[15] E. Larsson and Z. Peng, "An integrated system-on-chip test framework," in *Proc. Design, Automation, and Test in Europe (DATE)*, Mar. 2001, pp. 138–144.
[16] J. K. Lenstra, D. B. Shmoys, and E. Tardos, "Approximation algorithms for scheduling unrelated parallel machines," *Proc. 28th Annual IEEE Symp. Foundations of Computer Science (FOCS)*, pp. 217–224, 1987.
[17] E. J. Marinissen, S. K. Goel, and M. Lousberg, "Wrapper design for embedded core test," *Proc. IEEE Int. Test Conf. (ITC)*, pp. 911–920, Oct. 2000.
[18] E. J. Marinissen, V. Iyengar, and K. Chakrabarty. ITC '02 SOC Benchmark Website. [Online]. Available: http://www.extra.research.philips.com/itc02socbenchm/
[19] ——, "A set of benchmarks for modular testing of SOC's," in *Proc. Int. Test Conf. (ITC)*, Oct. 2002.
[20] V. Muresan *et al.*, "A comparison of classical scheduling approaches in power-constrained block-test scheduling," *Proc. IEEE Int. Test Conf. (ITC)*, pp. 882–891, Oct. 2000.
[21] P. Rosinger, B. Al-Hashimi, and N. Nicolici, "Power constrained test scheduling using power profile manipulation," in *Proc. Int. Symp. Circuits Syst. (ISCS)*, May 2001, pp. 251–254.
[22] M. Sugihara, H. Date, and H. Yasuura, "A novel test methodology for core-based system LSI's and a testing time minimization problem," *Proc. IEEE Int. Test Conf. (ITC)*, pp. 465–472, Oct. 1998.
[23] Y. Zorian, "Test requirements for embedded core-based systems and IEEE P1500," *Proc. IEEE Int. Test Conf. (ITC)*, pp. 191–199, Nov. 1997.
[24] Y. Zorian, E. J. Marinissen, and S. Dey, "Testing embedded-core based system chips," *Proc. IEEE Int. Test Conf. (ITC)*, pp. 130–143, Oct. 1998.

Digital Filter Synthesis Based on an Algorithm to Generate All Minimal Signed Digit Representations

In-Cheol Park and Hyeong-Ju Kang

***Abstract*—In this paper, the authors propose an algorithm to find all the minimal signed digit (MSD) representations of a constant and present an algorithm to synthesize digital filters based on the MSD representation. The hardware complexity of a digital signal processing system is dependent on the number system used for the implementation. Although the canonical signed digit (CSD) representation is widely employed, as it is unique and guarantees the minimal number of nonzero digits for a constant, the MSD representation provides multiple representations that have the same number of nonzero digits as the CSD representation. The proposed filter synthesis algorithm utilizes this redundancy of the MSD representation to make common subexpressions, as many as possible, leading to smaller filters. By applying the proposed algorithm to the hardware synthesis of finite impulse response filters, the authors obtained multiplier blocks that are 7% smaller than those generated from the CSD representation.**

***Index Terms*—Canonical signed digit, filter design, minimal signed digit, multiple constant multiplication, number system.**

I. Introduction

Digital filters are frequently used in digital signal processing by virtue of stability and easy implementation. Although programmable filters based on digital signal processing cores can take an advantage of flexibility, they are not suitable for recent consumer applications demanding high throughput and low-power consumption. In such an application, therefore, application specific digital filters are frequently adopted to meet the constraints of performance and power consumption. However, these filters are suffering from a large number of multiplications, leading to excessive area and power consumption even if implemented in full custom integrated circuits. Therefore, the problem of designing digital filters with small area and low-power consumption has received a great attention during the last decade. Early works have focused on replacing multiplications by decomposing them into simple operations such as addition, subtraction, and shift. As the coefficients of an application specific filter are constant, the decomposition is more efficient than employing general multipliers. The number of additions/subtractions used to implement the coefficient multiplications in this case dominates the complexity of filters.

