Microelectronics Journal 42 (2011) 1090–1097

Contents lists available at ScienceDirect

Microelectronics Journal

journal homepage: www.elsevier.com/locate/mejo

Efficient canonic signed digit recoding

Gustavo A. Ruiz *, Mercedes Granda

Departamento de Electrónica and Computadores, Facultad de Ciencias, Avda. de Los Castros s/n, Universidad de Cantabria, 39005 Santander, Spain

ARTICLE INFO

Article history:
Received 14 December 2010
Received in revised form
14 June 2011
Accepted 17 June 2011
Available online 6 July 2011

Keywords:
Canonic signed digit (CSD)
Digital arithmetic
Minimal signed digit
Signed digit representation

ABSTRACT

In this work novel-efficient implementations to convert a two's complement binary number into its canonic signed digit (CSD) representation are presented. In these CSD recoding circuits two signals, ***H*** and ***K***, functionally equivalent to two carries are described. They are computed in parallel reducing the critical path and they possess some properties that lead to a simplification of the algebraic expressions minimizing the overall hardware implementation. As a result, the proposed circuits are highly efficient in terms of speed and area in comparison with other counterpart previous architectures. Simulations of different configurations made over standard-cell implementations show an average reduction of about 55% in the delay and 29% in the area for a ripple-carry scheme, 47% in the delay and 17% the area in a carry look-ahead scheme, and 36% in the delay and 31% the area in a parallel prefix scheme.

1. Introduction

The canonical signed digit (CSD) representation is one of the existing signed digit (SD) representations with unique features which make it useful in certain DSP applications focusing on low-power, efficient-area and high-speed arithmetic [1]. The CSD code is a ternary number system with the digit set $\{\bar{1}\ 0\ 1\}$, where $\bar{1}$ stands for 1. Given a constant, the corresponding CSD representation is unique and has two main properties: (1) the number of nonzero digits is minimal, and (2) no two consecutive digits are both nonzero, that is, two nonzero digits are not adjacent. The first property implies a minimal Hamming weight, which leads to a reduction in the number of additions in arithmetic operations. The second property provides its uniqueness characteristic. However, if this property is relaxed, this representation is called the minimal signed digit (MSD) representation, which has as many nonzeros as the CSD representation, but which provides multiple representations for a constant [2,3].

CSD representation has proven to be useful for the design and implementation of digital filters such as the area-efficient programmable FIR digital filter architecture in Ref. [4], Chebyshev FIR filter design with some constrains in terms of hardware and frequency domain in Ref. [5], low-complexity algorithms for design filters in Ref. [6], 2D FIR and IIR filter design in Ref. [7] and the digit-serial CSD filter FPGA architecture for image conversion proposed in Ref. [8]. It has also been used in the reduction of the complexity of digital filters [9–11] or matrix multipliers [12] applying shared subexpression methods. CSD code has been largely exploited to implement efficient multipliers [13–15]. It enables the reduction of the number of partial products that must be calculated fast, and also low-power consumption and low area structure of a multiplier for DSP applications [16] or self-timed circuits [17]. In fixed-width multipliers, CSD succeeded in reducing the mean square error [18] or the compensation error using efficient sign extension [19]. Finally, other applications of CSD coding in reversible image color transforms [20], Montgomery exponentiation [21,22] or vector rotational CORDIC [23] have been proposed.

Many researchers have addressed the question of CSD recoding to convert two's complement into CSD code. Already in 1960, Reitwiesner proposed an algorithm for converting two's complement numbers to a minimum weight radix-2 (binary) signed digit representation [24]. From the practical point of view, the traditional approach to generate the CSD representation uses look-up table [25,26]. Here, there is a great similarity in the carry definition used in CSD recoding and in conventional adders, which suggest that the implementation of fast CSD converters should be based on well-known structures of classical adders. However, some algorithms to convert two's complement into CSD numbers try to reduce the computational complexity [27,28], but are not suitable for hardware implementation. Other hardware approaches propose the bypass method [29], fast carry look-ahead circuits [30] or parallel prefix schemes [31,32] to reduce hardware but they only focus on carry optimization without considering the overall CSD recoding. All of these algorithms generate the CSD code recursively from the least significant bit (LSB) to the most significant bit (MSB). However, in some applications, such as the computation of exponentiation, the conversion from MSB to LSB brings some advantages.

* Corresponding author.
E-mail addresses: ruizrg@unican.es (G.A. Ruiz), grandam@unican.es (M. Granda).

0026-2692/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.mejo.2011.06.006

Okeya et al. [33] proposed an algorithm to carry out this conversion but it requires additional memory to store some precomputed elements. Indeed, efficient MSB-to-LSB algorithms yield a MSD representation [34,35] but not a CSD one because of having consecutive nonzero digits.

This paper presents novel, efficient standard cell-based implementations to convert a two's complement binary number into its CSD representation based on two signals, ***H*** and ***K***, functionally equivalent to two carries. These signals were already defined in the implementation of 3X terms for radix-8 encoding [36], but here they are used for the CSD recoding in a more resourceful way. Implementations in a 130 nm standard cell CMOS technology of previously published architectures have been compared with the proposed scheme demonstrating that the circuits are highly efficient in area and speed. The remainder of this paper is organized as follows. In Section 2, a brief introduction and definitions related to CSD recoding are presented. New compact algebraic expressions for optimal CSD recoding in terms of signals ***H*** and ***K*** and their application in the efficient implementation of CSD recoders for different configurations are described in Section 3. Simulations and comparisons are listed in Section 4. Finally, the conclusions are stated in Section 5.

2. CSD recoding

The CSD representation of an integer number is a signed and unique digit representation that contains no adjacent nonzero digits. Given an n-digit binary unsigned number $\boldsymbol{X}=\{x_0, x_1, \ldots, x_{n-1}\}$ expressed as

$$\boldsymbol{X}=\sum_{i=0}^{n-1} x_i\cdot 2^i, \quad x_i\in\{0,1\} \tag{1}$$

then the $(n+1)$-digit CSD representation $\boldsymbol{Y}=\{y_0, y_1,\ldots, y_n\}$ of $\boldsymbol{X}$ is given by

$$\boldsymbol{Y}=\sum_{i=0}^{n-1} x_i\cdot 2^i=\sum_{i=0}^{n} y_i\cdot 2^i, \quad y_i\in\{\bar{1},0,1\} \tag{2}$$

The condition that all nonzero digits in a CSD number are separated by zeros implies that

$$y_{i+1}\cdot y_i=0, \quad 0\le i\le n-1 \tag{3}$$

From this property, the probability that a CSD n-digit has a nonzero value [13,26] is given by

$$P(|y_i|=1)=\frac{1}{3}+\frac{1}{9n}\left[1-\left(-\frac{1}{2}\right)^n\right] \tag{4}$$

As n becomes large, this probability tends to 1/3 while this probability becomes 1/2 in a binary code. Using this property, the number of additions/subtractions is reduced to a minimum in multipliers [14–17] and, as a result, an overall speed-up can be achieved.

