JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 19, 1015-1039 (2003)

Asymmetric High-Radix Signed-Digit Number Systems for Carry-Free Addition

SHAO-HUI SHIEH AND CHENG-WEN WU[*]
Department of Electronic Engineering
National Chinyi Institute of Technology
Taichung, 411 Taiwan
[*]*Department of Electrical Engineering*
National Tsing Hua University
Hsinchu, 300 Taiwan

We propose an asymmetric high-radix signed-digit (AHSD) number system for fast binary addition, and show that the AHSD number system supports carry-free (CF) addition. The CF additions in AHSD, to be classified as quasi-closed and subclosed additions, use only one redundant digit for any radix $r \geq 2$. Novel algorithms for constructing the three-stage CF adder (CFA) based on the AHSD number system are also presented. Moreover, if the radix is specified as $r = 2^m$, where m is any positive integer, the binary-to-AHSD conversion can be done in constant time regardless of the word-length. Hence, the AHSD-to-binary conversion dominates the performance of an AHSD-based arithmetic system. We also propose two efficient algorithms for converting AHSD numbers to binary format. The first uses a novel structure to achieve high speed, while the second uses simple transformations and conventional additions to provide hardware reusability. These results are important since the conversion from AHSD numbers to binary format has been considered the performance bottleneck of AHSD-based arithmetic systems.

Keywords: asymmetric high-radix signed-digit number, computer arithmetic, number system conversion, redundant number system, signed-digit numbers, VLSI design

1. INTRODUCTION

Addition is the most important and frequently used arithmetic operation in computer systems. Generally, two methods can be used to speed up the addition operation. One is to explicitly shorten the carry-propagation chain by using circuit design techniques, such as detecting the completion of the carry chain as soon as possible, carry look-ahead, etc. [1-3]. Another is to convert the operands from the binary number system to a redundant number system, e.g., the signed-digit number system or the residue number system, so that the addition become carry-free (CF) [2-5]. This implicitly eliminates the carry-propagation chain so that fast addition can be done, at the expense of conversion between the binary number system and the redundant number system. In this paper we focus on exploring signed-digit (SD) numbers.

A class of signed-digit number representations of symmetrical type, called the *ordinary signed-digit* (OSD) number system, were first proposed in [4]. These number sys-

Received January 29, 2002; revised July 23, 2002; accepted February 24, 2003.
Communicated by Gen-Huey Chen.

tems are defined for any radix $r \geq 3$ and for the digit set $\{-\alpha, \ldots, -1, 0, 1, \ldots, \alpha\}$, where α is an integer such that $r/2 < \alpha < r$. In an OSD number system, the *redundancy index* ρ, defined as $\rho = 2\alpha - r + 1$, ranges from the minimal redundancy $\lfloor r/2 \rfloor + 1$ to the maximal redundancy $r - 1$. The most important contribution of OSD is to explore the possibility of performing carry-free addition and borrow-free subtraction for fast parallel arithmetic, if enough redundancy is used. The OSD number system was later extended to the *generalized signed-digit* (GSD) number system [6-9]. The GSD number system for radix $r > 1$ has the digit set $\{-\alpha, \ldots, -1, 0, 1, \ldots, \beta\}$, where $\alpha \geq 0$ and $\beta \geq 0$. The redundancy is $\rho = \alpha + \beta + 1 - r$. So far, the most important contribution of the works on GSD [6-9] include unifying the redundant number representation and sorting the CF addition schemes for the GSD number system according to the radix r and redundancy index ρ. However, ideal single-stage CF addition has not been achieved, though two-stage CF addition has been shown to be doable for any GSD system with $r > 2$ and $\rho > 2$, or with $r > 2$ and $\rho = 2$ provided that $\alpha \neq 1$ and $\beta \neq 1$. For any GSD system with $r = 2$ and $\rho = 1$, or $\rho = 2$ and α or β equals 1, the limited-carry addition must be used.

There are many applications for the SD number representations, most notably in computer and digital signal processing systems. Specifically, the CF adder has been investigated based on the redundant positive-digit numbers [10] and the symmetrical radix-4 SD numbers [11, 12] for high-speed area-efficient multipliers. The symmetrical radix-2 SD number representation has been used in the implementation of a RSA cryptographic chips [13], high-speed VLSI multipliers [14, 15], FIR filters [16], IIR filters [17], dividers [18], etc. Though arithmetic operations using these number representations can be done carry free, they have common difficulty in conversion to and from the binary number representation. Hence, in the past, many researchers have proposed specific architectures for number system conversion [15, 17, 19-22].

In this paper, we present the *asymmetric high-radix signed-digit* (AHSD) number system. The idea of AHSD is not new. A particular AHSD number system was called the radix-r stored-borrow (SB) number system in [7]. Most earlier works have focused on binary stored-borrow (BSB) number systems, where $r = 2$ [6, 14, 23-25]. Instead of proposing a new number representation, our purpose is to explore the inherent CF property of AHSD. The CF addition in AHSD is the basis for our high-speed addition circuits. The conversion of AHSD to and from binary will be discussed in detail. By choosing $r = 2^m$, where m is any positive integer, a binary number can be converted to its canonical AHSD representation in constant time. We will also present two simple algorithms for converting AHSD numbers to binary: the first stresses high speed and the other provides hardware reusability. Since the conversion from AHSD to binary has been considered the bottleneck of AHSD-based arithmetic computation, these algorithms greatly improve the performance of AHSD systems. For illustration, we will discuss in detail the example on AHSD(4), i.e., the radix-4 AHSD number system. The proposed approach is practical thanks to the simple conversion.

2. THE AHSD NUMBER SYSTEM

In the selection of a signed-digit (SD) number system, we have to consider several things, including 1) conversion from binary to SD, 2) inherent CF addition, and 3) conversion from SD to binary. We also need to identify applications that can take advantage of the system. We will give some definitions first, and then discuss these issues.

Definition 1 (Closure of Addition)
A set S is *closed* with respect to addition if, for any pair of elements (a, b), the addition specifies a rule for obtaining a unique value s from (a, b) such that if $a, b \in S$ then $s \in S$.

The addition is denoted as $a + b = s$ or $s = a + b$, in which a is the augends, b the addend, and s the sum. If S is closed with respect to addition, we call the addition a *closed addition*.

Definition 2 (Quasi-Closure of Addition)
Let D be a proper subset of S. The set S is *quasi-closed* in D with respect to addition if, for any pair of elements (a, b), the addition specifies a rule for obtaining a unique value s from (a, b) such that if either $a \in D$ and $b \in S$ or $b \in D$ and $a \in S$ then $s \in S$.

Note that the quasi-closure of addition requires one operand to be in $D \subset S$, and the other operand and the sum to be in S. If S is quasi-closed in D with respect to addition, we call the addition a *quasi-closed addition*.

Definition 3 (Subclosure of Addition)
Let D be a proper subset of S. The set S is *subclosed* in D with respect to addition if, for any pair of elements (a, b), the addition specifies a rule for obtaining a unique value s from (a, b) such that if $a, b \in D$ then $s \in S$.

Again, if S is subclosed in D with respect to addition, we call the addition a *subclosed addition*. Based on the definitions, it is easy to show that the following proposition is true.

Proposition 1
The closure of addition implies the quasi-closure of addition; and the quasi-closure of addition implies the subclosure of addition.

The radix-r *asymmetric high-radix signed-digit* (AHSD) number system, denoted AHSD(r), is a positional weighted number system with the digit set $S_r = \{-1, 0, \ldots, r - 1\}$, where $r > 1$. The AHSD number system is a minimally redundant system with only one redundant digit in the digit set. It is also referred to as the radix-r stored-borrow (SB) number system in previous works [7]. In the past, only the binary (radix-2) stored-borrow (BSB) number system has been extensively studied [6, 14, 23-25]. We will explore the inherent carry-free property in AHSD and develop systematic approaches for conversion of AHSD numbers to and from binary ones.

