

# A Number System with Continuous Valued Digits and Modulo Arithmetic

Revised submission to IEEE Transaction on Computers

Aryan Saed<sup>\*</sup>, Majid Ahmadi, Graham A. Jullien<sup>†</sup>  
VLSI Research Group, University of Windsor, Windsor, Ontario, Canada  
ahmadi@uwindsor.ca

<sup>\*</sup>Nortel Networks, Nepean, Ontario, Canada  
saed@nortelnetworks.com

<sup>†</sup>ECE Dept., University of Calgary, Calgary, Alberta, Canada  
jullien@enel.ucalgary.ca (corresponding author)

## Abstract

*This paper presents a novel number system based on signed continuous valued digits. Arithmetic operations in this number system are performed using simple analog circuitry, in contrast to the conventional implementation of arithmetic units by Boolean or multiple-valued logic circuits. Unlike the limited precision offered by classical analog arithmetic circuits, the ensemble of continuous valued digits that comprises a number in this system, provides arbitrary implementation precision with standard analog circuitry. The number system also provides almost-carry-free arithmetic structures with digit level redundancy. In this paper we introduce the mathematical foundations for positive and negative numbers, addition, multiplication, redundancy, radix conversions, and also the digit value integrity for circuit implementations. Potential applications are in the area of low noise and low cross-talk circuitry for arithmetic circuits used in mixed-signal systems.*

**Keywords:** Computer Arithmetic; Continuous Digits; Modulo Arithmetic; Multiple-Valued Logic; Low-noise Circuitry.

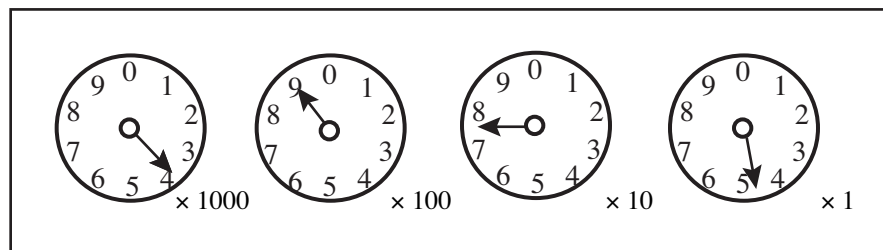
## 1. Introduction

Arithmetic processors in today's digital computers rely almost exclusively on Boolean logic circuits. These circuits are simple and, with current technologies, very fast. At current System-on-Chip (SOC) densities, however, speed is often not so much of a concern as is power reduction and system noise. Power reduction has been a research goal for several years, and there have been many important results achieved [1]; the reduction of system noise, however, is still a lower research priority at the architectural level. Power dissipation in most integrated systems is mainly dynamic and dependent upon the voltage swing magnitude and frequency across load capacitances. System noise, on the other hand, is dependent on the rate of change of voltage between nets (cross talk) and the rate of change of current (system noise). To some extent the reduction of power dissipation also aids in noise reduction; an extreme example is the use of adiabatic logic [2] where the requirements for relatively slow charging and discharging of parasitic capacitances also aids in reducing the noise level. If we look at a static CMOS logic inverter, for example, from an analog perspective, it is essentially a high gain amplifier with attendant large slew current and output voltage as the input moves through the transition region. Even if we reduce the frequency of the input digital signal transitions, a measure which certainly reduces power and RMS noise, the current and voltage slewing rates do not change; i.e., the instantaneous noise level is not reduced. This can be disruptive to synchronous systems where incorrect values can be latched based on system or cross-talk noise. Although there are technological solutions, e.g., decreasing the relative permittivity (and hence capacitance) between adjacent levels of connections, these solutions can only go so far in alleviating what has the potential to become a major problem as device densities increase.

In this paper we look at building low noise arithmetic systems using analog (continuous) representations and operations. Limited precision analog arithmetic circuits are not new; some of the first electronic computers used precision analog components to provide solutions to quite complex mathematical problems [3]. All of these machines, however, had the same limitation of

rather low precision (often measured in several percentage points which translates to fewer than 6-bits equivalent digital precision). We are interested in using analog operations for their potential low-noise qualities when implemented as circuits, and in finding ways around the low precision problem. One of the authors has already examined the use of analog circuits in self-timed arrays to emulate binary digit arithmetic [4] (though not necessarily using the binary number system); in this paper we look at a number system that is inherently analog, and thus can be naturally implemented with analog circuitry, but without the low-precision limitation. The number system we are introducing is based on multiple digits, each of which is continuous; we will refer to this as the Continuous Valued Number System (CVNS).

This is perhaps not as radical a concept as it first seems, since there is an example of such a system found in many house-holds; in North America it is referred to as the “Utility Meter”. Consider the diagram shown in Figure 1. There are four dials (digits), and each of the dials is continuous in the range  $[0, 10)$ .



**Figure 1: The “Utility Meter”**

The meter is read from left to right. The first dial provides an analog value of the entire reading; in this case it appears to be a number slightly less than 4000. The second dial provides a reading modulo 1000, and shows that the remainder must be approximately 900. The third and fourth dials provide further precision with a modulo 100 reading of about 76 and a modulo 10 reading of about 4.8. The reading can now be assembled as 4974.8. The dynamic range of  $10^4$  is given by the most significant dial. In fact, if the most significant dial were very precise, and we were able to read it to a precision of greater than 0.01%, then there would clearly be no need for the other 3

dials. The other dials are required to remove the uncertainty associated with the pointer position in the  $\times 1000$  dial. The  $\times 100$  dial appears to point to 9, but we see from the  $\times 10$  dial that the  $\times 100$  pointer must be in error by a positive amount (reading a value greater than actual) and so the  $\times 100$  digit can be corrected to 8. The  $\times 10$  dial appears to read a value greater than 75 but the  $\times 1$  dial shows that it must be 74.5.

At first sight this appears to be a Multiple-Valued Logic (MVL) [5] implementation of a number system with the number of values of each digit taken to the limit; however, there is a fundamental difference. The multiple digits in this new system are only used to refine higher order digits; in MVL implementations of number systems the weighted digits (or their mappings for non-positional representations - for example, the Chinese Remainder Theorem in Residue Arithmetic) are summed to obtain the value of the number being represented. Although in the CVNS digits are not limited to integers, conversion between a measured value and a CVNS number turns out to be relatively simple.

The CVNS and the concept of CVD's were first introduced in [7], and basic methods for addition and array multiplication were given in [8]. In [9] we presented a scheme for efficient interfacing between CVNS and binary. Circuit design issues have been discussed in [10] and [11], and in [12] we presented the concept of signed digits. In this paper we provide a review of our earlier findings, with a more complete emphasis on the mathematical background and several worked examples. We also introduce methods for number comparison and radix transform. The original name we used, Overlap Resolution Number System (ORNS), has since been changed to CVNS.

In section 2 we introduce the number system and the properties of the digits. In section 3 we discuss error recovery and the error tolerance of CVNS representations, an important point when considering future analog circuit implementations. Section 4 presents techniques to perform the elementary arithmetic operations of addition, subtraction, and multiplication, and also discusses

algorithms for number comparison and sign detection in the presence of errors. In section 5 we discuss the problem of converting between different CVNS radix representations, and in section 6 we briefly address alternate methods for representing continuous valued digits.

## 2. Continuous Valued Digits

Given a real number  $|x| < X$ , we shall represent it by a set of CVNS digits (CVD's),  $x_n$ , with  $K \leq n \leq L$ . We will define a radix  $B$  for the representation and, for the purposes of this paper, we will assume a maximum dynamic range of:

$$X = B^{L+1} \quad (1)$$

The format of the representation is given as follows:

$$x \Rightarrow (x_L, \dots, x_0 | x_{-1}, \dots, x_K) \quad (2)$$

where the bar (|) represents the radix point. In order to maintain a consistent notation with other number representations, we will refer to the part to the left of the radix point as the *integer part* and to the right of the radix point as the *fractional part*. It is possible to generalize the CVNS for real valued radices, but it increases the mathematical and implementation complexity and we have found no practical benefit in computer arithmetic [13]. For the purposes of this introductory paper we will maintain the relationship of eqn. (1). Note that we refer to the  $n$ th CVNS digit of  $x$  as  $x_n$ ; this is shorthand for  $(x)_n$  which we will also use where appropriate.

### 2.1 Generating the CVD's

We propose two different methods to generate the CVD's. The first method involves a cascaded technique, whereby we start with the Most Significant Digit (MSD)  $x_L$ , and compute it as

$x_L = B \cdot \frac{x}{X}$ , where  $B$  is the radix of the representation. Since this is a continuous valued representation,  $B$  can be any real number, though we will restrict it to be an integer for this

introductory paper. With this restriction we constrain  $B \geq 2$ . Further digits are calculated by the *cascade rule* as given below:

$$x_n = (x_{n+1} - \lfloor x_{n+1} \rfloor) \cdot B \quad (3)$$

where  $\lfloor \cdot \rfloor$  denotes the floor function. As a result we have  $|x_n| < B$  and  $\lfloor x_n \rfloor \leq B - 1$ ; where  $x_n$  is a real number. We select the value  $L$  such that  $B^{L+1} \geq |X|$ . A rule for selecting  $K$  will follow in the next section.