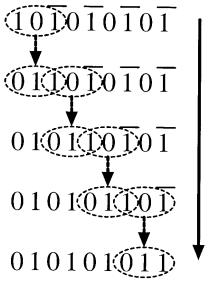

Fig. 1. Decomposition of a long conversion into several short conversions.

Many approaches have been proposed for the implementation of multiple constant multiplications (MCMs) [1]–[10]. In the approaches, a common subexpression is searched among multiple constants and implemented into one hardware in order to share the result of the subexpression in evaluating all the constants. Previous approaches have tried to select common subexpressions, as many as possible, after representing the constants in the canonical signed digit (CSD) representation. Although the CSD representation is good for one constant, it is not the best for multiple constants because the CSD representation of a constant is unique and independent of the other constants, leading to limited subexpressions for multiple constants. For the multiple constant multiplications, it is more efficient to use the minimal signed digit (MSD) representation that has the same number of nonzero digits as the CSD representation but provides multiple representations for a constant [9], [10].

As the CSD representation is unique, it has received much attention and there have been many methods of converting a given binary number into the CSD representation [11]–[15]. The uniqueness is important in terms of mathematics, but not in implementing hardware units. In general, the MSD representation providing multiple representations that yield the same value is more flexible than the CSD representation. This redundancy can result in smaller hardware units than those generated from the CSD representation, if an appropriate MSD representation is selected for each constant. Even better results could be obtained by using all signed digit representations including nonminimal signed digit representations. The search space, however, becomes so large that the exploration time becomes unreasonably long. Each digit in the signed digit representation can have one of three values, -1, 0 and 1. In n-digit representation, 3^n combination represents $2^{n+1}-1$ numbers. Therefore, the average number of representations per number is $(3/2)^n/2$ approximately, growing exponentially as n increases.

Manuscript received May 2, 2001; revised November 26, 2001 and March 4, 2002. This work was supported in part by the Korea Science and Engineering Foundation through the MICROS center, the Ministry of Science and Technology, and the Ministry of Commerce, Industry and Energy through the project System IC 2010 and IC Design Education Center. This paper was recommended by Associate Editor R. Gupta.

The authors are with the Department of Electrical Engineering, Korea Advanced Institute of Science and Technology, Yuseong-gu, Daejeon 305-701, Korea (e-mail: icpark@ee.kaist.ac.kr).

Digital Object Identifier 10.1109/TCAD.2002.804374

The MSD representation gives a search space small enough to explore in reasonable time and large enough to give good results. In spite of this advantage, the MSD number system has not been studied much because of the lack of formalism and there has been no report on how to find all the MSD representations for a given number except enumerating all cases. In this paper, we propose an algorithm that can find all the MSD representations for a given constant number. In addition to the MSD generation algorithm, we present a new digital filter synthesis algorithm to exploit the MSD representations of coefficients.

The rest of this paper is organized as follows. In Section II, the CSD and MSD representations are introduced and compared in more detail. In Section III, we suggest a theorem and an algorithm to generate all MSD representations of an arbitrary number. We explain the algorithm to synthesize digital filters with the MSD representation in Section IV and experimental results obtained from the filter synthesis are described in Section V. Finally, concluding remarks are made in Section VI.

II. CSD and MSD Representations

The CSD representation is a radix-2 signed digit system with the digit set $\{1, 0, \bar{1}\}$, where $\bar{1}$ denotes -1. Given a constant, the corresponding CSD representation is unique and has two properties; the first is that the number of nonzero digits is minimal and the second is that the product of adjacent two digits is zero, that is, two nonzero digits are not adjacent. Due to the first property, the CSD representation is widely used in implementing MCMs because it guarantees the least number of additions for a given constant. The second property is called "property M" in [11]. If a signed digit representation of a constant satisfies property M, it is the CSD representation. If the second property is relaxed, it is called the minimal signed digit (MSD) representation.

Although the CSD representation is optimal for one constant, it is difficult to consider the other constants in case of multiple constants because a number is uniquely represented in the CSD representation. Since the MSD representation is a superset of the CSD representation and provides a number of forms, the MSD representation is more appropriate in finding common subexpressions for multiple constants if a proper MSD representation is selected for each constant to be synthesized [9], [10]. Since the representation method affects the number of additions (or subtractions) in the decomposed multiplication block and the number of common subexpressions that can be eliminated, it has significant influence on the resulting area and power consumption.