The adoption of a ternary number system adds some flexibility to the CSD representation, since it allows the number of nonzero digits to be minimized, but it requires that each digit y_i must be encoded over two bits $\{y_i^s, y_i^d\}$. Table 1 shows the two most frequently used encodings in practice [1]. Encoding 1 can be viewed as a two's representation. However, encoding 2 is preferable since it satisfies the following relation

$$y_i=y_i^d-y_i^s \tag{5}$$

where y_i^s represents the sign bit and y_i^d the data bit. This encoding also allows an additional valid representation of 0 when $y_i^s=1$ and $y_i^d=1$, which is useful in some arithmetic implementations. In the whole paper, this encoding is used.

Table 1
Two most encodings used in the binary representation of a CSD digit ($\bar{1}$ stands for -1). In this paper, encoding 2 is used.

y_i	Encoding 1 $y_i^s y_i^d$	Encoding 2 $y_i^s y_i^d$
0	00	00
1	01	01
$\bar{1}$	11	10

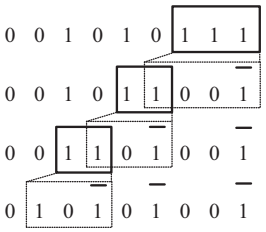

Fig. 1. Conversion process from binary to CSD code.

Hashemian [27] presented the binary coded CSD (BCSD) number to avoid extra data word representation. The BCSD recoding is based on a simple binary representation $\boldsymbol{B}=\{b_0, b_1,\ldots,b_{n-1}\}$ of a CSD code, which uses the same number of bits as the original two's complement representation. It takes advantage of the CSD property, in which no two adjacent digits can both be nonzero, to assign the next position of each nonzero CSD digit as sign bit while maintaining the same size data word. Thus, if the bit i in a CSD code is nonzero, $y_i\neq 0$ (which means that $y_{i+1}=0$), then the bit i is nonzero in the BCSD code, $b_i\neq 0$, and the following bit b_{i+1} acts as a sign bit: $b_{i+1}=0$ means y_i is positive and $b_{i+1}=1$ means y_i is negative. As a result, $y_i=0$ is encoded as $b_i=0$, $y_i=1$ as $b_{i+1}b_i=01$ and $y_i=\bar{1}$ as $b_{i+1}b_i=11$. A simple conversion between a CSD coding and BCSD coding is

$$\begin{aligned} b_i &= y_i^d+y_{i-1}^s \\ b_{i+1} &= y_{i+1}^d+y_i^s \end{aligned} \tag{6}$$

Since this conversion can be obtained by a simple operation, in this paper the two-bit encoding representation is used to make its reading and understanding easier, and, without loss of generality, it can be straightforwardly extended to other representations.

The conversion from a binary representation to CSD representation is mostly based on the following identity

$$2^{i+j-1}+2^{i+j-2}+\ldots+2^i=2^{i+j}-2^i \tag{7}$$

This means that a string of 1s can be replaced by a 1, followed by 0s, followed by a $\bar{1}$. Isolated 1s are left unchanged, but isolated 0s are re-examined in such a way that, after applying Eq. (7), pairs of type $1\bar{1}$ are changed to 01. For example, the binary number $(001010111)_2$ is equivalent in a CSD representation to $010\bar{1}0\bar{1}00\bar{1}$; the encoding process is graphically shown in Fig. 1. Traditionally, this encoding is performed from LSB to MSB using two adjacent bits and a carry signal according to the recoding algorithm shown in Table 2 [25,26]. Here, the carry-out $c_i=1$ if and only if there are two of three 1s among the three inputs x_{i+1}, x_i and c_{i-1}, that is

$$c_i=x_i x_{i+1}+(x_i+x_{i+1})c_{i-1}, \text{ being } c_{-1}=0 \tag{8}$$

The variable $\boldsymbol{D}=\{d_0, d_1,\ldots,d_{n-1}\}$, which flags all nonzero digits in the CSD representation, is defined as

$$d_i=x_i\oplus c_{i-1} \tag{9}$$

Since y_i takes one of the three values $\{0, 1, \bar{1}\}$, two bits $\{y_i^s, y_i^d\}$ are necessary to encode it, which are defined from Table 2 as

$$\begin{cases} y_i^d = \overline{x}_{i+1}(x_i \oplus c_{i-1}) = \overline{x}_{i+1} d_i \\ y_i^s = x_{i+1}(x_i \oplus c_{i-1}) = x_{i+1} d_i \end{cases} \tag{10}$$

For the sake of clarity, the following example describes the different signals used in the CSD recoding:

$X = 001010111$
$Carry = 011111110$
$D = 010101001$
$Y = 010\bar{1}0\bar{1}00\bar{1}$

In the case of a n-bit two's complement binary number $\boldsymbol{X}$, the CSD representation is given by

$$\boldsymbol{Y} = -x_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} x_i \cdot 2^i = \sum_{i=0}^{n-1} y_i \cdot 2^i \tag{11}$$

here, only n CSD digits are necessary as the value of the binary number is limited to $[-2^n, 2^{n-1}]$. Negative integers in CSD can be obtained trivially from their positive counterpart by changing the signs of all nonzero digits. For example, the CSD code $010\bar{1}$ represents the decimal number 3, while $0\bar{1}01$ represents the number -3. Therefore, the conversion of a negative n-digit two's complement binary number $\boldsymbol{X}$ into its CSD representation can be performed from the well-known property $-\boldsymbol{X} = \overline{\boldsymbol{X}} + 1$. Then reformulating the former equations for the case of a binary number $\boldsymbol{X}$ in a two's complement representation, the conversion into its CSD representation is given by

$$t_i = x_i \oplus x_{n-1}, \quad i < n-1 \tag{12}$$

$$c'_i = t_{i+1} t_i + (t_{i+1} + t_i) c'_{i-1}, \text{ being } c'_{-1} = x_{n-1} \tag{13}$$

where x_{n-1} represents the sign of $\boldsymbol{X}$. From (13), c'_i propagates c_i or its complement depending on sign of the $\boldsymbol{X}$ because of all inputs are computed according to (12). Therefore, another way to express c'_i is

$$c'_i = x_{n-1} \oplus c_i \tag{14}$$

Table 2
CSD coding.

