Multiple Constant Multiplications: Efficient and Versatile Framework and Algorithms for Exploring Common Subexpression Elimination

Miodrag Potkonjak
Department of Computer Science, University of California, Los Angeles, CA 90024
Mani B. Srivastava
AT&T Bell Laboratories, Murray Hill, NJ 07974
Anantha Chandrakasan
Dept. of EE, MIT, Cambridge, MA 02139

ABSTRACT

Many applications in DSP, telecommunications, graphics, and control have computations that either involve a large number of multiplications of one variable with several constants, or can easily be transformed to that form. A proper optimization of this part of the computation, which we call the *multiple constant multiplication* (MCM) problem, often results in a significant improvement in several key design metrics, such as throughput, area, and power. However, until now little attention has been paid to the MCM problem.

After defining the MCM problem, we introduce an effective problem formulation for solving it where first the minimum number of shifts that are needed is computed, and then the number of additions is minimized using common subexpression elimination. The algorithm for common subexpression elimination is based on an iterative pairwise matching heuristic. The power of the MCM approach is augmented by preprocessing the computation structure with a new scaling transformation that reduces the number of shifts and additions. An efficient branch and bound algorithm for applying the scaling transformation has also been developed.

The flexibility of the MCM problem formulation enables the application of the iterative pairwise matching algorithm to several other important and common high level synthesis tasks, such as the minimization of the number of operations in constant matrix-vector multiplications, linear transforms, and single and multiple polynomial evaluations. All applications are illustrated by a number of benchmarks.

1.0 Motivation and Problem Relevance

Transformations are alternations of a computation algorithm such that its functionality is maintained. Computational transformations have recently attracted much attention in the high level synthesis community [63], [49], [64], [35] for their ability to dramatically improve area [47], throughput [45], latency [60], power [17], transient [33] and permanent [29] fault tolerance, and other design metrics. Transformations have also been studied in areas such as algorithm design for numerical and DSP applications [6] and compilers [24]. In high level synthesis, the primary goal of transformations has been to optimize an ASIC design to reduce cost metrics (area, power) while meeting throughput constraints [64]. The main technical novelty of this use of transformations in high level synthesis has been the development of powerful optimization algorithms to apply these transformations.

The exceptional ability of the human brain to recognize and manipulate regularity, symmetry, and other structural properties of computation has been used to great advantage in the design of algorithms such as FFT and DCT [6], [50]. Often, sophisticated mathematical knowledge is applied [23], [6]. Software compilers, on the other hand, employ fast and relatively simple automatic techniques to apply algorithm transformations on very large programs [24]. Compared to the approaches used in algorithm design and software compilers, the high level synthesis approach to transformations is better suited to transformations that require handling of numerous details, involve complex combinatorial optimization, do not require a large and sophisticated mathematical knowledge base, and place relatively low importance to the regularity of the resulting computation structure. This paper addresses multiple constant multiplication, which is exactly such a transformation.

Multiple constant multiplication is a new transformation closely related to the widely used substitution of multiplications with constants by shifts and additions. While the latter considers multiplication with only one constant at a time, the new transformation considers several different constant multiplication with the same variable. This change in the scope of the transformation significantly enhances its power and domain of application.

Optimization of multiplication with a single constant has for a long time been recognized as being important for compilers and high level synthesis systems. It has been used for improving throughput [39], area [18], [19] and power [17]. In ASICs as well as many microprocessors, it is significantly less expensive to do additions, subtractions and shifts, than it is to do a multiplication. While the frequency of multiplication and division by a constant is quite small in general-purpose computation benchmarks (for example, it is often quoted at the level of 3% as indicated by the Gibson Mix [25]), it is not the case for application-specific computation.

In ASICs the substitution of multiplications with constants by shifts and additions influences not just the speed, but also the area. The effect on ASIC area is much more pronounced because the area of a shifter is many times smaller than the area of a multiplier. We analyzed more than two hundred industrial examples, mainly from DSP, communication, graphics, and control applications. More than 60% of the examples have more than 20% operations that are multiplications with constants. Similar results about the frequency of constant multiplication in DSP applications were also presented in [30]. The importance of multiplication with constants is also indicated by recent results [17] showing an order of magnitude reduction in power achieved by this transformation alone.

A close look at currently used optimization techniques for substitution by shifts and additions show that the full potential of this transformation has not been explored. Until now the attention has exclusively been paid to (i) optimization of isolated multiplication with constants, and (ii) minimization of the number of shifts, with little attention paid to the number of additions.

We formulate the multiple constant multiplication problem in such a way that the number of shifts is minimized. Our formulation has several distinctive advantages. First, it enables one to treat the problem of minimizing the number of addition as one of common subexpression elimination in a restricted domain. This allows efficient and low complexity algorithms to be used. Second, our formulation enables an interesting theoretical analysis of the effectiveness of the multiple constant multiplication transformation, which leads to the asymptotic result that both the number of shifts and additions stay bounded and finite regardless of the size of the instance of the problem (see Section 4). Also, the formulation allows one to establish an interesting relationship between the MCM problem and combinational logic synthesis (see Section 6). Finally, our restricted formulation also allows the MCM approach and associated software to be applied with minimal modification to a number of high level synthesis tasks that are based on common subexpression elimination. For example, there is a close parallel between the single constant multiplication problem, and the constant power evaluation problem [34] of efficiently computing X^n, where n is a constant. We generalize this relationship to exploit the new optimization algorithms not just for the MCM problem, but also to the parallel problems of single or multiple polynomial evaluations. The new technique is also applied to optimizing the implementation of error correcting codes - an application area not previously targeted by high level synthesis tools (see Section 9).

2.0 Related work

Optimization of multiplications with a constant has been studied for a long time, including pioneering work by von Neumann and his coworkers [14]. Later several schemes for minimizing the number of operations after substitution of multiplication with constants by shifts and additions were proposed [55], [46]. This work led to novel number representation schemes, such as the canonical signed digit (CSD) representation [54], [38], [31] that is often used in DSP to reduce the number of shifts and additions after a multiplication with a constant is replaced by shifts and additions. The CSD representation is also sometimes used in high level synthesis [49].

Booth [7] used similar ideas for binary number encoding to design his famous and widely used variable-variable multiplier.

Algorithms for optimizing the number of shifts and additions have also been described in the software compiler and computer architecture research. For example, Bernstein presented several heuristics approaches to this problem [4]. Magenheimer et. al. presented an in-depth study on frequency, optimization and importance of constant multiplication substitution with shifts and addition, and used their finding to influence the Hewlett-Packard line of high performance workstations [40].

The minimization of number of shifts is also a well studied topic in DSP, in particular in the digital filter design community. The emphasis is on achieving structures with minimal number of shifts (usually two or three) per constant multiplication. It is interesting to note that even on these already optimized computational structures [16], [57] our new techniques yields significantly better results.

While minimizing the number of shifts required for multiplication with a constant is a thoroughly studied problem, not much attention has been paid to the simultaneous optimization of the multiplication of a variable by multiple constants. Only recently Chatterjee et. al. [18], [19] addressed this problem by presenting two optimization approaches - greedy and simulated annealing-based - for the minimization of the number of operations in constant multiplications in vector-matrix product representation of linear systems. Although these two algorithms, based on a number spliting technique, are theoretically interesting and use sophisticated state-of-the-art optimization algorithm, the algorithm proposed in this paper is superior in its simplicity, effectiveness, and range of applications. The application range of the number splitting approach is restricted to linear systems, while the MCM transformation can be applied to an arbitrary computation. Table 1 shows the improvement over the initial and the best previously published results [18] on 4 examples (*mat1* - 1 input 3 state controller; *ellip* - 4-state 1-input elliptical wave

	Mul		8 bit			16 bit			32 bit		
	I	N	I	SA	N	I	SA	N	I	SA	N
mat	12	6	39	38	6	83	74	14	152	130	25
ellip	20	8	77	56	8	136	87	16	240	198	32
lin4	30	10	92	57	8	212	128	16	383	279	26
lin5	28	10	81	66	8	191	117	16	348	313	26

Table 1: Comparison of the MCM transformation with previously published results[Cha93] (I - Initial Design; SA - design optimized using simulated annealing; N - new design; The second and third columns show the number of multiplication; the next 9 columns show the number of shifts.

filter; and *lin4* and *lin5* two 5-state 1 input controllers) for 8, 16 and 32 bit designs obtained using the MCM transformation. The average improvement compared to the initial design is 9.92 times, and compared to designs optimized using simulated annealing is 7.41 times.

A problem closely related to constant multiplication is that of computing a constant power of a variable by using only multiplications. A lucid treatment of techniques for this problem, covering work dating as far back as two millennia, was done by Knuth [34]. Note that while the right-shift operation has the same complexity as the left-shift, division is usually significantly more expensive than multiplication. The similarity between the two problems of constant power evaluation, and constant multiplication, are exploited in Section 9 to reduce the number of operations in polynomial computations.

An alternate way of looking at the algorithms presented in this paper is as techniques for minimizing the number of operations using common subexpression elimination. A number of excellent summaries of efficient algorithms for common subexpression elimination, both within a basic block or using global flow analysis, are available in the literature [66], [2], [62]. The treatment of common subexpression in compiler literature and practice is usually based on value numbering [20] and Sethi-Ullman numbering [59] techniques in a local peephole optimization framework [42], [61]. Nakata [44] and Redziejowski [51] proposed earlier versions of the Sethi-Ullman common subexpression elimination labeling algorithm. Other approaches include [22], [15]. Fisher and Le Blank showed a simple and effective way to significantly enhance the effectiveness of common subexpression elimination by using commutativity and associativity [24].

Common subexpression elimination is widely used manually as a tool for reducing the number of operations in many algorithms in applied numerical analysis [26], and in DSP, video and image applications [6], [50].

It is interesting to note that while high level synthesis literature has an extensive coverage of common subexpression replication [37], [48], [60], there have been few efforts [32] dedicated to common subexpression elimination. Also, surprisingly, those efforts were mainly concentrated on the use of common subexpression elimination to reduce critical paths, instead of the more apparent goal of reducing the number of operations, and therefore area and power of designs. Iqbal et. al. [32] used common subexpression elimination within their algebraic speed-up procedure for throughput improvement.

There is another very interesting use of common subexpressions. Although the symbolic algebra manipulation system Mathematica provides only a limited mechanism for exploration of common subexpression during expression manipulation, it extensively uses common subexpression recognition for the reduction of storage requirements [65].

Exploration of common subexpression was also addressed in theoretical computer science literature. Research done at AT&T Bell Laboratories in the mid seventies indicates that common subexpression exploration makes scheduling problem computationally intractable for both machines with only one register [13] and machines with an arbitrary number of registers [2].

Common subexpression elimination is also often used in logic synthesis, most often in kernel based factoring framework [10]. While the initial application of factoring in logic synthesis was mainly directed toward area minimization [9], more recently it is also applied for power minimization [56].