III. MSD Generation Algorithm

Basically, the MSD representations of a number are derived from the corresponding CSD representation. In this section, we present a theorem related to the CSD-to-MSD conversion and an algorithm to generate MSD representations.

Theorem 1: Assume that a number N has an MSD representation $m_{n-1}m_{n-2}\cdots m_0$ being different from the CSD representation $c_{n-1}c_{n-2}\cdots c_0$. If there is a portion ranging from k to l, $0 \le l < k \le n-1$, which satisfies the following:

$$\begin{aligned} c_{k+1} &= m_{k+1} \\ c_k &\neq m_k \\ c_{k-1} &\neq m_{k-1} \\ &\vdots \\ c_l &\neq m_l \\ c_{l-1} &= m_{l-1} \end{aligned} \tag{1}$$

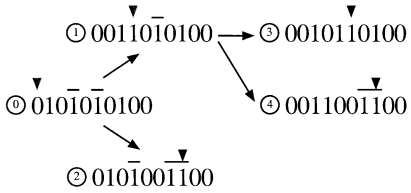

Fig. 2. The MSD generation procedure for 180.

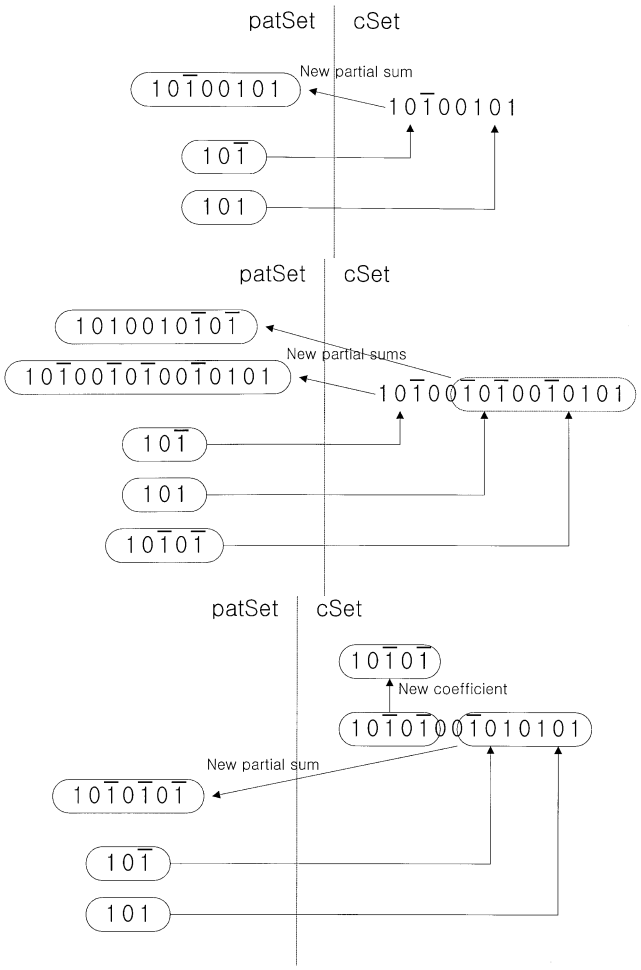

Fig. 3. Examples of (a) Step 5, (b) Step 6, and (c) Step 7 in the proposed filter synthesis algorithm.

then $k - l$ is even and

$$\begin{cases} c_k c_{k-1}\cdots c_l = 10\bar{1}0\bar{1}0\bar{1}0\cdots 0\bar{1} = 1(0\bar{1})^+ \\ m_k m_{k-1}\cdots m_l = 0\,101\,010\cdots 011 = (01)^+1 \end{cases} \tag{2}$$

or

$$\begin{cases} c_k c_{k-1}\cdots c_l = \bar{1}0\,101\,010\cdots 01 = \bar{1}(01)^+ \\ m_k m_{k-1}\cdots m_l = 0\bar{1}0\bar{1}0\bar{1}0\cdots 0\bar{1}\bar{1} = (0\bar{1})^+\bar{1} \end{cases} \tag{3}$$

where $+$ represents at least one repetition.

