Decomposition of logic networks with emphasis on signed digit arithmetic systems

W. Kuczborski
Y. Attikiouzel
G. Crebbin

Indexing terms: Decomposition of logic networks, Logic networks, Signed digit arithmetic

Abstract: This paper describes an attempt to combine advantages of the signed digit number representation, applied at the word-level, and the residue number system applied at the digit-level, to achieve arithmetic decomposition of high-radix systems. Also introduced is a new decomposition algorithm for multiple-output Boolean functions based on partition products. Analysis of the proposed new method of arithmetic decomposition, when compared to an approach based on the theory of digit sets, reveals a more efficient use of data storage plus a higher degree of structural uniformity. The practical importance of the proposed method has been tested on a number of designs for the field programmable gate arrays. Comparison with a commercially available CAD system indicates a significant reduction in implementation complexity.

1 Introduction

Interest in the arithmetic decomposition of high-radix systems and the Boolean decomposition for multiple-output functions was prompted by the needs of real-time grey-scale morphological processors. The enormous hardware complexity of processors applying the principles of umbra transform [4] or threshold decomposition [11] directed attention towards the direct implementation of morphological transformations. The direct approach demands a fast implementation of two conflicting operations: (1) addition and maximum search (dilation); (2) subtraction and minimum search (erosion). A conflict results from the characteristic of conventional arithmetic systems, which require opposite directions of digit processing.

Image data throughputs of the order of 10^8 pixels per second and a high degree of design regularity can be achieved by applying the principles of digit-level systolic arrays [7]. However, the conflict between addition and magnitude comparison introduces loops into the dependence graph, and the whole problem becomes non-computable. The solution adopted is to replace the conventional arithmetic system by the signed digit number representation (SDNR). The SDNR eliminates loops in the dependence graph by allowing a uniform direction of addition/subtraction and magnitude comparison — from the most to the least significant digit. Moreover, carry propagation is restricted to a single digit.

SDNR systems are more difficult to implement than conventional arithmetic circuits. Boolean functions, which define the SDNR systems, are more complex and have a larger number of independent variables. Apparently some new methods of logic synthesis are required for a wider application of this non-conventional data representation. In most cases, especially for systems of radices higher than two, a complex logic network must be replaced by a cluster of simpler ones.

Carter and Robertson [2] applied the theory of digit sets to replace high-radix modules by simpler modules of radix two. An alternative approach is proposed here, based on a combination of the SDNR, applied at the word level, and the residue number system (RNS), applied at the digit level. Such a combination simplifies the complexity of basic modules and reduces the number of inputs to these modules. If the selected implementation technology, e.g. programmable logic arrays (PLAs), gate arrays, or field programmable gate arrays (FPGAs), requires a further reduction of the input number, then another design step is applied — the decomposition of the multivalued Boolean functions [3]. A new decomposition algorithm is presented here, based on partition products. The algorithm has been tested on FPGAs from Xilinx which allow five Boolean variables per logic module.

Being aware of the speed and density limitations of the technology, when compared with full custom or semi-custom designs, the choice was influenced by two strengths of FPGAs: (1) fast and zero-cost modifications of the prototypes, a significant virtue for the unconventional data representation employed; (2) the very complex SDNR functions can be efficiently implemented by look-up tables of the Xilinx FPGAs.

2 Principles of the signed digit representation (SDNR) and the residue number system (RNS)

2.1 SDNR

The SDNR [1] has become an attractive alternative for dedicated VLSI systems such as morphological processors [6], CORDIC processors [13], and IIR filters [9].

The SDNR is a redundant data format which requires additional memory space. However, the redundancy reduces carry propagation to at most a single digit. The

Paper 1109G (C2), first received 22nd February and in revised form 20th December 1993

W. Kuczborski is at Edith Cowan University, Department of Computer and Communication Engineering, Joondalup WA6065, Australia

Y. Attikiouzel and G. Crebbin are with The University of Western Australia, Centre for Intelligent Information Processing Systems, Nedlands WA6009, Australia

SDNR has a number of important advantages: parallel processing of all digits, modularity, variable operand lengths, regular logic structures, and local connections. SDNR, unlike RNS, is a positional number system:

$$N_{SDNR} = \sum_{i=0}^{n} a_i r^i$$

The digits a_i can be negative, zero, or positive. For symmetric digit sets $\{-a, -a+1, \ldots, -1, 0, 1, \ldots, a-1, a\}$, the allowed range of a becomes

$$a_{min} = r/2 + 1$$

(minimal redundant set), and

$$a_{max} = r - 1$$

(maximal redundant set), where r is the radix. A sensible choice of carry values is within the range -1 to 1. The threshold value t, which determines carry generation, must be within the range:

$$1 \leqslant r - a \leqslant t \leqslant a - 1$$

For the addition $SUM = X + Y$, the sum digit at the ith position is a function of only four arguments, X_i, Y_i, X_{i-1}, Y_{i-1}, and can be calculated in two stages:

Stage 1:

$$\text{if} \quad X_i + Y_i > t \quad \text{then } C_{i-1} = 1$$

$$\text{else if} \quad X_i + Y_i < -t \text{ then } C_{i-1} = -1$$

$$\text{else} \quad C_{i-1} = 0$$

$$S_i = X_i + Y_i - rC_{i-1}$$

Stage 2:

$$SUM_i = S_i + C_i$$

2.2 RNS

The RNS [5, 12, 14] is defined by a set of relatively prime moduli

$$\{p1, p2, \ldots, pn\}$$

The dynamic range of the numbers is defined by the product of all moduli:

$$0 \cdots (p1 \times p2 \times \cdots \times pn) - 1$$

The use of the RNS for coding the SDNR digits requires a range which includes negative numbers as well:

$$-(p1 \times p2 \times \cdots \times pn/2) \cdots (p1 \times p2 \times \cdots \times pn/2) - 1$$

A numerical value N can be converted into its n-digit RNS equivalent

$$N = \langle XP1, XP2, \ldots, XPn \rangle$$

by modulo operations:

$$XP1 = N \bmod p1,$$

$$XP2 = N \bmod p2, \ldots, XPn = N \bmod pn$$

The main advantage of the residue arithmetic lies in its parallelism — additions, subtractions, and multiplications can be executed at each digit independently. This parallelism was utilised in the first stage of logic synthesis — arithmetic decomposition (next section).

Despite its parallelism, the RNS is not an efficient representation of image data in the morphological processor. The reason for this is the strongly non-linear character of mathematical morphology, requiring constant magnitude comparisons. According to Winograd's lower bound theorem [14], the speed potential of the RNS can be fully utilised only if the number of additions significantly outnumbers the number of magnitude comparisons. However, the usual difficulties with RNS magnitude comparisons or sign detections can be avoided if the application of the RNS is restricted to SDNR digit level.

3 Arithmetic decomposition based on the SDNR and the RNS

As mentioned above, the SDNR system is used to represent data of the grey-scale morphological processor. Although the signed digit representation demands additional storage space for signal/image data, memory requirements can be reduced for systems with higher radices. Higher radices have other advantages too. They reduce the interconnection complexity, which is an important consideration in view of the fact that interconnections occupy approximately 70% of silicon area of the VLSI devices. Finally, high-radix systems may be faster — for example, multiplication will be completed after fewer algorithmic steps.

The logic synthesis procedure is executed in two consecutive steps. The first step, based on a combination of SDNR and RNS, reduces the number of inputs to each module by a factor of two. The SDNR, applied at the word level, allows easy magnitude comparisons and sign detections. The RNS, applied at the digit level, decomposes each SDNR digit into simpler networks. Since the dynamic range of RNS values is very small and is determined by the SDNR digit set, the RNS conversions and sign detections are handled by unified and simple logic circuits. It is seen in the next section that the SDNR/RNS decomposition method reduces storage requirements, when compared to an alternative method based on the theory of digit sets [2].

An important decision for designers of an RNS system is the choice of relatively prime moduli. Assuming the system to be implemented using conventional digital circuits and restricting the maximum number of inputs in each module to six, then, the choice of moduli will depend on the required set of SDNR digits. Table 1 specifies the choices of moduli for various radices of the arithmetic system. The moduli guarantee maximum dynamic ranges of digit sets for three, four, five and six bits per SDNR digit. It is assumed that all digit sets are symmetric.

Table 1: Choice of moduli for various radices

Bits/SDNR digit	Moduli $p1, p2$	Dynamic range $p1 \times p2$	a $p1 \times p2 \geqslant 2a+1$	Maximum radix for minimum redundant SDNR $2a-1$
3	8	8	3	5
4	2, 7	14	6	11
5	4, 7	28	13	25
6	7, 8	56	27	53

The principle of the arithmetic SDNR/RNS decomposition is based on RNS parallelism, which allows independent handling of the RNS digits because of carry-free arithmetic.

Assume it is required to add two SDNR digits of a radix-10 system. To represent the maximum redundant digit set of $\{-9, \ldots, 9\}$, the sign-and-magnitude (or two's complement) code would require five bits. Then, a single-digit adder would require 10 inputs — too many for an

efficient implementation of the circuit using FPGAs or similar technology. However, if the sign-and-magnitude code is replaced by the RNS, no module will have more than six inputs.