x_{i+1}	x_i	c_{i-1}	y_i	c_i	Comments
0	0	0	0	0	String of 0s
0	0	1	1	0	End of 1s
0	1	0	1	0	A single 1
0	1	1	0	1	String of 1s
1	0	0	0	0	String of 0s
1	0	1	$\bar{1}$	1	A single 0
1	1	0	$\bar{1}$	1	Beginning of 1s
1	1	1	0	1	String of 1s

Applying these definitions, we get

$$d_i = t_i \oplus c'_{i-1} = (x_i \oplus x_{n-1}) \oplus (x_{n-1} \oplus c_{i-1}) = x_i \oplus c_{i-1} \tag{15}$$

This means that the definition of $\boldsymbol{D}$ is independent of sign $\boldsymbol{X}$. In a similar way, the expression of $\boldsymbol{Y}$ is the same as that in (10) as

$$\begin{cases} y_i^d = \overline{t}_{i+1} d_i = \overline{t}_{i+1}(t_i \oplus c'_{i-1}) = (\overline{x}_{i+1} \oplus x_{n-1})((x_i \oplus x_{n-1}) \oplus c'_{i-1}) = \overline{x}_{i+1} d_i \\ y_i^s = t_{i+1} d_i = t_{i+1}(t_i \oplus c'_{i-1}) = (x_{i+1} \oplus x_{n-1})((x_i \oplus x_{n-1}) \oplus c'_{i-1}) = x_{i+1} d_i \end{cases} \tag{16}$$

Fig. 2 shows the circuit to convert a $n=6$ digit binary number into its CSD representation according to Eqs. (8)–(10), valid for both unsigned and two's complement binary numbers. The only difference arises in the last CSD digit. By introducing an extra sign extension, $x_n=0$ for unsigned numbers and $x_n=x_{n-1}$ for two's complement numbers, the last section changes depending on the sign of $\boldsymbol{X}$ in the following general expression:

$$\text{For an unsigned number } \boldsymbol{X} \begin{cases} y_{n-1}^d = d_{n-1} \\ y_{n-1}^s = 0 \\ y_n^d = d_n = c_{n-1} = x_n c_{n-2} \\ y_n^s = 0 \end{cases} \tag{17}$$

$$\text{For a signed number } \boldsymbol{X} \begin{cases} y_{n-1}^d = \overline{x}_n d_i = \overline{x}_{n-1} c_{n-2} \\ y_{n-1}^s = x_n d_i = x_{n-1} \overline{c}_{n-2} \end{cases} \tag{18}$$

For unsigned $\boldsymbol{X}$, $n+1$ CSD digits are necessary to represent that binary number. However, as it is a positive number the two MSB digits of CSD are also positives and, indeed, only their positive data parts are necessary as shown in Eq. (17); the negative part is always zero. For signed numbers, only n CSD digits are used and the last digit is computed according to Eq. (18). As can be seen in Fig. 2, the critical path of the circuit is fixed by the propagation of the carry signal, in a similar way to a conventional ripple-carry adder. There is a clear similarity between the carry definition in conventional adders and the definition of that carry in Eq. (8). This suggests that implementation of fast CSD converters should be based on fast well-known carry look-ahead structures used in addition.

3. New CSD recoding

In the definition of carry in Eq. (8), two adjoining carries share the same input variable. This means that the same input x_i is used in the generation of both carries c_{i-1} and c_i. This characteristic allows the algebraic expressions of carry generation to be simplified in order to obtain efficient circuits.

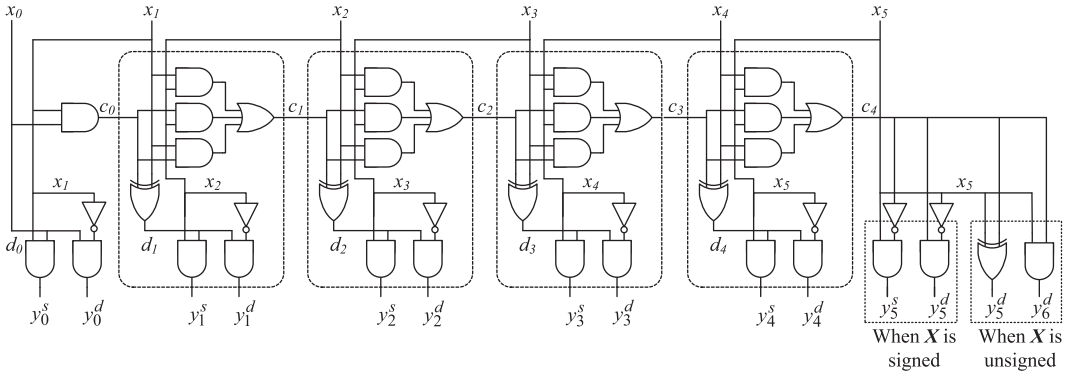

Fig. 2. Schematic circuit for the conversion of a binary number into its CSD representation ($n=6$).

Let $\boldsymbol{X}$ be a n-digit binary number. If we define two signals, $\boldsymbol{H}=\{h_0,h_1,\ldots,h_{n-1}\}$ and $\boldsymbol{K}=\{k_0,k_1,\ldots,k_{n-1}\}$, as

$$h_i=\begin{cases} x_i h_{i-1}, & \text{for } i \text{ odd} \\ x_i+h_{i-1}, & \text{for } i \text{ even} \end{cases} \tag{19}$$

$$k_i=\begin{cases} x_i+k_{i-1}, & \text{for } i \text{ odd} \\ x_i k_{i-1}, & \text{for } i \text{ even} \end{cases} \tag{20}$$

with $h_{-1}=0$ and $k_{-1}=0$. Then c_i can be formally expressed in term of odd or even index i by means of the following recursive relation:

$$c_i=\begin{cases} h_i+x_{i+1}k_i=h_i+k_{i+1}, & \text{for } i \text{ odd} \\ x_{i+1}h_i+k_i=h_{i+1}+k_i, & \text{for } i \text{ even} \end{cases} \tag{21}$$

A demonstration by induction of this equation can be found in appendix A of Ref. [36]. Moreover, h_i and k_i have the following properties:

$$\text{For } i \text{ odd} \begin{cases} a)\ h_i k_i=h_i \\ b)\ h_i\overline{k}_i=0 \\ c)\ \overline{h}_i+k_i=1 \\ d)\ h_i+k_i=k_i \end{cases} \tag{22}$$

$$\text{For } i \text{ even} \begin{cases} a)\ h_i k_i=k_i \\ b)\ \overline{h}_i k_i=0 \\ c)\ h_i+\overline{k}_i=1 \\ d)\ h_i+k_i=h_i \end{cases} \tag{23}$$