The second method involves the modulo operation of eqn. (4), where  $a$  is a real number.

$$(a) \bmod B = a - B \lfloor a/B \rfloor \quad (4)$$

The CVNS *digit generation rule* is given below:

$$x_n = \left( \frac{x}{X} \cdot B^{L-n+1} \right) \bmod B \quad (5)$$

Both of these methods may also be used to compute digits with index  $n > L$ , where these digits will be referred to as *excessively evolved digits* (EED's). EED's are equivalently computed by the cascade rule, using eqn. (3), which simplifies to  $x_n = x_{n-1}/B$ , or by the digit generation rule, using eqn. (5), which simplifies to  $x_n = B^{L-n+1} \cdot x/X$ . We can also use eqn. (5) to generalize digit generation to find  $x_n$  given  $x_m$  as follows:

$$x_m = (x_n \cdot B^{n-m}) \bmod B; \quad m \leq n \quad (6)$$

EEDs are often used in arithmetic operations where the result of the operation (e.g., multiplication) will require an increase in the value of  $L$ ; i.e., an increase in dynamic range.

## 2.2 Signed values

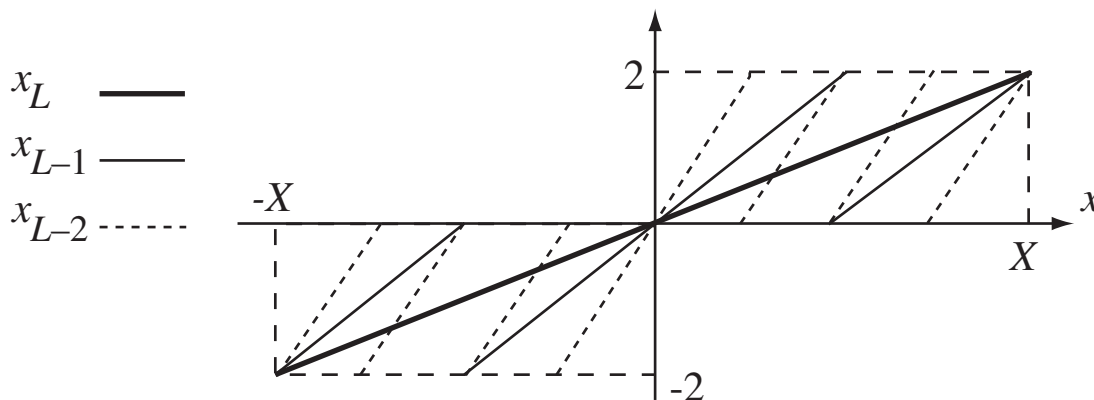
Our representation methods so far imply that the sign of the number is contained in each digit, and so we will formalise this by representing negative numbers with all the CVNS digits negative

(signed digits). Since all of the  $x_n$  follow the sign of  $x$ , then from eqn. (5) we can formalize the sign inversion (negation) of the CVNS representation as below:

$$(-x)_n = -x_n \quad (7)$$

Signed digits are well known in binary representations, though our signed representation is somewhat limited compared to the wealth of advantages offered by the redundancy of signed binary digits [14]. In section 6 we discuss two alternative representations that are more efficient than signed CVNS digits in terms of utilizing the range of the analog digit circuitry.

Figure 2 shows an example of signed CVNS digit values for  $B=2$  (binary CVNS). The horizontal axis represents the value of the input number,  $x$ , and the vertical axis shows the value of each continuous digit,  $x_n$ . We see that  $x_L$  is proportional to  $x$  over the maximum range of the representation ( $\pm X$ ), and digits  $x_{L-1}$  and  $x_{L-2}$  continuously follow  $x$  with saw-tooth discontinuities associated with the modulo operation with radix 2.



**Figure 2: Signed CVNS Digits for  $B=2$**

### 2.3 Analog implementations

Since the CVD's are continuous, we may also use the term *analog digits*. If a linear electronic variable, for instance a current, charge or voltage, ranges from 0 to  $\pm Q$  units, then each CVD of  $x$  is matched proportionately to the variable  $q_n$  by  $q_n = x_n \cdot Q/B$ .

**Example 1:** Consider a value  $x \geq 0$ , limited by  $X = 100$ , to be represented by CVD's in the range  $0\mu A$  to  $50\mu A$ . We select two radix values,  $B = 10$  for decimal CVNS, and  $B = 2$  for binary CVNS. To satisfy  $X \leq B^{L+1}$  we select  $L = 1$  for the decimal case, and  $L = 6$  for the binary case. For both, we select  $K = -1$ . The CVD's for  $x = 58.742$  are presented in Table 1. The decimal digit  $x_2$  is an EED. We observe that  $\lfloor x_n \rfloor$  are the PNS digits of  $(xB^{L+1})/X$ . For instance, here we have  $(x2^{6+1})/100 = 75.19$ , which is approximated in binary as (1001011.0).

**Table 1: CVNS Example for  $x=58.742$**

$n$	Decimal ( $B = 10$ )			Binary ( $B = 2$ )		
	$x_n$	$\lfloor x_n \rfloor$	$q_n/(\mu A)$	$x_n$	$\lfloor x_n \rfloor$	$q_n/(\mu A)$
6	-		-	1.17484	1	29.371
5	-		-	0.34968	0	8.742
4	-		-	0.69936	0	17.484
3	-		-	1.39672	1	34.918
2	0.58742	0	2.9371	0.79744	0	19.936
1	5.87420	5	29.3710	1.59488	1	39.872
0	8.74200	8	43.7100	1.18796	1	29.699
-1	7.42000	7	37.1000	0.37952	0	9.488

The concept of analog digits appears to be little known, though we have recently been made aware of some earlier efforts to use this concept for optical computing by Pulliam [15]. Pulliam's approach was also motivated by the Utility Meter, but his work was restricted to performing addition in optical systems. To the authors' knowledge, no other work exists and there does not seem to have been any attempt to rigorously define the number representation with analog circuitry applications. As discussed in section 1, a similar principle is in use with mechanical devices that employ a gear box and rotating indicators to represent a continuous value [16],[17]. The modular and continuous nature of Continuous Valued Digits is present in kilo-Watt-hour meters, gas volume meters, aeronautical altimeters and the analog watch.

### 3. Error Recovery

Theoretically a number is retrieved from its CVNS representation by the MSD alone, without any error:

$$x = x_L \cdot X/B \quad (8)$$

In practice this is not possible because of the inaccuracies in the hardware used to represent the digits. As stated in section 1, the multiple digits in the CVNS representation are only used to correct higher order digits; we will now use that feature to correct errors in the analog representation of the CVNS digits.

Let us consider an error in a CVNS digit such that  $x'_n = x_n + \varepsilon_{x_n}$ . Where obvious by context we will use the shorthand notation  $\varepsilon_{x_n} = \varepsilon_n$ . From eqn. (8) the value we recover from the MSD alone is given as follows:

$$\hat{x} = x + \varepsilon_L \cdot X/B \quad (9)$$

We now present a method for reducing this error by using the information from lower order digits,  $x_{L-1 \dots K}$ ; we refer to this method as *reverse evolution*.

#### 3.1 Reverse evolution

We can easily correct an error in the  $n$ th CVNS digit of  $x$ ,  $x'_n = x_n + \varepsilon_n$ , given that its adjacent lower digit,  $x'_{n-1} = x_{n-1}$ , is correct (i.e.,  $\varepsilon_{n-1} = 0$ ) and that the condition given below holds.

$$\lfloor x'_n \rfloor = \lfloor x_n \rfloor \quad (10)$$

Given these conditions we can restore  $x'_n$  to  $x''_n = x_n$  as shown below:

$$x''_n = \begin{cases} \lfloor x'_n \rfloor + x'_{n-1}/B & n > K \\ x'_K & n = K \end{cases} \quad (11)$$

We will refer to this as the reverse evolution process.

Reverse evolution requires a reliable method for computing the floor function and, based on eqn. (10), we see that the technique fails if either  $\lfloor x_n + \varepsilon_n \rfloor > \lfloor x_n \rfloor$  or  $\lfloor x_n + \varepsilon_n \rfloor < \lfloor x_n \rfloor$ . Since  $\lfloor x_n \rfloor = x_n - x_{n-1}/B$ , under the two conditions of eqns. (12) we see that this technique will fail for  $|\varepsilon_n| > |\varepsilon|$  and in the worst case, when  $x_n$  approaches an integer value, this implies failure for  $\varepsilon_n \rightarrow 0$ .

$$\begin{aligned} x_{n-1} &= \varepsilon \\ x_{n-1} &= B - \varepsilon \end{aligned} \quad (12)$$

We can improve this situation by replacing  $\lfloor x'_n \rfloor$  with a rounded computation of the floor function, as shown in the following theorem.