The proof of Theorem 1 is complex and thus omitted in this paper for simplicity. If there are readers who are interested in the exact proof, please refer to [16]. From Theorem 1, it can be induced that the only transformations needed to convert the CSD representation to MSD representations are $10\bar{1} \rightarrow 011$ and $\bar{1}01 \rightarrow 0\bar{1}\bar{1}$. Fig. 1 shows how a number of MSD representations are achieved by repeatedly applying the short transformations. The overall algorithm developed from that

fact is as follows. First, a number is represented in the CSD representation by using one of the algorithms presented in [11]–[14]. As the CSD representation is also an MSD representation, it is registered as the first MSD representation. Next, a pattern of either $10\bar{1}$ or $\bar{1}01$ is searched starting from the most significant digit and transformed to 011 or $0\bar{1}\bar{1}$, respectively. For each transformation, a new MSD representation is generated. The transformation is repeatedly applied to the new MSD representations found in the previous transformations until there is no such pattern. To avoid duplications, the patterns are searched in an MSD representation from the next position of the digit where a transformation is applied to generate the MSD representation.

The detailed explanation of the algorithm and related terms is described below.

Definitions:

N	n bit number to find all MSD representations.;
MSD_i	ith MSD representation found;
S	set including the MSD representations found;
$\|S\|$	number of MSD representations in S;
SearchMSD	MSD representation where a new one is being searched;
$SP[i]$	digit position where the transformation is applied to generate ith MSD representation;
SearchPoint	digit position where the search is being done.

Algorithm 1

Step 1. Convert N into the CSD representation. It is named MSD_0. $S = \{\mathrm{MSD}_0\}$. $|S| = 1$. $\mathrm{SearchMSD} = 0$. $SP[0] = n - 1$.

Step 2. $\mathrm{SearchPoint} = SP[\mathrm{SearchMSD}]$.

Step 3. If $\mathrm{SearchPoint} < 1$, go to Step 6.

Step 4. If the digits from position SearchPoint to SearchPoint $-$ 2 in $\mathrm{MSD}_{\mathrm{SearchMSD}}$ are $10\bar{1}$ or $\bar{1}01$, make a new MSD representation by changing $10\bar{1}$ to 011 or $\bar{1}01$ to $0\bar{1}\bar{1}$. The new MSD representation is named $\mathrm{MSD}_{|S|}$. $SP[|S|] = \mathrm{SearchPoint} - 2$. Insert the new MSD representation into S. Decrement SearchPoint by 2. Go to Step 3.

Step 5. Decrement SearchPoint. Go to Step 3.

Step 6. Increment SearchMSD. If SearchMSD is the same as $|S|$, end. Otherwise, go to Step 2.

As an example, Fig. 2 shows the procedure of finding all the MSD representations of 180. The circled number means the order of MSD representations generated by the proposed algorithm. The first MSD representation numbered as 0 is equivalent to the CSD representation. The inverse triangle of each MSD representation denotes the first SearchPoint of the representation.

IV. Filter Synthesis Algorithm

In this section, we explain the proposed multiplier block synthesis algorithm. In the previous methods, common subexpressions are searched and combined after all coefficients are expressed in the CSD representation, whereas in the proposed algorithm coefficients are considered and synthesized one by one, e.g., a coefficient is selected and synthesized sequentially one at a time.

The first step is to generate all MSD representations for each coefficient. Then, a coefficient that can be implemented using a minimal number of adders is selected for synthesis, that is, a coefficient with the minimal number of nonzero digits is selected for the first synthesis. The intermediate sums that can be obtainable from the adders used for the synthesized coefficients are registered as partial sums. Among not-yet synthesized coefficients, we select a coefficient that can be implemented with minimal additional adders. Therefore, the next coefficient to be synthesized is the one that can be implemented with the previously defined partial sums. The following is the flow of our algorithm where *cSet* is the set of coefficients that are to be synthesized and *patSet* is the set of patterns of partial sums that can be used to synthesize the selected coefficient.

