Efficient Substitution of Multiple Constant Multiplications by Shifts and Additions using Iterative Pairwise Matching

Miodrag Potkonjak
C&C Research Laboratories, NEC USA, Inc., Princeton, NJ 08540
Mani B. Srivastava
AT&T Bell Laboratories, Murray Hill, NJ 07974
Anantha Chandrakasan
University of California, Dept. of EECS, Berkeley, CA 94720

ABSTRACT

Many numerically intensive applications have computations that involve a large number of multiplications of one variable with several constants. 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.

After defining the MCM problem, we formulate it as a special case of common subexpression elimination. The algorithm for common subexpression elimination is based on an iterative pairwise matching heuristic. The flexibility of the MCM problem formulation enables the application of the iterative pairwise matching algorithm to several other important high level synthesis tasks. All applications are illustrated by a number of benchmarks.

1.0 Motivation and Problem Relevance

Computational transformations have recently attracted much attention in the high level synthesis community [Wal89, Rab91, Wal91, Ku92]. Transformations have also been studied in areas such as algorithm design for numerical and DSP applications [Bla85] and compilers [Fis91].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 [Bla85]. Software compilers, on the other hand, employ fast and relatively simple automatic techniques to apply algorithm transformations on very large programs [Fis91]. 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, and do not require a large and sophisticated mathematical knowledge base. 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 simultaneously.

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 area [Cha93] and power [Cha92]. 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. The importance of multiplication with constants is also indicated by recent results [Cha92] showing an order of magnitude reduction in power achieved by this transformation alone.

We formulate the MCM problem in such a way that a minimum number of shifts is selected, and after which the number of additions is minimized. The formulation has several distinctive advantages. First, it enables one to treat the problem of minimizing the number of additions as one of common subexpression elimination in a restricted domain. This allows efficient and low complexity algorithms to be used. Second, the formulation enables an interesting theoretical analysis of the effectiveness of the multiple constant multiplication transformation, which leads to the surprising asymptotic result that both the number of shifts and additions stay bounded and finite regardless of the size of the problem. Finally, our formulation also allows the MCM approach to be applied to a number of high level synthesis tasks that are based on common subexpression elimination.

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 [Bur47]. Later, several schemes were proposed for minimizing the number of operations after substitution of multiplication with constants by shifts and additions were [Rob60]. This work led to novel number representation schemes, such as the canonical signed digit (CSD) representation [Rei60, Hwa79] 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 [Rab91].

Algorithms for optimizing the number of shifts and additions have also been described in the software compiler research [Ber86]. The minimization of number of shifts is also a well studied topic in DSP, in particular in the digital filter design community [Cat88].

While minimizing the number of shifts required for multiplication with a constant is a thoroughly studied, not much attention has been paid to the simultaneous optimization of the multiplication of a variable by multiple constants. Only recently Chatterjee et. al. [Cha93] addressed this problem by presenting two optimization approaches - greedy and simulated annealing-based - for the minimization of the number of operations in vector-matrix product representation of linear systems. Although those two algorithms, based on a number spliting technique, are theoretically interesting, the algorithm proposed in this paper is superior in its simplicity, effectiveness, and range of applications.

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 [Knu81].

3.0 Problem Formulation and Examples

The examples shown in Figure 1 a-c introduce the multiple constant multiplication problem (MCM). Suppose the goal is to implement the computation in Figure 1a, using only shifts and additions. 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 1b. 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 + 1100101111*X, and b*X = 1000101101*X + 1001101101*X share the common term 1000101101*X.

Next consider the example in Figure 1c. The binary representation of all the constants is shown in Table 1.