**Theorem 1:** Given two adjacent CVD's,  $x'_n = x_n + \varepsilon_n$  and  $x'_{n-1} = x_{n-1} + \varepsilon_{n-1}$ , the *rounded floor function*,  $\lfloor x'_n \rfloor_{\mathbb{R}}$ , given below, will return a correct result providing  $|\varepsilon_n - \varepsilon_{n-1}/B| < 1/2$ .

$$\lfloor x'_n \rfloor_{\mathbb{R}} \equiv \lfloor x'_n - x'_{n-1}/B \rfloor_{\mathbb{R}} \quad (13)$$

**Proof:** Substituting the definitions for  $x'_n$  and  $x'_{n-1}$  into eqn. (13) we have:

$$\begin{aligned} \lfloor x'_n \rfloor_{\mathbb{R}} &= \lfloor x_n + \varepsilon_n - (x_{n-1} + \varepsilon_{n-1})/B \rfloor_{\mathbb{R}} \\ &= \lfloor \lfloor x_n \rfloor + \varepsilon_n - \varepsilon_{n-1}/B \rfloor_{\mathbb{R}} \end{aligned}$$

Hence, under the condition

$$|\varepsilon_n - \varepsilon_{n-1}/B| < 1/2 \quad (14)$$

we have  $\lfloor x'_n \rfloor_{\mathbb{R}} = \lfloor x_n \rfloor$ .

□

We improve the error rejection by using the corrected value,  $x''_{n-1}$ , in the floor function calculation. We now have a robust reverse evolution procedure as shown below:

$$\begin{aligned} \lfloor x'_n \rfloor_{\mathbb{R}} &= \lfloor x'_n - x''_{n-1}/B \rfloor_{\mathbb{R}} \\ x''_n &= \begin{cases} \lfloor x'_n \rfloor_{\mathbb{R}} + x''_{n-1}/B & n > K \\ x'_K & n = K \end{cases} \end{aligned} \quad (15)$$

The use of the *rounded floor function* vs. the normal floor function, in correcting higher order digits by reverse evolution, is demonstrated in Table 2 for  $B = 10$ . Using an exact variable,  $x = 9.9845$  (first line), we apply a 5% error to each digit, giving  $x'_n$  on the second line.  $x''_n$  (third line) is the direct computation from eqn. (11), and  $\lfloor x''_n \rfloor_{\mathbb{R}}$  (fourth line) uses the rounded floor function of eqn. (15). We see that the use of the direct computation will not tolerate a 5% error, where the MSD,  $x'_2$ , overflows the analog range (we assume that this is possible for the moment). The use of the rounded floor function tolerates this error. The final corrected value is shown in boldface; we also see that a small amount of the error in the LSD has been propagated upwards, and we will discuss this in section 3.3.

**Table 2: Comparison between the rounded and normal floor function in reverse evolution**

$n$	2	1	0	-1	-2
$x_n$	9.9845	9.845	8.45	4.5	5
$x'_n$	10.483725	10.33725	8.8725	4.725	5.25
$x''_n$	11.084525	10.84525	8.4525	4.525	5.25
$\lfloor x'_n \rfloor_{\mathbb{R}}$	$\lfloor 9.3992 \rfloor_{\mathbb{R}} = 9$	$\lfloor 9.492 \rfloor_{\mathbb{R}} = 9$	$\lfloor 8.42 \rfloor_{\mathbb{R}} = 8$	$\lfloor 4.2 \rfloor_{\mathbb{R}} = 9$	-
$\lfloor x''_n \rfloor_{\mathbb{R}}$	<b>9.984525</b>	9.84525	8.4525	4.525	5.25

We must point out here that we are making an implicit assumption that the floor function represents an exact integer value (or continuous variable equivalent) and that the cascade rule

summation does not contain errors. These are clearly strong assumptions and have to be resolved when the implementation is to be considered. Given a low value of  $B$  it is certainly possible to produce accurately controlled analog variables (e.g., based on bandgap principles [18]) that can be used in error recovery along with the inherent error tolerance of the reverse evolution procedure. This perhaps is our analog equivalent of the natural signal recovery associated with digital systems. Another approach might be to develop a hybrid (analog/digital) circuit in which the floor function values can be exactly represented as integers. These are questions that are being answered in a current study, to be published later.

### 3.2 Modulus overflow correction

In Table 2 we have a situation where an error in  $x_2$  causes the digit value,  $x'_2 = 10.01123$ , to exceed the modulus,  $B=10$ . In arithmetic operations (to be discussed in section 4) a modulo operator would change the digit value to  $x'_n = (10.01123) \bmod B = 0.01123$ . In this situation the rounded floor function of eqn. (13) no longer gives the correct value of  $\lfloor x'_2 \rfloor = 9$ , but instead provides a value of  $\lfloor x'_2 \rfloor = -1$ . We can correct for this by redefining the rounded floor function of eqn. (15) as shown below:

$$\lfloor x'_n \rfloor_{\mathbb{R}} \equiv \begin{cases} \lfloor x'_n - x''_{n-1}/B \rfloor_{\mathbb{R}} \bmod^+ B & \hat{x} \geq 0 \\ \lfloor x'_n - x''_{n-1}/B \rfloor_{\mathbb{R}} \bmod^- B & \hat{x} < 0 \end{cases} \quad (16)$$

$$x''_n = \begin{cases} \lfloor x'_n \rfloor_{\mathbb{R}} + x''_{n-1}/B & n > K \\ x'_K & n = K \end{cases}$$

where  $\hat{x}$  is the value measured from the MSD, as defined in eqn. (9). The  $\bmod^+$  and  $\bmod^-$  functions are defined below:

$$\begin{aligned} (a) \bmod^+ B &= a + I \cdot B; & 0 \leq (a) \bmod^+ B < B \\ (a) \bmod^- B &= a + I \cdot B; & -B < (a) \bmod^- B \leq 0 \end{aligned} \quad (17)$$

with  $I$  a signed integer. Table 3 demonstrates the effect of including the mod  $B$  computation in the floor function of eqn. (16) when the digit error causes modulus overflow.

**Table 3: Floor function comparison with Mod  $B$  computation**

$n$	<b>2</b>	<b>1</b>	<b>0</b>	<b>-1</b>	<b>-2</b>
$x_n$	9.9845	9.845	8.45	4.5	5
$x'_n$ with Mod $B$	0.483725	0.33725	8.8725	4.725	5.25
$\lfloor x'_n \rfloor_{\mathbb{R}}$ - eqn. (13)	0	-1	8	4	-
$\lfloor x'_n \rfloor_{\mathbb{R}}$ - eqn. (16)	9	9	8	4	-

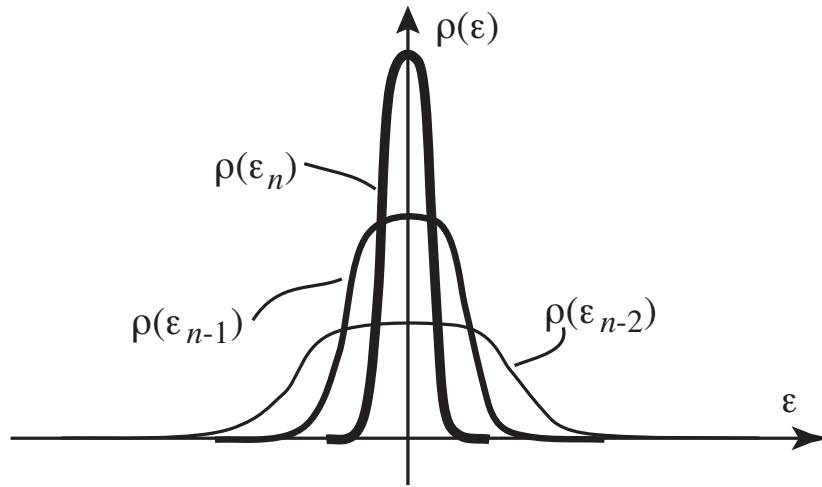
Thus we see that in the special cases where  $\lfloor x_n \rfloor = 0$  or  $\lfloor x_n \rfloor = B-1$ , this method eliminates potential catastrophic consequences in reverse evolution.

