VLSI DESIGN
2000, Vol. 11, No. 4, pp. 363–379
Reprints available directly from the publisher
Photocopying permitted by license only

Graphical Design Techniques for Fixed-point Multiplication

A. G. DEMPSTER*

University of Westminster, 115 New Cavendish St, London W1M 8JS

(Received 2 August 1999; In final form 15 September 1999)

This is a tutorial paper that examines the problem of performing fixed-point constant integer multiplications using as few adders as possible. The driving application is the design of digital filters, where it is often required that several products of a single multiplicand are produced. Thus two specific problems are examined in detail, *i.e.*, the one-input/one-output case and the one-input/several-output case. The latter is of interest because it can take advantage of redundancy in the different coefficient multipliers. Graphical methods can be used to design multipliers in both cases.

For the one-input/one-output case, both optimal and sub-optimal algorithms introduced by the author are shown to be the best methods for the design of these multipliers. The key to the new methods' success is the use of different graph topologies to those available under standard methods. The optimal method uses an exhaustive search and is limited to short wordlengths, so the suboptimal methods must be used for long wordlengths. The design is shown to be analogous to the design of algorithms for exponentiation, which is becoming increasingly important in cryptography.

For the one-input/several-output ("multiplier block") case, again new algorithms designed by the author are shown to be the best. When used for designing digital filters, the multiplier block method is more efficient (uses fewer adders) than any other method that has been examined. It is so successful at reducing the number of adders used for multiplication that the non-multiplier elements begin to dominate the overall filter cost. It also allows previously unpopular filter structures to compete with structures like the lattice wave structure, despite having more coefficients of longer wordlength. The use of multiplier blocks in filter banks is also described.

A third case is examined briefly, that of matrix multiplication (several-input/several-output), which is an area of further research.

Keywords: Digital filters; Multipliers; Multiplication; Low-complexity

*Tel.: (44) 20 7911 5891, Fax: (44) 20 7580 4319, e-mail: dempsta@cmsa.wmin.ac.uk

1. INTRODUCTION

This paper examines the problem of minimising the number of adders required to perform fixed-point shift-and-add multiplication by one or more constants. Constant integer multiplication can be implemented using a network of binary shifts and adders. In an integrated circuit implementation of parallel arithmetic, binary shifts (scaling by a power of 2) can be hardwired and therefore do not require gates. Gates are thus only required to implement adders and subtractors, which require approximately equal numbers of gates. The hardware cost of the multiplier is thus approximately proportional to the number of adders and subtractors required; for convenience, both will be referred to as "adders", and their number as "adder cost". The paper is divided into three parts, addressing the distinct problems of the single multiplier case, the case where several products are required of a single multiplicand, and the matrix multiplication case where the requirement is for several sums of products of several inputs. Almost all of the material discussed and presented has been published before, but this is the first paper where these findings have been gathered together.

The problem of reducing the number of adders required for a single fixed-point multiplication has been studied for some years. As early as 1951, Booth [1] recognised that if subtractors were also allowed, the total number of adders and subtractors, hereafter collectively called "adders", could be reduced. The 1960's saw the introduction of signed-digit (SD) representation by Avizienis [2]. A coefficient in "n-bit" signed-digit representation can be written:

$$C = \sum_{i=0}^{n-1} b_i 2^i \tag{1}$$

where b_i is taken from the set $\{\bar{1}, 0, 1\}$ where $\bar{1}$ represents -1. It is therefore a ternary representation (binary representation would be identical except the b_i would be taken from the set $\{0,1\}$). In general, there are several different SD representations for a given integer, and the representation that has fewest non-zero digits is known as the canonic signed-digit (CSD) representation. Reitweisner [3] showed that the representation with no string of non-zero bits is canonic, and an algorithm for finding this representation was presented by Hwang [4]. Garner [5] showed that, on average, and for long wordlengths, CSD requires 33% fewer non-zero digits than binary. Since non-zero digits represent additions (or subtractions), CSD therefore is significantly more efficient in adders than binary. The graph multiplier technique, developed by the author and described in Section 2, has proven to be significantly more efficient than CSD.