3.0 Problem Formulation and Examples

For simplicity and clarity in the next two sections we will use examples with constants that are integers. Obviously, there is no change in either the problem formulation or the algorithms when arbitrary fixed point numbers are used.

The examples shown in Figure 1 a-c introduce the multiple constant multiplication problem (MCM). Consider first a computation which has only one multiplication with constant a = 815, as shown in Figure 1a. This multiplication can be done using a series of shifts and additions as shown in Figure 1d. During substitution of the constant multiplication by shifts and additions, a binary representation of the constant a in the form 1100101111_2 is used.

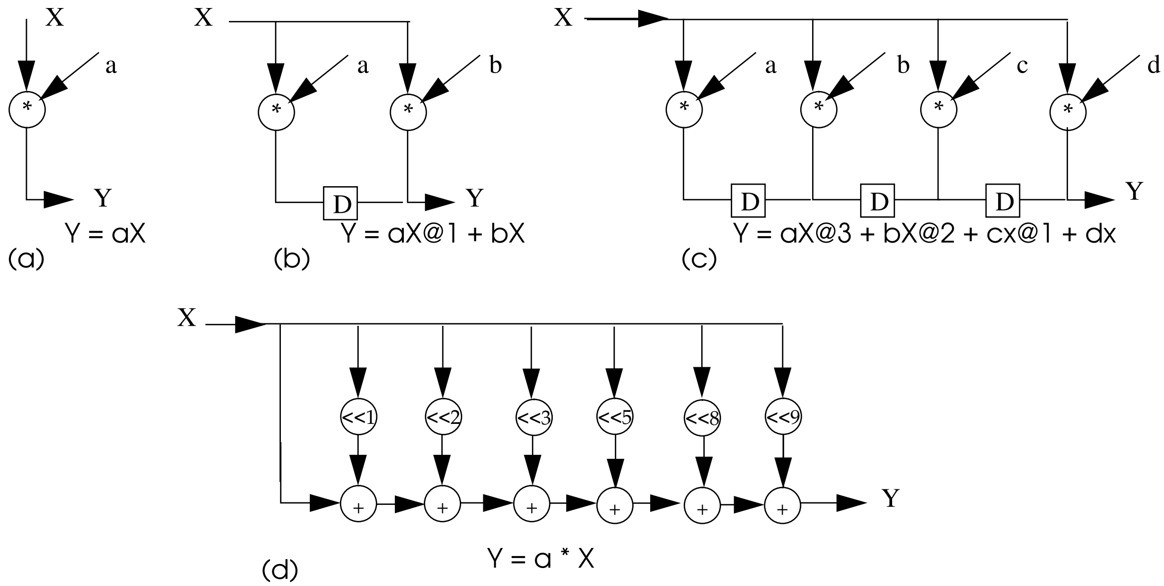
Suppose now the goal is to implement the computation in Figure 1b, using only shifts and additions. Once again the constants, a = 815 and b = 621 can be represented in the binary form as 1100101111_2 and 1001101101_2 respectively. Note that several of the shifts (first, third, fourth, sixth and tenth from the right) can be shared during the computation of the two different products as shown in Figure 1e. One needs a total of only 7 shifts (no shift is needed for the right-most digit) due to fact that both the constants are being multiplied with the same variable. The second important point is that many of intermediate results of additions can also be shared. For example, a*X = 1000101101*X + 0100000010*X, and b*X = 1000101101*X + 0001000000 *X share the common term 1000101101*X.

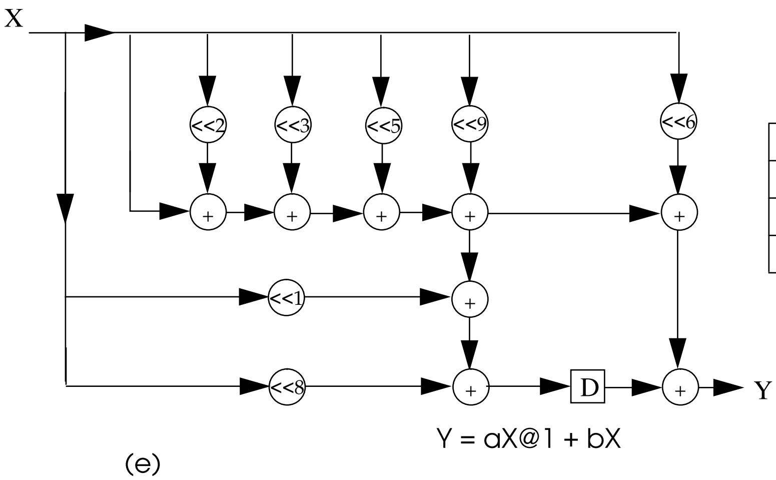
Finally consider the example in Figure 1c. The binary representation of all the constants is shown in Table 2. Obviously, now there exist even higher number of possibilities to share shifts and

a	815	1100101111
b	621	1001101101
c	831	1100111111
d	105	0001101001

Table 2: Binary representation of constants for example from Figure 1c.

additions while multiplying the variable X with the multiple constants. Table 3 shows the





a	815	1100101111
b	621	1001101101
c	831	1100111111
d	105	0001101001

(g)

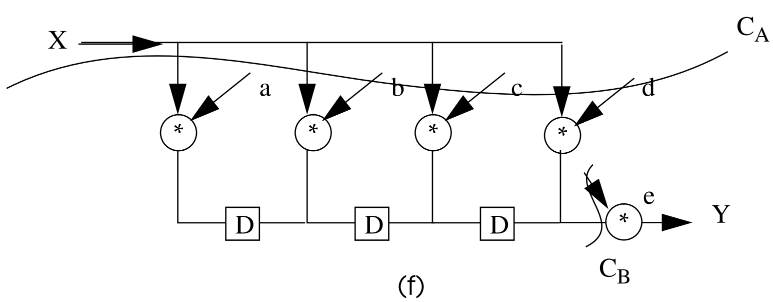


Figure 1: Motivational Examples: (a) single constant multiplication; (b) multiplication with two different constants; (c) multiplication with four constants; (d) the example from (a) after substitutions of multiplication with shifts and additions; (e) common subexpression exploration applied on the example shown In Figure (b); (f) cut selection for the application of scaling on example from Figure 1c; (g) decimal and binary representation of the values used in the examples

number of the identical shifts for all pairs of constant multiplications. If additions are shared between a and c, and between b and d, it is easy to see that one will save 9 additions. While initially 21 additions were needed, now only 12 additions are sufficient.

However, note that the optimization process can be continued further, and that two shifts (first and sixth from the right) are present in all multiplications. So if their sum is computed first, this intermediate result can be shared among all the constant multiplications, thus saving one more addition.

If the sharing of shifts and additions is not used, the example from Figure 1c requires 21 shifts (excluding shifts by 0) and 21 additions. However, when intermediate results in forming products are exploited, only 8 shifts and 11 additions are needed.

	a	b	c	d
a	-	5	7	3
b		-	5	4
c			-	3
d				-

Table 3: Number of the same shifts between all pairs of multiplications by constant for example from Figure 1c

An interesting and intuitively appealing idea is to employ signed digit (SD) encoding, or some other encoding that reduces the number of ones in the binary representation of the constants, and then attempt sharing of additions and subtractions. Table 4 shows the constants for Figure 1c

a	815	110011000N
b	621	1001101101
c	831	110100000N
d	105	0001101001

Table 4: Alternative representation of constants for example from Figure 1c. N represents -1

after one such encoding. Note that now only 7 shifts are needed. Table 5 shows the number of intermediate results that can be shared during multiplications by constants. Again, it is advantageous to combine some of intermediate results. As is suggested by Table 5, one will profit most if a and c are combined first, followed by combining b and d. This reduces the number of additions and subtractions to only 10.

	a	b	c	d
a	-	2	3	1
b		-	2	4
c			-	1
d				-

Table 5: Name shifts between all pairs of multiplications by constant, when both additions and subtractions are used

We conclude this section by stating the **multiple constant multiplication problem**: *Substitute all* *multiplications with constants by shifts and additions (and subtractions). so that the number of shifts and additions (and subtractions) is minimized*. The next section describes a systematic approach for accomplishing this. In the rest of the paper, we assume that barrel shifters are used, and therefore cost of all shifts is identical. However, even when this is not the case, our approach will minimize the number of shifts to such an extent that at most one shift operation of any given size is used. Therefore, the algorithm yields a high quality solution with respect to cost of shifts, regardless of the properties of the used implementation style and libraries.

4.0 Iterative Matching Algorithm

The analysis of the MCM problem in the previous section indicates that a natural way to solve the MCM problem is to execute *recursive bipartite matching*. Recursive bipartite matching will match at each level all constants in pairs, so that the payoff is maximized for each single level. There are a number of very efficient algorithms for bipartite matching [21]. However, this approach has several serious drawbacks, the most important of which is illustrated by the following example. Suppose that one needs to multiply a variable X with constants a, b and c, such that a = 111111111100000_2, b = 111110000011111_2, and c = 000001111111111_2. Obviously, bipartite matching will combine only two of these constants, and result in a saving of 4 additions so that 23 additions are needed after the application of the MCM transformation. However, if one first forms numbers d = 11111000000000_2, e = 000001111100000_2, and f = 000000000011111_2, then by noting that a * X = d * X + e * X, b * X = d * X + f * X, and c * X = e * X + f * X one needs only four addition each for computing d * X, e * X, and f * X, and 3 more for computing a * X, b * X, and c * X, for a total of only 15 additions.

We can summarize the drawbacks of the bipartite matching approach by pointing out that it is often advantageous to form intermediate constants by combining parts of more than two constants. Another important bottleneck is that bipartite matching at one level does not take into account how a particular match influences matching at the next level.

In order to preserve the advantages of using matching algorithms to solve the MCM problem while addressing the drawbacks of bipartite matching, we developed the algorithm described by the following pseudo-code:

Iterative matching for the MCM problem:

Express all constants using binary (or SD) representation;
Eliminate all duplicates of the same constants;
Eliminate all constants which have at most one nonzero digit in their binary (SD) representation;

Let CANDIDATES = Set of all constants in binary representation;
Calculate Matches Between all elements of the set CANDIDATES;
while there exist a match between two entries in at least 2 binary digits {
Select Best Match;
Update the set CANDIDATES;
Update Matches by adding matches between new entries and those already existing in the set CANDIDATES;
}

The first step is a simple conversion. SD representations refers to use of digits 1, 0, and -1 in the representation of constants [31]. There are numerous signed digit (SD) representations that may be suitable to different extents for the subsequent optimization using the iterative matching algorithm. We use a greedy heuristic for selection of SD representation that minimizes the number of shifters required. The next two steps are simple preprocessing steps which in practice often significantly reduce the run time of the algorithm. It is interesting to note that identical constants are quite common is certain type of benchmarks (e.g. digital filters and linear transform) and quite rare in other types (error-controlling codes and elementary function evaluation). Of all identical constants only one instance is included in the set of candidates. At the end of the program, all constants which had initially the same value as the some included constants, are calculated using the same set of common subexpressions. The third step is based on a simple and obvious observation that only constants which have at least two nonzero digits are suitable for common subexpression elimination.