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

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

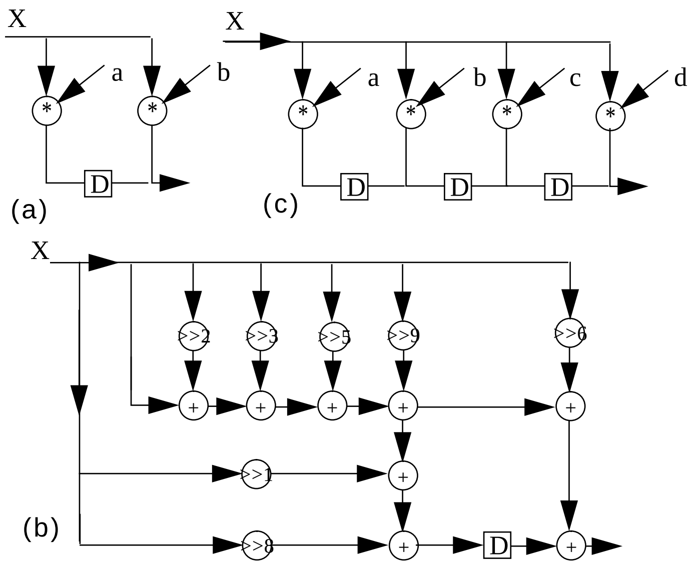


Figure 1: Motivational Examples: (a) multiplication with two different constants; (b) after substitutions of multiplication with shifts and additions; (c) more complex example

Obviously, now there exist an even greater number of possibilities to share shifts and additions while multiplying the variable X with the multiple constants. Table 2 shows the 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			-	4
d				-

Table 2: Number of matching shifts between all pairs of multiplication by constants for the example in Figure 1c

An interesting and intuitively appealing idea is to employ canonical signed digit encoding, or some other encoding which reduces the number of ones in the binary representation of the constants, and then attempt sharing of additions and subtractions. Table 3 shows the constants

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

Table 3: Alternative representation of constants for example from Figure 1c. N represents -1

for Figure 1c after one such encoding. Note that now only 7 shifts are needed. Table 4 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 4, 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 4: Name shifts between all pairs of multiplications by constant, when both additions and subtractions are used

We conclude this section by formally stating the **multiple constant multiplication problem**: *Substitute all multiplications with constants by shifts and additions (and subtractions), and use common subexpressions between various multiplications to minimize the number of additions (and subtractions)*. The next section describes a systematic approach for accomplishing this.

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 [Cor90]. 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 = 111110000000000_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 CSD) representation;
Eliminate all duplicates of the same constants;
Eliminate all constants which have at most one nonzero digit in their binary (CSD) 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. The next two steps are preprocessing steps, which in practice often significantly reduce the run time of the algorithm. 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.

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). 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 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 the 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.

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.

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 a 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). 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.

The following theorem, which we state without proof, 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.*

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.

5.0 Numerical Stability and Relationship with other Transformations

While the effectiveness of transformations in improving implementation metrics is well documented, the effect of transformations on word-length requirements is a rarely addressed topic in high level synthesis and compiler literature. The attitude toward numerical stability issues varies significantly, ranging from 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. An analysis by Golub and van Loan [Gol90] indicates that if 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 a 126 tap FIR filter which is part of a PCM system developed by NEC Japan. Figures 3a and 3b, show the transfer function of the filter corresponding to double precision floating-point arithmetic and finite precision fixed-point arithmetic with the same word-length after applying the MCM transformation respectively.

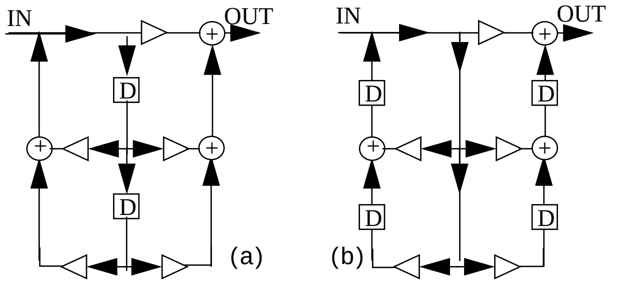
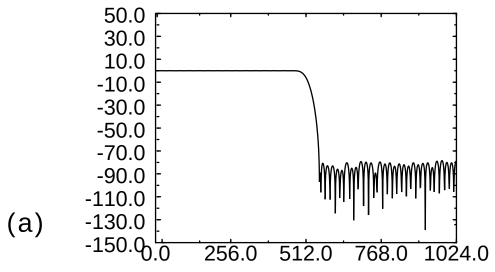