Demonstrations of these properties can be found in appendix B of Ref. [36]. However, using these properties, Eq. (21) can be transformed into another compact form as

$$c_i=\begin{cases} h_i+x_{i+1}k_i=h_{i+1}k_i, & \text{for } i \text{ odd} \\ x_{i+1}h_i+k_i=h_i k_{i+1}, & \text{for } i \text{ even} \end{cases} \tag{24}$$

For the sake of clarify, the definition of the carry based on $\boldsymbol{H}$ and $\boldsymbol{K}$ signals for $n=4$ are described. From Eq. (8), we get

$$c_0=x_1x_0$$
$$c_1=x_1x_2+(x_1+x_2)c_0=x_1x_0+x_2x_1$$
$$c_2=x_2x_3+(x_2+x_3)c_1=x_3(x_2+x_1x_0)+x_2x_1$$
$$c_3=x_3x_4+(x_3+x_4)c_2=x_3(x_2+x_1x_0)+x_4(x_3+x_2x_1)$$

From the recursive property in the definition of signals $\boldsymbol{H}$ and $\boldsymbol{K}$ of Eqs. (19) and (20), it is straightforward to obtain the following expressions

$$h_{-1}=0$$
$$h_0=x_0+h_{-1}=x_0$$
$$h_1=x_1h_0=x_1x_0$$
$$h_2=x_2+h_1=x_2+x_1x_0$$
$$h_3=x_3h_2=x_3(x_2+x_1x_0)$$
$$h_4=x_4+h_3=x_4+x_3(x_2+x_1x_0)$$

$$k_{-1}=0$$
$$k_0=x_0k_{-1}=0$$
$$k_1=x_1+k_0=x_1$$
$$k_2=x_2k_1=x_2x_1$$
$$k_3=x_3+k_2=x_3+x_2x_1$$
$$k_4=x_4k_3=x_4(x_3+x_2x_1)$$

Therefore, from Eqs. (21) and (24), the carry can be expressed in terms of those signals as

$$c_0=h_1+k_0=x_1x_0+0=h_0k_1=x_0x_1$$
$$c_1=h_1+k_2=x_1x_0+x_2x_1=h_2k_1=(x_2+x_1x_0)x_1$$
$$c_2=h_3+k_2=x_3(x_2+x_1x_0)+x_2x_1=h_2k_3$$
$$c_3=h_3+k_4=x_3(x_2+x_1x_0)+x_4(x_3+x_2x_1)=h_4k_3$$

The variable $\boldsymbol{D}$ in Eq. (9) can be directly obtained from $\boldsymbol{H}$ and $\boldsymbol{K}$ without it being necessary to generate c_i. If i is odd, this means that $i-1$ is even, we obtain

$$d_i=x_i\oplus c_{i-1}=x_i\overline{c}_{i-1}+\overline{x}_i c_{i-1}=x_i\cdot\overline{x_i h_{i-1}+k_{i-1}}+\overline{x}_i(x_i h_{i-1}+k_{i-1})$$
$$=x_i(\overline{h_{i-1}+k_{i-1}})+\overline{x}_i k_{i-1} \tag{25}$$

In Eq. (23), we get $h_{i-1}+k_{i-1}=h_{i-1}$ and $\overline{h}_{i-1}k_{i-1}=0$. Then, applying these properties to Eq. (25), it can be transformed as

$$d_i=x_i\overline{h}_{i-1}+\overline{x}_i k_{i-1}=(\overline{x}_i+\overline{h}_{i-1})(x_i+k_{i-1})=\overline{x_i h_{i-1}}(x_i+k_{i-1})=\overline{h}_i k_i \tag{26}$$

For i even, and likewise, we obtain

$$d_i=h_i\overline{k}_i \tag{27}$$

However, the variable $\boldsymbol{Y}$ for the CSD recoding in terms of signals $\boldsymbol{H}$ and $\boldsymbol{K}$ can be expressed as

$$y_i^d=\overline{x}_{i+1}d_i=\begin{cases} \overline{x}_{i+1}\overline{h}_i k_i=\overline{h}_{i+1}k_i, & \text{for } i \text{ odd} \\ \overline{x}_{i+1}h_i\overline{k}_i=h_i\overline{k}_{i+1}, & \text{for } i \text{ even} \end{cases} \tag{28}$$

$$y_i^s=x_{i+1}d_i=\begin{cases} x_{i+1}\overline{h}_i k_i=\overline{h}_i k_{i+1}, & \text{for } i \text{ odd} \\ x_{i+1}h_i\overline{k}_i=h_{i+1}\overline{k}_i, & \text{for } i \text{ even} \end{cases} \tag{29}$$

The proposed equations based on signals $\boldsymbol{H}$ and $\boldsymbol{K}$, which are functionally equivalent to two carries, lead to an efficient hardware implementation of CSD recoders. Fig. 3 shows the proposed implementation for the conversion of a 5-bit binary number $\boldsymbol{X}$ into its equivalent CSD representation. These signals can be computed by two parallel and independent paths of simple AND and OR gates in a ripple configuration. As a result, the critical path is reduced and the hardware implementation is minimized in comparison with other structures derived from conventional adders. The main body of this circuit is made up of alternative

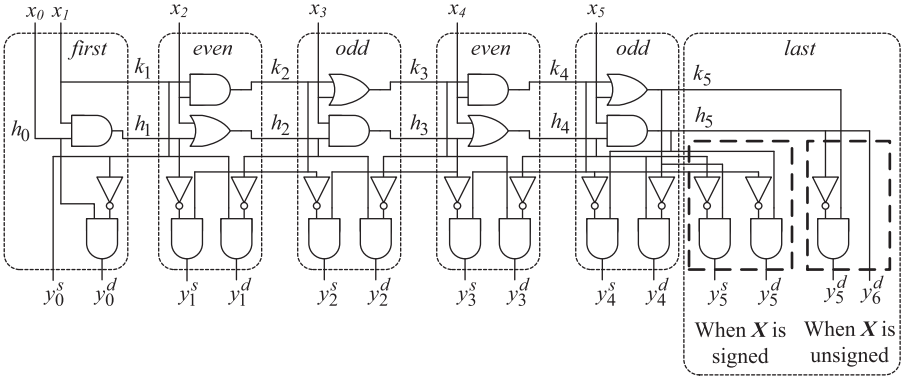

Fig. 3. Proposed circuit for the conversion of a binary number into its CSD representation ($n=6$).