Table 2 specifies the RNS representation of the digit set $\{-9, \ldots, 9\}$ for moduli $p1 = 4$ and $p2 = 7$. The single line indicates intermediate sums which require a correction by -10 and generate carry $= 1$; the double line indicates a correction by $+10$ and carry $= -1$.

Table 2: RNS representation of the digit set $\{-9, \ldots, 9\}$ for moduli $p1 = 4$, $p2 = 7$

	SDNR		RNS	SDNR	RNS	
	0		⟨0, 0⟩	14–14	⟨2, 0⟩	
	1		⟨1, 1⟩	15–13	⟨3, 1⟩	
	2		⟨2, 2⟩	16–12	⟨0, 2⟩	
	3		⟨3, 3⟩	17–11	⟨1, 3⟩	carry
	4		⟨0, 4⟩	18–10	⟨2, 4⟩	= −1
	5		⟨1, 5⟩	−9	⟨3, 5⟩	
	6		⟨2, 6⟩	−8	⟨0, 6⟩	
	7		⟨3, 0⟩	−7	⟨1, 0⟩	
	8		⟨0, 1⟩	−6	⟨2, 1⟩	
	9		⟨1, 2⟩	−5	⟨3, 2⟩	
	10	−18	⟨2, 3⟩	−4	⟨0, 3⟩	
	11	−17	⟨3, 4⟩	−3	⟨1, 4⟩	
carry	12	−16	⟨0, 5⟩	−2	⟨2, 5⟩	
= 1	13	−15	⟨1, 6⟩	−1	⟨3, 6⟩	

Table 3 clarifies SDNR addition for $r = 10$, $a = 9$ and $t = 8$. The equivalent SDNR/RNS addition operates independently on each module to calculate a non-corrected intermediate sum. If the intermediate sum generates $C_{i-1} \langle \rangle 0$, a correction by ± 10 is required. Finally, SUM_i is calculated by adding carries.

This example of the SDNR/RNS addition reveals that a proper correction of intermediate sums and a carry generation require identification of four regions of the non-corrected intermediate sums

region I $C_{i-1} = 0$ $(-8 \leqslant S_i \leqslant 8)$
region II $C_{i-1} = 1$ $(S_i = 9)$
region III $C_{i-1} = \pm 1$ $(10 \leqslant S_i \leqslant 18$ or $-18 \leqslant S_i \leqslant -10)$
region IV $C_{i-1} = -1$ $(S_i = -9)$

Identification of this region demands a magnitude test of non-corrected sums. Additionally, in the case of region III, where positive and negative values have identical RNS representations, it is required to identify the signs of both input arguments. The structure given in Fig. 1 implements all necessary operations. It should be stressed that no module requires more than six inputs despite the circuit's suitability for a wide range of radices, between 3 and 53.

It will be even possible to simplify the unified circuit of Fig. 1 if sufficient redundancy of module selection eliminates the ambiguous region III. We must make sure that the condition

$$n \geqslant 4 \times a + 1$$

is true, where n is the product of RNS moduli (the dynamic range) and a is the greatest element of the SDNR digit set. Fig. 2 shows the simplified structure of the SDNR/RNS adder.

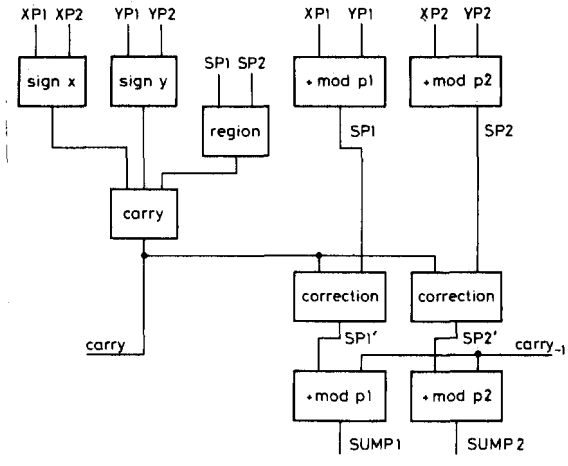


Fig. 1 *The unified structure of the SDNR/RNS adder (1. argument = ⟨XP1, XP2⟩; 2. argument = ⟨YP1, YP2⟩, non-corrected intermediate sum = ⟨SP1, SP2⟩, corrected intermediate sum = ⟨SP1', SP2'⟩, final sum = ⟨SUMP1, SUMP2⟩)*