An n-digit number X in AHSD(r) is represented as

$$X = (x_{n-1}, x_{n-2}, \ldots, x_0)_r, \tag{1}$$

where $x_i \in S_r$ for $i = 0, 1, \ldots, n - 1$, and $S_r = \{-1, 0, 1, \ldots, r - 1\}$ is the digit set of AHSD(r). The value of X can be represented as

$$X = \sum_{i=0}^{n-1} x_i r^i. \tag{2}$$

Clearly, the range of X is $[(1 - r^n)/(r - 1), r^n - 1]$.

2.1 Binary to AHSD Conversion

Since the binary number system is the most widely used, we have to consider the conversion between the AHSD and binary number systems. Although the radix r may be any positive integer, simple binary-to-AHSD conversion can be achieved if $r = 2^m$ for any positive integer m [26]. The reason for such simple conversion will be explained later. We assume $r = 2^m$ in what follows, unless otherwise specified. Note that there may be more than one AHSD(r) number representation for a binary number. For instance, the binary number $(0, 1, 1, 0, 0)_2$ can be converted to two different AHSD(4) numbers, i.e., $(1, -1, 0)_4$ and $(0, 3, 0)_4$. Hence, the binary-to-AHSD(r) conversion, being a one-to-many mapping, may be performed by several different methods. We would like to find an efficient and systematic conversion method that takes advantage of the carry-free property.

Proposition 2 (Existence of AHSD Representation)
Any n-bit unsigned binary number $X = (b_{n-1}, \ldots, b_1, b_0)$ has a unique *canonical form* AHSD(r) representation as

$$X = \sum_{j=0}^{q-1} x_j r^j = \sum_{j=0}^{q-1} (\sum_{k=0}^{m-1} b_{mj+k} 2^k) r^j, \tag{3}$$

where $x_j \in S_r$, $r = 2^m$, and $q = \lceil n/m \rceil$.

Proposition 2 suggests a simple one-to-one mapping that converts a binary number to its corresponding AHSD(r) representation. This conversion can easily be done by partitioning the binary number, from the LSB to the MSB, into groups of m bits each. Leading 0's can be added to the MSB end of the bit-string to ensure that the number of bits is m for each group. Then, the corresponding digit, with value ranging from 0 to $r - 1$, is assigned to each group. Obviously, this converts a binary number to a unique conventional radix-r number representation. In the conventional radix-r number system, the digit set is $D_r = \{0, 1, \ldots, r - 1\}$ for any radix $r \geq 2$. Thus, D_r is a proper subset of S_r. The equivalent conventional radix-r representation for a binary number we use here is a special form called the *canonical form*, which is one of many equivalent AHSD(r) representations. The digit -1 is introduced into the system for flexible representation of the AHSD number converted from its binary form, and for easy implementation of AHSD-based arithmetic operations. In the subsequent discussion, the inherent carry-free property of the AHSD number system is exploited, and the arithmetic based on AHSD is developed.

2.2 Carry-Free Addition

By using the AHSD representation, fast carry-free (CF) addition can be done. It is well known that the addition and subtraction of signed numbers, using the r's-complement representation, can be implemented by the unsigned addition with sign extension and overflow detection. However, even though OSD and GSD provide various CF additions, there can be insufficient redundancy to perform a CF addition for two AHSD(r) numbers when $r > 2$. In what follows, we focus only on certain unsigned additions. Suppose we are adding $X = (x_{q-1}, \ldots, x_1, x_0)_r$ and $Y = (y_{q-1}, \ldots, y_1, y_0)_r$, which are two unsigned AHSD(r) numbers. Let one of the two operands be in canonical form, and then, by extending our previous works [26, 27], we will give two CF algorithms that realize the addition $S = X + Y$.

2.2.1 CF addition with estimate digits

1. Internal Individual Summation (IIS): Adding corresponding digits of X and Y, i.e.,

$$z_j = x_j + y_j,\ 0 \le j \le q-1, \tag{4}$$

where z_j's are the digits of the sum. The estimate digits are generated concurrently, where

$$e_j = \begin{cases} 1, & \text{if } z_j < r-1 \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

Apparently, $z_j \in \{-1, 0, \ldots, 2(r-1)\}$ if one of the two operands is in the canonical form. If z_j is not in S_r, then conversion is required.

2. Self-Consistent Adjustment (SCA): For consistency in our AHSD operation, conversion must be made to z_j. For each z_j, an interim transfer digit c_j and an interim sum digit μ_j are chosen based on e_{j-1} such that

$$z_j = r \times c_j + \mu_j \tag{6}$$

subject to

$$(c_j, \mu_j) = \begin{cases} (0, z_j), & \text{if } (e_j) \vee ((z_j = r-1) \wedge (e_{j-1})) \\ (1, z_j - r), & \text{otherwise} \end{cases} \tag{7}$$

Table 1 lists the details of our conversion rule. It can be seen that the conversion results in two digits in AHSD for each z_j. In the table, “x” stands for any value in $\{-1, 0, \ldots, 2(r-1)\}$, and $z_{-1} \equiv 0$.

3. Adjacent Digit Modification (ADM): The final sum digit s_j is obtained by

$$s_j = \mu_j + c_{j-1},\ 0 \le j < q, \tag{8}$$

where $c_{-1} = 0$. In the final result $S = (s_q, s_{q-1}, \ldots, s_0)$, $s_q = c_{q-1}$, and so S is in our AHSD representation.

Table 1. Conversion of z_j in AHSD(r).

z_j	z_{j-1}	c_j	μ_j
-1	x	0	-1
0	x	0	0
1	x	0	1
.	.	.	.
.	.	.	.
.	.	.	.
$r-2$	x	0	$r-2$
$r-1$	$< r-1$	0	$r-1$
	$\geq r-1$	1	-1
r	x	1	0
$r+1$	x	1	1
.	.	.	.
.	.	.	.
.	.	.	.
$2(r-1)$	x	1	$r-2$

This procedure is illustrated in Fig. 1. Apparently, the transfer digits are local to their adjacent digits, and all μ_j's and c_j's are generated concurrently. Also, all the final sum digits are generated in parallel. Therefore, the proposed AHSD addition can be performed in a constant time independent of the length of the operands (q).

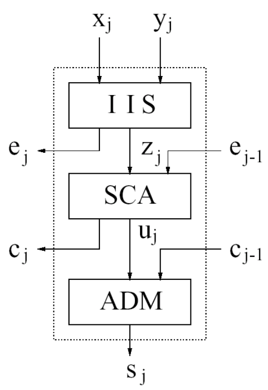

Fig. 1. The CF addition with estimate digits.

2.2.2 Double-transfer CF addition

There are two approaches to implementing the double-transfer CF addition: 1) the original approach, and 2) the simplified approach.

1. First Transfer Generation (FTG): Convert $x_j + y_j$ into a transfer digit, c_j, and an intermediate sum digit, μ_j, i.e.,

$$x_j + y_j = r \times c_j + \mu_j,\ 0 \leq j \leq q-1, \tag{9}$$

where

$$(c_j, \mu_j) = \begin{cases} (0, x_j + y_j), & \text{if } ((x_j + y_j) < r) \\ (1, x_j + y_j - r), & \text{otherwise.} \end{cases} \tag{10}$$

or

$$(c_j, \mu_j) = \begin{cases} (0, x_j + y_j), & \text{if } ((x_j + y_j) < r - 1) \\ (1, x_j + y_j - r), & \text{otherwise.} \end{cases} \tag{11}$$

Table 2 (a) lists the original conversion rules defined by (9) and (10), and Table 2 (b) shows the simplified conversion rules defined by (9) and (11). Apparently, $(x_j + y_j) \in \{-1, 0, \ldots, 2(r-1)\}$ if one of the two operands is in the canonical form. Note that $c_j \in \{0, 1\}$ and $\mu_j \in \{-1, 0, \ldots, (r-1)\}$ if we use the original approach, but $\mu_j \in \{-1, 0, \ldots, (r-2)\}$ if we use the simplified one. Further conversion is thus required.