Figure 2: Applying Retiming to enable the MCM transformation: (a) before retiming; (b) after retiming



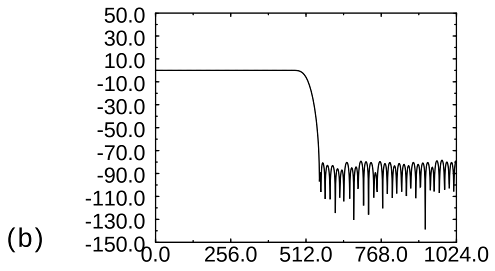


Figure 3: Simulation Results for the NEC FIR filter before (a) and after the MCM transformations

It is also well known that the application of isolated transformations is often not sufficient to achieve desired results. 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. On majority of the designs which we examined, retiming was most often required to effectively enable the MCM transformation. Figure 2 shows typical examples where retiming makes the MCM transformation much more effective. Performing retiming such that the payoff from applying the MCM transformation is 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.

6.0 Experimental Results

The iterative matching algorithm is very efficient and compact - it required slightly more than 1000 lines of C code. Table 5 shows the set of benchmarks examples on which it was applied The benchmark examples are: 126 tap NEC FIR filter (N FIR), NEC digital to analog converter (DAC), 100 and 123 FIR filters (M FIR1 and M FIR2), 64 tap FIR filter, 3 GE linear controllers (Con4, Con5 and Mat) [Cha93], Linear power controller [Power], 8th order direct form IIR [8IIR], and two dimension 10X10 FIR video filter [2D FIR].

DESIGN	INITIAL		MCM		INITIAL/MCM	
	# of >>	# of +/ -	# >>	# of +/ -	# of >>	# of +/ -
N FIR	160	202	17	138	9.41	1.46
DAC	349	416	98	277	3.56	1.50
M FIR1	177	231	19	158	9.32	1.46
M FIR2	138	170	15	125	9.25	1.36
64FIR	123	144	13	101	9.46	1.43
Con4	212	183	16	86	13.2	4.59
Con 5	383	358	25	137	15.3	2.61
MAT	83	74	14	49	5.93	1.51
Power	136	120	58	99	2.34	1.21
8IIR	184	200	15	102	12.3	1.96
2D FIR	804	807	141	377	5.70	1.56

Table 5: The application of the MCM Iterative Matching algorithm on 11 examples

The performance of the iterative matching algorithm on these examples is detailed in Table 5. Average (Median) Reduction in number of shiftsby factor of 8.71 (9.32). Average (Median) Reduction in number of additions by factor of 1.71 (1.50).

Using the MCM algorithm to reduce the number of operations also helps in reducing the implementation area and power consumption. 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 [Rab91] show factors of 2.76 and 2.55 reductions in area and power respectively. The iterative matching algorithm took less than 2 seconds for even the largest example (2D FIR filter).

7.0 Extensions to other High Level Synthesis 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 to be easily modified for a large number of important high level synthesis optimizing transformation tasks.

A direct application of the MCM methodology and software to a new high level synthesis task is to multiplication-free linear transforms. 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. Figure 4 shows the Hadamard matrix of size 8×8.

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

Figure 4: Hadamard Matrix of Size 8 X 8

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

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

Figure 5: Encoding matrix for (16,11) second-order Reed-Muller code.

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 5 shows the matrix G for the second-order (16,11) Reed-Muller code [Rhe89].

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.

Table 6 shows the set of error-correction benchmarks [Rhe89] on which we applied the MCM approach. The average and median reductions in the number of additions is by factors of 1.55 and 1.56.