### 3.3 LSD error propagation

In general we can assume that prior to using reverse evolution, all digit errors have a substantially equal error probability density function (EPDF),  $\rho(\varepsilon)$ . This is quite reasonable considering that all the analog digit operations will be implemented with similar circuits over the same analog range. With reverse evolution, the errors from all digits, except for the LSD, are removed providing the condition given by Theorem 1 (inequality (14)) is met and the modulus overflow is corrected using the modified floor function of eqns. (16). The LSD error is propagated upwards as the reverse evolution proceeds. Table 2 demonstrates that this error reduces by the factor  $B$  for each successive reverse evolved digit. In particular, given an EPDF for the  $n$ th digit,  $\rho(\varepsilon_n)$ , then  $\rho(\varepsilon_{n+1}) = B \cdot \rho(\varepsilon_n/B)$  and the error that propagates to the MSD in reverse evolution is given below:

$$\varepsilon'_L = \varepsilon_K / B^{L-K} \quad (18)$$

This propagation of EPDF (reduction of variance) for the successively evolved digits is sketched in Figure 3 for three adjacent digits using  $B=2$ .



**Figure 3: EPDF propagation for 3 adjacent evolved digits ( $B=2$ )**

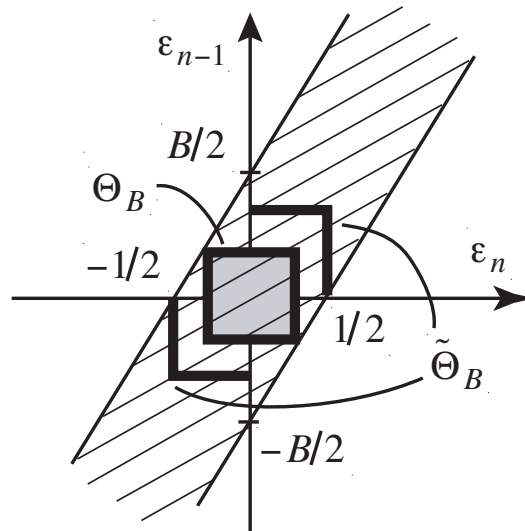
Since  $L-K$  represents the number of digits, we see that it also controls the precision of the final result, as is the case with a PNS.

### 3.4 Error Tolerance

The inequality in (14) provides a bound on adjacent digit errors,  $(\varepsilon_n, \varepsilon_{n-1})$  that can be tolerated for correct retrieval of the rounded floor function,  $\lfloor x'_n \rfloor_{\mathbb{R}}$ . In Figure 4 we have plotted  $\varepsilon_n$  vs  $\varepsilon_{n-1}$  and have hatched the region in which conditions provided by (14) are met. Since we also have the assumption of equal EPDF for all digits, then the actual bound on the adjacent errors is the square region,  $\Theta_B$ . If we are able to make an assumption of biased errors, for example offsets of the same sign, then we can relax the bound to the outer exploded square,  $\tilde{\Theta}_B$ .

The error threshold,  $\vartheta \geq \varepsilon_n$ , is obtained from  $\Theta_B$  as:

$$\vartheta = \frac{B}{2(B+1)} \quad (19)$$



**Figure 4: Error Map for Adjacent Digits.**

For error values  $\varepsilon_n$  and  $\varepsilon_{n-1}$  of the same sign then the error threshold, from  $\tilde{\Theta}_B$ , changes to the more tolerant  $\tilde{\vartheta} = 1/2$ .

In Table 4 we list values of these thresholds for selected radices. Implementation requirements are best formulated by relative error thresholds  $\vartheta/B$  and  $\tilde{\vartheta}/B$ , included as percentages in the table. We also compare these tolerances with tolerances in a Multiple-Valued Logic (MVL) circuit, where digit values are integers.

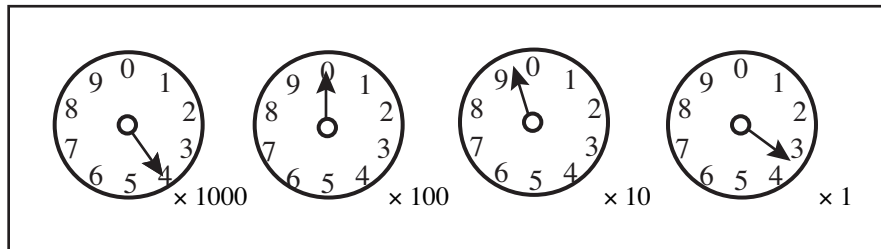
From the table we make the following observations: the integer valued digit tolerance is a constant, 0.5, and the CVD digit tolerance is radix dependent. Level discriminators tolerate at most an absolute digit error of 0.5, regardless of the radix. Hence, for digits that are by definition integers, we can write  $\varepsilon_n < \vartheta = 0.5$ . While the relative circuit tolerance for analog digits is stricter than for multiple-valued digits at low radix values, the tolerances converge with increasing radix. As with MVL circuits, higher radix values in CVNS can be implemented if more precise circuits are employed. In the next few sections we will show that CVNS, as opposed to MVL, does not

require a radix greater than  $B = 2$  to benefit from more precise circuits. A general drawback of higher radices is that the associated level detection circuitry becomes increasingly complex.

**Table 4: Threshold Values for Selected Radices**

$B$	2	4	8	10	100
<b>MVL</b>					
$\vartheta$	0.5	0.5	0.5	0.5	0.5
$\vartheta/B$	50.0%	16.7%	7.14%	5.56%	0.51%
<b>CVNS</b>					
$\vartheta$ - eqn. (19)	0.333	0.400	0.444	0.455	0.495
$\vartheta/B$	16.7%	10.0%	5.56%	4.55%	0.50%
$\tilde{\vartheta}/B$	25.0%	12.5%	6.25%	5.00%	0.50%

**Example 2:** In Figure 1, in the introduction, we gave the example of a utilities meter with  $B = 10$ . Let us consider the same meter, but with a different set of readings, as shown in Figure 5.



**Figure 5: Utilities meter reading for Example 2**

Using our notation, the meter is read as  $x \Rightarrow (4, 0, 9.5, 3.4|0)$  with  $X = 10^4$ . By applying reverse evolution with eqn. (16) we generate the values shown in Table 5. An imprecision in the setting of a dial, and an inaccuracy in the reading of it, is the equivalent of generating and sensing voltages, currents or charges in analog circuits with limited accuracy. We see that the CVNS representation allows much greater precision than any of the individual digit readings.

**Table 5: Reverse evolution results for Example 2**

$n$	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
$x'_n$	4	0	9.5	3.4
$\lfloor x'_n \rfloor_{\text{R}}$ - eqn. (16)	3	9	9	-
$x''_n$	3.9934	9.934	9.34	3.4

It is important to note that the applied method *with* modulo operations tolerates the critical cases where a dial setting of for instance  $x_n = 9.9$  is read as  $x'_n = 0$ . Reverse evolution considers only pairs of digits, and converges to a solution  $\hat{x}$  within  $L-K$  steps. As such it is practical and fast. Other methods may certainly be conceived that consider more digits in fewer steps and utilise the digit redundancy in a different way

## 4. CVNS Arithmetic

### 4.1 Addition

In this section we will show that the properties of the CVNS allow us to compute addition independently on each of the digits within the conversion accuracy specified in eqn. (14). This promises the use of analog circuitry to perform arbitrary precision arithmetic in an imperfect, noisy, and non-linear environment. We limit ourselves to unsigned addition in order to provide a more straightforward introduction. Signed addition (subtraction) is introduced in section 4.2.

**Theorem 2:** Given two CVNS represented numbers,  $x$  and  $y$ , the CVNS digits of the sum,  $z = x + y$ , are given as follows:

$$z_n = (x_n + y_n) \bmod B \quad (20)$$

**Proof:** From eqn. (5) we can write eqn. (20) as:

$$z_n = \left( \frac{(x+y)}{X} \cdot B^{L-n+1} \right) \bmod B \quad (21)$$

This can be expanded as:  $z_n = \left( \frac{x}{X} \cdot B^{L-n+1} + \frac{y}{X} \cdot B^{L-n+1} \right) \bmod B = (x_n + y_n) \bmod B$ .

□

It is clear that the same error recovery mechanism discussed in section 3 will still work for addition with the proviso that the digit error,  $\varepsilon_n$ , used in section 3 be defined as  $\varepsilon_n \equiv \varepsilon_{x_n} + \varepsilon_{y_n}$ .

In order to demonstrate this, we provide the following example.

**Example 3:** Consider the addition of  $z = 58.34 + 72.89 = 131.23$  with  $X = 1000$  and  $B = 10$ . We

will add a fixed error of  $\frac{0.05}{3}B = 0.16667$  to each of the CVNS representations of  $x$  and  $y$  and then

another fixed error of  $0.16667$  to the CVNS addition. Note that the accumulation of these fixed errors in the addition process satisfies the error bound of (14) and so the final result should be correct except for the propagated LSD error. We have selected  $L = 2$ , to hold the correct MSD of the addition (note that the MSD is an EED for the two input variables), and  $K = -2$  to reduce the propagated error by  $10^{-5}$ .