Table 2. Conversion of $x_j + y_j$ in FTG.

(a) Original Approach			(b) Simplified Approach		
$x_j + y_j$	c_j	μ_j	$x_j + y_j$	c_j	μ_j
-1	0	-1	-1	0	-1
0	0	0	0	0	0
1	0	1	1	0	1
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
$r-2$	0	$r-2$	$r-2$	0	$r-2$
$r-1$	0	$r-1$	$r-1$	1	-1
r	1	0	r	1	0
$r+1$	1	1	$r+1$	1	1
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
$2(r-1)$	1	$r-2$	$2(r-1)$	1	$r-2$

2. Second Transfer Generation (STG): Convert $c_{j-1} + \mu_j$ again into another transfer digit, c'_j, and intermediate sum digit, μ'_j, i.e.,

$$c_{j-1} + \mu_j = r \times c'_j + \mu'_j,\ 0 \le j \le q-1, \tag{12}$$

where

$$(c'_j, \mu'_j) = \begin{cases} (0, c_{j-1} + \mu_j), & \text{if } (c_{j-1} + \mu_j) < r - 1 \\ (1, c_{j-1} + \mu_j - r), & \text{otherwise.} \end{cases} \tag{13}$$

Table 3 lists the conversion rules. It is easy to see that $(c_{j-1} + \mu_j) \in \{-1, 0, \ldots, r\}$ in the original approach, and $(c_{j-1} + \mu_j) \in \{-1, 0, \ldots, r-1\}$ in the simplified approach. Accordingly, $c'_j \in \{0, 1\}$ and $\mu'_j \in \{-1, 0, \ldots, (r-2)\}$.

Table 3. Conversion of $c_{j\text{-}1} + \mu_j$ in STG.

(a) Original Approach | **(b) Simplified Approach**

$(c_{j\text{-}1} + \mu_j)$	c'_j	μ'_j	$(c_{j\text{-}1} + \mu_j)$	c'_j	μ'_j
−1	0	−1	−1	0	−1
0	0	0	0	0	0
1	0	1	1	0	1
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
$r-2$	0	$r-2$	$r-2$	0	$r-2$
$r-1$	1	−1	$r-1$	1	−1
r	1	0			

3. Final Sum Modification (FSM): The final sum digit, s_j, is obtained by

$$s_j = c'_{j-1} + \mu'_j, \qquad 0 \le j < q, \tag{14}$$

where $c_{-1} = 0$. In the final result, $S = (s_q, s_{q\text{-}1}, \ldots, s_0)$, $s_q = c_{q\text{-}1}$, and so S is in AHSD(r) representation.

The above CF addition scheme is illustrated in Fig. 2. Note that the transfer digits and intermediate sum digits are obtained concurrently at each and every digit position. Also, in the last step (i.e., FSM), all the final sum digits are generated in parallel. Clearly, the proposed AHSD addition is real CF.

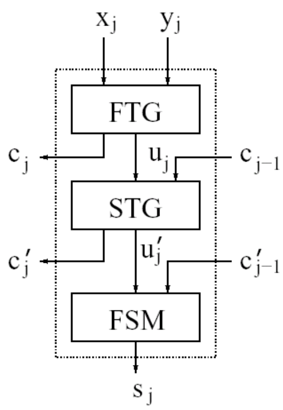

Fig. 2. The double-transfer CF addition procedure.

2.2.3 Comparison of CF addition schemes

The requirement that one of the operands be in canonical form preserves the consistency of our addition as described by Tables 1, 2, and 3. With the redundant digit − 1 introduced, the CF addition in AHSD systems can be achieved. Obviously, the AHSD addition is a quasi-closed addition. By Proposition 1, the proposed CF addition satisfies the properties of quasi-closure and subclosure. Note that although the proposed algorithms for implementing the CF addition in AHSD are similar to the three-stage limited-carry addition for GSD as reported in [9, 28], they are different. The latter addition approach can only be used in cases when $r = 2$ and $\rho = 1$, or $\rho = 2$ while $\alpha = 1$ or $\beta = 1$, both in GSD representation. Our proposed CF additions can be used when $\rho = 1$ and $r \geq 2$. Furthermore, if $r = 2^m$ our design can easily perform the simple conversion between ADSD and binary numbers. This makes our AHSD system a realistic one for fast arithmetic computation, especially for applications that need to perform massive amounts of addition operations. In such a case, the AHSD-to-binary conversion overhead can be amortized over a long sequence of CF additions.

We now make a brief review of the CF addition alternatives. The six approaches have been classified into three categories as follows [7, 9, 28].

1. *Ideal CF* (Type-I) addition: Fig. 3 shows the ideal CF addition scheme where the sum digit s_i depends only on the input digits x_i and y_i, not the carry-in. This is not possible for positional number representations.

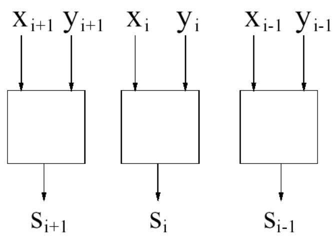

Fig. 3. Ideal CF addition (Type-I) [9].

2. *CF* addition: Any GSD system with $r > 2$ and $\rho > 2$, or $r > 2$ and $\rho = 2$, provided that $\alpha \neq 1$ and $\beta \neq 1$, supports CF addition. There are two different CF additions as shown below.

   (a) *Two-stage CF* (Type-II) addition (see Fig. 4)
   (b) *Single-stage CF* (Type-III) addition (see Fig. 5)

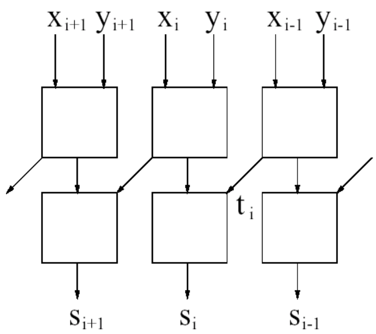

Fig. 4. Two-stage CF addition (Type-II) [9, 28].

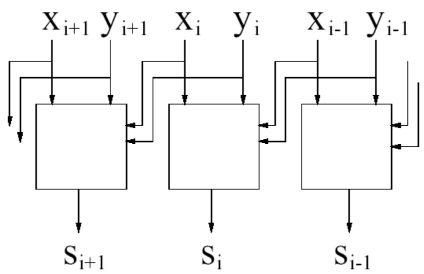

Fig. 5. Single-stage CF addition (Type-III) [28].

3. *Limited-carry* addition: The limited-carry addition is applicable for any GSD system with $r = 2$ and $\rho = 1$, or $\rho = 2$ and $\alpha = 1$ or $\beta = 1$.

(a) *Three-stage limited-carry* (Type-IV) addition with estimated transfer. The adder is shown in Fig. 6, where the estimate digit e_i is binary, and the transfer digit t_i is multivalued.
(b) *Three-stage limited-carry* (Type-V) addition with double transfers (see Fig. 7)
(c) *Two-stage limited-carry* (Type-VI) addition with parallel transfers (see Fig. 8)

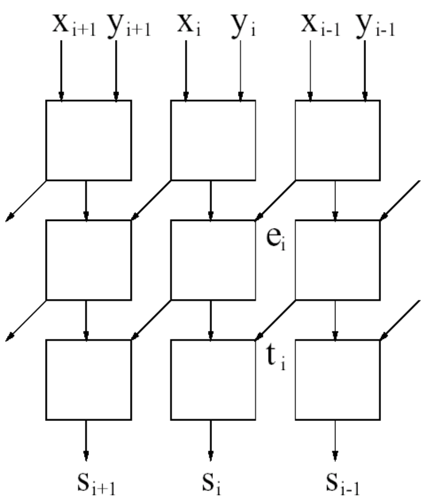

Fig. 6. Three-stage limited-carry addition with estimated transfer (Type-IV) [9, 28].