A match between two constants is equal to the number of identical nonzero digits at the same position in their binary representations, reduced by one (because n-1 additions are needed to add n numbers). This is equal to the number of operations saved if these two candidates are sharing addition/subtraction operations for forming common intermediate results.

The best match is selected according to an additive objective function which combines immediate saving and the change in likelihood for later savings. The immediate payoff is, obviously, equal to the reduction in the number of operations when a particular pair of constants is chosen. To estimate the potential future savings after a particular match is selected, we estimate the influence of selecting this match on our future ability to reduce the number of additions using matching between the remaining constants. We do this by evaluating the difference in the average of the top k (where we use k=3 based on empirical observations) best bitwise matches in the set CANDIDATES, and the average of the top k best bitwise matches in the set CANDIDATES excluding the two constants being considered for the current match. The intuition behind this measure is that this average is a good indicator of potential for matching the remaining candidates among themselves. While it is unrealistic to expect that we will be able to select best matches for all remaining constants, in many instances for majority of them we were able to select some of those high ranked matches.

The set CANDIDATES is updated by first removing the two constants which constituted the best match, and then adding the constant corresponding to the matched digits, as well as the differences between the two matched constants and this newly formed constant. If any of those new constants is already in the set of candidates, only one, original instance of that constant, is kept in the set of candidates. For example, after constants 111100_2 and 110011_2 are matched, they are replaced by new elements 110000_2, 001100_2 and 000011_2.

The algorithm works the same way whether only additions are used, or both additions and subtractions The only difference is how matches are computed. Suppose that we have two numbers A and B such that both have digit 1 on an identical binary positions, both have digit -1 on b binary positions, A has digit 1 and B has digit -1 on c binary positions, and A has digit -1 and B has digit 1 on d positions number. The number of matches is computed as sum a + b + max (0, c+d-1). For example if A = 815 and B = 831, then a = 2, b = 3, c = 0, and d = 0; and the sum of matches is 5. The motivations for this matching function is based on the observation that we can always match all identical digits, and that nonzero digits can be also matched, but this type of matching will result in one less saved operation. This is so because the intermediate result for positions where A and B have complementary values has to be computed only once and that either add (for one of the numbers) or subtract (for the another number) from the intermediate result where A and B have the same values.

The worst case run time analysis of the iterative pairwise matching algorithm for the MCM transformations can be conducted as follows. Suppose that we have N numbers and that each number has at most B bits. At each step of the algorithm the resulting set of constants will have at least two nonzero bits less than the number of nonzero bits in the previous set of constants. The total number of bits initially is equal at most N * B. The largest number of constants during the constant decomposition is bounded by N * B, because each constant can be decomposed in at most (B-1) constituents. Each step of the iterative pairwise matching algorithm takes at most time $O(B^2 * N^2)$. This is so because there can be at most N * B constants during the numbers decompositions, and at each step it takes at most quadratic time to select the best match. Therefore, we can conclude that the worst case run time is at most cubic function of the number of initially different constant and quadratic function of the number of bits. The number of bits is almost always very small, so the second part of the run time can be considered constant, resulting in overall $O(N^3)$ worst case run time.

It is interesting to note that for a given size the run time is positively correlated with the reduction of the number of additions. It is so because the algorithm continues to run only if it is able to find new matches, i.e. reduce the number of required operations.

The experimental validation of the effectiveness of the MCM transformation, and the iterative matching algorithm is presented in Section 8. We will finish this section by a theoretical analysis of the asymptotic effectiveness of the MCM transformation with increasing problem size. The following theorem clearly indicates the effectiveness of the MCM approach on large instances of the problem.

Asymptotic Effectiveness Theorem: *An arbitrarily large instance of the multiple constant multiplication problem can always be implemented with a bounded finite constant number of shifts and additions irrespective of the problem size. Furthermore, if the iterative matching algorithm is used, at most (B-1) shift is required regardless of the number of constant multiplications. (B is the number of bits used for binary representation of the constants.*

Proof: Consider that we have a very large number of constants, of at most B bits, being multiplied to a variable. Then it is clear that there are at most (B-1) distinct shifted versions of the variable. Further, the number of distinct constants is also bounded by 2^B, and due to the *pigeonhole principle*[1] [28] the constants will start to repeat when their number becomes large. Each constant-variable multiplication needs at most (B-1) additions. Thus, the total number of operations needed is bounded by $(B-1)(1+2^B)$

Observation: The bound for the number of additions in the above proof is very pessimistic for most cases. Suppose that we have a very large number of constant multiplications, say M, with the same variable, and that for these M constant-variable multiplications we need N shifts and additions. When multiplication with a new constant is considered, this constant will quite likely share one or more common sub-expressions with one or more of the previously considered constants. This is a consequence of the fact that there are only finitely many different constants that can be represented using a given number of bits. As the number of constants increases, the number of intermediate results and constants will start to repeat due to the probabilistic version of the pigeon hole principle. Eventually, as the size of the problem becomes very large, the probability that all intermediate results for forming new constants are already available will become very high. Therefore, note that significantly sharper bounds can be derived by

1. This principle states that if there are N holes and N+1 pigeons, then at least one hole will have at least two pigeons Sometimes the pigeonhole principle is quoted as a special case of Dirichllet box principle: “If n objects are put into m boxes, at least one box must contain $\geq \left\lceil \frac{n}{m} \right\rceil$ objects, and at least one must contain $\leq \left\lceil \frac{n}{m} \right\rceil$ objects”. The pigeonhole principle has a number of application in combinatorics and theoretical computer science [3].

considering particular algorithms (e.g. iterative pairwise matching) for the minimization of the number of additions.

An important consequence of the asymptotic effectiveness theorem is that as the size of the MCM problem increases, the MCM transformation is increasingly more effective. Interestingly, a limited experimentation with real life examples indicates that in most cases the probabilistic pigeon hole principle starts showing its effects for relatively small instances of the MCM problem.

5.0 Using Scaling as a Preprocessing Optimization Step for the MCM Transformation in Linear Computation

The effectiveness of the iterative matching algorithm is often very high. For example in the number of filters and controllers the number of shifts is reduced by an order of magnitude, and the number of additions by more than 50%. Nevertheless, in the special but widely used case of linear computation structures[2], the effectiveness of the MCM transformation can be significantly improved by preprocessing the linear computation structure with the scaling transformation introduced in this section. A branch and bound algorithm has been developed for scaling based optimization of the MCM problem.

For the sake of simplicity we will assume that the initial linear computational structure has only one strongly connected undirected component. Most real-life examples, as well as examples presented in the literature, have only one strongly connected undirected component. Generalization to computational structures with more than one isolated components is straightforward because each component can be treated separately.

Let us define a *cut* as a set of variables which divides the computational structure into two components such that all the inputs are on one side of the cut, and all the outputs are on the other side of the cut. Feedforward cut is a cut that consists only of feedforward edges. Feedforward edges can be easily detected using an algorithms for strongly connected components [21] - they are the edges that go from one strongly connected component to another (different) strongly connected component. Figures 1f, 2c and 3c show examples of two such cuts, denoted by C_A and C_B. The following theorem, which is easy to prove, forms the basis for the scaling:

Scaling Theorem: *Suppose that two feedforward cuts are identified in a linear computational structure. If all variables on one of the feedforward cuts are multiplied by a constant* β*, and all variable on the other feedforward cut are multiplied by a constant* $1/\beta$*, the overall input-output relationship of the linear*

2. A *Linear Computation Structure* is composed of variable-variable additions and constant-variable multiplications. Such computation structures are quite common in control and signal processing systems.

computation is not altered.

The theorem can be applied to a computation structure an arbitrary number of times. Figures 2c and 3c show its application to FIR and IIR filters. The idea behind the scaling transformation is to select the cuts and the constant $1/\beta$ required by the scaling theorem in such a way that the number of shifts and additions in the MCM transformation is minimized. Therefore, at least one cut is always placed such that the maximum number of constant-variable multiplications are affected by that cut. Note that no new multiplications are introduced when the cut is made only along constant-variable multiplications. Otherwise, additional multiplications are introduced, but the overall number of operations after substitution of constant multiplications by shifts and additions is reduced by a proper selection of the constant β. For instance, if two cuts are introduced in the example from Figure 1c, as shown in Figure 1f, and $\beta = 1.03125$ is chosen, then a simple calculation indicates that only 8 additions are now sufficient, even though one of the additions is used to form the constant $\beta = 1 + 0.5 >> 4$. The new coefficients for the MCM transformation after scaling are shown in Table 6.

a	815 * β	1101001000
b	621 * β	1010000000
c	831 * β	1101011000
d	105 * β	0001101100

Table 6: The coefficients a, b, c, and d after scaling by $\beta = 1.03125$

To effectively apply the scaling transformation, one has to address the following two problems: selecting the cuts, and selecting the constants β. As already indicated, one cut is always placed to affect the maximum possible number of multiplications by constants. If there is no possibility to place another cut that is only across other multiplications with constants, then the cut which needs the smallest number of additional multiplication is selected. The smallest cut can be efficiently found using the min-cut algorithm. For all computations with only one input, or only one output, putting the cut just after the input, or just before the output, results in only one new multiplication. In general, it is advantageous to apply scaling theorem repeatedly. For this task we use a greedy algorithm, that successively applies the cut selection and multiplication constant procedures. The algorithm is terminated when no new cuts with improvements are observed.

The constant β is selected using a branch and bound algorithm. All constants are multiplied iteratively by β in range 1 to 2, with the step equal to the user specified 2^{-n} value. The best current solution is maintained, and a new constant β is attempted only if the number of non-zero elements in the binary representation of the resulting constants (obtained after multiplying by β or $1/\beta$) is within k% of the number of nonzero elements in binary representation of the best solution, and the sum of positions where any two pairs of constants match is within m% from the corresponding number in the best current solution. In our experiments we used an empirically derived value of 5% for both k and m.

It is interesting to note that in a number of applications it is advantageous to select one of the cuts required by the scaling theorem on the primary output or on all primary outputs. It is so because those outputs are anyhow additionally scaled by transmission parts of the overall system, and all the information required for the receiving part is contained only in the relative ratios of data values of the received signal. Note, that in this case the semantic of the targeted applications allows that essentially only one cut is applied. For more detailed treatment of the design strategy, see [50].

Also note that even when the computation is overall nonlinear, the scaling theorem can be applied on linear subparts. In that case, a useful approach is to first identify all maximally large linear computations [48].

6.0 Relationship between the MCM Problem and Logic Synthesis

Our formulation of the MCM problem as one of minimizing the number of additions by using recursive application of common subexpression elimination is equivalent to a restricted case of multilevel combinational logic synthesis. This equivalence is interesting because it potentially allows one to develop alternate algorithms using multilevel logic synthesis systems such as MISII from University of California, Berkeley [11].