We show the results in Table 6. Note that the corrected representation of  $z''_n = \lfloor z'_n \rfloor_{\text{R}} + z''_{n-1}/B$  has removed all errors except for the propagation of the LSD error, to provide a result of  $z'' = 131.235$ .

An important observation is that the addition operation itself is digit-wise with no carry or other interaction required between neighbouring digits. The interaction between digits takes place in the reverse evolution process and this only needs to take place when the error rejection tolerance of eqn. (14) is likely to be exceeded [13].

**Table 6: Results from Example 3, CVNS addition  $x=58.34, y=72.89$** 

$n$	<b>2</b>	<b>1</b>	<b>0</b>	<b>-1</b>	<b>-2</b>
$x_n$	0.5834	5.834	8.34	3.4	4
$x'_n$	0.75007	6.00067	8.506667	3.56667	4.06667
$y_n$	0.7289	7.289	2.89	8.9	9
$y'_n$	0.89557	7.45567	3.056667	9.06667	9.16667
$z_n = (x' + y')_n$	1.64563	3.45633	1.563333	2.63333	3.33333
$z'_n$	1.8123	3.623	1.73	2.8	3.5
$\lfloor z'_n \rfloor_{\mathbb{R}} - (16)$	1	3	1	2	3.5
$z''_n$	1.31235	3.1235	1.235	2.35	3.5

#### 4.2 Subtraction (signed addition)

We can perform subtraction by the addition of negative numbers. Using the standard CVNS we have the negation rule of eqn. (7),  $(-x)_n = -x_n$ . We therefore can use Theorem 2 to define subtraction between digits as shown in eqn. (22).

$$z_n = (x_n + (-y)_n) \bmod B = (x_n - y_n) \bmod B \quad (22)$$

Since the sign of a number after the addition process is only guaranteed for the MSD,  $n = L$ , we need to use the sign of the MSD to convert lower order digits,  $n < L$ , of the result to the correct sign, if required, in order to return the result to a signed CVNS representation. In order to accomplish this we form the addition computation of eqn. (23), which generates the correct CVNS digit sign for the mod  $B$  operation.

$$z_n = \begin{cases} (x_n + y_n) \bmod^+ B & (x_L + y_L) \geq 0 \\ (x_n + y_n) \bmod^- B & (x_L + y_L) < 0 \end{cases} \quad (23)$$

We also modify the rounded floor function procedure of eqn. (16) in a similar way as shown below:

$$\lfloor z'_n \rfloor_{\mathbb{R}} \equiv \begin{cases} [z'_n - z'_{n-1}/B]_{\mathbb{R}} \bmod^+ B & (x_L + y_L) \geq 0 \\ [z'_n - z'_{n-1}/B]_{\mathbb{R}} \bmod^- B & (x_L + y_L) < 0 \end{cases} \quad (24)$$

This is best explained by means of the following example.

**Example 4:** We will modify Example 3 by forming the addition  $z = 58.34 + (-72.89) = -14.55$ , which is the same as the subtraction  $z = 58.34 - 72.89 = -14.55$ ; again we let  $X = 1000$  and  $B = 10$ , and we add a fixed error of  $\frac{0.05}{3}B = 0.16667$  to each of the CVNS representations of  $x$  and  $y$  and then add another fixed error of  $0.16667$  to the CVNS addition. Here we select  $L = 1$ , since the result will be within the range  $-100 < z < 100$ , and we let  $K = -2$  as before. The results are provided in Table 7. Note the propagation of the LSD error to provide a result of  $z'' = 14.545$ .

**Table 7: Results from Example 4, CVNS addition,  $x=58.34$ ,  $y=-72.89$**

$n$	1	0	-1	-2
$x_n$	5.834	8.34	3.4	4
$x'_n$	6.00067	8.506667	3.56667	4.06667
$y_n$	-7.289	-2.89	-8.9	-9
$y'_n$	-7.12233	-2.723333	-8.73333	-8.83333
$z_n = (x' + y')_n$	-1.12167	-4.216667	-5.16667	-4.66667
$z'_n$	-0.955	-4.05	-5	-4.5
$\lfloor z'_n \rfloor_{\mathbb{R}} - (24)$	-1	-4	-5	-4.5
$z''_n$	-1.4545	-4.545	-5.45	-4.5

### 4.3 Multiplication

Unlike addition, multiplication cannot be directly performed independently digit-wise. We can see this from the following inequality:

$$(x \times y)_n = \left( \frac{x \times y}{X} \cdot B^{L-n+1} \right) \bmod B \neq x_n \times y_n = \left( \frac{x \times y}{X^2} \cdot B^{2(L-n+1)} \right) \bmod B \quad (25)$$

We can, however, emulate multiplication by summation of partial products, as is done with conventional PNS arithmetic. This is possible because of the following theorem.

**Theorem 3:** The CVD's of a product  $\lambda \cdot x$  with integer  $\lambda$  are given as:

$$(\lambda x)_n = (\lambda \cdot x_n) \bmod B \quad (26)$$

**Proof:** The proof simply follows from the definition of the  $n$ th CVD given by eqn. (5).

$$(\lambda x)_n = \left( \frac{1}{X} \left( \sum_{i=1}^{\lambda} x \right) \cdot B^{L-n+1} \right) \bmod B = \left( \sum_{i=1}^{\lambda} \frac{x}{X} \cdot B^{L-n+1} \right) \bmod B = (\lambda x_n) \bmod B$$

□

For the special case  $\lambda = B^k$ , we have  $(B^k \cdot x)_n = x_{n-k}$ , and if  $y = \sum_{\forall k} \lambda_k B^k$ , then

$$(y \cdot x)_n = \left( \sum_{\forall k} \lambda_k \cdot x_{n-k} \right) \bmod B \quad (27)$$

Eqn. (27) represents the PNS equivalent of multiplication by the summation of partial products, where  $x$  is represented in CVNS and  $y$  is represented in a PNS, normally of the same radix. If  $B = 2$ , then  $y$  is represented in binary, and  $\lambda_k$  serves as an on-off switch for summing analog voltages, currents or charges,  $x_{n-k}$ . We can also use a signed digit representation of  $y$  in order to reduce the number of partial products being summed.

Errors in the CVD's of  $x$ ,  $x'_n$ , cause errors in the partial product digits. The integers,  $\lambda_k$ , amplify these digit errors, and so it is beneficial to limit their range. If  $y \geq 0$ , then the range  $0 \leq \lambda_k \leq B-1$  is required, but if we use signed values of  $\lambda_k$ , then we can reduce the range to  $0 \leq |\lambda_k| \leq B/2$ . Since signed digits provide a redundant representation, we can select a representation that will minimise errors in the CVNS representation of  $x$ . This can include limiting the size of the  $\lambda_k$  as well as using mixed signs to reduce the accumulated error in the partial product summations. The latter implies, in general, a smaller amplification of  $\varepsilon_n$  when  $B \geq 3$ , and it is preferable to represent  $y$  such that  $|\lambda_k|$  are minimised. Using a signed binary representation for  $y$  appears to offer the most flexibility in error reduction. As with addition, the mitigation of the errors in the multiplication process is based on observing the bound set by (14).

Decimal multiplication is illustrated in the following example.

**Example 5:** We will perform a CVNS multiplication of  $z = x \cdot y$  for  $x = 31.89$  and  $y = 19.5$ , using a decimal radix of  $B = 10$  with a range of  $X = 1000$ . We use the representation,  $x \Rightarrow (0.3189, 3.189, 1.89 | 8.9, 9)$ , for the multiplicand, which includes an EED,  $x_2$ , in order to handle the shifts required in the partial product computations. Thus  $L = 2$  and  $K = -2$ . We perform the computation using two different representations for  $y$ : a direct representation,  $\lambda \Rightarrow (1, 9, 5)$ , and a signed digit, or redundant representation,  $\lambda \Rightarrow (2, 0, -5)$ . The results are provided in Table 8.

In order to demonstrate the error rejection differences between the two representations for  $y$  we have included an added fixed error,  $\varepsilon^+$  to  $x$ ,  $x' = x + \varepsilon^+$ , and to the partial product sum,

$$z' = \sum_{i=-1}^1 \vec{\lambda}_i \cdot x' + \varepsilon^+.$$

We are using the notation,  $\vec{\lambda}_i$ , to indicate a shifted version of the multiplier

digit; i.e.,  $\overrightarrow{\lambda}_n$  provides a left shift of  $n$  places and  $\overleftarrow{\lambda}_{-n}$  a right shift of  $n$  places. We have also included a multiplicative, or gain, error for each of the partial product computations  $(\lambda_i \cdot x') = (1 + \varepsilon^*) \cdot (\lambda_i \cdot x')$ . The errors,  $\varepsilon^+$  and  $\varepsilon^*$ , have been set at values just below the point where errors occur in the rounded floor function. We note that the redundant multiplier representation tolerates a considerably larger error than the direct representation, as we would expect.