The comparison of AHSD with other CF addition schemes is shown in Table 4. From the table, we see that AHSD is the most flexible and compact representation since $\rho = 1$ for any radix $r \geq 2$. Lower redundancy means less storage needed to represent the numbers.

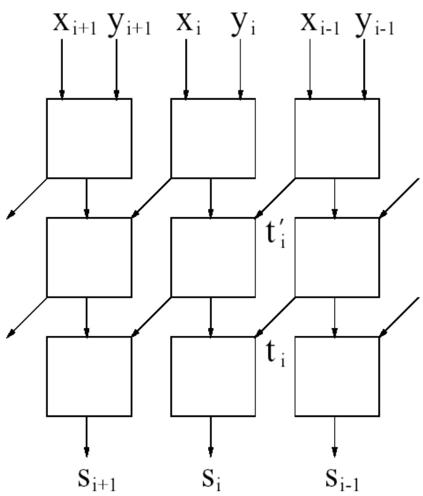

Fig. 7. Three-stage limited-carry addition with double transfer (Type-V) [28].

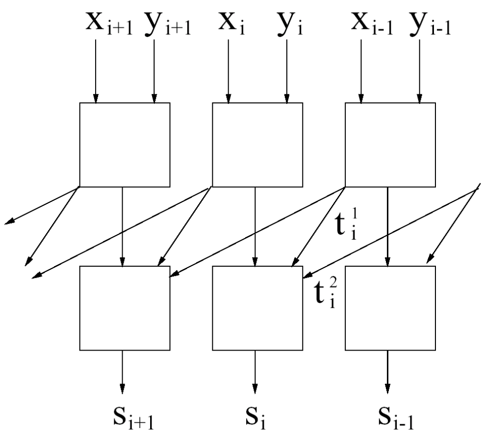

Fig. 8. Two-stage limited-carry addition with parallel transfers (Type-VI) [9, 28].

2.3 AHSD-to-Binary Conversion

Generally, a binary result is preferred in most applications. The AHSD-to-binary conversion is the most time-consuming procedure in AHSD-based systems, and its complexity is $O(\log\frac{n}{m})$. We describe two approaches to convert the final sum $S = (s_q, s_{q-1}, \ldots, s_0)$ from AHSD form to binary representation.

2.3.1 Direct conversion

The following three-step algorithm directly converts the final AHSD sum to its equivalent binary sum by using a dedicated circuit. It has the highest conversion speed and the smallest hardware overhead.

Table 4. Comparison of different CF addition schemes.

System	r	Digit Set	ρ	CF Scheme	Constraints
OSD [4]	≥ 3	$\{-\alpha, \ldots, -1, 0, 1, \ldots \alpha\}$ $\lfloor \frac{r}{2}+1 \rfloor \leq \alpha \leq (r-1)$	$\lfloor r/2 \rfloor + 1 \leq \rho \leq r-1$	Type-II	symmetric
MSD [4]	≥ 2	$\{-\frac{r}{2}, \ldots, 0, \ldots, \frac{r}{2}\}$	1	Type-V	even radix
		$\{-\frac{r\pm1}{2}, \ldots, 0, \ldots, \frac{r\mp1}{2}\}$			odd radix
GSD [7]	≥ 3	$\{-\alpha, \ldots, 0, \ldots, \beta\}$	≥ 3	Type-II Type-III	
[8]			2		$\alpha \neq 1$ and $\beta \neq 1$
[9]	2		1	Type-IV Type-V Type-VI	
[28]			2		$\alpha = 1$
					$\beta = 1$
AHSD	≥ 2	$\{-1, .., r-1\}$	1	Type-IV Type-V	

1. Decompose each digit s_j into s_j^+ and s_j^- :

$$s_j = s_j^+ + s_j^-, \quad 0 \leq j < q, \tag{15}$$

where $s_j \in S_r = \{-1, 0, 1, \ldots, r-1\}$, $s_j^+ \in D_r = \{0, \ldots, r-1\}$, and $s_j^- \in \{0, -1\}$. The digit s_q is equivalent to a binary carry-out bit c_{q-1} for the CF addition with estimate digits and ($c_{q-1} \vee c'_{q-1}$) for the double transfer CF addition, so there is no need to convert it. Using the *signed-number* representation, each s_j can be represented as an (m + 1)-bit number (z, v_{m-1}, …, v_1, v_0), as shown in Table 5, where z is the sign bit. Therefore, decomposition of s_j is equivalent to partitioning it into the sign bit z (for s_j^-) and the magnitude bits (v_{m-1}, …, v_1, v_0) (for s_j^+). Note that this can be done by simple wire routing, and no additional circuitry is required.

Table 5. Signed-number representation for AHSD(r).

S_r	z	v_{m-1}	…	v_1	v_0
-1	1	0	…	0	0
0	0	0	…	0	0
1	0	0	…	0	1
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
$r-2$	0	1	…	1	0
$r-1$	0	1	…	1	1

2. Add s_j^- and s_j^+ to generate the digit d_j in canonical form, i.e.,

$$d_j = s_j^+ + s_j^-, \quad 0 \leq j < q, \tag{16}$$

where $d_j \in D_r = \{0, 1, \ldots, r-1\}$. An implementation using the radix-r borrow-lookahead (BLA) generator is shown in Fig. 9. In the figure, n_j represents s_j^- and v_j stands for s_j^+. The function table of the jth-digit position in the direct AHSD(r)-to-binary converter is listed in Table 6. We define two auxiliary functions, the high-radix borrow generate function G_j and borrow propagate function P_j, as

$$G_j = n_j \bar{v}_j, \tag{17}$$

$$P_j = \bar{v}_j + n_j(v_j = 1). \tag{18}$$

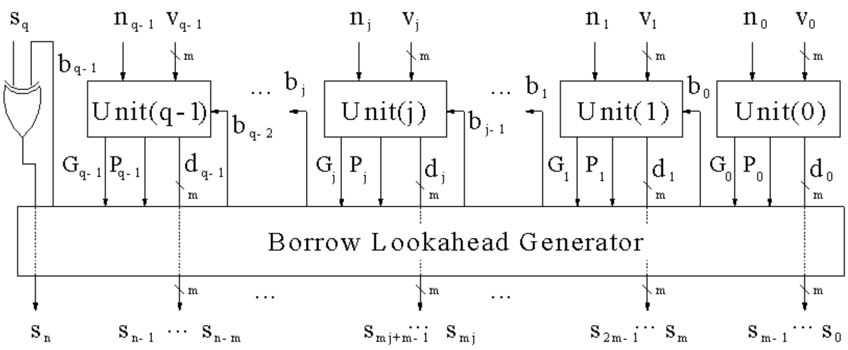

Fig. 9. The direct AHSD(r)-to-binary converter.

Table 6. The function for direct AHSD(r)-to-binary conversion.

b_{j-1}	n_j	v_j	d_j	b_j
0	0	v_j	v_j	0
0	1	$(v_j > 0)$	$(v_j - 1)$	0
0	1	0	$(r - 1)$	1
1	0	$(v_j > 0)$	$(v_j - 1)$	0
1	0	0	$(r - 1)$	1
1	1	$(v_j \geq 2)$	$(v_j - 2)$	0
1	1	$(v_j = 1)$	$(r - 1)$	1
1	1	0	$(r - 2)$	1

Thus, the *borrow* b_j at position j can be formulated in terms of G_j and P_j as

$$b_j = G_j + G_{j-1}P_j + G_{j-2}P_{j-1}P_j + \ldots + G_0P_1P_2\ldots P_j,\ 0 \leq j < q, \tag{19}$$

where $b_{-1} = 0$. All the digits d_j, $0 \leq j < q$, in canonical AHSD(r) form can then be generated in parallel:

$$d_j = b_j r + \{v_j - (b_{j-1} + n_j)\}. \tag{20}$$

3. According to Proposition 2, each d_j can be converted into its equivalent m-bit binary form without additional hardware.

