EFFECT OF NUMBER REPRESENTATION ON THE ACHIEVABLE MINIMUM NUMBER OF OPERATIONS IN MULTIPLE CONSTANT MULTIPLICATIONS

Levent Aksoy, Ece Olcay Gunes
Istanbul Technical University
Istanbul, Turkey
{aksoyl, ece.gunes}@itu.edu.tr

Eduardo Costa
Universidade Catolica de Pelotas
Pelotas-RS, Brazil
ecosta@ucpel.tche.br

Paulo Flores, José Monteiro
INESC-ID/IST, TU Lisbon
Lisbon, Portugal
{pff, jcm}@inesc-id.pt

ABSTRACT

In this work, we analyze the effect of representing constants under binary, CSD, and MSD representations on the minimum number of operations required in a multiple constant multiplications problem. To this end, we resort to a recently proposed algorithm that computes the exact minimum solution. To extend the applicability of this algorithm to much larger instances, we propose problem reduction and model simplification techniques that significantly reduce the search space. We have conducted experiments on a rich set of instances including randomly generated and FIR filter instances. The results show that, contrary to common belief, the binary representation clearly yields better solutions than CSD, and even provides slightly better solutions than MSD. Moreover, the superiority of the binary solutions increases as the number and bit-width of the constants increase.

Index Terms— Multiple Constant Multiplication (MCM), Common Subexpression Elimination (CSE), Canonical Signed Digit (CSD), Minimal Signed Digit (MSD).

1. INTRODUCTION

Linear systems such as finite impulse response (FIR) filters and discrete signal processing transforms are widely used in a number of applications, e.g., audio and video processing and wireless communication. The computations in these systems include multiple constant multiplications (MCM) that lead to excessive area, delay, and power consumption in hardware even if implemented in a full custom integrated circuit. The proposed methods have focused on the minimization of area by replacing the multiplication operations with constants by addition, subtraction, and shifting operations. Since shifts are free in terms of hardware, the MCM problem can be defined as the minimization of the number of addition/subtraction operations to implement the constant multiplications.

The proposed algorithms for the optimization of the number of operations in MCM can be categorized in two classes: common subexpression elimination (CSE) and graph-based techniques. In CSE algorithms, constants are represented in a number representation, namely, binary, canonical signed digit (CSD), and minimal signed digit (MSD). Both CSD and MSD representations use a signed digit system with the digit set $\{-1, 0, 1\}$, where $\bar{1}$ denotes -1, and have the property that the number of non-zero digits is minimum. The CSD representation provides a unique representation for every constant, since two non-zero digits are not adjacent. This representation is generally used in multiplierless implementations due to the number of non-zero digits being reduced by 33% on average when compared with the binary representation [1]. A constant can have several MSD representations, because non-zero digits can be consecutive in MSD. For example, suppose the constant 23 in six bits. The representation of 23 in binary, 010111, includes 4 non-zero digits. The constant is represented as $10\bar{1}00\bar{1}$ in CSD and both $10\bar{1}00\bar{1}$ and $01100\bar{1}$ represent 23 in MSD with 3 non-zero digits. CSE algorithms generally find the most common non-zero digit combinations while optimizing the number of operations. In [2], a two-term subexpression elimination technique is presented under CSD representation. The algorithm of [3] applies two-term common subexpression elimination iteratively while generating two-term divisors. Also, the use of different selection criteria for the common subexpressions in CSE algorithms are described in [4] and [5]. In [6], it is shown that using MSD representation yields better solutions than CSD in MCM problems, since it has the same minimum number of non-zero digits as CSD, but provides multiple alternative representations for a constant. However, all these algorithms are heuristics, i.e., provide no indication how far from the minimum their solutions are. An exact CSE algorithm that considers the maximum sharing of partial terms is introduced in [7]. In this algorithm, all the possible implementations of constants are found, represented as a Boolean network and then converted to a 0-1 integer linear programming (ILP) problem. The cost function to be minimized is the linear function of optimization variables that represent partial terms. Finally, an exact solution is found by a generic satisfiability (SAT)-based 0-1 ILP solver. In graph-based methods, constants are implemented without a restriction to any particular number representation. In [8, 9], two prominent heuristics are introduced for multiple constants. The reader is referred to [9] for more information on graph-based methods.

