPM under grant CS-1384-3-02 and by Shahid Beheshti University under grant 600/2818.

9 References

1 Parhami, B.: 'Computer arithmetic: algorithms and hardware designs' (Oxford University Press, New York, 2000)
2 Metze, G., and Robertson, J.E.: 'Elimination of carry propagation in digital computers'. Proc. Int. Conf. on Information Processing, Paris, 1959, pp. 389–396
3 Avizienis, A.: 'Signed-digit number representations for fast parallel arithmetic', *IRE Trans. Electron. Comput.*, September 1961, **10**, pp. 389–400
4 Edamatsu, H., Taniguchi, T., Nishiyama, T., and Kuninobu, S.: 'A 33 MFLOPS floating point processor using redundant binary representation'. Digest of IEEE Int. Conf. on Solid-State Circuits, February 1988
5 Balakrishnan, W., and Burgess, N.: 'Very-high-speed VLSI 2s-complement multiplier using signed binary digits', *IEE Proc. E, Comput. Digit. Tech.*, 1992, **139**, pp. 29–34
6 Fahmy, H., and Flynn, M.J.: 'The case for a redundant format in floating-point arithmetic'. Proc. 16th IEEE Symp. on Computer Arithmetic, 2003, pp. 95–102
7 Atkins, D.E.: 'An introduction to the role of redundancy in computer arithmetic', *Computer*, 1975, **8**, (6), pp. 74–76
8 Parhami, B.: 'Tight upper bonds on the minimum precision required of the divisor and the partial remainder in high-radix division', *IEEE Trans. Comput.*, 2003, **52**, (11), pp. 1509–1514
9 Matula, D.W.: 'Basic digit sets for radix representation', *J. ACM*, 1982, **29**, (4), pp. 1131–1143
10 Kornerup, P.: 'Digit-set conversions: generalizations and applications', *IEEE Trans. Comput.*, 1994, **43**, (5), pp. 622–629
11 Takagi, N., Yasuura, H., and Yajima, S.: 'High-speed VLSI multiplication algorithm with a redundant binary addition tree', *IEEE Trans. Comput.*, 1985, **34**, (9), pp. 789–796
12 Phatak, D.S., and Koren, I.: 'Hybrid signed-digit number systems: a unified framework for redundant number representations with bounded carry propagation chains', *IEEE Trans. Comput.*, 1994, **43**, pp. 880–891
13 Phatak, D.S., and Koren, I.: 'Constant-time addition and simultaneous format conversion based on redundant binary representations', *IEEE Trans. Comput.*, 2001, **50**, (11), pp. 1267–1278
14 Jaberipur, G., and Ghodsi, M.: 'High radix signed digit number systems: representation paradigms', *Sci. Iranica*, 2003, **10**, (4), pp. 383–391
15 Jaberipur, G., Parhami, B., and Ghodsi, M.: 'Weighted two-valued digit-set encodings: unifying efficient hardware representation schemes for redundant number systems'. *IEEE Trans. Circuits Syst. I*, 2005, **52**, (7), pp. 1348–1357
16 Jaberipur, G., Parhami, B., and Ghodsi, M.: 'A class of stored-transfer representations for redundant number systems'. Proc. 35th Asilomar Conf. on Signals Systems and Computers, 2001, pp. 1304–1308
17 Parhami, B.: 'Generalized signed-digit number systems: a unifying framework for redundant number representations', *IEEE Trans. Comput.*, 1990, **39**, (1), pp. 89–98
18 Kobayashi, H.: 'A multioperand two's complement addition algorithm'. Proc. 7th IEEE Symp. on Computer Arithmetic, June 1985, pp. 16–19
19 Radhakrishnan, D.: 'Low-voltage low-power CMOS full adder', *IEE Proc., Circuits, Devices Syst.*, February 2001, **148**, (1), pp. 19–24
20 Flynn, M.J., and Oberman, S.F.: 'Advanced computer arithmetic design' (Wiley, 2001)

10 Appendix

This Appendix contains formal proofs of lemmas and theorems appearing in the main body of the paper.

10.1 Proof of Lemma 1

To ensure that no new transfer is generated in step 3 of Algorithm 1, the following inequalities should hold:

$$\alpha - c_0 \le w_i \le \beta - c_{d-1}$$

The minimum (maximum) p_i value associated with the choice $t_{i+1} = c_0$ ($t_{i+1} = c_{d-1}$) in step 2 of Algorithm 1 is $\alpha - c_0 + rc_0$ ($\beta - c_{d-1} + rc_{d-1}$). To ensure that all possible p_i values in $[2\alpha, 2\beta]$ can be decomposed to their corresponding interim sum and transfer digit values, two conditions must hold: (i) The position sum range $[2\alpha, 2\beta]$ must be a subinterval of $[\alpha - c_0 + rc_0, \beta - c_{d-1} + rc_{d-1}]$, leading to the requirements $c_0 \le \lfloor \alpha/(r-1) \rfloor$ and $c_{d-1} \ge \lceil \beta/(r-1) \rceil$; (ii) There should be no gap between the maximum and minimum position sum values associated with an arbitrary pair of consecutive transfer values c_j and c_{j+1}, leading to the condition $\beta - c_{d-1} + rc_j + 1 \ge \alpha - c_0 + rc_{j+1}$, which can be converted to $\delta_j \le 1 + (\rho - \delta)/r$ through replacing $\beta - \alpha + 1$ by $\rho + r$, $c_{d-1} - c_0$ by δ, and $c_{j+1} - c_j$ by δ_j. □