2.3.2 Conversion by transforms

This algorithm is based on our previous work [29]. It converts the final AHSD(r) sum to binary by first transforming the final sum digit s_j to another digit set, then performing conventional radix-r addition and finally converting the result to its corresponding binary representation. Thanks to transformation of the digit sets, any existing good radix-r adder can be used in the converter. Although more area and latency may result, the main advantages of this algorithm are hardware reusability, design flexibility, and reduced design complexity. To pave the way to developing systematic conversion rules, we define a function $f(x) = (r - 1) - x$, $x \in S_r$, that maps S_r into $P_r = \{0, 1, \ldots, r\}$. Since $f(x)$ is bijective and self-inversed, $x = f(y) = f(f(x))$ for all $x \in S_r$, and $y = f(x) = f(f(y))$ for all $y \in P_r$. The mapping is summarized in Table 7.

Table 7. Mapping of $f(x)$ between S_r and P_r.

S_r	P_r
-1	r
0	$r-1$
1	$r-2$
.	.
.	.
.	.
$r-2$	1
$r-1$	0

By applying $f(x)$ to each digit in an AHSD(r) number, a $(q + 1)$-digit number $S = (s_q, s_{q-1}, \ldots, s_0)$ can be represented as

$$\begin{aligned} S &= \sum_{j=0}^{q} s_j r^j \\ &= \sum_{j=0}^{q} f(y_j) r^j \\ &= (r-1)\sum_{j=0}^{q} r^j - \sum_{j=0}^{q} y_j r^j \\ &= (r^{q+1} - 1) - Y \\ &= r^{q+1} + \overline{Y} \end{aligned} \tag{21}$$

In this equation, $\overline{Y}$ standards for the $(r - 1)$'s complement of Y. The $(q + 1)$-digit number Y is denoted by $Y = (y_q, \ldots, y_j, \ldots, y_0)_r$, where $y_j \in P_r$ for $j = 0, \ldots, q$. The number system that uses P_r as its digit set is called the *minimally redundant positive digit* (MRPD) number system. From (21), we obtain a three-stage algorithm for converting a $(q + 1)$-digit final sum $S = (s_q, s_{q-1}, \ldots, s_0)$ to binary form as follows.

1. AHSD-to-MRPD Conversion (AMC): Using the previously defined function $f(x)$, we convert each digit s_j of the final sum S in AHSD(r) to the corresponding digit y_j of a (q + 1)-digit number Y in MRPD(r), by the mapping as shown in Table 7. If we use the signed-number representation as shown in Table 5 to represent each digit s_j, the operation can be performed by an array of NOR gates in constant time. This is shown in the upper part of Fig. 10, which is the entire AHSD(r)-to-binary converter.

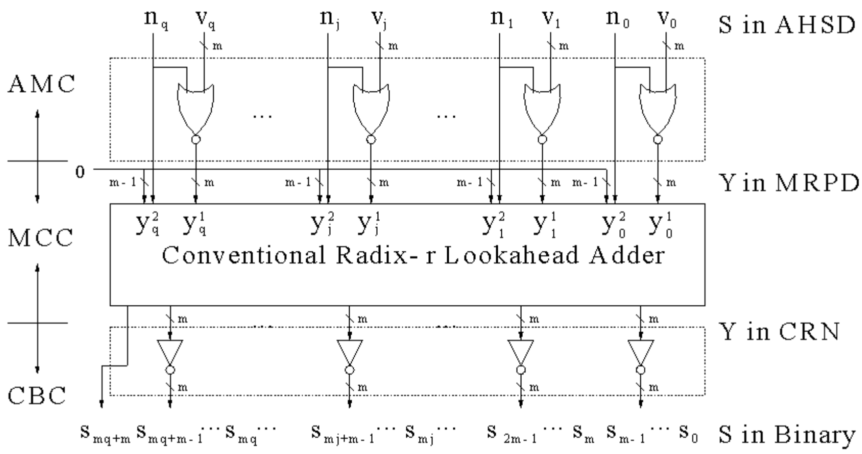

Fig. 10. AHSD(r)-to-binary conversion by transforms.

2. MRPD-to-CRN Conversion (MCC): In this stage, we transform the (q + 1)-digit number Y to its equivalent (q + 2)-digit conventional radix-r number (CRN), in which the additional digit is just one-bit. We first decompose Y into two operands, Y^1 and Y^2. Now $y_j = y_j^1 + y_j^2$, and both y_j^1 and y_j^2 are in $\{0, 1, \ldots, r-1\}$, i.e., Y^1 and Y^2 are conventional radix-r numbers. The radix-r number Y can be obtained by just adding Y^1 and Y^2 using a conventional radix-r adder, e.g., the carry-lookahead adder. This is the most time-consuming procedure in the entire AHSD-based system.
3. CRN-to-Binary Conversion (CBC): Assuming $r = 2^m$, we can directly convert each radix-r digit into an m-bit equivalent binary number. We then invert the entire vector to obtain the 1's-complement representation $\overline{Y}$, which is equal to the ($r - 1$)'s complement of Y. Finally, we add r^{q+1} to $\overline{Y}$. This stage can be implemented by an array of NOT gates in constant time, as shown in the lower part of Fig. 10.

Fig. 10 depicts the AHSD(r)-to-binary converter. The computation time for both AHSD-to-binary conversion schemes discussed above is proportional to $\log\frac{n}{m}$, so a higher radix results in higher computational speed.

The entire adder structures using the two approaches, including their input and output converters, are shown in Figs. 11 and 12, respectively. In these adders, both binary-to-AHSD converter (BAC) and CF adders operate in constant time, so the AHSD-to-binary converter (ABC) dominates the time complexity of the adders.

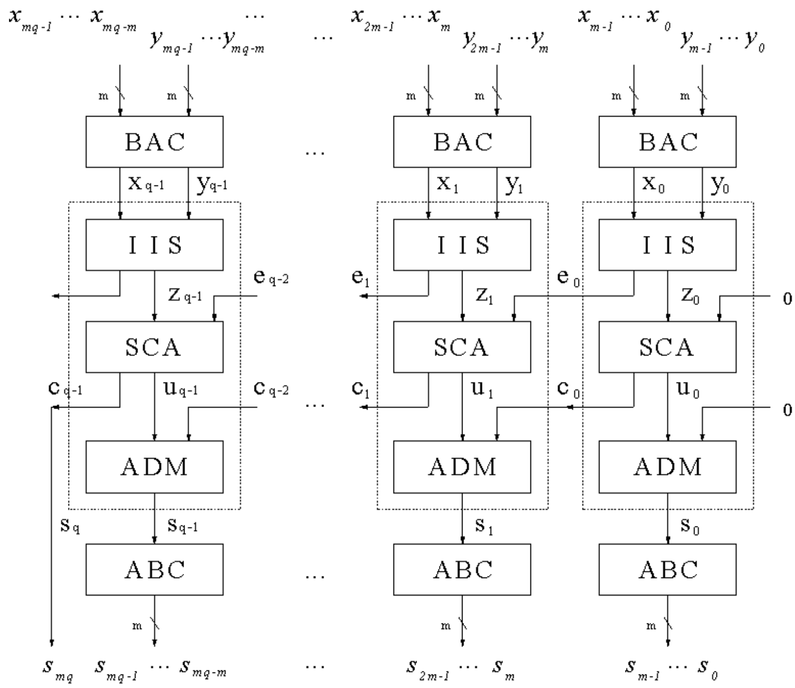

Fig. 11. CF adder with estimate digits.