odd and even sections, which implement the Eqs. (19), (20), (28) and (29). The first section is simplified as

$$\begin{cases} y_0^d = h_0\overline{k}_1 = x_0\overline{x}_1 \\ y_0^s = h_1\overline{k}_0 = h_1 = x_0x_1 \end{cases} \tag{30}$$

Only the last section is changed depending on sign of $\boldsymbol{X}$ in the following general expression

For a signed number

$$\boldsymbol{X}\begin{cases} y_{n-1}^d = \begin{cases} \overline{h}_{n-2}k_{n-1}, & \text{for } n-1 \text{ odd} \\ h_{n-1}\overline{k}_{n-2}, & \text{for } n-1 \text{ even} \end{cases} \\ y_{n-1}^s = \begin{cases} \overline{h}_{n-1}k_{n-2}, & \text{for } n-1 \text{ odd} \\ h_{n-2}\overline{k}_{n-1}, & \text{for } n-1 \text{ even} \end{cases} \end{cases} \tag{31}$$

For an unsigned number

$$\boldsymbol{X}\begin{cases} y_{n-1}^d = \begin{cases} \overline{h}_{n-1}k_{n-1}, & \text{for } n-1 \text{ odd} \\ h_{n-1}\overline{k}_{n-1}, & \text{for } n-1 \text{ even} \end{cases} \\ y_n^d = \begin{cases} h_{n-1}, & \text{for } n-1 \text{ odd} \\ k_{n-1}, & \text{for } n-1 \text{ even} \end{cases} \end{cases} \tag{32}$$

A more efficient implementation, suitable for a CMOS technology, is shown in Fig. 4. In this circuit, a straightforward manipulation of the Boolean algebra enables the implementation of the signals $\boldsymbol{H}$ and $\boldsymbol{K}$ in terms of NAND and NOR gates, which are the fastest gates in CMOS circuits. Moreover, these signals, and alternatively their complementary ones, facilitate the generation of $\boldsymbol{Y}$, thus reducing the complexity of the overall circuit in comparison with that of Fig. 3 as additional inverters are unnecessary.

3.1. Carry look-ahead CSD recoding

The carry look-ahead principle is one of the most widely used methods to implement fast adders by introducing some kind of parallelism in order to reduce the critical path of the circuit. The expressions of $\boldsymbol{H}$ and $\boldsymbol{K}$ defined in Eqs. (19) and (20) are of a similar form to those used in conventional carry look-ahead adders [1,39]. For example, for $i=6$ we get

$$\begin{aligned} h_6 &= x_6 + x_5(x_4 + x_3(x_2 + x_1x_0)) \\ k_6 &= x_6(x_5 + x_4(x_3 + x_2x_1)) \end{aligned} \tag{33}$$

These expressions are a simplified version of those used in carry look-ahead structures where the classical propagate and generate signals are directly removed by input signals. However, new generation and propagation signals associated with $\boldsymbol{H}$ and $\boldsymbol{K}$ enable a significant reduction of the critical path. Using the expressions in Eq. (33), h_6 can be rewritten as

$$h_6 = x_6 + x_5x_4 + x_5x_3(x_2 + x_1x_0) = gh_1 + ph_0h_2 \tag{34}$$

where gh_j and ph_j are, respectively, the generation and propagation signals of $\boldsymbol{H}$. In this case, $gh_1 = x_6 + x_5x_4$, $ph_0 = x_5x_3$ and $h_2 = x_2 + x_1x_0$. In a general way

$$h_{4(j+1)+2} = gh_{j+1} + ph_jh_{4j+2}, \quad j = 0,1,2,\ldots \tag{35}$$

where

$$\begin{cases} gh_j = x_{4j+2} + x_{4j+1}x_{4j} \\ ph_j = x_{4(j+1)+1}x_{4(j+1)-1} \end{cases} \tag{36}$$

Note that these signals are defined for a group of four inputs and $h_2 = gh_0$. Similarly, we can rewrite k_6 as

$$k_6 = x_6(x_5 + x_4)(x_5 + x_3 + x_2x_1) = gk_1(pk_0 + k_2) \tag{37}$$

where gk_j and pk_j are, respectively the generation and propagation signals of $\boldsymbol{K}$. In this case, $gk_1 = x_6(x_5 + x_4)$, $pk_0 = x_5 + x_3$ and $k_2 = x_2x_1$. In a general way, we get

$$k_{4(j+1)+2} = gk_{j+1}(pk_j + k_{4j+2}), \quad j = 0,1,2,\ldots \tag{38}$$

where

$$\begin{cases} gk_j = x_{4j+2}(x_{4j+1} + x_{4j}), & j > 0 \\ pk_j = x_{4(j+1)+1} + x_{4(j+1)-1} \end{cases} \tag{39}$$

The recurrence in Eqs. (35) and (38) based on propagation and generation signals obtained from a group of 4 input signals leads to fast implementations of CSD recoders. Fig. 5a depicts a schematic of a circuit for performing the conversion of a signed binary number ($n=16$) into its CSD representation. Here, $\boldsymbol{X}_{<3:5>}$ stands for $\{x_3, x_4, x_5\}$ and $\boldsymbol{Y}_{<2:5>}$ stands for $\{\{y_2^s, y_2^d\}, \{y_3^s, y_3^d\}, \{y_4^s, y_4^d\}, \{y_5^s, y_5^d\}\}$. Two parallel circuits generate the signal $\boldsymbol{H}$ and $\boldsymbol{K}$ for 4-bit groups in such a way that the biggest bits are expressed as

$$\begin{aligned} h_{14} &= gh_3 + ph_2(gh_2 + ph_1(gh_1 + ph_0h_2)) \\ k_{14} &= gk_3(pk_2 + gk_2(pk_1 + gk_1(pk_0 + h_2))) \end{aligned} \tag{40}$$

From these signals, the CSD representation $\boldsymbol{Y}$ is computed by three different kinds of compact modules: a first module (Fig. 5b), 4-bit CSD modules (Fig. 5c) and a last module (Fig. 5d). Only this last module changes depending on whether $\boldsymbol{X}$ is an unsigned or signed number. However, when the number of modules is large, it is necessary to apply multi-level carry look-ahead schemes to reduce the delay; any interested reader can find more information in Section 2.6 of Ref. [37].