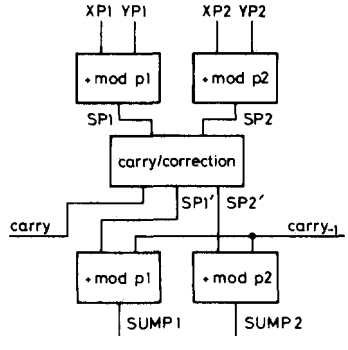


Fig. 2 *SDNR/RNS adder for disjoint sets*

4 SDNR/RNS arithmetic decomposition versus arithmetic decomposition based on the theory of digit sets

The importance of new design methods for the high-radix systems was appreciated by Carter and Robertson [2]

Table 3: SDNR addition for $r = 10$, $a = 9$ and $t = 8$

SDNR addition		SDNR/RNS addition	
$\bar{9}\ 3\ \bar{7}\ 3$	(−8767)	⟨3, 5⟩⟨3, 3⟩⟨1, 0⟩⟨3, 3⟩	
$+1\ 8\ \bar{6}\ 4$	(+1744)	$+\langle 1, 1\rangle\langle 0, 1\rangle\langle 2, 1\rangle\langle 0, 4\rangle$	
		⟨0, 6⟩⟨3, 4⟩⟨3, 1⟩⟨3, 0⟩	non-cor. S
$\bar{8}\ 1\ \bar{3}\ 7$	S	⟨0, 0⟩⟨2, 4⟩⟨2, 3⟩⟨0, 0⟩	correct. S
$0\ 1\ \bar{1}\ 0$	carries	⟨0, 6⟩⟨1, 1⟩⟨1, 4⟩⟨3, 0⟩	
		⟨0, 0⟩⟨1, 1⟩⟨3, 6⟩⟨0, 0⟩	carries
$0\ \bar{7}\ 0\ \bar{3}\ 7$	SUM(−7023)	⟨0, 0⟩⟨1, 0⟩⟨0, 0⟩⟨1, 4⟩⟨3, 0⟩	SUM

who developed a unified design technique based on the theory of digit sets. The theory applies the concept of the digit set and, not unlike our method, is suitable for redundant SDNR systems.

A digit set $\langle \delta^{\Omega} \rangle$ is defined by two parameters — δ, the diminished cardinality (that is, the number of digits less one) and Ω, the offset (that is, the magnitude of the smallest digit).

For example, the conventional digit set of a radix-10 system is represented by $\langle 9^0 \rangle$ or the radix-10 maximum redundant SDNR digit set is defined by $\langle 18^9 \rangle$.

The decomposition process replaces a digit set of a high diminished cardinality by weighted sums of digit sets of lower diminished cardinalities (ternary and binary sets).

For example,

$$\langle 18^9 \rangle \rightarrow 8\langle 1^1 \rangle + 4\langle 1^0 \rangle + 2\langle 2^0 \rangle + \langle 2^1 \rangle$$

An important characteristic of any decomposition method is storage requirements. The above decomposed digit set requires $1 + 1 + 2 + 2 = 6$ bits (two bits/ternary set, one bit/binary set).

The decomposition method described here reduces storage requirements for the same digit set to 5 bits (moduli $p1 = 4$ and $p2 = 7$ are to be selected).

Assuming 32-bit words for a signal/image processing system, then Fig. 3 compares the dynamic ranges for

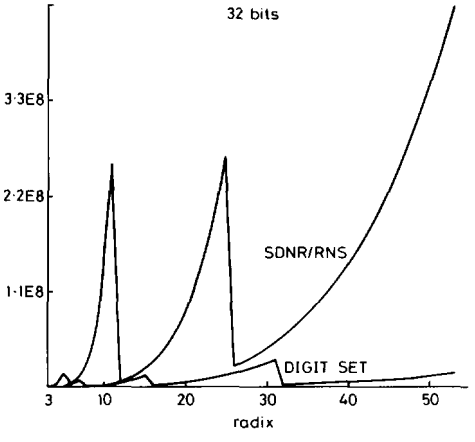


Fig. 3 *Dynamic ranges of SDNR/RNS and digit set systems for 32-bit words*

radices between 3 (smallest radix for the SDNR) and 53 (the largest radix for six bits per SDNR digit). For each radix an optimum digit set is selected (between minimum and maximum redundancy) which provides a maximum dynamic range. The figure clearly shows that the SDNR/RNS decomposition method allows a more efficient use of data storage. The additional advantage of our approach is a higher degree of structural uniformity for a wide range of radices. The structure of Fig. 1 can be applied to any radix between 3 and 53. On the other hand, the set theoretical approach, although very interesting from a mathematical point of view, requires different structures for different radices.