In signal processing applications, several products of a single multiplicand are often required, *e.g.*, in a direct form finite impulse response (FIR) digital filter, the input data at some stage is multiplied by each of the coefficients. Transposition of the FIR filter as in Figure 1 allows all of these multiplications to be performed at once, with the individual multipliers replaced by the "multiplier block". Redundancy between the coefficient multipliers can be exploited in order to reduce the number of adders required to produce all of the products. Various techniques have been proposed to maximise these savings. The multiplier block method described in Section 3 was developed by the author from a technique first described by Bull and Horrocks [6]. It uses the same graphical methods used for the graph multipliers of Section 2 and has proven to be more efficient in terms of adders than any other method examined by the author.

2. GRAPH MULTIPLIERS

2.1. Graph Representation of Multiplication

Multiplication by a constant integer can be described in terms of a graph as follows. There is an initial vertex of the graph, which can nominally be assigned the value 1. There is a terminal vertex of the graph, which is assigned the value of the multiplier being designed. The multiplicand can be considered as being input to the initial vertex. The product is output from the terminal vertex. Each

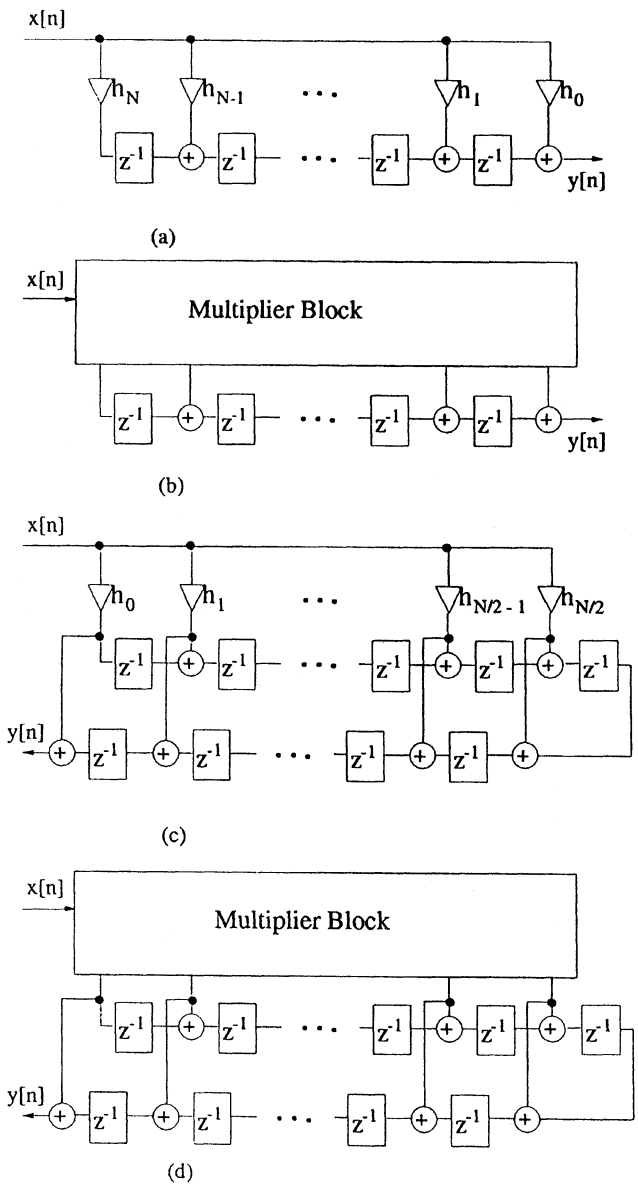


FIGURE 1 Replacement of individual coefficient multipliers by a single multiplier block. In the general transposed Nth-order direct form (a), there are $N+1$ multipliers, which are replaced by a multiplier block (b). Similarly, for an even order-N symmetric filter (c), the $N/2+1$ multipliers are replaced with a single multiplier block. Note that in all cases, there are N adders intrinsic to the filter structure, *i.e.*, those not used for multiplication.

vertex of the graph except the initial vertex represents an adder, which has two inputs, and can have any number of outputs. Each edge of the graph can be assigned a value of ($\pm 1, 2, 4, 8, \ldots$), representing multiplication of the value of the initial vertex of that edge by a value which can be implemented as a binary shift. Since adders and subtractors are treated as having equal cost, positive and negative values can be freely chosen. The example in Figure 2 is 45, the smallest integer that be represented using fewer adders than in CSD. Figure 2(a) shows $45x$ computed as $((x-4x)-16x)+64x$, using 3 "adders", *i.e.*, this (cost-3) graph has adder cost 3. This corresponds to