The equivalence between MCM and logic synthesis is based on the observation that the common subexpression elimination to minimize additions in MCM is equivalent to a multilevel factorization on a set of Boolean functions obtained by encoding the coefficients in the MCM problem.

For example, if one has the following three coefficients in the MCM problem:

$$C_1 = 111110$$
$$C_2 = 111101$$
$$C_3 = 011111$$

then he can encode these coefficients as the following purely disjunctive Boolean functions:

$$C_1 = x_1 + x_2 + x_3 + x_4 + x_5$$
$$C_2 = x_1 + x_2 + x_3 + x_4 + x_6$$
$$C_3 = x_2 + x_3 + x_4 + x_5 + x_6$$

where x_1, x_2, ... correspond to bits at positions (from the left) 1, 2, ... in the coefficients.

Now, it is easy to see that doing common subexpression elimination on the coefficients for the MCM problem of minimizing additions is equivalent to finding a multilevel implementation of the corresponding Boolean functions that are pure disjunctions with each input variable appearing at most once in the disjunction. The resulting multilevel decomposition should use only 2-input OR gates, must have no reconvergent paths (as is obviously true for the input functions), and should use a minimum number of the OR gates. The restriction of no reconvergent paths is needed because while $x + x = x$ is true in boolean domain, it is not in the algebraic domain of MCM - the restriction ensures that the property $x + x = x$ is not exploited during logic synthesis.

Making use of this equivalence we developed a script to do the above multilevel decomposition in MISII [10], [11] - this script is basically an alternative to the iterative pairwise matching algorithm. However, we found that on all examples our iterative matching algorithm performs as good as, and often better, than the corresponding MISII script. The reason for this is that MISII is a general case multilevel logic synthesis tool - it fails to exploit the fact that the input and output Boolean networks have the special forms that use only OR gates and are free of reconvergent paths.

Finally, it must be mentioned that another advantage of this equivalence between MCM and the special case of multilevel logic synthesis is that techniques from logic synthesis also allow the more convenient way to the development of optimization algorithms that do an area-time trade-off (i.e. small/slow versus large/fast solutions) when the common subexpression elimination is explored in the MCM problem- the iterated pairwise matching algorithm is targeting only the minimization of the number of operations, and therefore area.

7.0 Numerical Stability and Relationship with other Transformations

While the effectiveness of transformations in improving key implementation parameters such as critical path, available parallelism and iteration bounds is well documented, the effect of transformations on word-length requirements during fixed point computation is a rarely addressed topic in high level synthesis and compiler literature. However, it is well-known that transformations sometimes have very strong impact on numerical properties of computations and required wordlength [36], [27]. The attitude toward this numerical stability problems varies significantly, ranging from total denial of the problem to avoidance of applying transformations.

One of the few transformations which is well suited for a theoretical analysis of numerical stability is the MCM transformation. The analysis by Golub and van Loan [26] indicates that if

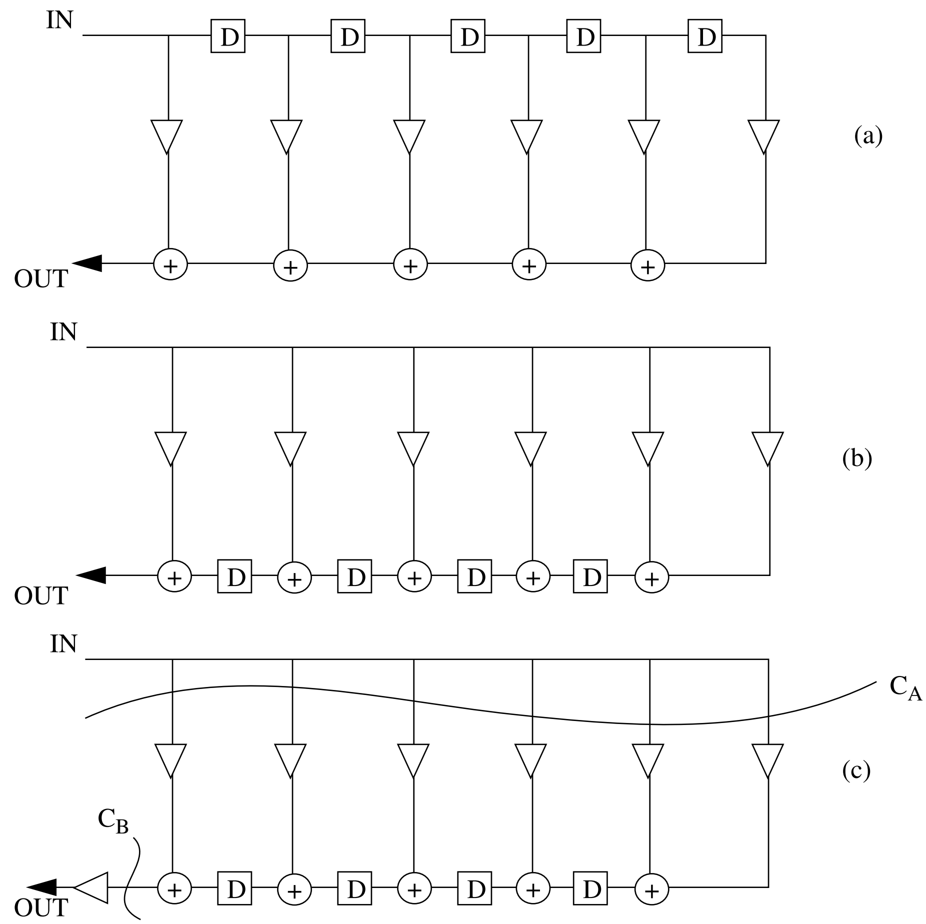


Figure 2: FIR filter - Direct Form: Applying Retiming to Enable the MCM Transformation: (a) before retiming; (b) after retiming; (c) cut selection for scaling transformation

one additional binary digit is used, one can apply an arbitrary combination of common subexpressions without disturbing the correctness of the answer. This analysis also indicates that even this additional digit is statistically very unlikely to be needed. We experimentally verified this claim on several examples, including a 126 tap FIR filter which is part of a PCM system developed by NEC Japan, and the results of our investigation are shown in Figure 4. Figures 4a, 4b, and 4c show the transfer function of the filter corresponding to double precision floating-point arithmetic (essentially infinite precision arithmetic), finite precision fixed-point arithmetic before applying the MCM transformation, and finite precision fixed-point arithmetic with the same word-length after applying the MCM transformation respectively. Figures 4d, 4e, and 4f show the same three transfer functions such that 3db range is well exposed, and Figures 4g, 4h, and 4i show the same transfer function at a much finer scale for the low-pass part of the frequency spectrum. As the three comparison presented on the transfer function plots clearly demonstrate, the effect of the MCM transformation on numerical accuracy is negligible. All other considered examples showed very similar numerical properties.

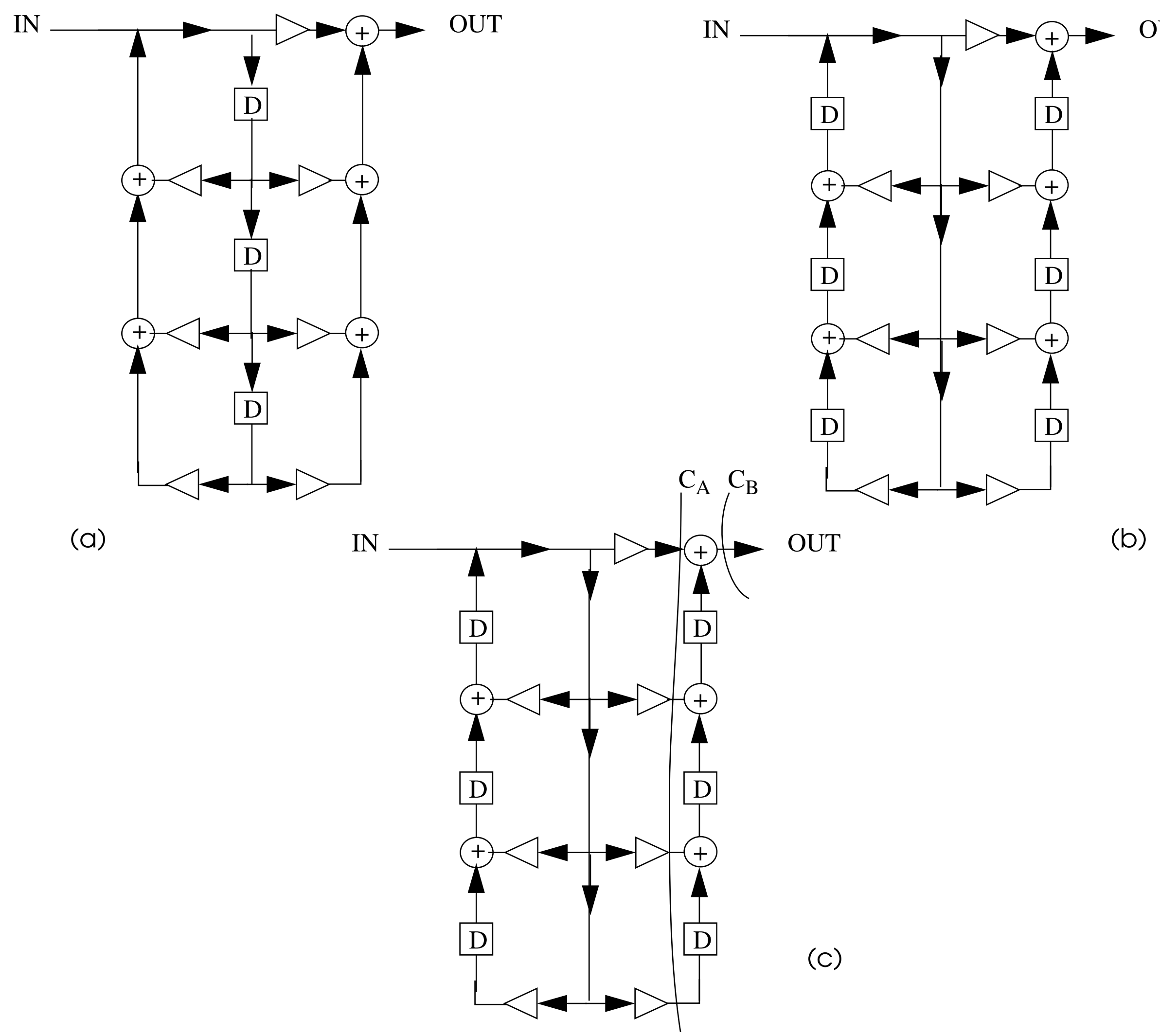


Figure 3: IIR Filter- Direct Form: Applying Retiming to enable the MCM transformation: (a) before retiming; (b) after retiming; (c) cut selection for scaling

It is well known that the application of isolated transformations is often not sufficient to achieve desired results, and that the successive or simultaneous application of several transformations is often much more effective due to the ability of some transformations to increase the effectiveness of others [47], [48]. It is easy to see that majority of transformations has either direct on indirect impact on the effusiveness of the MCM transformation. For example, using associativity one can transform expressions V = a* (x * y); and W = (b * x) * y); to V = (a* x) * y; and W = (b * x) * y). While initially it was impossible to apply the MCM transformation, after the application of associativity, the MCM can be used for optimization.