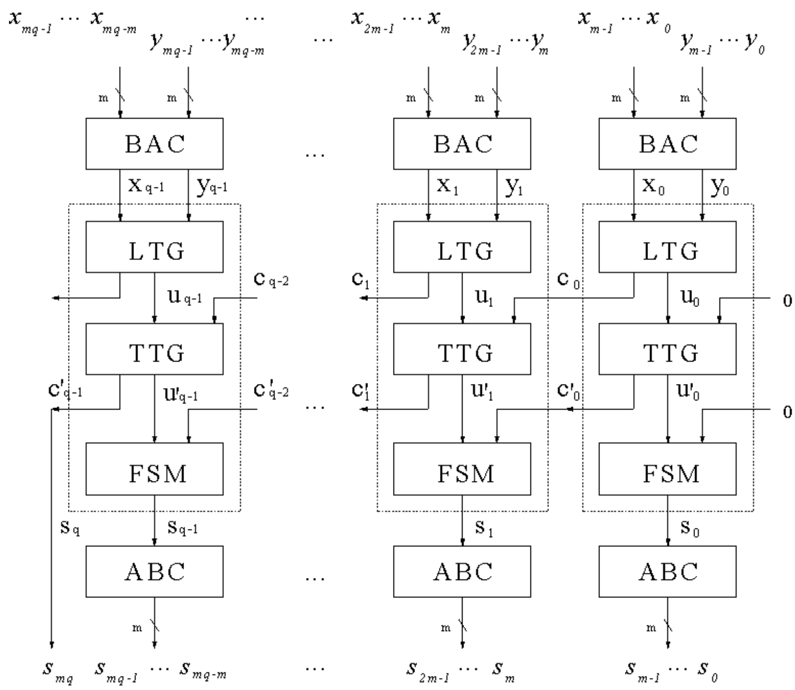

Fig. 12. Repeated-transfer CF adder.

2.3.3 Comparison of AHSD-to-binary conversion schemes

A comparison of the hardware flexibility, circuit reusability, and conversion time complexity between these two AHSD-to-binary conversion schemes is given in Table 8. In the table, a stands for the area of a full radix-r adder or subtractor. By (16), the direct conversion scheme basically performs the addition $s_j^- + s_j^+$, which is simpler than a full radix-r addition since one of its operands (s_j^-) is a binary signal. The hardware that implements this simple addition is called the radix-r partial adder. The computation time for the direct conversion is

$$T_{AB}^d = \log(q)t_{CGP}, \tag{22}$$

where T_{AB}^d, t_{CGP}, and q denote the processing time for AHSD-to-binary direct conversion, carry generation per digit of the radix-r partial adder, and digit $q = \lceil n/m \rceil$, respectively. Note that the processing time is computed based on the well-known result that the average carry (or borrow) chain in a q-digit addition is of length $\log(q)$.

Table 8. Comparison of AHSD-to-binary conversion schemes.

Features	Direct Conversion	Conversion by Transforms
Converter Circuitry	Dedicated partial radix-r adder	Any radix-r adder + (q + 1) (NOR + NOT)
Area	$< a$	$> a$
Time Complexity	$O(\log\lceil n/m \rceil)$	$O(\log(\lceil n/m \rceil + 1))$
Conversion Speed	Fast	Slow
Hardware Reusable	No	Yes
Design Complexity	High	Low

A fast (q + 1)-digit radix-r adder (e.g., carry-lookahead adder, carry-skip adder, etc.) also can be used to implement the AHSD-to-binary conversion as shown in Fig. 10. The computation time for such conversion by transform is

$$T_{AB}^t = t_{NOR} + \log(q+1)t_{CGF} + t_{NOT}, \tag{23}$$

where T_{AB}^t, t_{NOR}, t_{CGF}, t_{NOT}, and q denote the processing time for AHSD-to-binary conversion by transform, NOR operation, carry generation per digit of a conventional radix-r adder, NOT operation, and digit $q = \lceil n/m \rceil$, respectively. It should be noted that the carry generation time t_{CGF} is larger than t_{CGP}, because a partial radix-r adder is less complex, requiring fewer gates to implement. This also implies that the direct conversion is faster than the conversion by transforms.

3. APPLICATION EXAMPLES

3.1 AHSD(4) CF Adder

The radix $r = 2^m$ should be used in practical AHSD-based implementations to achieve the simplest conversion from and to binary. In this section we use the AHSD(4) number system as an example to show the proposed CF addition schemes.

Consider the two binary numbers $X = (11111111)_2$ and $Y = (00000001)_2$ that are to be added. Apparently the longest carry propagation chain occurs in this case if an ordinary ripple-carry adder is used. By using the proposed addition algorithms, we can generate the sum without any carry propagation in the CF adders. An entire AHSD(4) CF adder can be implemented using the following procedure.

1. Binary-to-AHSD(4) Conversion
   By Proposition 2, any binary number can be partitioned, from the LSB to the MSB, into 2-bit blocks. Each block can then be converted directly into its corresponding canonical AHSD(4) digits concurrently. The two binary numbers $X = (11111111)_2$ and $Y = (00000001)_2$ are thus converted to the canonical AHSD(4) numbers $X = (3333)_4$ and $Y = (0001)_4$. As discussed above, the three-bit signed number (n, v_1, v_0) can be used to represent an AHSD(4) digit.
2. CF Adder Based on AHSD(4)
   There are two CF addition schemes that can be used to implement the CF adder. The example based on the addition by estimated digits is shown in Fig. 13. In the IIS stage, the two AHSD(4) numbers are added to generate the digit sum $Z = (3334)_4$. In the SCA stage, each digit of Z is converted to an intermediate transfer digit and an intermediate sum digit according to Table 9. The final sum $S = (10000)_4$ in AHSD(4) representation is generated by the ADM stage by adding these two interim digits.

	1 1	1 1	1 1	1 1	X	Binary	BAC
+	0 0	0 0	0 0	0 1	Y		
	3	3	3	3	X	AHSD(4)	
+	0	0	0	1	Y		
	3	3	3	4	Z	I I S	CPF Adder
1	1	1	1		C	SCA	
	-1	-1	-1	0	u		
1	0	0	0	0	S	ADM	

Fig. 13. Example for addition by estimated digits.

Table 9. Conversion rules for AHSD(4).

z_j	z_{j-1}	c_j	μ_j
− 1	x	0	− 1
0	x	0	0
1	x	0	1
2	x	0	2
3	< 3	0	3
	≥ 3	1	− 1
4	x	1	0
5	x	1	1
6	x	1	2

The addition procedure for AHSD(4) based on the repeated-transfer algorithm discussed above is shown in Fig. 14. In the FTG stage, the two AHSD(4) numbers are added to generate an interim transfer digit and an interim sum digit, as listed in Table 10. During the STG stage, the previous interim transfer digit and its corresponding interim sum digit are added to form a new interim transfer digit and a new interim sum digit as listed in Table 11. Each digit of the final sum $S = (10000)_4$ is produced in the FSM stage by adding these two interim digits.

	1 1	1 1	1 1	1 1	X	Binary	BAC
+	0 0	0 0	0 0	0 1	Y		
	3	3	3	3	X	AHSD(4)	
+	0	0	0	1	Y		
0	0	0	1		c	FTG	CPF Adder
	3	3	3	0	u		
1	1	1	0		c'	STG	
	-1	-1	0	0	u'		
1	0	0	0	0	S	FSM	

Fig. 14. Repeated-transfer addition example.

Table 10. Conversion of $x_j + y_j$ in FTG.

$x_j + y_j$	c_j	μ_j
− 1	0	− 1
0	0	0
1	0	1
2	0	2
3	0	3
4	1	0
5	1	1
6	1	2

Table 11. Conversion of $(c_{j-1} + \mu_j)$ in STG.

$(c_{j-1} + \mu_j)$	c'_j	μ'_j
− 1	0	− 1
0	0	0
1	0	1
2	0	2
3	1	− 1
4	1	0

3. AHSD(4)-to-Binary Conversion

There are also two AHSD(4)-to-binary conversion schemes. The direct AHSD(4)-to-binary conversion scheme was shown in Fig. 9. A gate-level design of the jth bit slice of the AHSD(4)-to-binary converter is shown in Fig. 15. It can be seen that b_j can easily be generated according to (19). The other approach converts the final sum to binary form by digit set conversion and conventional radix-4 addition, as shown in Fig. 16.