In CSE algorithms, CSD representation is widely used than binary and MSD, since binary representation of a constant includes more non-zero digits and MSD representation provides alternative representations of a constant increasing the complexity of an algorithm. The complexity analysis of the 0-1 ILP problem constructed by the exact algorithm is done in [7] considering the worst case and it is shown that problem size grows exponentially with the number of non-zero digits. Moreover, binary representation offers less flexibility on the implementation of a constant, i.e., only additions. Although it offers more non-zero digits than CSD and uses only positive sign, we argue that in an exact approach considering multiple constants, the use of binary representation increases the possibility of partial term sharing yielding better solutions than CSD.

The contributions of this paper are twofold: first, we introduce model simplification and problem reduction techniques to the previously proposed exact CSE algorithm [7] to deal with larger size 0-1 ILP problems and second, we compare the use of different number representations in MCM under a rich set of instances based on exact solutions rather than evaluations based on solutions of heuristic algorithms. In this paper, we show that binary representation is superior than commonly preferred CSD representation and is better than MSD as the number of constants is increased.

The rest of the paper is organized as follows. In Section 2, the proposed exact algorithm is introduced and the improvements to the previously proposed exact algorithm are described in Section 3. Experimental results are given in Section 4. Finally, the paper concludes in Section 5.

2. THE EXACT ALGORITHM

In this section, initially, we give the problem definition and then present the exact CSE algorithm that can handle multiple constants in binary, CSD, and MSD representations as it deals with the multiplierless implementation of a digital FIR filter.

2.1. Problem Definition

In the optimization of the number of operations in digital filter synthesis, filter coefficients and partial terms are considered as odd numbers, since shifts can be implemented with only wires in hardware. So, an *operation* represents an addition or a subtraction with two odd inputs, I_1 and I_2, input shifts, S_1 and S_2, and an output, O, given as $O = I_1 \ll S_1 \pm I_2 \ll S_2$, where $S_1 = 0, S_2 > 0$ or $S_1 > 0, S_2 = 0$, i.e., without loss of generality one of the shifts at the input is zero and the other is greater than zero. A *partial term* is an odd constant that is neither a coefficient nor a filter input and is determined as an input of an operation that implements a coefficient. Thus, *the problem of the optimization of the number of operations* can be defined as finding the minimum number of partial terms to be added to a set that contains filter coefficients and filter input such that each coefficient and partial term in the set can be implemented with the set elements using only one operation. So, the minimum number of operations is the sum of the number of odd coefficients and the minimum number of required partial terms.

2.2. Partial Term Generation

In this work, the optimization of the number of operations is defined as a binate covering problem, a special case of a 0-1 ILP problem where every constraint is interpreted as a propositional clause. In the preprocessing phase of the algorithm, after the filter coefficients are made positive and odd, they are stored without repetition in a set called *Cset*. They are labeled as filter coefficients and unimplemented. If the filter input, i.e., 1, does not exist in *Cset*, it is also inserted into *Cset* and labeled as implemented. The part of the algorithm where the partial terms are found for each element in *Cset* is as follows:

1. Take an unimplemented element from *Cset*, $Cset_i$. Form an empty set of arrays called $Pset_i$ associated with $Cset_i$. $Pset_i$ will contain all partial terms that are required to implement $Cset_i$.
2. Find an operation that implements $Cset_i$;
   (a) Find the non-repeated inputs of the operation that are neither a filter coefficient nor a filter input and store them in an empty array called *Iarray*. Note that *Iarray* may contain a single partial term or a pair of partial terms.
   (b) If *Iarray* is empty, then make $Pset_i$ empty and go to Step 5. In this case, $Cset_i$ can be implemented with an operation whose inputs are filter coefficients or filter inputs and this is the minimum implementation.
   (c) If *Iarray* is not empty, then check for each array of $Pset_i$, $Pset_i(k)$, if $Pset_i(k) \subseteq Iarray$. If *Iarray* is included in $Pset_i$, then go to Step 3.
   (d) If *Iarray* is not empty, then check for each array of $Pset_i$, $Pset_i(k)$, if $Iarray \subset Pset_i(k)$. If *Iarray* dominates $Pset_i(k)$, then delete $Pset_i(k)$.
   (e) Add *Iarray* to $Pset_i$.
3. Repeat Step 2 until all possible implementations of $Cset_i$ are considered.
4. Add all partial terms in $Pset_i$ to *Cset*, if they are not in *Cset* and label them as unimplemented.
5. Label $Cset_i$ as implemented and repeat Step 1 until all elements in *Cset* are labeled as implemented.