Examples	Additions		
	I	MCM	I/MCM
Goppa	18	11	1.64
Hadamard	21	16	1.31
Reed-Muller	61	43	1.41
Hamming	105	66	1.59
Fire	44	29	1.52
BCH	183	99	1.85

Table 6: Benchmark examples for linear codes

The algorithms for the minimization of the number of shifts and additions, with a slight modifications, were applied on several widely used general linear transforms shown in Table 7. Again, a very significant reduction in the number of operations is achieved.

Ex	# of bits	# of >>		# of +		I	
		I	M	IN	M	I/M	I/M
DC	8	308	72	300	94	4.2	3.2
DC	12	376	74	368	100	5.1	3.7
DC	16	529	107	521	129	4.9	4.0
DC	24	797	190	789	212	4.2	3.7
HV	8	122	91	119	95	1.3	1.2
HV	12	261	153	255	161	1.7	1.6
HV	16	367	211	361	226	1.7	1.7
HV	24	586	313	580	334	1.9	1.7

Table 7: Benchmark example for linear transforms: DC- discrete cosine transform; HV - IEEE Human Vision Sensitivity Transform; I - initial; M - after MCM.

8.0 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 studied A simple generalization of the problem was used to significantly enlarge the application range of the proposed approach. On a large set of industrial examples the MCM approach yielded significant improvements in the number of operations was achieved.

Acknowledgments

The authors would like to thank M. Cheong for help in developing the final version of the MCM iterative pairwise matching program.

9.0 References:

[Ber86] R. Bernstein: “Multiplication by Integer Constants”, Software - Practice and Experience, Vol. 16, No. 7, pp. 641-652, 1986.

[Bla85] R.E. Blahut: “Fast Algorithms for Digital Signal Processing”, Addison-Wesley, 1985.

[Cat88] F. Catthoor, et. al.: “SAMURAI: a General and Efficient Simulated Annealing Schedule with Fully Adaptive Annealing Parameters”, Integration, Vol. 6, 1988.

[Cha92] A. Chandrakasan, et. al.“HYPER-LP: A System for Power Minimization Using Architectural Transformations”, IEEE ICCAD-92, pp. 300-303, 1992.

[Cha93] A. Chatterje, R.K. Roy, M.A. d’Abreu: “Greedy hardware optimization for linear digital systems using number splitting and repeated factorization”, IEEE Tran. on VLSI) Systems, Vol. 1, No. 4, pp. 423-431, 1993.

[Cor90] T.H. Cormen, C.E. Leiserson, R.L. Rivest: “Introduction to Algorithms”, The MIT Press, Cambridge, MA, 1990.

[Fis91] C.N. Fischer, R.J. LeBlanc, Jr.: “Crafting a Compiler”, Benjamin/Cummings, Menlo Park, CA, 1991.

[Gol89] G.H. Golub, C. van Loan: “Matrix Computation”, The Johns Hopkins University Press, 1989.

[Knu81] D.E. Knuth: “The Art of Computer Programming: Volume 2: Seminumerical Algorithms”, 2nd edition, Addison-Wesley, Reading, MA, 1981.

[Ku92] D. Ku, G.De Micheli: “Constrained Synthesis and Optimization of Digital Circuits from Behavioral Specifications”, Kluwer Academic Publishers, 1992.

[Rab91] 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.

[Rhe89] M.Y. Rhee: “Error-Correcting Coding Theory”, McGraw Hill. New York, NY, 1989.

[Rei60] G.W. Reitwiesner: “Binary Arithmetic”, in Advances in Computers”, Vol. 1, Academic Press, pp. 261-265, New York, NY, 1960.

[Wal89] 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.

[Wal91] R.A. Walker, R. Camposano: “A Survey of High-Level Synthesis Systems”, Kluwer Academic Publishers, Boston, Ma, 1991.