3.2. Parallel prefix CSD recoding

Another alternative for fast implementations of a CSD recoding is derived from the binary associative operator "○" developed for prefix adders [38]. As long as the recurrence relations of $\boldsymbol{H}$ and $\boldsymbol{K}$ defined in Eqs. (35) and (38) have identical properties to those of conventional adders, new associative operators "○" and "●",

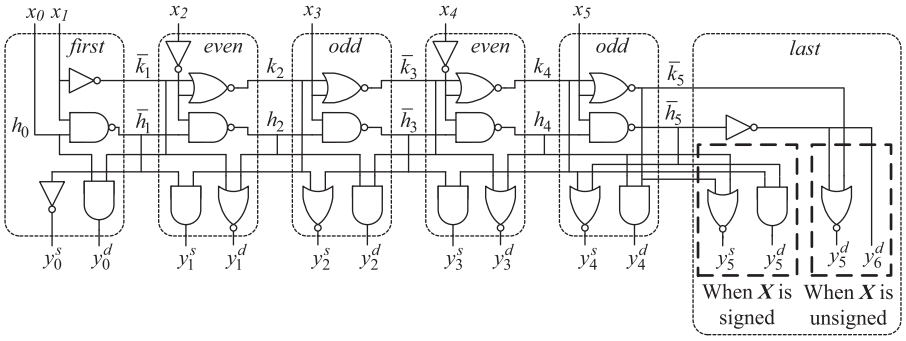

Fig. 4. Efficient implementation for the circuit in Fig. 3.

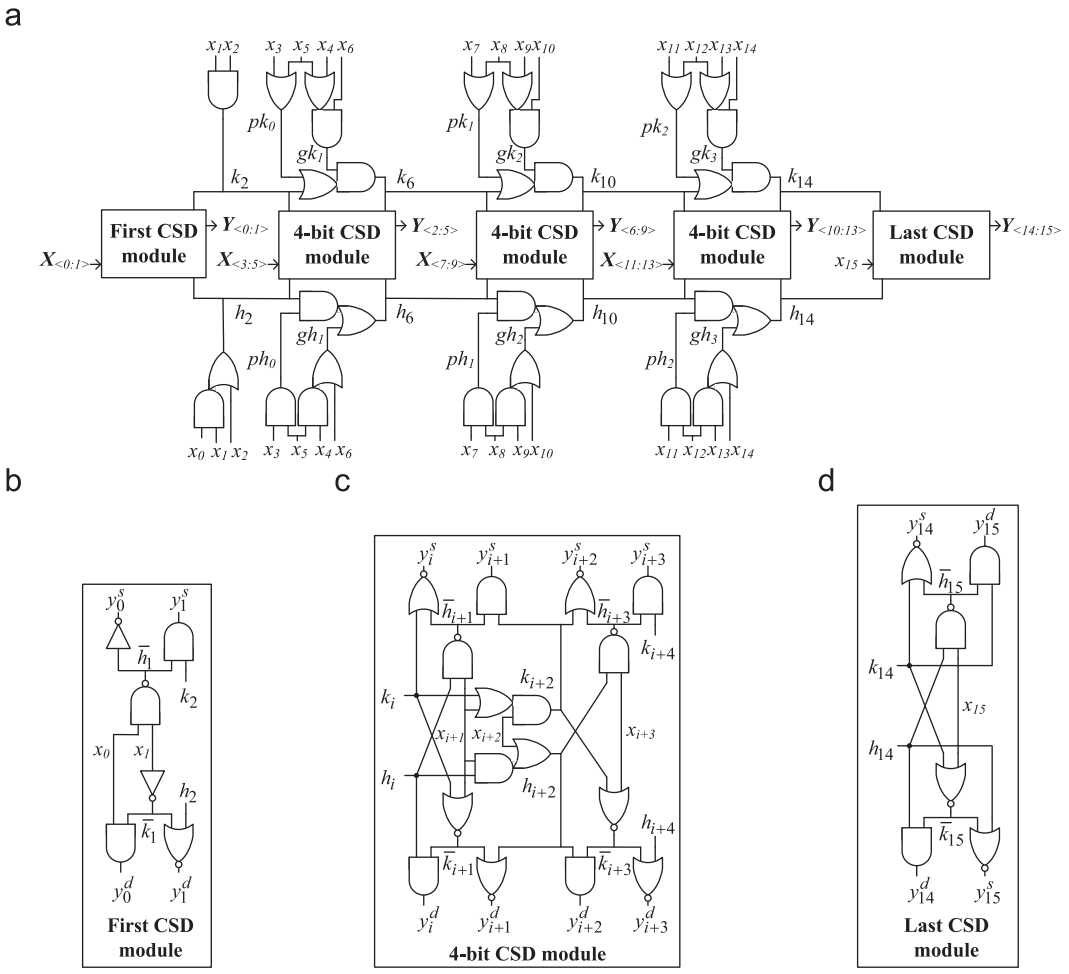

Fig. 5. Carry look-ahead scheme of a CSD recoder for $n=16$. (a) Schematic of full circuit, (b) schematic of first CSD module, (c) schematic of 4-bit CSD module ($i=2,6,10$) and (d) schematic of Last CSD module.

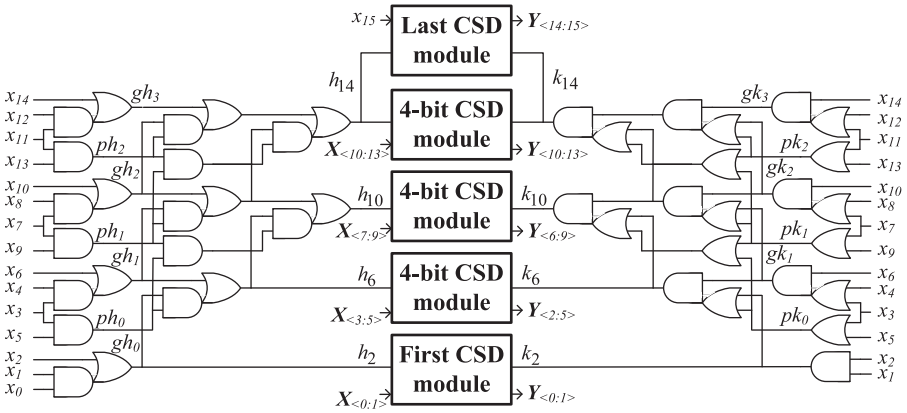

Fig. 6. Parallel prefix scheme of a CSD recoder for $n=16$.

dependent on $\boldsymbol{H}$ and $\boldsymbol{K}$ can be defined as

$$\begin{cases} (gh,\ ph)\circ(gh^*,\ ph^*) = (gh + ph\cdot gh^*,\ ph\cdot ph^*) \\ (gk,pk)\bullet(gk^*,pk^*) = (gk(pk+gk^*),\ pk+pk^*) \end{cases} \tag{41}$$

Based on these operators, parallel prefix circuits for computing signals $\boldsymbol{H}$ and $\boldsymbol{K}$ can be implemented in an efficient way. Fig. 6 shows an example of a parallel prefix scheme for CSD recoding ($n=16$). A first stage computes the individual generation and propagation signals. The remaining stages constitute a parallel prefix circuit with an organization based on the operators "○" and "●", which generates the 4-bit group signals. This circuit uses the same modules specified in Fig. 5 for computing the CSD representation $\boldsymbol{Y}$. The proposed implementation has a total number of stages of $\log_2(n/4)+1$ in comparison with $\log_2(n)+1$ used in a conventional scheme.