TABLE I
Test Filter Specification

Filter	Passband	Stopband	#tap	Width
1	0.15	0.25	40	12
2	0.15	0.25	60	14
3	0.15	0.20	60	14
4	0.15	0.20	100	16
5	0.10	0.15	60	14
6	0.10	0.15	100	16
7	0.10	0.12	100	16
8	0.10	0.12	120	18

Algorithm 2

Step 1. Insert 1 into *patSet*.

Step 2. Even coefficients are made odd by dividing them by a power of 2. Negative coefficients are converted to positive ones. The coefficients that have the same value with other coefficients are removed. The remaining coefficients are inserted into *cSet*.

Step 3. Obtain all the MSD representations of the elements in *cSet*.

Step 4. If there is an element in *cSet* that has the same MSD representation as the shifted value of an element in *patSet*, it is removed from *cSet*. Repeat this step until there is no such element in *cSet*. If no element is in *cSet*, end.

Step 5. If there is an element in *cSet* that has the same MSD representation as a shifted combination of two elements in *patSet*, it is removed from *cSet* and inserted into *patSet*. **Fig. 3(a)** is an example that a combination of two partial sums matches a coefficient. Repeat this step until there is no such element in *cSet*. If no element is in *cSet*, end.

Step 6. If there is an element in *cSet* that has the same MSD representation as a shifted combination of three elements in *patSet*, it is removed from *cSet* and inserted into *patSet*. One combination of two partial sums is inserted into *patSet*. An example is shown in **Fig. 3(b)**. If no element is remained in *cSet*, end. If no element is removed from *cSet* in Steps 4–6, go to Step 7. Otherwise, go to Step 4.

Step 7. For each shifted combination of two partial sums in *patSet* and each MSD

TABLE II
EXPERIMENTAL RESULTS FOR FIR FILTERS

Filter	Simple		Potkonjak[1]		Paš ko [3]		Hartley[4]		Proposed algorithm (MSD-based)	
	#adders	#adder-steps	#adders	#adder-steps	#adders	#adder-steps	#adders	#adder-steps	#adders	#adder-steps
1	33	3	22	4	19	4	19	3	18	3
2	54	3	30	4	24	5	25	3	22	4
3	77	3	48	5	37	4	35	3	35	3
4	131	3	82	4	53	5	55	4	50	4
5	88	3	54	4	36	5	37	4	35	4
6	140	3	87	5	56	5	55	4	50	4
7	173	3	95	5	71	5	71	4	66	4
8	246	3	136	6	98	5	96	4	91	4
Avg.	240%	-	141%	-	99%	-	100%	-	93%	-

representation of elements in *cSet*, check if the pattern of the combination is included in the MSD representation and count the number of nonzero digits in the MSD representation that are uncovered by the combination. Select a pair of a combination and an MSD representation that has the minimal number of uncovered nonzero digits. The pattern of the combination is registered as a new partial sum. A new element obtained by removing the selected combination is inserted into *cSet*. **Fig. 3(c)** presents an example, where $10\bar{1}0\bar{1}0\bar{1}$ becomes a new partial sum and $10\bar{1}0\bar{1}$ becomes a new coefficient. Go to Step 4.

When combining elements, shifting of elements is allowed, but there must be no conflict between them, e.g., there must be no digit place where two or more elements have nonzero digits simultaneously. Steps 1 and 2 are the preparing steps. In Step 1, 1 is inserted into *patSet* because it needs no adder. In Step 2, the coefficients are modified. The division or multiplication by a power of 2 can be implemented by wiring and the negation can be implemented by changing the adders to the subtractors. We can use positive odd numbers in place of negative or even numbers and the original negative or even coefficients can be produced from the positive odd numbers with a little hardware overhead. Step 3 generates the CSD representation and all of the MSD representations for each coefficient. The MSD representations are created with the algorithm in Section III. In Step 4, the elements in *cSet* that are already in *patSet* are removed because they are already made. In Step 5, the elements that can be synthesized with only one adder are selected and synthesized. In Step 6, the elements that need two adders are synthesized. In Step 7, we modify a coefficient by including the most matched combination of partial sums into *patSet* when there is no element that can be synthesized with one or two adders.