5 Theory of decomposition of Boolean functions

The purpose of the second stage of logic synthesis, Boolean decomposition, is the replacement of a complex logic network by a number of simpler ones. In the case of the FPGAs, the primary objective is the elimination of networks with more than five inputs using a minimum number of configurable logic blocks (CLBs). Each CLB can implement any function of five variables or two functions of four variables. For functions of more than five variables, there are two optimisation criteria: minimal number of required CLBs and minimal number of logic levels. The complexity of decomposed functions is irrelevant, since functions are implemented via look-up tables.

The above optimisation criteria require new synthesis algorithms which return efficient designs. The very general method of decomposition based on Shannon's expansion theorem is not very useful because it requires a large number of CLBs. For example, a function of seven variables would require as many as seven CLBs. A less general but more efficient approach is simple disjoint decomposition [3]. It expresses the Boolean function $F(X)$ as $G(H(Y), Z)$ where Y, the set of bound variables, and Z, the set of free variables, are disjoint sets and the union of Y and Z equals X. The problem here is that only a tiny percentage of all possible functions meets the above requirement; for instance, 0.00046% in the case of five arguments [10]. However, many functions applied in practice, especially functions which are not fully defined or sequential functions with optimum state assignments, can be decomposed in this way.

The less restrictive simple non-disjoint decomposition allows common elements in the bound and free variables: $F(X) = G(H(C, Y), C, Z)$, where C is a set of common variables (Fig. 4).

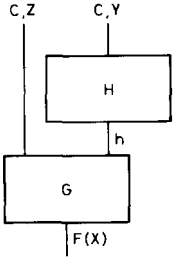


Fig. 4 *Simple non-disjoint decomposition*

Other possibilities include multiple disjoint, iterative disjoint, and complex disjoint decompositions.

Conditions for all types of decompositions have been well documented, althoutgh there is a shortage of efficient implementation algorithms. The spectral technique (Walsh transform) [10] is an attempt to avoid time-consuming searches, although its complexity has prevented its practical implementation. Another approach has been proposed in Reference 8. It uses symbolic partition description (SPD) for multiple-output functions. Although the decomposition procedure is very interesting from a mathematical point of view, the final stage of the procedure requires time-consuming merge-and-check operations on partitions.

6 Boolean decomposition based on partition products

The task of the decomposition algorithm is to find a simple disjoint or non-disjoint decomposition of a multiple-output function. The objective is to replace a logic network with too many arguments by an intercon-

nected pair of networks with a reduced number of inputs. The algorithm is explained with a specific example. Then, the general pseudo-code is given, followed by a comparison between the algorithm and a commercially available CAD system.

Consider the three output function S_i, representing the first stage of a radix-4 SDNR adder:

$$\begin{aligned} &\text{if} && X_i + Y_i > 2 && \text{then } C_{i-1} = 1 \\ &\text{else if} && X_i + Y_i < -2 && \text{then } C_{i-1} = -1 \\ &\text{else} && && C_{i-1} = 0 \end{aligned}$$

$$S_i = X_i + Y_i - 4 \times C_{i-1}$$

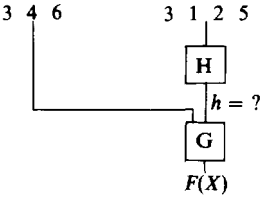
Table 4 specifies all possible function values. Letters A, ..., E represent function vectors 000, ..., 110. It is assumed that the adder uses the maximum redundant set of digits $\{-3, -2, -1, 0, 1, 2, 3\}$ and each digit is represented by the three-bit radix-magnitude code.

For any valid permutation of common, free, and bound arguments, the algorithm calculates the number of required outputs of the network H in Fig. 4. If the total number of inputs of the network G does not exceed a present maximum, then the algorithm accepts the permutation and, in the case of FPGAs, calculates the number of required CLBs.

If the selected permutation contains common argument(s), the algorithm creates a function table for each combination of common argument(s), otherwise a single function table is generated. For example, the set combinations

$$C = \{3\} \quad Z = \{4, 6\} \quad Y = \{1, 2, 5\}$$

which correspond to the network



leads to the following tables:

C : 0 (i.e. $x_3 = 0$)

		Y:						
	0	1	2	3	4	5	6	7
Z: 0	A	C	C	A	.	.	E	A
1	B	D	D	B	.	.	D	B
2	.	E	.	A	.	.	.	A
3	D	B	B	D	.	.	.	D

0, 3, 7, ; 1, 2, ; 6, ; (partition products)
0, 3, 7, ; 1, 2, ; 6, ;
0, 3, 7, ; 1, 2, ; 6, ;

C : 1 (i.e. $x_3 = 1$)