**Table 8: Example of multiplication using direct and redundant multiplier representations**

$n$	<b>2</b>	<b>1</b>	<b>0</b>	<b>-1</b>	<b>-2</b>	<b>2</b>	<b>1</b>	<b>0</b>	<b>-1</b>	<b>-2</b>
$x'_n$	0.3489	3.219	1.92	8.93	9.03	0.4689	3.339	2.04	9.05	9.15
$(\overrightarrow{\lambda}_1 \cdot x')$	3.2254	1.9238	8.9479	9.0481	0	6.7579	4.1209	8.2012	8.4042	0
$(\overrightarrow{\lambda}_0 \cdot x')$	3.1464	8.989	7.2946	0.3707	1.2725	0.14	0.14	0.14	0.14	0.14
$(\overleftarrow{\lambda}_{-1} \cdot x')$	0.2045	1.7745	6.125	9.63	4.68	-0.233	-2.329	-6.745	-0.152	-5.278
$z'_n$	6.6063	2.7173	2.3974	9.0788	5.9825	6.680	1.9320	1.5965	8.3919	4.862
$\lfloor z'_n \rfloor_{\mathbb{R}}$	6	2	1	8	5.9825	6	2	1	8	4.862
$z''_n$	6.2186	2.1859	1.8595	8.5953	5.9825	6.2185	2.1849	1.8486	8.4862	4.862

$$\lambda \Rightarrow (1, 9, 5); \quad \varepsilon^+ = 0.03, \varepsilon^* = 0.2\%$$

$$\lambda \Rightarrow (2, 0, -5); \quad \varepsilon^+ = 0.14, \varepsilon^* = 1.5\%$$

#### 4.4 Number Comparison

The ability to compare numbers is of essential importance to the implementation of practical algorithms, including the basic arithmetic operation of division.

The comparison procedure successively examines corresponding pairs of CVD's in  $x$  and  $y$ . If the difference lies outside of the error tolerance,  $2\vartheta$  (from eqn. (19)), then a definite comparison has been made and the algorithm may be terminated. If the digits are within the error tolerance, then we need to examine the next lower order digits. The procedure is more rigorously described in Algorithm 1.

**Algorithm 1:** Magnitude Comparison

*Input:*  $x \Rightarrow (x_L, \dots, x_0 | x_{-1}, \dots, x_K)$ ;  $y \Rightarrow (y_L, \dots, y_0 | y_{-1}, \dots, y_K)$

*Output:* Booleans:  $x = y$ ,  $x > y$ ,  $x < y$

*Step 1.* Set  $n = L$ . Compute  $\vartheta = \frac{B}{2(B+1)}$ .

*Step 2.* If  $x_n > y_n + \vartheta$  set  $\begin{cases} x > y & \text{TRUE} \\ x = y & \text{FALSE} \\ x < y & \text{FALSE} \end{cases}$  and terminate.

If  $x_n < y_n - \vartheta$  set  $\begin{cases} x > y & \text{FALSE} \\ x = y & \text{FALSE} \\ x < y & \text{TRUE} \end{cases}$  and terminate.

Otherwise set  $\begin{cases} x > y & \text{FALSE} \\ x = y & \text{TRUE} \\ x < y & \text{FALSE} \end{cases}$  and continue.

*Step 3.*

Set  $n-1 \Rightarrow n$

If  $n = K$  terminate.

Otherwise go to Step 2

The maximum number of iterations is clearly  $L-K+1$ .

We use the following example to illustrate Algorithm 1.

**Example 6:** Given a number  $x = 5.839$  which has a decimal CVNS approximation of  $x' \Rightarrow (5.8 | 8.4, 3.9)$ , and a second number  $y = 5.829$  with an approximation  $y' \Rightarrow (5.9 | 8.4, 2.9)$ , we

require to determine whether  $x = y$ ,  $x > y$  or  $x < y$ . We note that both numbers have imperfect but correctible CVD's. For the Step 2 magnitude comparisons we use  $\vartheta = \frac{B}{2(B+1)} = 0.45455$ . Table 9 shows the results of applying Algorithm 1. For the first 2 iterations, Step 2 returns a comparison of  $(x=y) \Rightarrow \text{TRUE}$ , and in fact if we terminate the algorithm at either iteration 1 or 2 we are simply defining equality within the error tolerances 0.45455 and 0.045455 respectively. At the final iteration, with an error tolerance of 0.0045455, we find that  $x > y$ .

**Table 9: Number comparison results for Example 6**

$n$		<b>1</b>	<b>0</b>	<b>-1</b>
$x'_n$		5.8	8.4	3.9
$y'_n$		5.9	8.4	2.9
Iteration 1	$x > y$			
	$x = y$	TRUE		
	$x < y$			
Iteration 2	$x > y$			
	$x = y$		TRUE	
	$x < y$			
Iteration 3	$x > y$			TRUE
	$x = y$			
	$x < y$			

#### 4.5 Sign detection

Sign detection is performed in a similar manner to magnitude comparison, but we set one of the operands in the comparison to precisely 0. The comparison is halted when the error tolerance is exceeded. Example 7 demonstrates the procedure.

**Example 7:** Consider a CVNS representation with  $B=10$ ,  $L=1$  and  $K=-1$ . A negative number  $x = -1.645$  has an ideal CVNS representation of  $x \Rightarrow (-0.1645, -1.645, -6.45)$  but let us assume

that the analog circuitry used to store the number has offset each of the digits by  $+0.167$ . We now have the approximate representation  $x' \Rightarrow (0.00217, -1.4783, -6.2833)$  in which the MSD has a changed sign. Using Algorithm 2, Iteration 1 produces a comparison within the error tolerance of 0.45455 and so we do not terminate. The comparison in Iteration 2 is outside of this tolerance and so we terminate the algorithm with the result that the number is negative. Note that as an alternative we can perform all iterations and then examine all of the results that produce comparisons outside of the error tolerance. They should all produce the same sign, but this technique does allow for a more robust voting procedure if there is a conflict in the results.

#### 4.6 Division

For brevity, we will not discuss any division algorithms here except to point out that we have implementations for all the operations required by most division algorithms; namely, multiplication, subtraction, sign detection and magnitude comparison. Finding efficient division algorithms will be the subject of a future publication.

### 5. Radix Conversion

In Table 4 we have illustrated the relationship between the choice of the radix, and the required implementation accuracy. For flexibility, the implementation tolerance for arithmetic operations may be different than the tolerance for storage or signalling. As in PNS, where we perform arithmetic with (signed) integer digits, the radix of choice for a multiplier may be quite different than for storage. We could, for example, decide to build a 4-level storage memory, and independently implement multipliers in 3-level bi-directional circuits. In CVNS we have similar flexibility. Radix conversion allows us, for instance, to perform signalling with radix 4, and arithmetic with radix 2. The advantage of a high radix is the lower number of digits required to cover a given number range and accuracy, and the advantage of a low radix is the high digit-error tolerance. The discussion of the decisions involved in selecting a radix for storage or for

arithmetic circuits lies beyond the scope of this paper. Here we simply present the mathematical principles.

Theorem 4 provides a simple relationship between digits in two different CVNS representations.

**Theorem 4:** Consider a number,  $x$ , that is represented by CVNS digits  $x_n^{[B_1]}$  with MSD index  $L_1$  and radix  $B_1$ . Given a second CVNS representation, with digits  $x_n^{[B_2]}$ , MSD index  $L_2$  and radix  $B_2$ , then:

$$x_n^{[B_2]} = x_n^{[B_1]} \cdot \frac{B_2}{B_1} \quad (28)$$

where indices  $n$  and  $k$  are related by  $B_2^{L_2-k} = B_1^{L_1-n}$ .