On the majority of the several hundred designs which we examined, retiming was by far most often needed to effectively enable the MCM transformation. Figures 2 and 3 show typical examples where first retiming the computation makes the MCM transformation much more effective. Performing retiming such that the payoff from applying the MCM transformation is

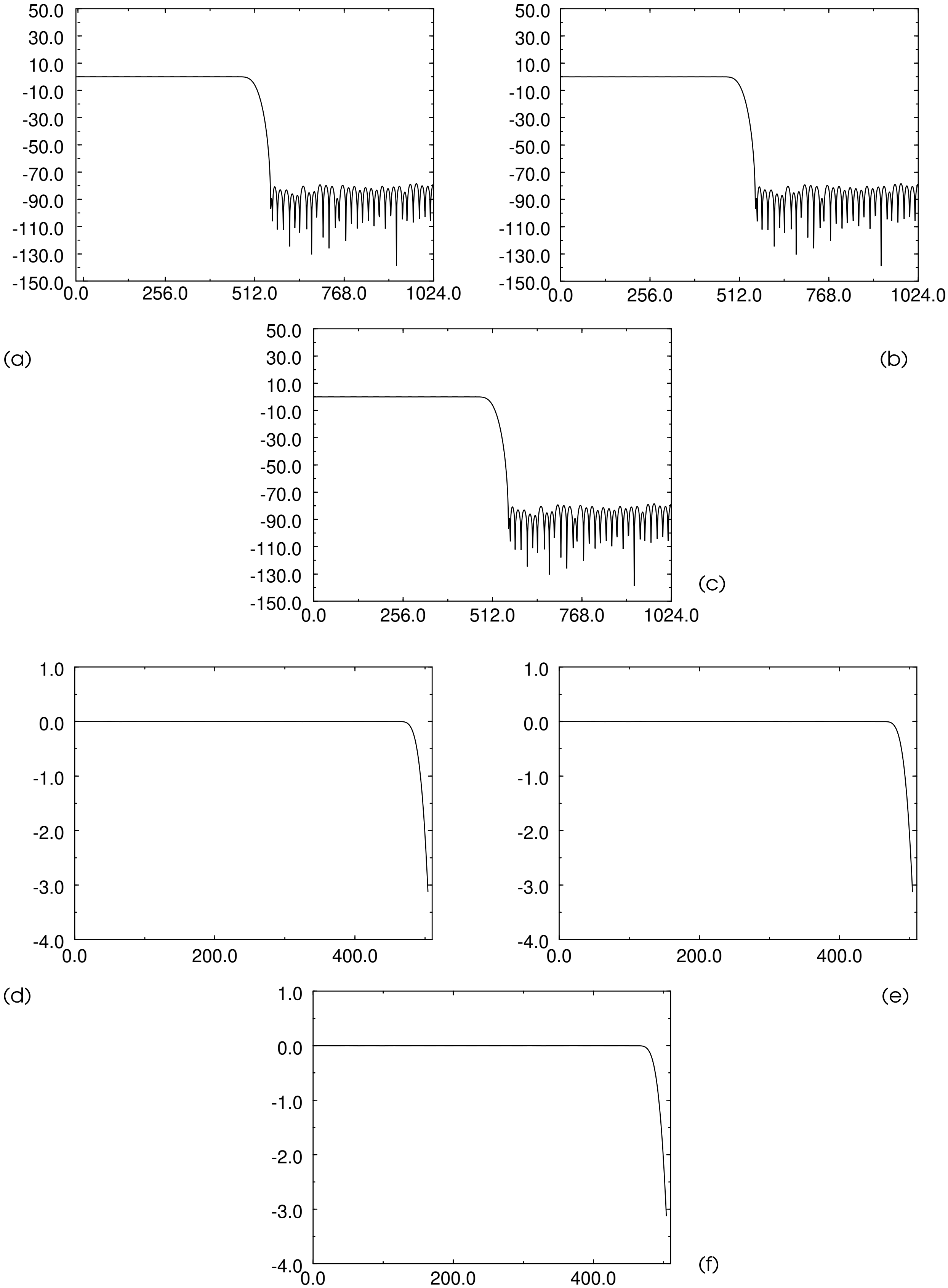


Figure 4: Simulation Results for the NEC FIR filter before the MCM transformations; (a), (d), and (g) double floating precision; (b), (e), and (h) bit true simulation; (c), (f), and (i) bit true simulation after the application of the MCM transformation

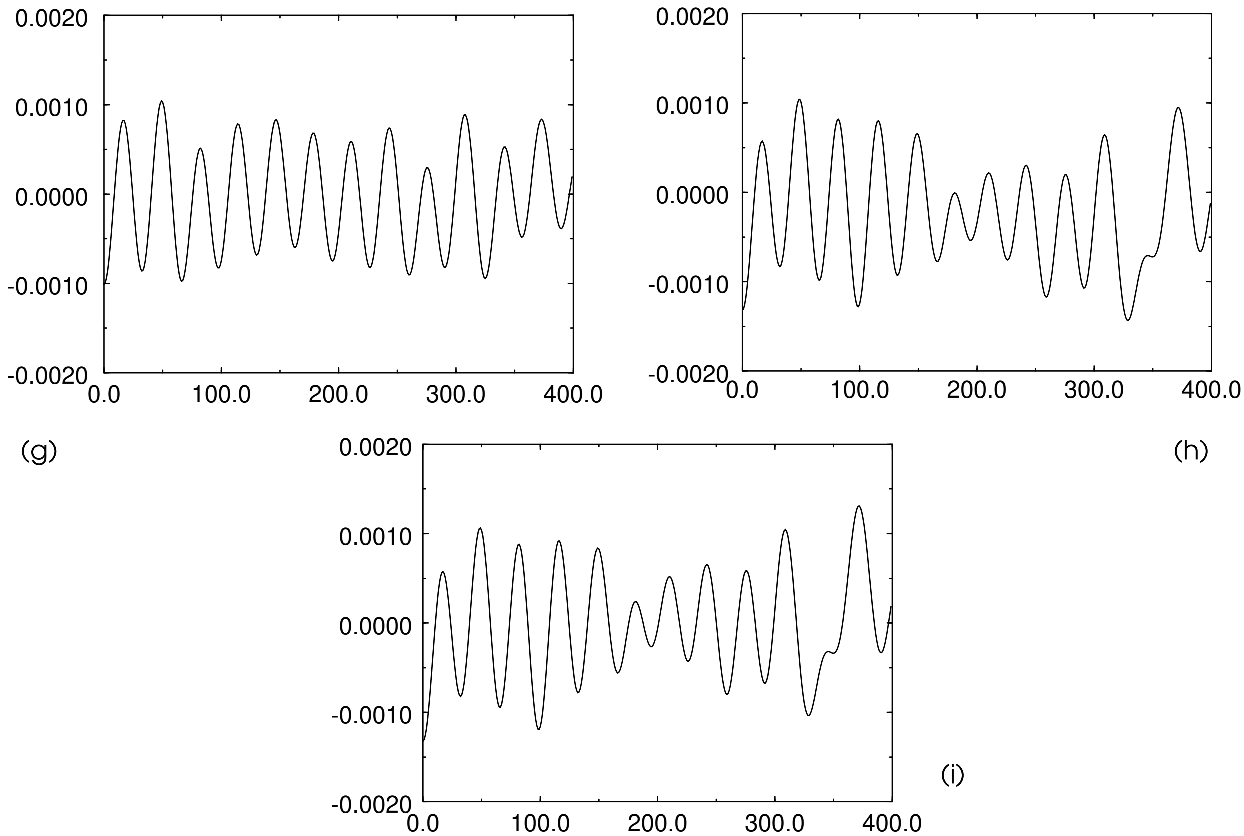


Figure 4 (cont'ed):

maximized is an involved combinatorial problem. However, on all examples that we considered, it was sufficient to adopt a simple retiming approach where the delays were just moved from edges where they were preventing the application of MCM transformations. Furthermore, note that even this retiming is actually not necessary - the shifts and additions in the MCM transformations can be shared across delays by taking into account that some of the intermediate results will be utilized in future iterations of the ASIC computation which is always done on semi-infinite streams of data. This approach eliminates the disabling effect of delays (states) on the application range of the MCM transformation.

8.0 Experimental Results

The iterative matching algorithm is very efficient and compact - it required slightly more than 1,000 lines of C code. Table 7 shows the set of benchmarks examples on which it was applied The benchmark examples are: 126 tap NEC FIR filter (NEC FIR), NEC digital to analog converter (DAC), 100 and 123 FIR Motorola filters (Mot FIR1 and Mot FIR2), 64 tap FIR filter, 3 GE linear controllers (Lin Con4, Lin Con5 and Mat) [19], linear controller for Power control [Power], linear controller for steam machine [steam], 8th order direct form IIR [8IIR], 10th order parallel form

DESIGN	INITIAL			MCM			IMPROVEMENTS		
	# of >>	# of +/ -	Total	# of >>	# of +/ -	Total	# of >>	# of +/ -	Total
NEC FIR	160	202	362	17	138	155	9.41	1.46	2.34
NEC DAC	349	416	765	98	277	375	3.56	1.50	2.04
Mot FIR1	177	231	408	19	158	177	9.32	1.46	2.31
Mot FIR2	138	170	308	15	125	140	9.25	1.36	2.20
64FIR	123	144	267	13	101	114	9.46	1.43	2.34
Lin Con4	212	183	395	16	86	100	13.25	4.59	3.95
Lin Con 5	383	358	741	25	137	162	15.32	2.61	4.57
MAT Cont	83	74	157	14	49	63	5.93	1.51	2.49
Power	136	120	256	58	99	157	2.34	1.21	1.63
Steam	260	255	515	101	178	279	2.57	1.43	1.85
8IIR	184	200	384	15	102	117	12.27	1.96	3.28
10IIR	177	162	339	54	96	140	3.28	1.69	2.42
11IIR	594	554	1148	30	266	296	19.47	2.08	3.88
12IIR	358	338	696	161	193	354	2.22	1.75	1.97
2D FIR	804	807	1611	141	377	518	5.70	1.56	3.11

Table 7: The application of the MCM Iterative Matching algorithm on 11 examples

IIR filter [10IIR], 11th order direct form IIR filter [11IIR], 12th order cascade form IIR filter [12IIR], and two dimensional 10X10 FIR video filter [2D FIR].