Observe that in the first iteration of the algorithm, *Cset* contains the filter coefficients and in later iterations, it contains also the partial terms.

2.3. Conversion to 0-1 ILP Problem

After all the partial terms required to implement each coefficient and partial term are found, the optimization problem is converted into a combinational network. The network includes only AND and OR gates. An OR gate, representing a coefficient or a partial term, combines all the partial terms that can be used for the synthesis of the associated coefficient or partial term. An AND gate, representing a pair of partial terms, combines two partial terms. The primary inputs of the network are filter coefficients and partial terms that can be implemented with a single operation whose inputs are filter coefficients or filter inputs.

The network is converted into a 0-1 ILP problem, after additional hardware (a 2-input AND gate for each partial term) with the optimization variables is added to the network. The optimization variables that represent the filter coefficients are assigned to 1 and the conjunctive normal form (CNF) formulas of each gate in the network are found. Each clause in CNF formulas is expressed as a linear inequality. A cost function, i.e, the linear function of the optimization variables that represent the partial terms, is constructed. Finally, the problem is given to the SAT-based 0-1 ILP solver, MiniSat+ [10], to obtain an exact solution. In the construction of the network and the translation of the network into CNF, the issues described in [11] that speed-up the 0-1 ILP solver were also considered.

In the post-processing phase of the algorithm, after the minimum solution including filter coefficients and required partial terms is obtained, the filter coefficients and partial terms are synthesized from inputs to outputs. In the selection of an operation for each coefficient and partial term among possible implementations whose inputs are in the found solution or filter inputs, the minimization of the maximum number of operations in series, i.e., the delay generally called adder-step, is considered. Thus, the minimum delay synthesis of the found minimum area solution is realized.

3. MODIFICATIONS TO THE PREVIOUSLY PROPOSED EXACT ALGORITHM

In CSE algorithms, the operations that implement a constant are determined as the decompositions of the non-zero digits of the constant (Step 2 of the algorithm given in Section 2.2). As an example, consider 51 ($10\bar{1}010\bar{1}$ in CSD) as a filter coefficient. The operations that implement 51 are given in Fig. 1. Observe that the implementation of 64-13 is the same as the implementation of 52-1, since these operations have the same odd inputs. Therefore, 52-1 is not listed in Fig. 1. Similarly, the duplications of implementations are not presented in this figure, e.g., 63-12 is equal to -12+63.

In the exact algorithm of [7], after all possible operations for each filter coefficient and partial term are found as illustrated in Fig. 1, the Boolean network is constructed using AND and OR gates. In this network, an AND gate represents

With 1 non-zero digit combinations	With 2 non-zero digits combinations
$51=1000000+00\bar{1}010\bar{1}=1\ll 6 - 13\ll 0$	$51=10\bar{1}0000+000010\bar{1}=3\ll 4 + 3\ll 0$
$51=00\bar{1}0000+100010\bar{1}=-1\ll 4 + 67\ll 0$	$51=1000100+00\bar{1}000\bar{1}=17\ll 2 - 17\ll 0$
$51=0000100+10\bar{1}000\bar{1}=1\ll 2 + 47\ll 0$	$51=100000\bar{1}+00\bar{1}0100=63\ll 0 - 3\ll 2$

Fig. 1. Implementations of 51 in CSD.

an operation and an OR gate associated with a filter coefficient or a partial term combines all operations that implement the related constant. Since an operation implements only one constant, it is different from operations that implement other constants even if they include the same odd inputs. While the primary inputs of the network are the filter inputs or their shifted versions, the primary outputs of the network are the outputs of OR gates that represent the filter coefficients. The network constructed by the algorithm of [7] for the coefficient 51 in CSD is given in Fig. 2 with the elimination of 1-input OR gates for the partial terms 3, 17, and 63.