		Y:						
	0	1	2	3	4	5	6	7
Z: 0	B	D	D	B	D	B	B	D
1	C	A	A	C	A	C	E	A
2	.	D	.	B	.	B	.	D
3	A	E	C	A	E	A	A	E

0, 3, 5, ; 6, ; 1, 2, 4, 7, ; (partition products)
0, 3, 5, ; 1, 2, 4, 7, ; 6, ;
0, 3, 5, ; 2, ; 1, 4, 7, ; 6, ;

Each column of the table(s) represents a unique combination of bound variables (one, two and five for our specific example). Rows of the function table represent combinations of free arguments (4, 6). The next step creates partitions for each line using function values as equivalence criteria. For example, the first line of the lower table is represented by the partition

0, 3, 5, 6; 1, 2, 4, 7

In the case of rows which are not fully defined, don't-cares will be added to every partition block. For example, the third row of the lower table is represented by the pseudo-partition

1, 7, 0, 2, 4, 6; 3, 5, 0, 2, 4, 6

Finally, the algorithm calculates the product of all partitions and pseudo-partitions. After each product calculation, all redundant blocks caused by don't-cares are

Table 4: Function values for radix-4 SDNR adder

X_i			Y_i			S_i			
x_1 1	x_2 2	x_3 3	y_1 4	y_2 5	y_3 6	s_1	s_2	s_3	
0	0	0	0	0	0	0	0	0	A
0	0	0	0	0	1	0	0	1	B
0	0	0	0	1	0	0	1	0	C
0	0	0	0	1	1	1	0	1	D
0	0	0	1	0	1	1	0	1	D
0	0	0	1	1	0	1	1	0	E
0	0	0	1	1	1	0	0	1	B
0	0	1	0	0	0	0	0	1	B
0	0	1	0	0	1	0	1	0	C
0	0	1	0	1	0	1	0	1	D
0	0	1	0	1	1	0	0	0	A
0	0	1	1	0	1	0	0	0	A
0	0	1	1	1	0	1	0	1	D
0	0	1	1	1	1	1	1	0	E
0	1	0	0	0	0	0	1	0	C
0	1	0	0	0	1	1	0	1	D
0	1	0	0	1	0	0	0	0	A
0	1	0	0	1	1	0	0	1	B
0	1	0	1	0	1	0	0	1	B
0	1	0	1	1	0	0	0	0	A
0	1	0	1	1	1	1	0	1	D
0	1	1	0	0	0	1	0	1	D
0	1	1	0	0	1	0	0	0	A
0	1	1	0	1	0	0	0	1	B
0	1	1	0	1	1	0	1	0	C
0	1	1	1	0	1	0	1	0	C
0	1	1	1	1	0	0	0	1	B
0	1	1	1	1	1	0	0	0	A
1	0	1	0	0	0	1	0	1	D
1	0	1	0	0	1	0	0	0	A
1	0	1	0	1	0	0	0	1	B
1	0	1	0	1	1	0	1	0	C
1	0	1	1	0	1	1	1	0	E
1	0	1	1	1	0	0	0	1	B
1	0	1	1	1	1	0	0	0	A
1	1	0	0	0	0	1	1	0	E
1	1	0	0	0	1	1	0	1	D
1	1	0	0	1	0	0	0	0	A
1	1	0	0	1	1	0	0	1	B
1	1	0	1	0	1	0	0	1	B
1	1	0	1	1	0	0	0	0	A
1	1	0	1	1	1	1	0	1	D
1	1	1	0	0	0	0	0	1	B
1	1	1	0	0	1	1	1	0	E
1	1	1	0	1	0	1	0	1	D
1	1	1	0	1	1	0	0	0	A
1	1	1	1	0	1	0	0	0	A
1	1	1	1	1	0	1	0	1	D
1	1	1	1	1	1	1	1	0	E

deleted. The number of blocks in the final product determines the required number of network H output lines.

In the above example, the partition products are formed sequentially, row by row. However, for a larger problem, it is possible to calculate many partition products simultaneously, as the partition product is an associative operation.

This algorithm is at least one order of magnitude faster than the XACT-CAD system and significantly reduces the required number of CLBs (Table 5).