Table 3
Comparison of CSD recoders using different configurations.

Configuration	8-bit			16-bit			32-bit			64-bit			Average reduction	
	t_p (ns)	Area (μm^2)	NAND Equiv.	t_p (ns)	Area (μm^2)	NAND Equiv.	t_p (ns)	Area (μm^2)	NAND Equiv.	t_p (ns)	Area (μm^2)	NAND Equiv	t_p	Area
Ripple-carry scheme														
[25,26]	0.96	282	35	2.07	605	75	4.28	1250	155	8.71	2542	315		
Fig. 4	0.46	205	25	0.93	431	53	1.88	883	109	3.79	1767	221		
Reduction (%)	*52.1*	*27.3*		*55.6*	*28.8*		*57.0*	*29.4*		*57.6*	*29.7*		***55***	***29***
Carry look-ahead scheme														
[30]	0.49	327	41	0.96	685	85	1.94	1403	174	3.92	2840	414		
Fig. 5	0.31	252	31	0.53	567	70	0.90	1196	148	1.77	2487	352		
Reduction (%)	*36.7*	*22.9*		*44.8*	*17.2*		*53.6*	*14.8*		*54.8*	*12.4*		***47***	***17***
Parallel prefix scheme														
[32]	0.51	387	48	0.67	922	114	0.92	2234	277	1.70	5459	677		
Fig. 6	0.34	256	32	0.53	647	80	0.62	1575	195	0.72	3721	461		
Reduction (%)	*33.3*	*33.9*		*20.9*	*29.8*		*32.6*	*29.5*		*57.6*	*31.8*		***36***	***31***

4. Simulation and comparisons

For the purposes of comparison, different standard cell-based implementations and configurations of CSD recoders based on previously published circuits have been compared and analyzed with those proposed in this paper. All of them were synthesized with Synopsys® design compiler release 2010.03 under the HCMOS9 STMicroelectronics 130 nm standard cell technology with a 1.2 V power supply. Table 3 lists the synthesis results in terms of worst-case propagation time (t_p in ns) including a size circuit dependent wire-load model, and total area expressed in terms of μm^2 or in an equivalent number of two input NAND gates (8.0688 μm^2 for this technology). In all cases, a digit two's complement binary number $\boldsymbol{X}$ with different values of $n=8$, 16, 32 and 64 has been used as input.

To obtain representative results, we have selected the configurations based on the ripple-carry principle derived from the expressions in Refs. [25,26], the optimized carry look-ahead scheme derived from the expressions presented in Ref. [30] and the parallel prefix scheme presented in Ref. [32]. All of these configurations have been implemented in terms of standard cells for comparison purposes and they have their counterpart in the proposed circuits shown in Figs. 4, 5 and 6, respectively, where the signals $\boldsymbol{H}$ and $\boldsymbol{K}$ are computed in a similar way to those carries in the former schemes. The results in Table 3 highlight the significant advantages of the circuits proposed in terms of speed and area. The implementation in Fig. 4 reduces the delay by roughly 55 and the area by 29% with respect to the counterpart derived from Refs. [25,26]. This is possible because of parallelism in the signals $\boldsymbol{H}$ and $\boldsymbol{K}$ and the simplification in the complexity of the circuit. Another important improvement is the decrease by up to 47% (average) in delay in the carry look-ahead scheme. Here, the generate and propagate signals associated with $\boldsymbol{H}$ and $\boldsymbol{K}$ significantly reduce the critical path in comparison with the scheme presented in Ref. [30], which computes the carries directly from input signals by cascading of 4-bit look-ahead circuits. This comparison has been made in terms of standard cells because a full CMOS complex gate to implement the carry look-ahead circuit is proposed in Ref. [30]. However, the reduction in area is limited, varying from 22.9% for the 8-bit configuration to 12.4% for the 64-bit. The reason is the similar area that both 4-bit look-ahead circuits have decreasing the ratio when the size of encoder increases. Finally, the fastest implementation corresponds to the parallel prefix scheme. Although for 8-bit, the carry look-ahead scheme is slightly better both in terms of area and in speed, the delay of the parallel prefix scheme is only 0.72 ns for 64-bit, which is suitable for high-speed circuits, and, moreover, without a significant increase in area. As a result, Table 3 demonstrates that the expressions developed to implement the CSD recoding lead to circuits, whose area and speed are notably improved in comparison with previously proposed implementations.

5. Conclusion

The conversion of a two's complement binary number into its canonic signed digit (CSD) representation can be efficiently implemented using two signals, $\boldsymbol{H}$ and $\boldsymbol{K}$, functionally equivalent to two carries. These signals have two advantageous features: they are computed in parallel reducing the critical path and they simplify the algebraic expressions minimizing the overall hardware implementation. Simulations performed with the proposed circuits show high efficiency in terms of speed and area in comparison with other previous counterpart architectures. Moreover, other schemes used for accelerating carries based on transistor structures such as domino carry look-ahead, multilevel carry look-ahead and carry-skip circuits can be directly used. Finally, the new formulation presented in this work enables the CSD number system to be made available to many DSP applications as the CSD recoding can be performed at high speed with a low area cost.

Acknowledgment

We wish to acknowledge the financial help of the Spanish Ministry of Education and Science through TEC2006-12438/TCM received to support this work.

References

[1] I. Koren, Computer Arithmetic Algorithms (Second ed.), A.K. Peters, Ltd. (Ed.), 2002.
[2] B. Phillips, N. Burgess, Minimal weight digit set conversions, IEEE Transactions on Computers 53 (6) (2004) 666–677.
[3] D.S. Phatak, I. Koren, Hybrid signed-digit number systems: a unified framework for redundant number representations with bounded carry propagation chains, IEEE Transactions on Computers 43 (8) (August 1994) 880–891.
[4] K.Y. Khoo, A. Kwentus, A.N. Wilson, A programmable FIR digital filter using CSD coefficients, IEEE Journal of Solid-State Circuits 31 (6) (June 1996) 869–874.
[5] Y.M. Hasan, L.J. Karam, M. Falkinburg, A. Helwig, M. Ronning, Canonic signed digit Chebyshev FIR filter design, IEEE Signal Processing Letters 8 (6) (2001) 167–169.