10.2 Proof of Lemma 2

Using the results of Lemma 1, a lower bound on $\delta_{\min}$ is easily derived as

$$\begin{aligned}\delta_{\min} &= \mathrm{lb}(c_{d-1}) - \mathrm{ub}(c_0)\\ &= \lceil \beta/(r-1) \rceil - \lfloor \alpha/(r-1) \rfloor\\ &\ge (\beta - \alpha)/(r-1)\\ &= 1 + \rho/(r-1)\end{aligned}$$

For an upper bound, assume that $\alpha = u_1(r-1) + v_1$ and $\beta = u_2(r-1) - v_2$, where $0 \le v_1, v_2 \le r-2$. Substituting for β and α in the equality above and in $\rho = \beta - \alpha + 1 - r$ yields $\delta_{\min} = u_2 - u_1$ and

$$\begin{aligned}\rho &= (u_2 - u_1)(r-1) - (v_1 + v_2) + 1 - r\\ &= (r-1)(\delta_{\min} - 1) - (v_1 + v_2)\end{aligned}$$

Solving the latter equation for $\delta_{\min}$ and using the bounds for v_1 and v_2 leads to

$$\begin{aligned}\delta_{\min} &= 1 + (\rho + v_1 + v_2)/(r-1)\\ &\le 1 + (\rho + 2r - 4)/(r-1)\\ &= 3 + (\rho - 2)/(r-1)\end{aligned}$$

Recalling the upper bound for δ_j from Lemma 1 and noting that transfer values in T are distinct, we have $1 \le \delta_j \le 1 + \lfloor (\rho - \delta_{\min})/r \rfloor$, leading to $\rho \ge \delta_{\min}$. □

10.3 Proof of Theorem 1

The second component of the upper bound is obvious by noting the maximum value $2(r-2)$ for $v_1 + v_2$ from the definitions of v_1 and v_2. For the first component of the bound, we resort to the equality $\delta_{\min} = 1 + (\rho + v_1 + v_2)/(r-1)$, derived in the proof of Lemma 2, to obtain

$$v_1 + v_2 = \delta_{\min}(r-1) - (\rho + r - 1)$$

Using the bound $\delta_{\min} \le \rho$ from Lemma 2, and after some manipulation, we obtain the desired bound $v_1 + v_2 \le (r-2)(\rho-1) - 1$. □

10.4 Proof of Theorem 2

The conditions $\rho = 2$ (which is assumed) and $r \ge 3$ (required by corollary 2) lead to $\delta_{\min} = 2$ by corollary 1. As per the definition of $\delta_{\min}$ (see Lemma 1), $\lceil \beta/(r-1) \rceil - \lfloor \alpha/(r-1) \rfloor = 2$. Substituting $\rho + r - 1 + \alpha$ for β and 2 for ρ converts the preceding equation to $\lceil (\alpha + 2)/(r-1) \rceil = \lfloor \alpha/(r-1) \rfloor + 1$. Letting $v_1 = \alpha \bmod (r-1)$, it is easy to see that the last equation holds except when $v_1 = r - 2$, or $\alpha = u_1(r-1) + r - 2$ and $\beta = (u_1 + 2)(r-1) + 1$. Therefore carry-free addition of Algorithm 1 is inapplicable to digit sets $[\alpha, \beta]$ satisfying the latter equations if $\rho = 2$. $\square$

10.5 Proof of Theorem 3

According to Theorem 1, the condition $(\rho - 1)\ (r - 2) > v_1 + v_2$ is necessary and sufficient for carry-free addition, where $v_1 = \alpha \bmod (r-1)$ and $v_2 = (-\beta) \bmod (r-1)$. For $\rho > 3$, the condition always holds, given that $(\rho - 1)(r-2) > 2(r-2) \geq v_1 + v_2$. For $\rho = 3$, the condition reduces to $v_1 + v_2 < 2(r-2)$, which always holds, except when $v_1 = v_2 = r - 2$. So the proof will be complete if we show that for $v_1 = r - 2$, we cannot have $v_2 = r - 2$ for any digit set $[\alpha, \beta]$ with $\rho = 3$. For $\alpha = u_1(r-1) + r - 2$ and $\rho = 3$, we have $\beta = \rho + r - 1 + \alpha = 2 + r + \alpha = u_1(r-1) + 2r = (u_1 + 3)(r-1) - (r-3)$; hence, $r - 3$ is seen to be the only possible solution for v_2. $\square$

10.6 Proof of Theorem 4

From $d \leq \delta + 1$ and the upper bound for $\delta_{\min}$ from Lemma 2, we have $d \leq \min(1 + \rho, 4 + \lfloor(\rho - 2)/(r-1)\rfloor)$, leading to $d \leq 4$ for $3 \leq \rho \leq r$ and $d \leq 3$ for $\rho = 2$. For a lower bound, note that $\delta = \sum_{0 \leq j \leq d-2} \delta_j$. We thus obtain $1 + \lceil \rho/(r-1) \rceil \leq (d-1)(1 + \lfloor(\rho - \delta)/r\rfloor)$ by replacing δ with the lower bound for $\delta_{\min}$ from Lemma 2 and each δ_j with the upper bound from Lemma 1. Assuming $\rho = u(r-1) - v$, where $u > 0$ and $0 \leq v \leq r - 2$, substitution for ρ in the latter inequality results in $d - 1 \geq (1 + u)/(1 + u - (u + v + \delta)/r) > 1$, which is equivalent to $d \geq 3$. $\square$