The performance of the iterative matching algorithm on these examples is detailed in Table 7. The following summarize the overall performance of the algorithm:

Average (Median) Reduction in number of Shifts by factor of 8.22 (9.25);

Average (Median) Reduction in number of Additions by factor of 1.84 (1.51);

Average (Median) Reduction in Total number of Operations by factor of 2.69 (2.34).

Using the iterative pairwise matching MCM algorithm to reduce the number of shift and addition operations and therefore in many situations also helps in reducing the implementation area and power consumption.Note, that although in general there is no direct relationship between the number of operations in the design specification and the area and power of the final implementation, in many cases there is a strong positive correlation. For example, in the case of NEC's 126-tap low-pass FIR filter, the estimates obtained by using the HYPER high-level synthesis system [49] show factors of 2.76 and 2.55 reductions in area and power respectively. The corresponding area and power reductions for the 44.1 KHz oversampling DAC example were by factors of 2.50 and 2.48 and for steam controller were by factors 1.71 and 2.09. The iterative matching algorithm and scaling preprocessing step took less than 2 seconds for even the largest example (2D FIR filter).

9.0 Extensions to other High Level Synthesis and Compiler Optimization Tasks

At the heart of the MCM approach is the recursive application of common subexpression elimination to reduce the number of operations needed for a given computation. This relationship with common subexpression elimination allows the MCM approach, and the iterative pairwise matching algorithm, to be easily generalized and modified for a large number of important high level synthesis optimizing transformation tasks. In this section several such applications are described.

9.1 Multiplication-Free Linear Transforms

A direct application of the MCM methodology and software to a new high level synthesis task is to multiplication-free linear transforms. Multiplication is often considered an expensive operation, and developers in many cost sensitive applications have been designing algorithms that do not use multiplications. Multiplication-free linear transform are extensively used in several engineering areas: DSP and in particular image compression and analysis [1], error correcting codes [5], [53] and neural networks [41]. The general form of multiplication free linear transform is Y = A * X, where Y and X are n-dimensional vectors and A is $n \times n$ quadratic matrix. The matrix A has as entries only values 1, -1, and 0.

We will introduce the application of the MCM transform for the optimization of multiplication free computations using Hadamard matrix transform [1]. Hadamard matrix transform decompose an arbitrary function on the set of Walsh orthonormal functions [1]. They are used for image and video compression, in signal processing of bioelectric signals, and in a number of other applications [1]. Hadamard matrix can be defined in several ways, but most often it is introduced using the following recursive definition:

$$H(1) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$H(m+1) = H(m) \otimes H(m)$$

$\otimes$ denotes the direct (Kronecker) product between two matrices, i.e.:

$$H(1) \otimes H(m) = \begin{bmatrix} H(m) & H(m) \\ H(m) & -H(m) \end{bmatrix}$$

Figure 5 shows the Hadamard matrix of size 8×8.

The 8x8 Hadamard transform is computed by evaluating Y = A * X, where X and Y are vectors of input and output samples respectively. The analogy with the MCM problem is apparent. We can use the following parallel to derive an efficient algorithm for computing the Hadamard transform. Matrix A corresponds to the binary representation of the constants in the MCM problem, and elements of vector X corresponds to the variable shifted by various amounts in the

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 \\ 1 & -1 & -1 & 1 & -1 & 1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & 1 & 1 \\ 1 & -1 & 1 & -1 & -1 & 1 & 1 & -1 \end{bmatrix}$$

Figure 5: Hadamard Matrix of Size 8 X 8

MCM problem. While the direct computation of the Hadamard transform requires 56 additions, the MCM approach reduces this number to 24.

The encoding and decoding algorithms of many error correcting codes can be represented as vector-matrix product in the form c = d * G. c is vector of k encoded bits, d is vector of m information bits, and G is m X k matrix, which describes the used encoding algorithm. Figure 6 shows the matrix G for the second-order (16,11) Reed-Muller code [52], [43].

$$G = \begin{bmatrix} 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1 \\ 0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1 \\ 0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1 \\ 0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1 \\ 0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1 \\ 0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1 \\ 0&0&0&0&0&1&0&1&0&0&0&0&0&1&0&1 \\ 0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1 \\ 0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&1 \\ 0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1 \end{bmatrix}$$

Figure 6: Encoding matrix for (16,11) second-order Reed-Muller code.

Once again, the analogy with the MCM transform is apparent. If the columns of matrix G are interpreted as the elements of the set of constants, the minimization of addition to produce the set of constants (65535, 21845, 13107, 3855, 255, 4369, 1285, 85, 771, 51, 15) is equivalent to minimizing the number of additions needed for encoding using the Reed-Muller code. Note that although all additions in error-correction computations are done using addition modulo-2, there is no any difference in the application of common subexpression elimination. It is so because on both types of additions (standard and modulo-2) the same set of algebraic transformations (associativity and commutativity) and redundancy elimination techniques are equally applicable.

The MCM approach can be also used in many error correcting codes to minimize the required number of operation not just in encoders and decoders, but also in error-correction logic. Figure 7 shows the computations required to compute error-correcting values for the (15,7) BCH code [8]. Again it is easy to see that it is possible to explore common subexpression eliminations by considering the MCM instance on the set of constants (22536, 23076, 23632, 22659, 26432, 17792, 19760, 17675, 25368, 19026, 17969, 21140, 24610, 28994, 25102, and 26770), This analogy is established by translations $e_i = 2^i$, for i = 0, ..., 14. After the application of the MCM transformation one third less additions are needed (only 48 additions are sufficient compared to initially required 72).

$$\xi_{1,1} = e_3 + e_{11} + e_{12} + e_{14} \qquad \xi_{2,1} = e_6 + e_8 + e_9 + e_{10} + e_{13} + e_{14}$$

$$\xi_{1,2} = e_2 + e_5 + e_9 + e_{11} + e_{12} + e_{14} \qquad \xi_{2,2} = e_7 + e_8 + e_{10} + e_{14}$$

$$\xi_{1,3} = e_4 + e_6 + e_{10} + e_{11} + e_{12} + e_{14} \qquad \xi_{2,3} = e_4 + e_5 + e_8 + e_{10} + e_{11} + e_{14}$$

$$\xi_{1,4} = e_0 + e_1 + e_7 + e_{11} + e_{12} + e_{14} \qquad \xi_{2,4} = e_0 + e_1 + e_3 + e_8 + e_{10} + e_{14}$$

$$\xi_{3,1} = e_3 + e_4 + e_8 + e_9 + e_{13} + e_{14} \qquad \xi_{4,1} = e_1 + e_5 + e_{13} + e_{14}$$

$$\xi_{3,2} = e_1 + e_4 + e_6 + e_9 + e_{11} + e_{14} \qquad \xi_{4,2} = e_1 + e_6 + e_8 + e_{12} + e_{13} + e_{14}$$

$$\xi_{3,3} = e_0 + e_4 + e_5 + e_9 + e_{10} + e_{14} \qquad \xi_{4,3} = e_1 + e_2 + e_3 + e_9 + e_{13} + e_{14}$$

$$\xi_{3,4} = e_2 + e_4 + e_7 + e_9 + e_{12} + e_{14} \qquad \xi_{4,4} = e_1 + e_4 + e_7 + e_{11} + e_{13} + e_{14}$$

Figure 7: Error checking sums for the (15,7) BCH code

Table 8 shows the set of error-correction benchmarks [5], [53] on which we applied the MCM approach. Only the addition operation makes sense in this case because no shifting is needed. The average reduction in the number of additions is by factors of 1.55; the median reduction is by factors of 1.56.

Examples	Additions		
	I	MCM	I/MCM
(8,2,5) Goppa	18	11	1.64
(7,4) Hadamard	21	16	1.31
(16,11) Reed-Muller	61	43	1.41
(15,4) Hamming	105	66	1.59
(1651,1631) Fire	44	29	1.52
(31,15) BCH	183	99	1.85

Table 8: Benchmark examples for linear codes

A similar analogy can be used to derive efficient algorithms for other multiplication free linear transforms. Moreover, the same type of computation is often used in neural network research.

For example, if we apply the MCM approach the number of additions needed to calculate the states and the output of a gradient descent neural network used for repairability of memory elements ([41], page 483), the number of additions is reduced from 192 to 93 (reduction by 52%) by using the iterative matching algorithm for the MCM problem. Note that, although in this example all entries of matrix are actually values 2, and -2, it is easy to apply the MCM approach by noting that most efficient way to treat multiplications (or, even better, corresponding shifts) is that all of them are applied as the last step when the final result is formed.

9.2 Linear Transforms

Of course, not all linear transforms have as entries only 1, 0, and -1, or can be as easily reduced to this form, as was the case with the neural network example in the previous subsection. In fact, many of the most widely used transforms, such as FFT and DCT, do not belong to this group. In this subsection we will present an approach which is based on multiple use of the basic MCM approach so that the number of operations (shifts and additions) is minimized when an arbitrary linear transform is targeted.

The general linear transform have the form: $y_i = \sum_{j=1}^{n} c_{ij} x_j, \quad (i = 1, ..., n)$. Note that in many of the most popular transformations there is a significant number of coefficients c_{ij} which have identical values.

The algorithm for this task can be introduced using the following pseudo-code:

Minimization of the Number of Operations in Linear Transformations

1. Minimize, using the MCM approach, the number of shifts and additions needed to compute all products of type $c_{ij} * x_j$;

2. Form a new instance of the MCM problem by considering how each y_j is formed using unique products formed in the first step;

3. Using the MCM approach minimize the number of additions in the new instance of the MCM problem.

The first step is the direct application of the MCM transformation on n separate vector-scalar products. During this step all coefficients are considered as positive by targeting only their absolute values. In the second step we assign a unique intermediate variable to each unique product. As a result we have an instance of matrix-vector product problem similar to one discussed in the previous subsection. The matrix has as entries only values -1, 0, and 1. Again, the MCM iterative pairwise matching algorithm is used for the minimization of the number of operations. This step is best explained using an example.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
y_1	0.354	0.490	0.462	0.416	0.354	0.278	0.191	0.098
y_2	0.354	0.416	0.191	-0.098	-0.354	-0.490	-0.462	-0.278
y_3	0.354	0.278	-0.191	-0.490	-0.354	0.098	0.462	0.416
y_4	0.354	0.098	-0.462	-0.278	0.354	0.416	-0.191	-0.490
y_5	0.354	-0.098	-0.462	0.278	0.354	-0.416	-0.191	0.490
y_6	0.354	-0.278	-0.191	0.490	-0.354	-0.098	0.462	-0.416
y_7	0.354	-0.416	0.191	0.098	-0.354	0.490	-0.462	0.278
y_8	0.354	-0.490	0.462	-0.416	0.354	-0.278	0.191	-0.098

Figure 8: Coefficients for corresponding products in the one dimensional 8 point DCT.