**Proof:** The proof is based on the notion that the MSD,  $x_{L_1}^{[B_1]}$  can always be retrieved from any

digit  $x_n^{[B_1]}$  and associated integers  $\left\lfloor x_{L_1}^{[B_1]} \right\rfloor \dots \left\lfloor x_{n+1}^{[B_1]} \right\rfloor$  as given below:

$$x_{L_1}^{[B_1]} = \frac{x_n^{[B_1]}}{B_1^{L_1-n}} + \sum_{i=0}^{L_1-n-1} \frac{\left\lfloor x_{L_1-i}^{[B_1]} \right\rfloor}{B_1^i} \quad (29)$$

For the MSD (considered as an EED - see section 2.1) of a CVNS with radix  $B$  we have  $x_L = \frac{B \cdot x}{X}$

and so  $\frac{x}{X} = \frac{x_{L_1}^{[B_1]}}{B_1} = \frac{x_{L_2}^{[B_2]}}{B_2}$  yielding:

$$x_{L_2}^{[B_2]} = B_2 \cdot \frac{x_{L_1}^{[B_1]}}{B_1} \quad (30)$$

Furthermore, from eqn. (5), any digit  $x_k^{[B_2]}$  can be calculated from the MSD,  $x_{L_2}^{[B_2]}$ , by:

$$x_k^{[B_2]} = \left( x_{L_2}^{[B_2]} \cdot B_2^{L_2-k} \right) \bmod B_2 \quad (31)$$

If we now substitute eqn. (29) into eqn. (30), and eqn. (30) into eqn. (31) we have the following result:

$$x_k^{[B_2]} = \left( x_n^{[B_1]} \cdot \frac{B_2}{B_1} \cdot \frac{B_2^{L_2-k}}{B_1^{L_1-n}} + B_2 \cdot \sum_{i=0}^{L_1-n-1} \left[ x_{L_1-i}^{[B_1]} \cdot \frac{B_2^{L_2-k}}{B_1^{i+1}} \right] \right) \bmod B_2 \quad (32)$$

Using the relationship  $B_2^{L_2-k} = B_1^{L_1-n}$ , we can rewrite eqn. (32) as:

$$x_k^{[B_2]} = \left( x_n^{[B_1]} \cdot \frac{B_2}{B_1} + B_2 \cdot \sum_{i=0}^{L_1-n-1} \left[ x_{L_1-i}^{[B_1]} \cdot B_1^{L_1-n-i-1} \right] \right) \bmod B_2 \quad (33)$$

It is easy to show that  $L_1-n-i-1 \geq 0$  and so  $\left( B_2 \cdot \sum_{i=0}^{L_1-n-1} \left[ x_{L_1-i}^{[B_1]} \cdot B_1^{L_1-n-i-1} \right] \right) \bmod B_2 = 0$ . Since

$$x_n^{[B_1]} < B_1, \text{ then } \left( x_n^{[B_1]} \cdot \frac{B_2}{B_1} \right) \bmod B_2 = x_n^{[B_1]} \cdot \frac{B_2}{B_1} \text{ and so } x_k^{[B_2]} = x_n^{[B_1]} \cdot \frac{B_2}{B_1}.$$

□

If we select  $B_2 = B_1^\alpha$ , then for  $B_2^{L_2-k} / B_1^{L_1-n} = 1$  we find  $k = \frac{(n+1)}{\alpha} - 1$ . With  $B_1, B_2 \geq 2$  we

have  $\alpha = \frac{n+1}{k+1} > 0$ . Since  $n, k \in \mathbb{Z}$  then we will not have a full cover of digits for the conversion.

For an error-free computation, this is not an issue since we can recover all other digits from a single converted digit. With errors present in the original CVNS digits and in the computations, then the best strategy is to use the mapping of eqn. (28) for the matched digit indices and eqn. (6)

with  $m > n$  for the missing digits. In this way any errors in the original digits will be reduced by  $B_2^{n-m}$ . The conversion algorithm is therefore defined as below.

**Algorithm 2:** CVNS Conversion

*Input:*  $B_1, B_2; L_1, L_2; K_1, K_2; x^{[B_1]} \Rightarrow (x_{L_1}^{[B_1]}, \dots, x_0^{[B_1]} | x_{-1}^{[B_1]}, \dots, x_{K_1}^{[B_1]})$

*Output:*  $x^{[B_2]} \Rightarrow (x_{L_2}^{[B_2]}, \dots, x_0^{[B_2]} | x_{-1}^{[B_2]}, \dots, x_{K_2}^{[B_2]})$

*Step 1.* Find the set of all tuples,  $\kappa \equiv \{(k, n) : L_2 \leq k \leq K_2; L_1 \leq n \leq K_1\}$  such that  $k = \frac{(n+1)}{\alpha} - 1$  with  $B_2 = B_1^\alpha$  and  $L_1 \geq n \geq K_1$ . Order the tuples in the set  $\kappa$  such that  $k_i > k_{i-1}$ .

*Step 2.* Compute  $x_k^{[B_2]} = x_n^{[B_1]} \cdot \frac{B_2}{B_1}$ ;  $\forall (k, n) \in \kappa$ .

*Step 3.* For each  $(k_i, n_i) \in \kappa$  compute  $x_m^{[B_2]} = (x_{k_i}^{[B_2]} \cdot B^{k_i - m}) \bmod B_2$ ; for all values of  $m$  such that  $\min(L_2, k_{i+1}) \geq m \geq \max(k_{i+1}, K_2)$ . Concatenate the digits with those from Step 2.

The following example illustrates the procedure.

**Example 8:** Given a radix-8 CVNS number  $x^{[8]} \Rightarrow (6.54|4.32, 2.56)$  in storage, it shall be converted to a binary CVNS number,  $x^{[2]}$ , with  $K = -4$  for use in an arithmetic unit. With  $B_1 = 8$  and  $B_2 = 2$ , we find  $\alpha = 1/3$ .

*Step 1.*  $\kappa = \{(2, 0), (-1, -1), (-4, -2)\}$ .

*Step 2.*  $x_2^{[2]} = 1.635; x_{-1}^{[2]} = 1.08; x_{-4}^{[2]} = 0.64$

Step 3.  $x^{[2]} \Rightarrow (1.635, 1.27, 0.54 | 1.08, 0.16, 0.32, 0.64)$

## 6. Alternate Representations

This paper has developed the concept of CVNS from the point of view of a signed number representation, as depicted in Figure 2. It is, in fact, quite possible to find other ways of representing analog digits that may have advantages over the signed number approach. Below we briefly discuss two other possible representation forms.

### 6.1 Rounded Digits

An alternative definition of the cascade rule employs the rounded value  $[x_n]_{\text{R}}$  as opposed to the floor value in eqn. (3). We will name this alternative representation, CVNS-R. The *rounding cascade rule* becomes:

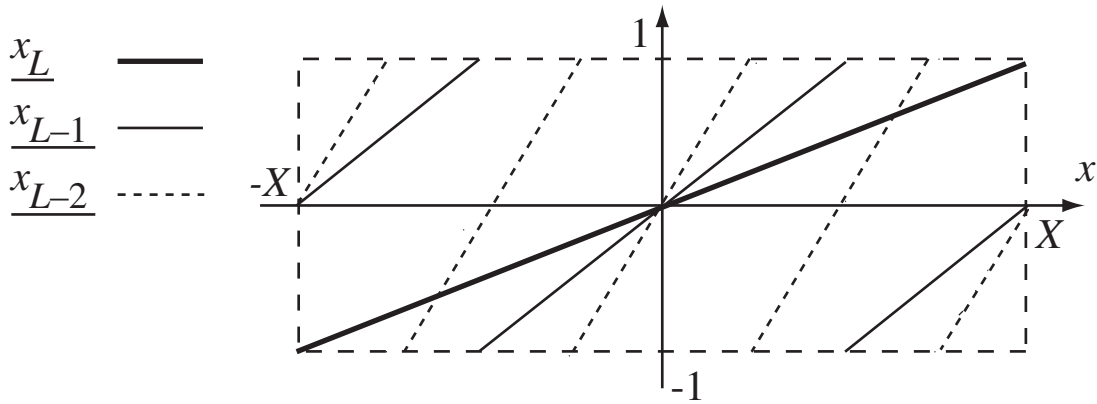
$$\underline{x}_n = (\underline{x}_{n+1} - [\underline{x}_{n+1}]_{\text{R}}) \cdot B \quad (34)$$

where the rounded digits are represented by the underscore  $\underline{x}_n$ . Using this modified definition, the dynamic range of the digit, for a signed number, is reduced by 50% to  $-B/2 < \underline{x}_n \leq B/2$ . We recall that in the CVNS, digits are mapped to quantities by  $q_n = x_n \cdot Q/B$ ; in CVNS-R they are mapped as  $q_n = \underline{x}_n \cdot Q/B$ . A plot of binary CVNS-R digit values is given in Figure 6. Comparing Figure 2 to Figure 6, we see a much better utilisation of the physical range  $-Q \dots Q$ .