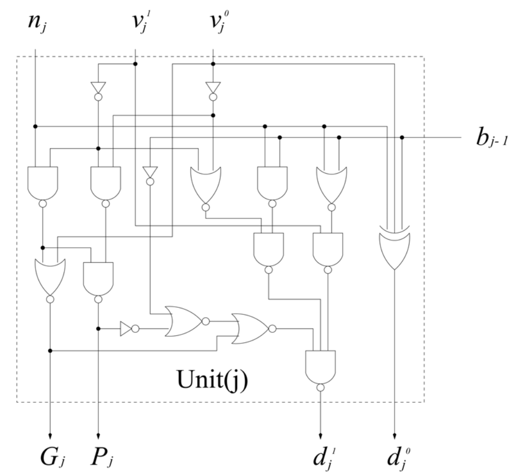

Fig. 15. A bit slice of the AHSD(4)-to-binary converter.

1	0	0	0	0	S	AHSD(4)	
2	3	3	3	3	Y	MRPD(4)	
2	3	3	3	3	Y^1	CR(4)	
0	0	0	0	0	Y^2		
2	3	3	3	3	Y	CR(4)	ABC
0 1 0	1 1	1 1	1 1	1 1	Y	Binary	
1 0 1	0 0	0 0	0 0	0 0	$\overline{Y}$		
0 0 1	0 0	0 0	0 0	0 0	S	Binary	

Fig. 16. Example for conversion by transforms.

We first convert $S = (10000)_4$ to $Y = (23333)_4$ in MRPD(4) by the mapping function specified in Table 12. Then we transform Y into its equivalent conventional radix-4 number $Y = (23333)_4$ by partitioning $Y = (23333)_4$ into $Y^1 = (23333)_4$ and $Y^2 = (00000)_4$, and then performing the conventional radix-4 addition. Note that since $r = 4$, the conversion seems unnecessary, i.e., we may skip the first two stages. However, for most other cases the first two stages are essential. In the final stage, each digit in the conventional radix-4 number $Y = (23333)_4$ is first converted to its two-bit binary equivalent with one additional leading bit as explained previously, and then the binary form $Y = (01011111111)_2$ is converted to its 1's-complement $\overline{Y}(10100000000)_2$. Finally, the bit 2^{10}, which is 4^5, is added to the MSB of $\overline{Y}$ to generate the desired binary result $S = (00100000000)_2$.

Table 12. Mapping of *f*() in radix-4.

S_r	P_r
− 1	4
0	3
1	2
2	1
3	0

3.2 Sequential Adder

A sequential adder calculates sequentially the sum of p numbers: X_0, X_1, …, X_{p-1}. The sum can be expressed as

$$S = \sum_{i=0}^{p-1} X_i = (...((X_0 + X_1) + X_2) + ...) + X_{p-1}. \tag{24}$$

This sequential adder can be realized by using a CF adder, including the binary-to-AHSD converter (BAC), AHSD-to-binary converter (ABC), and a latch as shown in Fig. 17. Initially, switch SW1 selects input A for the CF Adder, and switch SW2 places the CF Adder output at port C for storing in the latch. At this moment, the first two inputs (X_0 and X_1) are loaded and added to generate the temporary sum S_1, which is stored in the latch. After that, SW1 selects input B, so S_1 is then added by X_2 to generate the temporary sum S_2 and is subsequently stored in the latch. We repeat the process for S_i, i = 2, …, p − 2. In the final iteration, SW2 places the CF Adder output at port D, and the final sum S is produced by adding the temporary sum S_{p-2} to X_{p-1}.

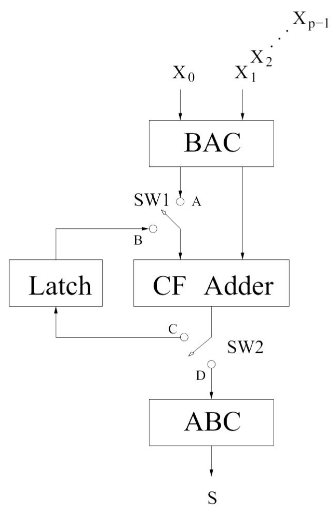

Fig. 17. An AHSD(r) sequential adder design.

The time to perform p additions sequentially based on AHSD(r) is

$$T_{add} = t_{BA} + (p - 2)[max(t_{BA}, t_L)] + (p - 1)t_{CF} + t_{AB}, \tag{25}$$

where t_L, t_{BA}, t_{CF}, and t_{AB} are the times for latching, binary-to-AHSD(r) conversion, CF addition, and AHSD(r)-to-binary conversion, respectively. Since t_L, t_{BA}, and t_{CF} are all constant, regardless of the input word length, t_{AB} dominates the time complexity. Speedup results from adding a large number of inputs. If p is large, the conversion time can be small in comparison with other terms and can be neglected, so the computation time can be approximated by

$$T_{add}^{p} \simeq p[\max(t_{BA}, t_L) + t_{CF}]. \tag{26}$$

On the other hand, if the word length $q >> p$, then the computation time can be approximated by

$$T_{add}^{q} \simeq t_{AB} \simeq \log(q)t_{CG}, \tag{27}$$

where t_{CG} is the per-digit carry generation time of the AHSD-to-binary conversion.

In comparison, the time to perform p additions sequentially based on the traditional radix-r number representation is

$$T_{add}^{trad} = (p-1)[\log(q)t_{CG}^{trad}], \tag{28}$$

where t_{CG}^{trad} is the per-digit carry generation time of a traditional radix-r adder. It can be seen that sequential addition based on AHSD(r) is superior to that based on traditional radix-r number representation, especially for a large amount of successive additions and/or subtractions, e.g., in multiplications and divisions.

4. CONCLUSIONS

In this paper two novel CF addition schemes based on an asymmetric high-radix signed-digit (AHSD) number system have been proposed. Although the proposed algorithms for implementing the CF addition in AHSD are similar to the three-stage limited-carry addition for GSD, the latter can only be used when $r = 2$ and $\rho = 1$, or $\rho = 2$ with $\alpha = 1$ or $\beta = 1$. Our AHSD CF additions can be used when $\rho = 1$ and $r \geq 2$. Additionally, if $r = 2^m$ for any positive integer m, the interface between AHSD and the binary number system is shown to be easily implementable. Two AHSD-to-binary conversion algorithms have also been proposed, one used to achieve high speed with smaller area, the other for hardware reusability and design flexibility. Since both the binary-to-AHSD converter and CF adder operate in a constant time, we can conclude that the AHSD-to-binary converter dominates the performance of the entire AHSD-based system. Hence, CF adders based on an AHSD number representation are especially suited to applications that process massive addition operations with only few input and output conversions needed. The time complexity of the entire AHSD CF adder is $O(\log(q))$ =

$O(\log(n/m))$. We also need additional studies including not only the implementation of CF adder with different design skills, but also the applications of various CF adders to design fast arithmetic. Systematic work in signed-digit representations remains to be done.

ACKNOWLEDGEMENT

The Simplified Approach in section 2.2.2 was suggested by one of the anonymous reviewers. We thank him for the suggestion. We also thank all the reviewers for their valuable comments and suggestions.