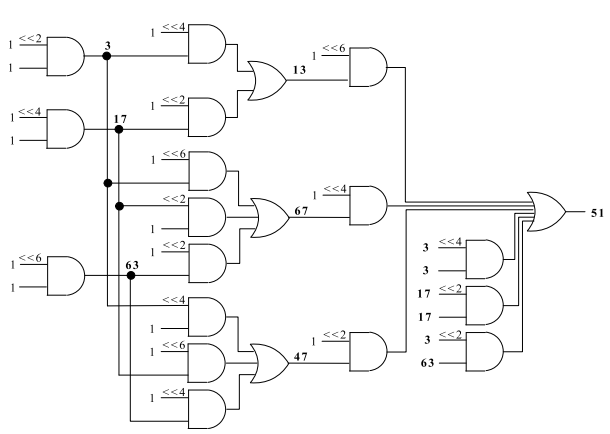

Fig. 2. The network constructed by the algorithm of [7] for the coefficient 51 in CSD.

As model simplification techniques, we consider the minimization of the number of partial terms required to synthesize the filter coefficients. So, in our algorithm, the partial terms are obtained from the operations that implement a constant. Also, since the variables that represent filter input and coefficients are assigned to 1 in the 0-1 ILP problem, the implications of these assignments are considered while determining the partial terms for a constant. So, if a constant can be implemented with an operation whose inputs are filter inputs or filter coefficients, this constant is determined as the primary input of the network. Otherwise, a pair of partial terms that is required to implement a filter coefficient or a partial term is represented with an AND gate and a single partial term is represented as the output of an OR gate associated with the partial term. As can be seen from Fig. 1, since 1 represents the filter input, the implementation of 51 requires 3,13,17,47,67 as single partial terms and 3 and 63 as a pair of partial terms. The network constructed by our algorithm for the coefficient

51 in CSD is given in Fig. 3. Observe that both networks given in Fig. 2 and 3 represent the same binate covering problem defined for the minimization of the number of operations.

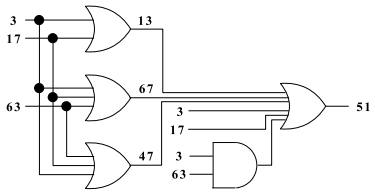

Fig. 3. The network constructed by our algorithm for the coefficient 51 in CSD.

As problem reduction techniques, we apply the dominance rule for each constant to reduce the number of inputs of an OR gate. For example, consider the network given in Fig. 3. Since the single partial term 3 dominates the pair of partial terms 3 and 63, the AND gate that represents this pair is redundant and can be removed from the network (Step 2d of the algorithm given in Section 2.2). Also, when a pair of partial term is required to implement more than one constant, it is shared in the network reducing the number of constraints and variables in the 0-1 ILP problem. As can be easily observed with the comparison of the networks given in Fig. 2 and 3, the proposed techniques reduce the 0-1 ILP problem size, consequently the CPU time required by the 0-1 ILP solver to find an exact solution as presented in Table 3.

4. EXPERIMENTAL RESULTS

In this section, results of the exact algorithm on randomly generated and real-sized FIR filter instances under binary, CSD, and MSD representations are presented and compared.

As the first experiment set, randomly generated instances that include between 10 and 120 constants were used. There are 30 instances for each number of constants and constants are defined in 10, 12, and 14 bit-widths. In Fig. 4, the use of binary representation on minimum number of operations is compared with CSD representation.

As can be observed from Fig. 4, using binary representation yields better results than CSD on average. Also, as the number and range of the constants increase, the difference of the number of operations on average between CSD and binary representations tends to increase. While the difference of the average number of operations under 10 bit-widths on problem instances with 120 constants between CSD and binary is 1.1, this value on problem instances with 120 constants under 14 bit-widths is 3.9. In Fig. 5, we compare the exact solutions obtained using binary representation with MSD.