The above algorithm, however, cannot consider all the coefficients simultaneously. The partial sums generated in the later coefficients cannot be shared with the former coefficients. The iterative scheme can solve this problem. Algorithm 2 is repeated once more with the partial sums generated in the first process. Some of the partial sums should be eliminated in order not to generate the same results. More iterations might provide better results, but our experiments reveal that more-than-three iterations do not give better results. The following algorithm provides the best results in most cases.

Algorithm 3
Step 1. Process Algorithm 2.
Step 2. Eliminate the partials sums that are not used in generating other partial sums. Go to Step 3 of Algorithm 2.
Step 3. Eliminate the partial sums that are not shared. Go to Step 3 of Algorithm 2.

V. EXPERIMENTAL RESULTS

The proposed algorithms are applied to several finite impulse response filters (FIRs) and compared with previous algorithms. The specification of those filters is summarized in Table I, where f_p and f_s are normalized passband frequency and stopband frequency, respectively, *#tap* is the number of taps, and *Width* denotes the word size of fixed-point representation. The passband and stopband frequency of the first filter in Table I are quoted from the example in [5]. Given the parameters, the floating-point coefficients of test filters are generated using the Remez algorithm in MATLAB and then converted to integer numbers by rounding. We assume transposed-form filters that can accept the subexpression sharing.

In Table II, the results obtained by the previous and proposed algorithms are shown. The results are compared in two terms. The first one is the number of adders required in each multiplier block. A subtractor is also counted as an adder since the area of a subtractor is almost the same as an adder. The other one is the number of adder steps. It means the number of adders in a maximal path of a multiplier block. A large number of adder steps increases the length of the critical path and decreases the speed. In addition, it generates more glitch and consumes more power. The column denoted as 'Simple' represents the results obtained by constructing a separate adder tree for each coefficient. The simple method requires a lot of adders but provides the fastest results requiring the minimal number of adder steps. The next three columns show the results of previous CSD-based algorithms. Among them, Paško's and Hartley's method provide better results. Comparing to the simple method, the methods significantly reduce the number of adders needed to synthesize the filters at the cost of delay increase for some filters. In terms of the number of adders, Paško's results are slightly better than those of Hartley, whereas the number of adder steps resulting from Paško's method is usually larger than that of Hartley's method. The last column describes the results obtained by the proposed algorithm. The MSD-based algorithm provides fast results that need the minimal number of adders without increasing the number of adder steps. It can reduce the number of adders by 7% even compared to Hartley's method.

TABLE III
EXECUTION TIMES

Paš ko [3]	Hartley[4]	Proposed algorithm (MSD-based)
3214	12	299

TABLE IV
PLACEMENT AND ROUTING RESULTS

Filter	Hartley[4]	Proposed algorithm (MSD-based)
	Delay(ns)	Delay(ns)
1	13.53	15.23
2	14.94	15.06
3	15.41	14.96
4	18.85	18.44
5	16.70	15.91
6	19.85	17.36
7	19.22	18.39
8	21.86	20.72
Avg.	100%	97.68%

Table III compares the execution time (in seconds) taken to process 16 filters at a Sun Ultra60 workstation. All algorithms are written in C language and compiled in GNU C compiler with -O3 option. The proposed algorithm with the MSD representations takes more time than Hartley's algorithm because it has larger search space. For a filter, however, the average execution time of the proposed algorithm is about 20 s.

The filters generated by algorithms are described in Verilog hardware description language and synthesized by Synopsys Design Compiler with 0.35 μm technology standard cell library. We have done placement and routing with Avanti's Astro. Power consumptions of filters generated with Hartley's algorithm and the proposed algorithm are almost the same (less than 0.5% on the average). Table IV shows the delays after placement and routing. The filters of the proposed algorithm are slightly faster than those of Hartley's.