Table 5

Device	Arguments	Functions	CLBs	
			XACT	Decompo
Radix-4 SDNR adder (*s1*, *s2*, *s3*)	6	3	12	4
Radix-4 SDNR adder (*s1*, *s2*, *s3*, *c1*, *c2*)	6	5	17	8
Radix-4 SDNR max (*d1*, *d2*, *d3*, *m1*, *m2*, *m3*)	9	6	35	18

The following decomposition algorithm is applicable to both partial and total functions.

```
MAIN PROGRAM

INPUT:
n - number of arguments, m - number of function outputs,
function definition, MAX - maximum number of inputs.
OUTPUT:
Ordered triples (C, Z, Y) allowing simple decompositions,
number of Configurable Logic Blocks.
ALGORITHM:
Input.
Repeat
  Generate (C, Z, Y).
  If | C U Z |+1 ≤ MAX and | C U Y | ≤ MAX then
    DECOMPOSITION TEST.
    If | C U Z |+h ≤ MAX then
      Calculate number of Configurable Logic Blocks
      OUTPUT.
Until all non trivial (C, Z, Y) tested.

DECOMPOSITION TEST FOR PARTIAL OR TOTAL FUNCTIONS

INPUT:
Function definitions.
OUTPUT:
h - required number of the H network outputs.
ALGORITHM:
i=0.
For each combination of C do
  Create a table of function values with 2^|Z| rows
  and 2^|Y| columns.
  Identify empty rows.
  Identify empty columns.
  For every row do
    If row not empty then
      Create a partition excluding empty positions.
      Convert the partition into a pseudo-partition
      adding empty positions, excluding positions
      of empty columns, to each partition block.
  Calculate intersection of all pseudo-partitions.
  Delete all blocks which are subblocks of other blocks
  Delete blocks containing only redundant positions.
  If number of intersection blocks > i then
    i=number of intersection blocks.
h=⌈log2(i)⌉.
```

7 Design examples

A number of experiments indicated that the XACT-CAD systems could not efficiently handle the implementation of SDNR modules with radices higher than two. In contrast, a decomposition of the modules based on the unified structure of the SDNR/RNS adder (see Fig. 1) and the new decomposition algorithm leads to highly efficient hardware implementations. To illustrate the potential of the new approach, the complex case of a radix-53 SDNR/RNS is presented. As was mentioned in Section 3, high-radix arithmetic has a number of functional and technological advantages. Most importantly, the freedom of radix choice offered by the new decomposition method may dramatically widen the data dynamic range (see Fig. 3).

According to Table 1 the $\langle 52^{27} \rangle$ digit set may apply moduli $p1 = 7$ and $p2 = 8$. All modules of the SDNR/RNS adder which have more than five inputs must be decomposed by the algorithm from the previous section. For instance, the modulo-8 adder (the '+mod p2' block of Fig. 1) is a network of six Boolean arguments $XP21$, $XP22$, $XP23$, $YP21$,

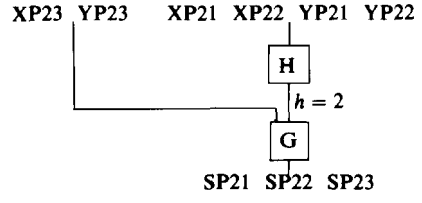
$YP22$, $YP23$ which encode values 0, ..., 7 of $XP2$ and $YP2$. The algorithm found that the following segregation of arguments reduces hardware requirements to just three CLBs:

	$XP21$, $XP22$, $YP21$, $YP22$:															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$XP23$, $YP23$: 0	A	C	E	G	C	E	G	A	E	G	A	C	G	A	C	E
1	B	D	F	H	D	F	H	B	F	H	B	D	H	B	D	F
2	B	D	F	H	D	F	H	B	F	H	B	D	H	B	D	F
3	C	E	G	A	E	G	A	C	G	A	C	E	A	C	E	G

Symbols A, ..., H represent values 0, ..., 7 of the non-corrected intermediate sum $SP21$, $SP22$, $SP23$. The product of all partitions has only four blocks:

{0, 7, 10, 13; 1, 4, 11, 14; 2, 5, 8, 15; 3, 6, 9, 12}

and, as a result, only two internal outputs ($h = 2$) are required:



Repeating the decomposition algorithm for all modules with more than five inputs leds to a very efficient mapping of the radix-53 SDNR/RNS adder with just 25 CLBs (Fig. 5).

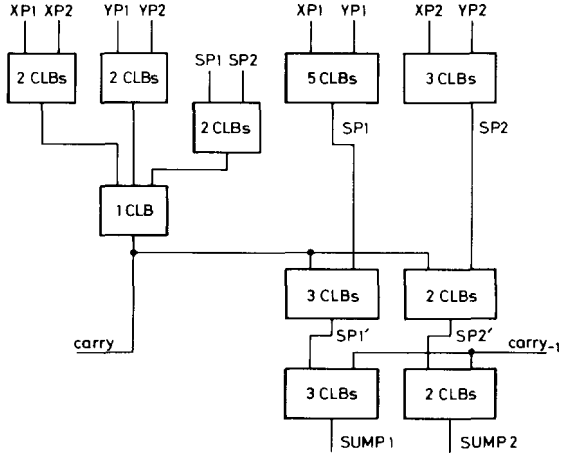


Fig. 5 *Radix-53 SDNR/RNS adder*

The significance of the SDNR/RNS data representation concept is not restricted to linear systems. The magnitude comparison, required by morphological transformations, median filters, and other non-linear methods, can be based on a subtractor and a sign detector. The sign of an SNDR/RNS difference is determined by the sign of the most significant non-zero digit [1]. Again, this algorithm leads to an efficient mapping of the network. For example, only two CLBs are required for a sign detector of a comparator with the dynamic range of