As can be seen in Fig. 5, the exact solutions obtained under binary and MSD are quite similar. However, as the number and range of constants are increased, using binary representation achieves better solutions than MSD. For example,

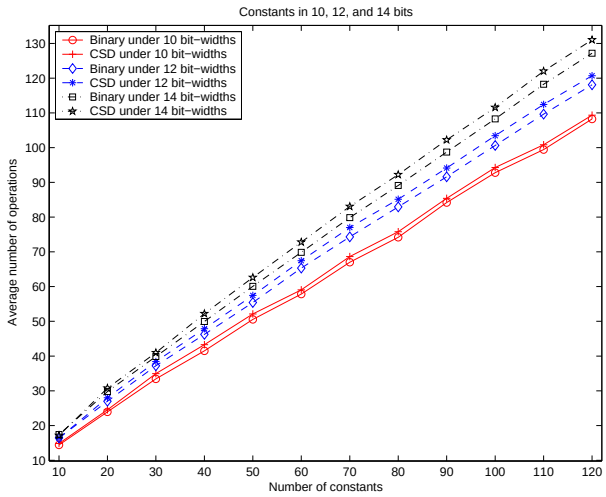

Fig. 4. Comparison of the use of binary and CSD representations on randomly generated instances.

while the difference of the average number of operations under 10 bit-widths on problem instances with 120 constants between MSD and binary is 0.6, this value on problem instances with 120 constants under 14 bit-widths reaches to 1.1.

In this experiment, we observe that as opposed to common usage, CSD representation does not tend to give the minimum number of operation solutions in MCM. Because, using a single representation of a constant with the minimum number of non-zero digits and both positive and negative signs may produce partial terms that are less common in the implementations of constants. This drawback can be overcome using MSD that considers alternative representations of a constant with the minimum number of non-zero digits. However, we observe that binary representation achieves more promising solutions than CSD, since using a unique representation of a constant with more non-zero digits and only positive sign increases the partial term sharing. Also, we note that the use of binary representation becomes more effective on finding the minimum number of operation solutions, as the number and range of constants increase. But, the main disadvantage of using binary representation is that the design can be obtained in a greater delay than the design obtained using CSD or MSD. The average adder-step of exact solutions obtained on randomly generated instances in 14 bit-widths under binary, CSD, and MSD is presented in Fig. 6. We note that while the average number of adder-step of solutions obtained under binary representation on problem instances with 120 constants is 6.1, this value is 4.6 for both CSD and MSD.

As the second experiment set, FIR filters where the coefficients were computed with the Remez algorithm in MATLAB were used. The filter specifications are given in Table 1 where *pass* and *stop* are normalized frequencies that define the passband and stopband respectively; *#tap* is the number of coefficients; and *width* is the bit-width of the coefficients.

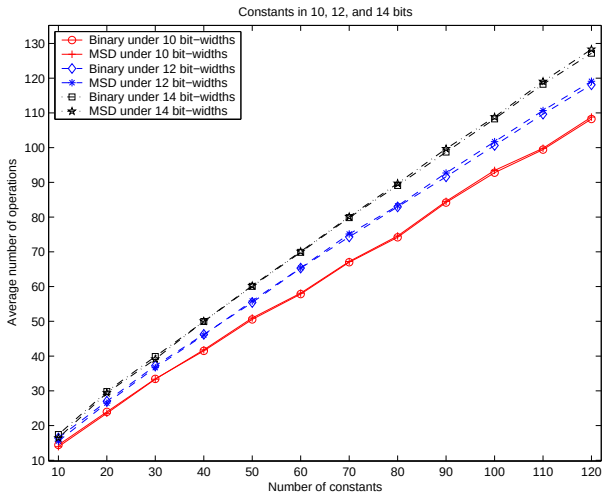

Fig. 5. Comparison of the use of binary and MSD representations on randomly generated instances.

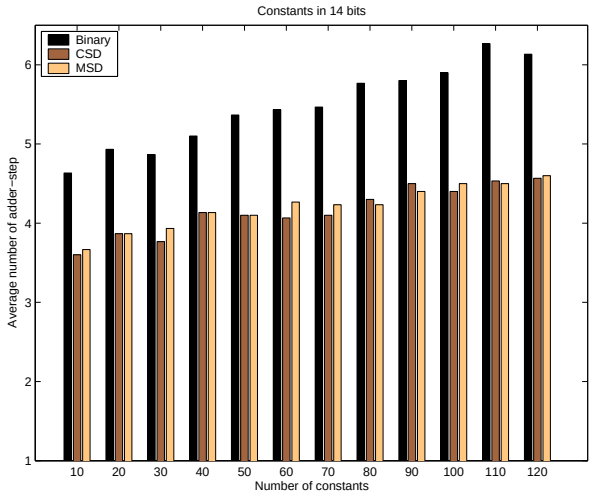

Fig. 6. Comparison of adder-step of solutions obtained under binary, CSD, and MSD representations.