### 6.2 B-Complement Digits

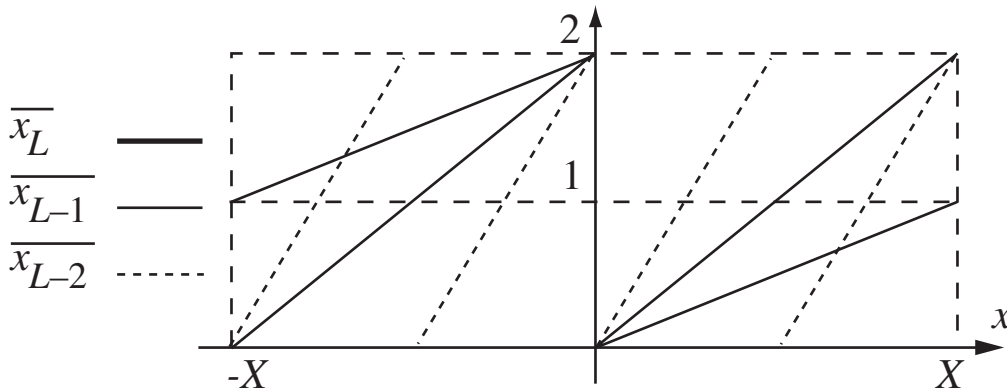
As an alternative to CVNS and CVNS-R, we also introduce a form of CVNS that allows the registration of negative and positive values of  $x$  with non-negative digits. Its digits are represented by the overscore  $\overline{x}_n$  and defined as below, with  $|x| < X/2$ :

$$\overline{x}_n = \left( \frac{x}{X} \cdot B^{L-n+1} \right) \text{mod}^+ B \quad (35)$$



**Figure 6: Signed Binary CVNS-R Digits**

Clearly now, CVD's are limited in range to  $0 \leq \bar{x}_n < B$ , and with  $q_n = \bar{x}_n \cdot Q/B$  negative values of  $q_n$  are never required. B-Complement digit values are plotted in Figure 7.



**Figure 7: B-Complement CVNS**

These digits relate to signed CVNS digits as  $\bar{x}_n = (x_n) \bmod^+ B$ , or

$$\bar{x}_n = \begin{cases} x_n & x_n \geq 0 \\ x_n + B & x_n < 0 \end{cases} \quad (36)$$

Since the radix  $B$  is added to negative values of  $x_n$  we call this form of CVNS *B-complement* (similar to 2's complement in binary arithmetic). Sign inversion is obtained as follows:

$$\overline{(-x)}_n = (B - \overline{(x)}_n) \bmod^+ B \quad (37)$$

which follows immediately from eqn. (7) and eqn. (36).

Examples for signed digits and B-complement digits are presented in Table 10. Note that the CVNS digits,  $x_n$ , are all negative, the CVNS-R digits,  $\underline{x}_n$ , have positive and negative sign, and the CVNS B-complement digits,  $\overline{x}_n$ , are all positive. Note also that  $-1 < x_n \leq 1$  for binary ( $B=2$ ).

**Table 10: Ordinary, Signed and B-Complement CVD's for  $x = -58.742$**

$n$	Decimal ( $B = 10$ )			Binary ( $B = 2$ )		
	$x_n$	$\underline{x}_n$	$\overline{x}_n$	$x_n$	$\underline{x}_n$	$\overline{x}_n$
6	-		-	-1.17484	0.82516	0.82516
5	-		-	-0.34968	-0.34968	1.6503
4	-		-	-0.69936	-0.69936	1.3006
3	-		-	-1.39672	0.60128	0.60128
2	-0.58742	-0.58742	9.4126	-0.79744	-0.79744	1.2026
1	-5.87420	4.12580	4.12580	-1.59488	0.40512	0.40512
0	-8.74200	1.25800	1.25800	-1.18796	0.81024	0.81024
-1	-7.42000	2.58000	2.58000	-0.37952	-0.37952	1.6205

## 7. Conclusions

We have introduced a novel number system, based on continuous valued digits. The formal foundations for addition and multiplication with signed continuous valued digits have been laid, and an important method for digit correction has been presented that is robust against error terms that result from implementation, as well as arithmetic residue operations. The basic operation of addition is digitwise, and carry free, but error recovery requires the equivalent of carry propagation [13].

A correction operation is performed if accumulated digit errors exceed a known threshold. With knowledge of circuit tolerances, correction hardware can be built into the arithmetic unit at design

time. The correction operation is from LSD to MSD, and as such it resembles carry propagation in normal PNS arithmetic. The fact that it is not required to correct after every addition can be exploited in the development of partial sums and the final sum in multiplication. Whether this represents the equivalent savings in the critical path of a carry-save adder array in binary multipliers remains to be seen. Certainly the reverse evolution process can be likened to the final adder required following a carry-save multiplier array.

Multiplication, together with number comparison and sign detection, open up avenues to iterative division algorithms. Radix conversion allows the implementation of memory and arithmetic circuits with different radices, catering to speed, complexity, and implementation tolerances.

### Acknowledgements

The authors acknowledge financial support from the Natural Sciences and Engineering Research Council and from the Micronet Network of Centres of Excellence, and the infrastructure support of workstations and design tools from the Canadian Microelectronics Corporation. The authors also acknowledge the many valuable comments and suggestions from the anonymous reviewers of this paper.

### References

- [1] A.P Chandrakasan, S. Sheng, R.W. Broderson, "Low-Power CMOS Digital Design", *IEEE Journal of Solid-State Circuits*, vol. 27, pp. 473-483, 1992.
- [2] W. Athas, L.J. Svensson, J.G. Koller, N. Tzartzanis, E. Chou, "Low-Power Digital Systems Based on Adiabatic Switching Principles", *IEEE Transactions on VLSI Systems*, vol. 2, pp. 398-407, 1994.
- [3] G. Korn, and T. Korn, *Electronic Analog Computers*, 2nd ed., McGraw Hill, New York, NY, 1956.

- 
- [4] S. Sadeghi-Emamchaie, G. A. Jullien, V. S. Dimitrov and W. C. Miller, 1999, "Cellular Neural Network Implementation of Digital Arithmetic using Symbolic Substitution", *Journal Circuits, Systems and Computers*, Special Issue on Neural Networks, Vol. 8, Nos. 5&6, pp. 615-635, 1998.
- [5] J. T. Butler, ed., *Multiple-Valued logic in VLSI*, IEEE Computer Society Press, 1991.
- [6] M.A. Soderstrand, W.K. Jenkins, G.A. Jullien, F.J. Taylor, *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*, IEEE Press, New York, 1986.
- [7] A. Saed, M. Ahmadi, G.A. Jullien, W.C. Miller, "Overlap Resolution: Continuous Valued Digits for Hybrid Architectures", *40th Midwest Symposium on Circuits and Systems*, Sacramento, California, vol. 1, pp. 377 -380, August 1997.
- [8] A. Saed, M. Ahmadi, G.A. Jullien, W.C. Miller, "Overlap Resolution: Arithmetic with Continuous Valued Digits in Hybrid Architectures", *Thirty-First Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, California, vol. 2, pp. 1188 -1191, November 1997.
- [9] A. Saed, M. Ahmadi, G.A. Jullien, W.C. Miller, "Analog Digits: Bit Level Redundancy in a Binary Multiplier", *Thirty-Second Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, California, vol.1, pp. 236 -240, November 1998.
- [10] A. Saed, M. Ahmadi, G.A. Jullien, W.C. Miller, "Circuit Tolerances and Word Lengths in Overlap Resolution", *1998 IEEE International Symposium on Circuits and Systems (ISCAS'98)*, Monterey, California, vol. 1, pp. 197 -200, June 1998.
- [11] A. Saed, M. Ahmadi, G.A. Jullien, "Arithmetic Circuits for Analog Digits", *29th International Symposium on Multiple Valued Logic (ISMVL)*, Freiburg, Germany, pp. 186 -191, May 1999.
- [12] A. Saed, M. Ahmadi, G.A. Jullien, "Arithmetic with Signed Analog Digits", *14th IEEE Symposium on Computer Arithmetic (ARITH14)*, Adelaide, Australia, pp. 134 -141, April 1999.
- [13] A. Saed, *Continuous Valued Digits: a Novel Direction in Multiple-Valued Arithmetic*, Ph.D. Thesis, University of Windsor, Windsor Ontario, Canada, 1998.
- [14] A. Avizienis, "Signed Digit Number Representations for Fast Parallel Arithmetic," *IRE Trans. Elec. Comput.*, pp. 389-400, Sept. 1961.
- [15] K. Pulliam, "Cyclic number codes for optical computing," *Proceedings of SPIE*, Vol. 495, Real Time Signal Processing VII, Keith Bromley, Ed., pp. 217-225, Aug. 1984.
- [16] *Handbook for Electricity Metering*, Ninth Edition, Edison Electric Institute, 1992
-

- 
- [17] Chironis, Nicholas P., ed., *Mechanics, Linkages and Mechanical Controls*, Mc Graw-Hill Book Company, 1965
- [18] H. Banba, H. Shiga, A. Umezawa, T. Miyaba, T. Tanzawa, S. Atsumi, K. Sakui. "A CMOS bandgap reference circuit with sub-1-V operation." *IEEE Journal of Solid-State Circuits*, Vol. 34, No.5 (Special Issue on the 1998 Symposium on VLSI Circuits), pp. 670-674, May 1999.