Figure 8 shows the functional dependences between output and input variables during the computation of a 8 point one dimensional DCT transform. Figure 9 shows the resulting unique products and how they are combined to form the final output values.

$$
\begin{array}{ll}
e_1 = 0.354*x_1; & e_{12} = 0.354*x_5; \\
e_2 = 0.490*x_2; & e_{13} = 0.278*x_6; \\
e_3 = 0.416*x_2; & e_{14} = 0.490*x_6; \\
e_4 = 0.278*x_2; & e_{15} = 0.098*x_6; \\
e_5 = 0.098*x_2; & e_{16} = 0.416*x_6; \\
e_6 = 0.462*x_3; & e_{17} = 0.191*x_7; \\
e_7 = 0.191*x_3; & e_{18} = 0.452*x_7; \\
e_8 = 0.416*x_4; & e_{19} = 0.098*x_8; \\
e_9 = 0.098*x_4; & e_{20} = 0.278*x_8; \\
e_{10} = 0.278*x_4; & e_{21} = 0.416*x_8; \\
e_{11} = 0.490*x_4; & e_{22} = 0.490*x_8;
\end{array}
$$

(a)

$$
\begin{aligned}
y_1 &= e_1 + e_2 + e_6 + e_8 + e_{12} + e_{13} + e_{17} + e_{19}; \\
y_2 &= e_1 + e_3 + e_7 - e_9 - e_{12} - e_{14} - e_{18} - e_{20}; \\
y_3 &= e_1 + e_4 - e_7 - e_{10} - e_{12} - e_{15} + e_{18} + e_{21}; \\
y_4 &= e_1 + e_5 - e_6 + e_{11} + e_{12} + e_{16} - e_{17} - e_{22}; \\
y_5 &= e_1 - e_5 - e_6 - e_{11} + e_{12} - e_{16} - e_{17} + e_{22}; \\
y_6 &= e_1 - e_4 - e_7 + e_{10} - e_{12} - e_{15} + e_{18} - e_{21}; \\
y_7 &= e_1 - e_3 + e_7 + e_9 - e_{12} + e_{15} - e_{18} + e_{20}; \\
y_8 &= e_1 - e_2 + e_6 - e_8 + e_{12} - e_{13} + e_{17} - e_{19};
\end{aligned}
$$

(b)

Figure 9: (a) The resulting products after the application of the first step of the algorithms for the minimization of the number of operations in linear transforms. The targeted example is one dimensional 8 point DCT shown in Figure 8; (b) The structure of the resulting MCM problem during the second step of the algorithm.

The third step is again the straightforward application of the MCM transformation.

The algorithms for the minimization of the number of shifts and additions were applied on several widely used linear transforms shown in Table 9. Again, a very significant reduction in the number of operations is achieved. The average (median) reduction of the number of shifts and additions were by factors of 3.10 (2.80) and 2.67 (2.60) respectively.

We implemented the DCT transform using 12 bit coefficients and 9 bits data. For this task we used the Hyper high level synthesis [49] and the Lager silicon compilation systems [12] from

Example	# of bits	# of >>		# of +		Improvement Ratios	
		IN	MCM	IN	MCM	IN/MCM	IN/MCM
DCT	8	308	72	300	94	4.27	3.19
DCT	12	376	74	368	100	5.08	3.68
DCT	16	529	107	521	129	4.94	4.04
DCT	24	797	190	789	212	4.19	3.72
HVS	8	122	91	119	95	1.34	1.25
HVS	12	261	153	255	161	1.71	1.58
HVS	16	367	211	361	226	1.74	1.60
HVS	24	586	313	580	334	1.87	1.74
CHVS	8	232	60	237	67	3.86	3.54
CHVS	12	398	140	391	146	2.84	2.68
CHVS	16	535	194	527	208	2.75	2.53
CHVS	24	808	308	800	328	2.62	2.44

Table 9: Benchmark example for linear transforms: DCT - discrete cosine transform; HVS - IEEE Human Vision Sensitivity Transform; CHVS - IEEE compound Human Vision Sensitivity transform; All examples are one dimensional, eight point transforms.

Berkeley. The area of the implementation using a 0.8 micron library was initially 35.86 mm^2. After the application of the MCM transformation the area was reduced to 8.78 mm^2, corresponding to the reduction by 4.09 times. The power consumption for reduced from 49.57 nJ to 21.64 nJ per sample.

During the optimization of linear transforms and several other domains of computations described in this Section, the MCM transformation is subsequently applied twice. Although, in general, multistep approaches are less effective than optimizations that consider the entire solution space at once, the improvements observed during experimentation indicate that multistep approaches provide a good trade-off between run-time and quality of results for our target, MCM transformation problems.

9.3 Single and Multiple Polynomial Evaluation

As mentioned earlier, there is also a close parallel between the problems of constant power evaluation using multiplications, and constant multiplication using shifts and additions. The power evaluation problem asks for computing x^n using only multiplications. For example, if we want to compute x^{11}, one way is to use the binary representation of the exponent 11_{10}=1011_2, and first compute x^2 and x^8 and then multiply $x * x^2 * x^8$ as indicated by the binary representation. Several researchers in compiler community [4], [39] have used the results from the constant power evaluation problem to derive efficient algorithms for the constant multiplication problem. Note that the mapping between these two problems is such that shifting and additions in the constant multiplication problem are mapped to forming exponents of the type 2^n using squaring, and multiplications of intermediate results respectively in the constant power evaluation problem. Note that this mapping between two problems is not complete: while the cost of subtractions and additions is comparable, division is significantly more expensive operation. Also, note that the cost of shifting by k positions corresponds to k multiplications (squaring). Again, the fact that the iterative matching algorithm very efficiently handles the minimization of the number of shift operations used (see Asymptotic Effectiveness Theorem, Section 4.) makes it amenable for the new task.

exponent	binary representation
1	00001
7	00111
14	01110
28	11100
29	11101

Table 10: Mapping multiple power evaluation problem to the MCM problem: Binary representation of exponents for the example explained in Section 9.3.

The parallel between the constant multiplication problem and the constant power evaluation problem can be used to extend the application domain of the MCM transformation to the interesting and important problem of calculating multiple constant powers of a variable x. We now explore the inverse of the problem studied by the compiler community, namely applying the results for the constant multiplication problem to the multiple constant power evaluation problem. Suppose that it is required to calculate the polynomial $x + x^7 + x^{14} + x^{28} + x^{29}$. As in the case of linear transformations, we will consider first the case when all coefficients are from the set {-1, 0, 1}, and then incorporate the proposed solution in the algorithm for solving more general problem. The binary representation of exponents is given in Table 10. It is easy to see that by using the MCM methodology we can share intermediate results between the second and third terms as well between the two last terms, and thus reduce the required number of multiplications by 2 compared to the case when the intermediate results are not shared.

Another application of the MCM transformation is to the calculation of multiple polynomials over the same variable. This application is illustrated by the example of G^2 blend surface calculation using quintic polynomials, which is an important and frequently used graphics task in mechanical CAD applications [64]. The example requires repeated calculation of the following 6 equations:

$$f_0(t) = -6t^5 + 15t^4 - 10t^3 + 1$$
$$f_1(t) = 6t^5 - 15t^4 + 10t^3$$
$$g_0(t) = -3t^5 + 8t^4 - 6t^3 + t$$
$$g_1(t) = -3t^5 + 7t^4 - 4t^3$$
$$h_0(t) = -(1/2)t^5 + (3/2)t^4 - (3/2)t^3 + (1/2)t^2$$
$$h_1(t) = (1/2)t^5 - t^4 + (1/2)t^3$$

Interestingly, the iterative matching algorithm can be applied twice: first in computing all the used powers of t, and then for sharing intermediate results during the multiplication of those power terms by the constant coefficients. For example, when calculating $7t^4$ and $15t^4$, the shifts of t^4 by 1, 2, and 3 can be shared. A complete analysis of this example shows that by using the MCM approach, only 4 multiplications, 25 additions, and 11 shifts are needed, instead of 55 multiplications, 42 additions, and 28 shifts.

Example	# of >>			# of +			# of *		
	I	MCM	MCM/ I	I	MCM	MCM/ I	I	MCM	MCM/I
Lag	61	61	1.00	85	53	0.62	11	5	0.45
Her	33	30	0.91	64	24	0.38	11	5	0.45
Leg	58	55	0.95	94	49	0.52	11	5	0.45
Ber	79	40	0.51	102	39	0.38	11	5	0.45
Eul	35	31	0.89	60	26	0.43	16	6	0.38

Table 11: The reduction in the number of shifts, additions and multiplications when computation of the set of polynomials is targeted. Lag - Lagguerre polynomials; Her - Hermite Polynomials; Leg - Legendre polynomials; Ber - Bernoulli polynomials; Eul - Euler polynomials

Calculation of polynomials dominates many widely used application domains. Table 11 shows reduction in the required number of operations for five different sets of polynomials The average reduction for the number of multiplications, additions and shifts were by 56.5%, 53.3%, and 15%. The corresponding median reductions were by 54,5%, 56.7%and 9.1%.

10.0 Conclusion

We formulated the multiple constant multiplication (MCM) problem, and proposed an iterative pairwise matching algorithm for solving it. The relationship of the MCM transformation to other transformations is also studied We used a newly established scaling theorem to enhance the effectiveness of the MCM approach. A simple generalization of the problem and augmentation of the algorithm were used to significantly enlarge the application range of the proposed approach. The new applications include the minimization of the number of operations in calculation linear transforms and polynomials. On a large set of industrial examples the MCM approach yielded large average and median improvements in the number of operations. The algorithm was very effective in reducing area and power requirements. We also studied the numerical stability of the MCM transformation and showed that the numerical stability is negligibly affected by the transformation, and our experiment results validated this conclusion.

Acknowledgments

The authors would like to thank K. J. Singh for his help in developing the MISII script. The authors would also like to thank M. Cheong for help in developing the final version of the MCM iterative pairwise matching program.

11.0 References:

[1] N. Ahmed, K.R. Rao: "Orthogonal Transform for Digital Signal Processing", Springer-Verlag, New York, NY, 1975.

[2] A.V. Aho, S.C. Johnson, J.D. Ullman: "Code Generation for Expressions with Common Subexpressions", Journal of the ACM, pp. Vol. 24, No. 1, pp. 146-160, 1977.

[3] P. Beame, R. Impaglgiazzo, J. Krajicek, T. Pitassi, P. Pudlak, A. Woods: "Exponential Lower Bounds for the Pigeonhole Principle", 24th Annual ACM Symposium on the Theory of Computing, pp. 200-221, 1992.

[4] R. Bernstein: "Multiplication by Integer Constants", Software - Practice and Experience, Vol. 16, No. 7, pp. 641-652, 1986.

[5] R.E. Blahut: "Theory and Practice of Error Control Codes", Addison-Wesley, Reading, MA, 1983.

[6] R.E. Blahut: "Fast Algorithms for Digital Signal Processing", Addison-Wesley, 1985.

[7] A.D. Booth: "A Signed Binary Multiplication Technique", Quat. Journal Mach. and Applied Mathematics, Vol. IV, No. 2, pp. 236-240, 1951.