REFERENCES

1. V. Kantabutra, "Designing optimum one-level carry-skip adders," *IEEE Transactions on Computers*, Vol. 42, 1993, pp. 759-764.
2. K. Hwang, *Computer Arithmetic: Principles, Architecture, and Design*, John Wiley & Sons, New York, 1979.
3. I. Koren, *Computer Arithmetic Algorithms*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1993.
4. A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," *IEEE Transactions on Electronic Computers*, Vol. EC-10, 1961, pp. 389-400.
5. F. J. Taylor, "A VLSI residue arithmetic multiplier," *IEEE Transactions on Computers*, Vol. 31, 1982, pp. 540-546.
6. B. Parhami, "Carry-free addition of recoded binary signed-digit numbers," *IEEE Transactions on Computers*, Vol. 37, 1988, pp. 1470-1476.
7. B. Parhami, "Generalized signed-digit number systems: a unifying framework for redundant number representations," *IEEE Transactions on Computers*, Vol. 39, 1990, pp. 89-98.
8. B. Parhami, "On the implementation of arithmetic support functions for generalized signed-digit number systems," *IEEE Transactions on Computers*, Vol. 42, 1993, pp. 379-384.
9. B. Parhami, "Comments on 'High-speed area-efficient multiplier design using multiple-valued current-mode circuits'," *IEEE Transactions on Computers*, Vol. 45, 1996, pp. 637-638.
10. S. Kawahito, M. Ishida, M. Kameyama, and T. Higuchi, "High-speed area-efficient multiplier design using multiple-valued current-mode circuits," *IEEE Transactions on Computers*, Vol. 43, 1994, pp. 34-42.
11. S. Kawahito, M. Kameyama, T. Higuchi, and H. Yamada, "A 32 × 32-bit multiplier using multiple-valued MOS current-mode circuits," *IEEE Journal of Solid-State Circuits*, Vol. 23, 1988, pp. 124-132.
12. M. Kameyama, S. Kawahito, and T. Higuchi, "A multiplier chip with multiple-valued bidirectional current-mode logic circuits," *IEEE Computer*, Vol. 21, 1988, pp. 43-56.
13. A. Vandemeulebroecke, E. Vanzieleghem, T. Denayer, and P. G. A. Jespers, "A new carry-free division algorithm and its application to a single-chip 1024-b RSA

processor," *IEEE Journal of Solid-State Circuits*, Vol. 25, 1990, pp. 748-756.
14. N. Takagi, H. Yasuura, and S. Yajima, "High-speed VLSI multiplication algorithm with a redundant binary addition tree," *IEEE Transactions on Computers*, Vol. 34, 1985, pp. 789-796.
15. H. Makino, Y. Nakase, H. Morinaka, H. Shinohara, and K. Mashiko, "An 8.8-ns 54 × 54-bit multiplier with high speed redundant binary architecture," *IEEE Journal of Solid-State Circuits*, Vol. 31, 1996, pp. 773-782.
16. R. Hartly, "Subexpression sharing in filters using canonic signed digit multipliers," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, Vol. 43, 1996, pp. 677-688.
17. R. Woods and J. McCanny, "Design of a high-performance IIR digital filter chip," *IEE Proceedings Part E.*, Vol. 139, 1992, pp. 195-202.
18. H. R. Srinivas, K. K. Parhi, and L. A. Montalvo, "Radix 2 division with over-redundant quotient selection," *IEEE Transactions on Computers*, Vol. 46, 1997, pp. 85-92.
19. M. Ercegovac and T. Lang, "On-the-fly conversion of redundant into conventional representations," *IEEE Transactions on Computers*, Vol. 36, 1987, pp. 895-897.
20. S. M. Yen, C. S. Laih, C. H. Chen, and J. Y. Lee, "An efficient redundant-binary number to binary number converter," *IEEE Journal of Solid-State Circuits*, Vol. 27, 1992, pp. 109-112.
21. H. R. Srinivas and K. K. Parhi, "A fast VLSI adder architecture," *IEEE Journal of Solid-State Circuits*, Vol. 27, 1992, pp. 761-767.
22. G. M. Blair, "The equivalence of twos-complement addition and the conversion of redundant-binary to twos-complement numbers," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, Vol. 45, 1998, pp. 669-671.
23. J. E. Roberson, "A new class of digital division methods," *IEEE Transactions on Computers*, Vol. 7, 1958, pp. 218-222.
24. O. L. MacSorley, "High-speed arithmetic in binary computers," in *Proceedings of IRE*, Vol. 49, 1961, pp. 67-91.
25. C. Y. Chow and J. E. Robertson, "Logical design of a redundant binary adder," in *Proceeding of Symposium on Computer Arithmetic*, 1978, pp. 109-115.
26. S. H. Shieh, B. H. Lin, and C. W. Wu, "Carry-propagation-free adder based on an asymmetric high-radix signed-digit number system," in *Proceedings of International Computer Symposium* (*ICS*), 1996, pp. 199-204.
27. S. H. Shieh and C. W. Wu, "Carry-free adder design using asymmetric high-radix signed-digit number system," in *Proceedings of 11th VLSI Design/CAD Symposium*, 2000, pp. 183-186.
28. B. Parhami, "Implementation alternatives for generalized signed-digit addition," in *Proceedings of IEEE 28th Asilomar Conference on Signals, Systems, and Computers*, 1994, pp. 157-161.
29. S. H. Shieh and C. W. Wu, "On *r*'s-complement addition and conversion from asymmetric high-radix signed-digit numbers to binary numbers," in *Proceedings of 10th VLSI Design/CAD Symposium*, 1999, pp. 161-164.

Shao-Hui Shieh (謝韶徽) received the B.S. degree in electronic engineering and the M.S. degree in automatic control engineering from Feng Chia University, Taichung, Taiwan, in 1977 and 1981, respectively, and the Ph.D. degree in electrical engineering form National Tsing Hua University, Hsinchu, Taiwan, in 2003. Since 1983 he has been with the Department of Electronic Engineering, National Chinyi Institute of Technology (NCIT), Taichung, Taiwan, where he is currently an associate professor. His research interests include VLSI design and testing, computer arithmetic and architecture, and redundant number representations.

Cheng-Wen Wu (吳誠文) received the BSEE degree in 1981 from National Taiwan University, Taipei, Taiwan, and the MS and Ph.D. degrees in electrical and computer engineering, in 1985 and 1987, respectively, from the University of California, Santa Barbara. From 1981 to 1983, he was an ensign instructor at the Chinese Naval Petty Officers' School of Communications and Electronics, Tsoying, Taiwan. From 1983 to 1984, he was with the Information Processing Center of the Bureau of Environmental Protection, Executive Yuan, Taipei, Taiwan. From 1985 to 1987 he was a post-graduate researcher at the Center for Computational Sciences and Engineering at UCSB. Since 1988 he has been with the Department of Electrical Engineering, National Tsing Hua University (NTHU), Hsinchu, Taiwan, where he is currently a professor. He also has served as the director of the university's Computer and Communications Center from 1996 to 1998, and the director of the university's Technology Service Center from 1998 to 1999. From August 1999 to February 2000, he was a visiting faculty of the ECE Department, UCSB. Since August 2000, he has been the Chair of the EE Department of NTHU. He also is the Director of the NTHU IC Design Technology Center.

Dr. Wu was the Technical Program Chair of the IEEE Asian Test Symposium (ATS'96), and the General Chair of ATS'00. He is an Associate Editor for the Journal of the Chinese Institute of Electrical Engineers (JCIEE) and a Guest Editor of the JCIEE Special Issue on Design and Test of System-on-Chip, and was a Guest Editor of the Journal of Information Science and Engineering (JISE), Special Issue on VLSI Testing. He received the Distinguished Teaching Award from NTHU in 1996, the Outstanding Electrical Engineering Professor Award from the Chinese Institute of Electrical Engineers (CIEE) in 1997, the Distinguished Research Awards from the Chinese Institute of Electrical Engineers (CIEE) in 1997, the Distinguished Research Awards from National Science Council in 2000 and 2002, the Industrial Collaboration Award from the Ministry of Education in 2001, the Best Paper Award of the 2002 IEEE International Workshop on Design & Diagnostics of Electronic circuits & Systems, the Best Paper Award of the 2003 IEEE Asia & South Pacific Design Automation Conference (ASP-DAC), and the Special Feature Award of the 2003 ASP-DAC University IC Design Contest. He is interested in design and test of high performance VLSI circuits and systems. Dr. Wu is a life member of CIEE and a senior member of IEEE.