Table 1. Filter specifications.

Filter	*pass*	*stop*	*#tap*	*width*
1	0.10	0.15	200	16
2	0.10	0.15	240	16
3	0.10	0.25	180	16
4	0.10	0.25	200	16
5	0.10	0.20	240	16
6	0.10	0.20	300	16
7	0.15	0.25	200	16
8	0.15	0.25	240	16
9	0.20	0.25	240	16
10	0.20	0.25	300	16

The 0-1 ILP problem sizes of filter instances and solutions obtained by the exact algorithm under binary, CSD, and MSD representations are given in Table 2. In this table, *vars*, *cons*, and *optv* stand for the number of variables, constraints, and optimization variables respectively. Also, *adder* denotes the number of operations and *step* denotes the maximum of number of operations in series needed to synthesize the filter coefficients. CPU is the CPU time in seconds that is used by MiniSat+ to compute the exact solutions on a PC with Intel Xeon at 3.16GHz with 8GB of main memory. Since the CPU time required to construct the network in the preprocessing phase and to find a solution with minimum delay in the post-processing phase are also negligible, CPU only indicates the CPU time of MiniSat+.

In this experiment, we observe that the size of 0-1 ILP problems under binary representation is generally larger than the size of problems defined under CSD and MSD. This is because the binary representation of a constant includes more non-zero digits. As can be seen from Table 2, this property helps to obtain FIR filter designs with less number of operations than CSD and MSD on every instances. Using binary representation leads solutions less than 2 and 1 operation on average with respect to CSD and MSD respectively. But, the delay of the filter designs is increased compared to the solutions obtained under CSD and MSD. Also, we note that since the CSD representation of a constant includes the minimum number of non-zero digits yielding 0-1 ILP problems in a smaller size, exact solutions can be found in less amount of CPU time than binary and MSD representations.

In Table 3, we compare the use of proposed model simplification and problem reduction techniques with the previously proposed exact algorithm [7] in terms of 0-1 ILP problem sizes and CPU time required to find the minimum solution on FIR filters given in Table 1 under binary representation.

In this experiment, we observe that a 0-1 ILP problem that represents an MCM problem can be obtained in a smaller problem size, when the proposed model simplification and problem reduction techniques are applied. On these filter instances, while the number of variables and constraints is reduced by almost 90%, the number of optimization variables is reduced by 45%. Also, the reduction of problem size enables the 0-1 ILP solver to obtain exact solutions with a very low computational effort.

5. CONCLUSIONS

In this work, previously proposed exact algorithm designed for the minimization of the number of operations in digital filter synthesis is improved with model simplification and problem reduction techniques to cope with larger search space. The exact algorithm is tested on a rich set of instances where constants are defined under binary, CSD, and MSD representations. It is shown by the experimental results that binary representation achieves better solutions than widely used CSD representation and gives more promising solutions than MSD as the number of constants increases.

Table 2. 0-1 ILP problem sizes and results of FIR filter instances.