VI. CONCLUSION

In this paper, we have presented an algorithm to find all the MSD representations of a constant number and a digital filter synthesis algorithm based on the MSD representation. Starting from the CSD representation, all the MSD representations are discovered by repeatedly applying simple transforms, $10\bar{1} \rightarrow 011$ and $\bar{1}01 \rightarrow 0\bar{1}\bar{1}$. The proposed filter synthesis algorithm is to select one coefficient at a time and synthesize it using previously synthesized patterns, which is different from the conventional method that searches subexpressions common to multiple constants. To overcome the disadvantage that the algorithm can view only one coefficient at a time, the proposed algorithm is repeated by keeping the partial sums generated from the previous iteration. To show the effectiveness of the proposed algorithm, several filters are implemented and the results are compared with those generated from previous algorithms. The experimental results show that the proposed algorithm yields better results than the conventional ones obtained from the CSD representation.

REFERENCES

[1] M. Potkonjak, M. B. Srivastava, and A. Chandrakasan, "Efficient substitution of multiple constant multiplication by shifts and additions using iterative pairwise matching," in *Proc. 31st ACM/IEEE Design Automation Conf.*, 1994, pp. 189–194.

[2] M. Mehendale, S. D. Sherlekar, and G. Venkatesh, "Synthesis of multiplier-less FIR filters with minimum number of additions," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1995, pp. 668–671.

[3] R. Paš ko, P. Schaumont, V. Derudder, S. Vernalde, and D. Ďuraèková, "A new algorithm for elimination of common subexpressions," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 58–68, Jan. 1999.

[4] R. I. Hartley, "Subexpression sharing in filters using canonic signed digit multipliers," *IEEE Trans. Circuits Syst. II*, vol. 43, pp. 677–688, Oct. 1996.

[5] H. Samueli, "An improved search algorithm for the design of multiplier-less FIR filters with powers-of-two coefficients," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 1044–1047, July 1989.

[6] D. Li, J. Song, and Y. C. Lim, "A polynomial-time algorithm for designing digital filters with power-of-two coefficients," in *Proc. 1993 IEEE Int. Symp. Circuits Syst.*, 1993, pp. 84–87.

[7] D. R. Bull and D. H. Horrocks, "Primitive operator digital filters," *IEE Proc. G*, vol. 138, no. 3, pp. 401–412, 1991.

[8] A. G. Dempster and M. D. Macleod, "Use of minimum adder multiplier blocks in FIR digital filters," *IEEE Trans. Circuits Syst. II*, vol. 42, pp. 569–577, Sept. 1995.

[9] T. S. Chang, C. S. Kung, and C. W. Jen, "A simple processor core design for DCT/IDCT," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 10, pp. 439–447, Mar. 2000.

[10] J. T. Kim, "Design and implementation of computationally efficient FIR filters and scalable VLSI architectures for discrete wavelet transform," Ph.D. dissertation, Advanced Institute of Science and Technology, Korea, 1998.

[11] G. W. Reitweisner, "Binary arithmetic," in *Advances in Computers*, F. L. Alt, Ed. New York: Academic, 1960, vol. 1, ch. 5, pp. 232–308.

[12] K. Hwang, *Computer Arithmetic Principles, Architecture and Design*: Wiley, 1979.

[13] I. Koren, *Computer Arithmetic Algorithms*. Englewood Cliffs, NJ: Prentice-Hall, 1993.

[14] Y. C. Lim, J. B. Evans, and B. Liu, "Decomposition of binary integers into signed power-of-two terms," *IEEE Trans. Circuits Syst.*, vol. 38, pp. 667–672, June 1991.

[15] R. Hashemian, "A new method for conversion of a 2's complement to canonic signed digit number system and its representation," in *Proc. Asilomar Conf. Signals, Syst., Computers*, 1997, pp. 904–907.

[16] H. J. Kang. Relationship between CSD and MSD. [Online]. Available: http://ics.kaist.ac.kr/~dk/CSDandMSD.pdf

[17] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*. Rockville, MD: Computer Sci., 1978.