$\pm 27\ 27\ 27\ 27_{53} = \pm 233{,}280.$

8 Conclusions

The two-stage decomposition process leads to a unified structure, suitable for a wide range of linear and non-linear systems. It is applicable to high-radix systems which could not be designed using conventional methods.

A comparison of the first decomposition stage, based on SDNR/RNS data representation with a set theoretical approach, indicates a significant reduction of memory requirements. Another benefit is a higher degree of structural uniformity for a wide range of radices.

An additional synthesis stage, which uses a new algorithm for simple Boolean decompositions, allows a further reduction in module complexity and an efficient hardware implementation based on conventional technologies, such as PLAs, gate arrays, and FPGAs. Both stages lead to a reduction in hardware complexities by factors of 2–3 when compared to designs generated by a complex, commercially available CAD system.

Although the research results presented here have been inspired by our work on morphological processors, their significance carries over to other linear and non-linear signal and image processing systems, such as FFT, rank-order filters, and encryption devices.

9 References

1 AVIZIENIS, A.: 'Signed-digit representations for fast parallel arithmetic', *IRE Transactions on Electronic Computers*, 1961, pp. 389–400

2 CARTER, T.M., and ROBERTSON, J.E.: 'The set theory of arithmetic decomposition', *IEEE Trans. Comput.*, 1990, **39**, (8), pp. 993–1005

3 CURTIS, H.A.: 'A new approach to the design of switching circuits' (D. Van Nostrand, Princeton, 1962)

4 GIARDINA, C.R., and DOUGHERTY, E.R.: 'Morphological methods in image and signal processing' (Prentice-Hall, Englewood Cliffs, 1988)

5 JULIEN, G.A., BIRD, P.D., CARR, J.T., TAHERI, M., and MILLER, W.C.: 'An efficient bit-level systolic cell design for finite ring digital signal processing applications', *J. VLSI Signal Process.*, 1989, **1**, (3), pp. 189–207

6 KUCZBORSKI, W., ATTIKIOUZEL, Y., and CREBBIN, G.: 'Video rate morphological processor based on a redundant number representation', *in* B.G. BATCHELOR, M.J.W. CHEN, and F.W. WALTZ (Eds.): 'Machine vision, architectures, integration, and applications' (SPIE, Bellingham, 1992), pp. 249–260

7 KUNG, S.Y.: 'VLSI array processors' (Prentice Hall, Englewood Cliffs, 1988)

8 LUBA, T., JASINSKI, K., and KRASNIEWSKI, A.: 'Combining serial decomposition with topological partitioning for effective multi-level PLA implementations', *in* P. MICHEL, and G. SAUCER (Eds.): 'Logic and architecture synthesis' (North-Holland, Amsterdam, 1991), pp. 243–252

9 McNALLY, O.C., McCANNY, J.V., and WOODS, R.F.: 'A 40 Magasample IIR filter chip', *in* M. VALERO, S.Y. KUNG, T. LANG, and J.A.B. FORTES (Eds.): 'Special purpose architectures' (IEEE Computer Society Press, 1991), pp. 416–430

10 POSWIG J.: 'Disjoint decomposition of Boolean functions', *IEE Proc. E*, 1991, **138**, (1), pp. 48–56

11 YEONG-CHYANG SHIH, F., and MITCHELL, O.R.: 'Threshold

decomposition of gray-scale morphology into binary morphology', *IEEE Trans. Pattern Anal. Mach. Intell.*, 1989, **11**, (1), pp. 31–42
12 SODERSTRAND, M.A., JENKINS, W.K., JULLIEN, G.A., and TAYLOR, F.J.: 'Residue number systems arithmetic: modern applications in digital signal processing' (IEEE Press, New York, 1986)
13 TAKAGI, N., ASADA, T., and YAJIMA, S.: 'Redundant CORDIC methods with a constant scale factor for sine and cosine computation', *IEEE Trans. Comput.*, 1991, **40**, (9), pp. 989–994
14 TAYLOR, F.J.: 'Residue arithmetic: a tutorial with examples', *Computer*, 1984, pp. 50–62
15 TORNG, H.C.: 'Switching circuits — theory and logic design' (Addison-Wesley, Reading, 1972)