[8] R.C. Bose, D.K. Ray-Caudhuri: "On a Class of Error-Correcting Binary Group Codes", Information and Control, Vol. 3, No. 1, pp. 68-79, 1960.

[9] R.K. Brayton, G.D. Hachtel, C.T. McMullen, A.L. Sangiovanni-Vincentelli: "Logic Minimization Algorithms for VLSI Synthesis", Kluwer, Boston, MA, 1984.

[10] R.K. Brayton: "Algorithms for Multi-Level Logic Synthesis and Optimization", in "Design Systems for VLSI Circuits: Logic Synthesis and Silicon Compilation", ed. G. De Micheli, A. Sangiovanni-Vincentelli, P. Antogneti, pp. 197-248, Martinus Nijhoff Publishers, Dordrecht, The Netherlands, 1987.

[11] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, A. R. Wang: "MIS: A multiple-level logic optimization system", IEEE Tran. on Computer-Aided Design, Vol. 6, No. 6, pp. 1062-1081, 1987.

[12] R. W. Brodersen, editor. "Anatomy of a Silicon Compiler.", Kluwer Academic Publishers, Boston, MA, 1992.

[13] J. Bruno, R. Sethi: "Code Generation for a one-register machine", Journal of the ACM, pp. Vol. 23, No. 4, pp. 502-510, 1976.

[14] A.W. Burks, H. Goldstine, J. von Neumann: "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument", Princeton, NJ, June 1947.

[15] V.A. Busam, D.E. Englund: "Optimization of expressions in FORTRAN", Communications of the ACM, Vol. 12, No. 2, pp. 666-674, 1969.

[16] F. Catthoor, et. al.: "SAMURAI: a General and Efficient Simulated Annealing Schedule with Fully Adaptive Annealing Parameters", Integration, Vol. 6, pp. 147-178, 1988.

[17] A. Chandrakasan, M. Potkonjak, J. Rabaey, R.W. Brodersen, "HYPER-LP: A System for Power Minimization Using Architectural Transformations", IEEE ICCAD-92, pp. 300-303, 1992.

[18] A. Chatterjee, R.K. Roy: "An Architectural Transformation Program for Optimization of Digital Systems by Multi-Level Decomposition", 30th ACM/IEEE Design Automation Conference, pp. 343-348, 1993.

[19] A. Chatterjee, R.K. Roy, M.A. d'Abreu: "Greedy hardware optimization for linear digital systems using number splitting and repeated factorization", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 1, No. 4, pp. 423-431, 1993.

[20] J. Cocke, J.T. Schwartz: Programming Languages and Their Compilers: Preliminary Notes, Second Revised Version, New York: Courant Institute of Mathematical Science, 1970.

[21] T.H. Cormen, C.E. Leiserson, R.L. Rivest: "Introduction to Algorithms", The MIT Press, Cambridge, MA, 1990.

[22] A.P. Ershov: "On programing of arithmetic operations", Communications of the ACM, Vol. 1, No. 8, pp. 3-6, 1958.

[23] E. Feig, S. Winograd "Fast Algorithms for the Discrete Cosine Transform", IEEE Transactions on Signal processing, Vol. 40, No. 9, pp. 2174-2193, 1992.

[24] C.N. Fischer, R.J. LeBlanc, Jr.: "Crafting a Compiler", The Benjamin/Cummings Publishing Co. Inc., Menlo Park, CA, 1991.

[25] J.C. Gibson: "The Gibson Mix", Report TR 00.2043, IBM Systems Development Division, Poughkeepsie, N.Y., 1970.

[26] G.H. Golub, C. van Loan: "Matrix Computation", The Johns Hopkins University Press, 1989.

[27] D. Goldberg, "What Every Computer Scientist Should Know About Floating-Point Arithmetic", ACM Computing Surveys, Vol. 23, No. 1, pp. 5-48, 1991.

[28] Graham, D.E. Knuth, O.Patashnik: "Concrete Mathematics: A Foundation for Computer Science", Addison-Wesley, Reading, MA, 1989.

[29] L. Guerra, M. Potkonjak, J. Rabaey: "High Level Synthesis for Reconfigurable Datapath Structures", ICCAD, pp. 26-29, 1993.

[30] L. Guerra, M. Potkonjak, J. Rabaey: "Concurrency Characteristics in DSP Programs", 1994 International Conference on Acoustic, Speech, & Signal Processing, paper 39.1, 1994.

[31] K. Hwang: "Computer Arithmetic: Principle, Architecture, and Design", John Wiley, New York, NY, 1979.

[32] Z. Iqbal, M. Potkonjak, S. Dey, A. Parker: "Critical path Minimization using retiming and algebraic speed-up, ACM/IEEE Design Automation Conference, pp. 573-577, 1993.

[33] R. Karri, A. Orailoglu: "Transformation-Based High-Level Synthesis of Fault-Tolerant ASICs", Design Automation Conference, pp. 662-665, 1992.

[34] D.E. Knuth: "The Art of Computer Programming: Volume 2: Seminumerical Algorithms", 2nd edition, Addison-Wesley, Reading, MA, 1981.

[35] D. Ku, G.De Micheli: "Constrained Synthesis and Optimization of Digital Circuits from Behavioral Specifications", Kluwer Academic Publishers, 1992.

[36] U.W. Kulish, W.L. Miranker, "The Arithmetic of the Digital Computer: A new approach", SIAM Review, Vol. 28, No. 1, pp. 1-36, 986.

[37] D.A. Lobo, B.M. Pangrle: "Redundant Operator Creation: A Scheduling Optimization Technique", 28th Design Automation Conference, pp. 775-778, 1991.

[38] O.L. MacSorley: "High-Speed Arithmetic in Binary Computers", Proceedings of the Institute of Radio Engineers, Vol. 49, No. 1, pp. 67-91, 1961.

[39] D.J. Magenheimer, L. Peters, K. Pettis, D. Zuras: "Integer Multiplication and Division on the HP precision Architecture", Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS II), pp. 90-99, IEEE Computer Society Press, Washington, DC, 1987.

[40] D.J. Magenheimer, L. Peters, K. Pettis, D. Zuras: "Integer Multiplication and Division on the HP precision Architecture", Trans. on Computers, Vol. 37, No. 8, pp. 980-990, 1988.

[41] P. Mazumder, J-S. Yih: "Neural Computing for Build-In-Self-Repair of Embedded Memory Arrays", International Symposium on Fault-Tolerant Computing", pp. 480-487, Chicago, IL, 1989.

[42] W.M. McKeeman: "Peephole optimization", Communications of the ACM, Vol. 8, No. 7, pp. 443-444, 1965.

[43] D.E. Muller: "Application of Boolean Algebra to Switching Circuits Design and Error Detection, IRE Transaction on Electronic Computers, Vol. 3, No. 6, pp. 6-12, 1954.

[44] I. Nakata: "On Compiling Algorithms for Arithmetic Expressions", Communications of the ACM, Vol. 10, Vol. 8, pp. 492-494, 1967.

[45] K.K. Parhi. "Algorithm Transformation Technique for Concurrent Processors." Proceedings of the IEEE, 77(12):1879–1895, 1989.

[46] J.O. Penhollow: “Study of Arithmetic Recording with Applications in Multiplication and Division”, Ph.D. Thesis, University of Illinois, Urbana, Il, September 1962.

[47] M. Potkonjak, J. Rabaey: “Optimizing Resource Utilization Using Transformations”, *IEEE ICCAD91*, pp. 88-91, Santa Clara, CA, 1991.

[48] M. Potkonjak, J. Rabaey: “Maximally Fast and Arbitrarily Fast Implementation of Linear Computations”, ICCAD-92, pp. 304-308, 1992.

[49] J. Rabaey, C. Chu, P. Hoang, M. Potkonjak: “Fast Prototyping of Data Path Intensive Architecture”, *IEEE Design and Test*, Vol. 8, No. 2, pp. 40-51, 1991.

[50] K.R. Rao, P. Yip: Discrete Cosine Transform: Algorithms, Advantages, Applications”, Academic Press, Boston, MA, 1990.

[51] R.R. Redziejowski: “On Arithmetic Expressions and Tress”, Comuunications of the ACM, Vol. 12, No. 2, pp. 81-84, 1969.

[52] I.S. Reed: “A Class of Multiple-Error-Correcting Codes and the Decoding Scheme”, IRE Trans. on Electronic Computers, Vol. 4, No. 1, pp. 38-49, 1954.

[53] M.Y. Rhee: “Error-Correcting Coding Theory”, McGraw Hill. New York, NY, 1989.

[54] G.W. Reitwiesner: “Binary Arithmetic”, in Advances in Computers”, Vol. 1, Academic Press, pp. 261-265, New York, NY, 1960.

[55] J.E. Robertson: “Theory of Computer Arithmetic Employed in the Design of the Computer at the University of Illinois”, Digital Computer Laboratory, University of Illinois, File No. 319, Urbana, Il, June 1960.

[56] K. Roy, S.C. Prasad: “Circuits activity Based Logic Synthesis for Low Power Reliable Operations”, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 1, No. 4, pp. 503-513, 1993.

[57] H. Samueli: “An Improved search algorithm for optimization of the FIR coefficients represented by a canonic signed-digit code”, IEEE Trans. on Circuits and Systems, Vol. 34, No. 9, pp. 1192-1202, 1987.

[58] M. Schictel: “G2 Blend Surfaces and Filling of N-Sided Holes”, IEEE Computer Graphics and Applications, Vol. 13, No. 5, pp. 68-73, 1993.

[59] R. Sethi, J.D. Ullman: “The generation of optimal code for arithmetic expressions”, Journal of the ACM, Vol. 17, No. 4, pp. 715-728, 1970.

[60] M.B. Srivastava, M. Potkonjak: “Transforming Linear Systems for Joint Latency and Throughput Optimization”, EDAC-94, paper 5B-2, 1994.

[61] A.S. Tanenbaum, H. van Straven, J.W. Stevenson: “Using peephole optimization on intermediate code”, ACM Transactions on Programming Languages and Systems, Vol. 4, No. 1, pp. 21-36, 1982.

[62] W.M. Waite, G. Goos: “Compiler Construction”, Springer-Verlag, New York, NY, 1984.

[63] R.A. Walker, D.E. Thomas: “Behavioral Transformation for Algorithmic Level IC Design” IEEE Transactions on CAD, Vol 8. No.10, pp. 1115-1127, 1989.

[64] R.A. Walker, R. Camposano: “A Survey of High-Level Synthesis Systems”, Kluwer Academic Publishers, Boston, Ma, 1991.

[65] S. Wolfram: “Mathematica: A System for Doing Mathematics by Computer”, Addison-Wesley, Redwood City, CA, 1991.

[66] W. Wulf, R.K. Johnson, C.B. Weinstock, S.O. Hobbs, C.M. Geschke: The Design of Optimizing Compiler”, North Holland, New York, NY, 1975.