Filter	0-1 ILP Problem Sizes									Minimum Number of Operation Solutions								
	Binary			CSD			MSD			Binary			CSD			MSD		
	vars	cons	optv	vars	cons	optv	vars	cons	optv	adder	step	CPU	adder	step	CPU	adder	step	CPU
1	3862	13550	944	633	1427	316	2103	6877	602	81	7	7	83	5	0.1	82	5	0.7
2	9904	38038	1500	618	1460	289	1776	5024	623	86	6	5.5	88	5	0	87	5	0.2
3	16226	67753	1433	1833	6014	476	10054	38972	1354	52	5	14.2	56	4	1.3	53	5	20
4	15992	63884	1928	1210	3545	420	656	1460	333	92	7	7	94	5	0.1	93	5	0.1
5	6808	27119	873	827	2174	329	2606	8127	751	65	6	22	66	4	0.1	66	5	8.4
6	13581	55759	1012	1121	3059	417	2778	8862	763	71	6	3.1	74	5	0.2	72	4	0.3
7	2413	8674	567	371	808	188	434	1043	200	62	5	0.1	65	4	0.1	64	4	0
8	2781	10119	642	394	824	221	861	2272	370	71	5	0.1	73	4	0	72	4	0.1
9	140	162	119	231	344	166	348	562	227	79	7	0	80	4	0	80	4	0
10	171	289	122	126	147	109	152	211	119	82	7	0	84	4	0	84	4	0
Total	71878	285347	9140	7364	19802	2931	21768	73410	5342	741	61	59	763	44	1.9	753	45	29.8

Table 3. The effect of using the proposed techniques on FIR filters under binary representation.

Filter	[7]				This work			
	vars	cons	optv	CPU	vars	cons	optv	CPU
1	60416	194552	1595	405	3862	13550	944	7
2	79707	262024	1886	278	9904	38038	1500	5.5
3	59069	191764	1510	678.2	16226	67753	1433	14.2
4	129530	444146	2366	503.1	15992	63884	1928	7
5	63076	207012	1519	52.3	6808	27119	873	22
6	58286	188294	1533	44.9	13581	55759	1012	3.1
7	47004	154086	1142	3.6	2413	8674	567	0.1
8	32044	98816	1048	3	2781	10119	642	0.1
9	133493	461220	2300	2.8	140	162	119	0
10	116294	405186	1880	2.6	171	289	122	0
Avg.	100%	100%	100%	100%	9.2%	10.9%	54.5%	3.0%

6. REFERENCES

[1] H. Garner, "Number Systems and Arithmetic," *Advances in Computers*, vol. 6, pp. 131–194, 1965.

[2] R. Hartley, "Subexpression Sharing in Filters using Canonic Signed Digit Multipliers," *IEEE Transactions on Circuits and Systems II*, vol. 43, no. 10, pp. 677–688, 1996.

[3] A. Hosangadi, F. Fallah, and R. Kastner, "Reducing Hardware Complexity of Linear DSP Systems by Iteratively Eliminating Two-Term Common Subexpressions," in *Proceedings of the IEEE Asia and South Pacific Design Automation*, 2005, pp. 523–528.

[4] M. Potkonjak, M. Srivastava, and A. Chandrakasan, "Multiple Constant Multiplications: Efficient and Versatile Framework and Algorithms for Exploring Common Subexpression Elimination," *IEEE Transactions on Computer-Aided Desig of IC Systems*, vol. 15, no. 2, pp. 151–165, 1996.

[5] R. Pasko, P. Schaumont, V. Derudder, S. Vernalde, and D. Durackova, "A New Algorithm for Elimination of Common Subexpressions," *IEEE Transactions on Computer-Aided Design*, vol. 18, no. 1, pp. 58–68, 1999.

[6] I-C. Park and H-J. Kang, "Digital Filter Synthesis Based on Minimal Signed Digit Representation," in *Proceedings of Design Automation Conference*, 2001, pp. 468–473.

[7] P. Flores, J. Monteiro, and E. Costa, "An Exact Algorithm for the Maximal Sharing of Partial Terms in Multiple Constant Multiplications," in *Proceedings of International Conference on Computer-Aided Design*, 2005, pp. 13–16.

[8] A. Dempster and M. Macleod, "Use of Minimum-Adder Multiplier Blocks in FIR Digital Filters," *IEEE Transactions on Circuits and Systems II*, vol. 42, no. 9, pp. 569–577, 1995.

[9] Y. Voronenko and M. Puschel, "Multiplierless Multiple Constant Multiplication," *to appear in ACM Transactions on Algorithms*.

[10] N. Een and N. Sorensson, "Translating Pseudo-Boolean Constraints into SAT," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 2, pp. 1–26, 2006.

[11] M. Velev, "Efficient Translation of Boolean Formulas to CNF in Formal Verification of Microprocessors," in *Proceedings of the IEEE Asia and South Pacific Design Automation*, 2004, pp. 310–315.