[6] J. Skaf, S.P. Boyd, Filter design with low complexity coefficients, IEEE Transactions on Signal Processing 56 (7) (2008) 3162–3169.
[7] T. Williams, M. Ahmadi, W.C. Miller, Design of 2D FIR and IIR digital filters with canonical signed digit coefficients using singular value decomposition and genetic algorithms, Circuits Systems Signal Processing 26 (1) (2007) 69–89.
[8] H. Lee, G. Sobelman, FPGA-based digit serial CSD FIR filter for image signal format conversion, Microelectronic Journal 22 (2002) 501–508.
[9] R.I. Hartley, Subexpression sharing in filters using canonic signed digit multipliers, IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing 43 (10) (1996) 677–688.
[10] C.Y. Yao, H.H. Chen, T.F. Lin, C.J. Chien, C.T. Hsu, A novel common-subexpression-elimination method for synthesizing fixed-point FIR filters, IEEE Transactions on Circuits and Systems-I: Regular Papers 51 (11) (2004) 2215–2221.
[11] S.T. Pan, A canonic-signed-digit coded genetic algorithm for designing finite impulse response digital filter, Digital Signal Processing 20 (2) (2010) 314–327.
[12] M.D. Macleod, A.G. Dempster, Common subexpression elimination algorithm for low-cost multiplierless implementation of matrix multipliers, Electronics Letters 40 (11) (2004) 651–652.
[13] G.K. Ma, F.J. Taylor, Multiplier policies for digital signal processing, IEEE ASSP Magazine 1 (1990) 6–20.
[14] M.A. Soderstrand, CSD multipliers for FPGA DSP applications, in: Proceedings of the International Symposium on Circuits and Systems (ISCAS), pp. V-469–V-472, 2003.
[15] D.L. Iacono and M. Ronchi, Binary canonic signed digit multiplier for high-speed digital signal processing, in: Proceedings of the 47th IEEE International Midwest Symposium on Circuits and Systems, II, pp. 205–208, 2004.
[16] Y. Wang, L.S. DeBrunner, D. Zhou, V.E. DeBrunner, A multiplier structure based on a novel real-time CSD recoding, IEEE International Symposium on Circuits and Systems (ISCAS) (2007) 3195–3198.
[17] G.A. Ruiz., M.A. Manzano, Self-timed multiplier based on canonical signed-digit recoding, IEE Proceedings-Circuits, Devices and Systems 148 (5) (2001) 235–241.
[18] N. Petra, D. de Caro, A.G.M. Strollo, V. Garofalo, E. Napoli, M. Coppola, P. Todisco, Fixed-width CSD multipliers with minimum mean square error, IEEE International Symposium on Circuits and Systems (ISCAS) (2010) 4149–4152.
[19] S.M. Kim, J.G. Chung, K.K. Parhi, Low error fixed-width CSD multiplier with efficient sign extension, IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing 50 (12) (2003) 984–993.
[20] S. Yang, B.U. Lee, Efficient transform using canonical signed digit in reversible color transforms, Journal of Electronic Imaging 18 (3) (2009) 033010.
[21] D.C. Lou, J.C. Lai, C.L. Wu, T.J. Chang, An efficient Montgomery exponentiation algorithm by using signed-digit-recoding and folding techniques, Applied Mathematics and Computation 185 (2007) 31–44.
[22] D.C. Lou, J.C. Lai, C.L. Wu, T.J. Chang, An efficient Montgomery exponentiation algorithm by using signed-digit-recoding and folding techniques, Applied Mathematics and Computation 185 (1) (2007) 31–44.
[23] A.Y. Wu, C.S. Wu, A unified view for vector rotational CORDIC algorithms and architectures based on angle quantization approach, IEEE Transactions on Circuits and Systems-I: Fundamental, Theory and Applications 49 (10) (2002) 1442–1456.
[24] S.W. Reitwiesner, Binary arithmetic, Advances in Computers 1 (1966) 231–308.
[25] K. Hwang, Computer Arithmetic, Principles, Architecture and Design, John Wiley & Sons, New York, NY, 1979.
[26] A. Peled, On the hardware implementation of digital signal processors, IEEE Transactions on Acoustics, Speech, and Signal Processing ASSP-24 (1) (1976) 76–86.
[27] R. Hashemian, A new method for conversion of a 2's complement to canonic signed digit system and its representation, Proceedings of the Asilomar Conference on Signals, Systems and Computers (1997) 904–907.
[28] F. Xu, C.H. Chang, C.C. Jong, Hamming weight pyramid—a new insight into canonical signed digit representation and its application, Computers and Electrical Engineering 33 (2007) 195–207.
[29] M. Faust, O. Gustafsson, C.H. Chang, Fast and VLSI efficient binary-to-CSD encoder using bypass signal, Electronics Letters 47 (1) (2011).
[30] A. Herrfeld, S. Hentschke, Look-ahead circuit for CSD-code carry determination, Electronics Letters 31 (6) (1995) 434–435.
[31] S.K. Das, M.C. Pinotti, Fast VLSI circuits for CSD coding and GNAF coding, Electronics Letters 32 (7) (1996) 632–634.
[32] C.K. Koc, Parallel canonical recoding, Electronics Letters 32 (22) (1996) 2063–2065.
[33] K. Okeya, K. Schmidt-Samoa, C. Spahn, T. Takagi, Signed binary representations revisited, Chapter of book "Advances in Cryptology—CRYPTO 2004, Lecture Notes in Computer Science, vol. 3152, Springer Berlin, Heidelberg, pp. 123–139, 2004.
[34] M. Joye, S.M. Yen, Optimal left-to-right binary signed-digit recoding, IEEE Transactions on Computers 49 (7) (July 2000) 740–748.
[35] E. Backenius, E. Säll, O. Gustafsson, Bidirectional conversion to minimum signed-digit representation, in: Proceedings of the IEEE International Symposium on Circuits & Systems (ISCAS 2006). pp. 2413–2416, September 2006.
[36] G.A. Ruiz, M. Granda, Efficient implementation of 3X for radix-8 encoding, Microelectronic Journal 39 (1) (2008) 152–159.
[37] M.D. Ercegovac, T. Lang, Digital Arithmetic, Morgan Kaufmann Publishers, Elsevier Science, 2004.
[38] P.M. Kogge, H.S. Stone, A parallel algorithm for the efficient solution of a general class of recurrence equations, IEEE Transactions on Computers C-22 (8) (1973) 783–791.
[39] M.D. Ercegovac, T. Lang., Digital Arithmetic, Morgan Kaufmann Publishers, 2004.