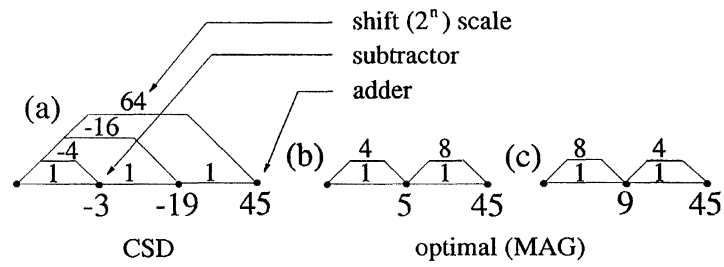


FIGURE 2 (a) CSD representation of 45, (b) and (c) 45: represented with fewer adders than for CSD.

$45 = 64 - 16 - 4 + 1$, or $10\bar{1}0\bar{1}01$ in signed-digit representation. Because there are no zero-free substrings longer than one signed-digit, this signed-digit representation is canonic [4], *i.e.*, this is a CSD graph.

Optimum (cost-2) representations, which require only two adders, are shown in Figure 2(b) and (c) – note the optimum representation is not necessarily unique. The important thing to note about these graphs is that they have a different topology from the CSD graph. Shifted versions of the newly created vertex values are used to produce the result, rather than simply adding shifted versions of the input (multiplicand), which is the basis of both binary and CSD representations.

We have used the term "fundamental" to describe the values assigned to the vertices of the graphs. The three graphs of Figure 2 have sets of fundamentals $\{-3,-19,45\}$, $\{5,45\}$ and $\{9,45\}$, respectively.

2.2. General Algorithms for Graph Design

2.2.1. General Algorithms

There are two types of algorithm that have been devised for the design of graph multipliers. Exhaustive algorithms, discussed in Section 2.3, evaluate all multipliers possible for all graph topologies. This is a very time-consuming task, and is typically done once only to produce a lookup table that includes the graphs for a given multiplier. This lookup table is also expensive in terms of memory, so in practice, the algorithms are restricted to shorter wordlengths. Due to their exhaustive nature, these algorithms produce optimal results, *i.e.*, they use fewest possible adders.

On the other hand, general algorithms, the subject of this section, are suboptimal in general. They are given the multiplier value required and design the graph with no other information. This tends to produce results relatively quickly using far less memory, so long wordlengths can be readily accounted for. The problem of producing an *optimal* general algorithm remains a target of research.

Both types of algorithm operate on positive odd integers only. Even integers can be produced with a shift, and simply replacing some adders with subtractors can produce negative integers.

2.2.2. The Bull and Horrocks Algorithm

The algorithm of Bull and Horrocks [6] was not designed to produce single multipliers, although it can be used for this purpose. Originally, it exploited redundancy in designing multiplier blocks, a problem more completely addressed in Section 3. The reader is referred to [6] for a detailed description of the algorithm, hereafter denoted "BH", and to [7] for a version modified by the author, denoted "BHM" which produced significantly improved results. The basis of the algorithm is that it starts with the input ("1") vertex, and takes all pairwise sums of powers-of-two multiples of that vertex. The sum closest to the required multiplier value is added to the graph as a vertex. This process is repeated, using powers-of-two multiples of all vertices in the graph, until the multiplier value is added to the graph.

2.2.3. *The Bernstein Algorithm*

The problem of reducing the number of adders required for a hardware integer multiplier is closely related to the problem of reducing the number of ADD, SUB and SHIFT operations required to perform an integer multiplication in software. An algorithm for this purpose was proposed by Bernstein [8], and has been used in various software compilers (*e.g.* [9, 10]). Recently, the algorithm was extended by Wu [11] to account for the LEA (shift/add) instruction of the Pentium processor.

Bernstein measured cost in instructions, where SHIFT had the same cost as ADD and SUB. For the hardware case, shifts can be wired and are thus essentially costless. Therefore the algorithm, modified to evaluate the number of adders required to produce a product by integer n is:

$$\begin{aligned}\text{Cost}(1) &= 0\\ \text{Cost}(n) &= \min(\text{Cost}(n-1), \text{Cost}(n+1),\\ &\qquad \text{Cost}(n/(2^i-1)))\\ &\quad + 1\text{Cost}(n/(2^i+1)),\end{aligned} \tag{2}$$

where i and $n/(2^i \pm 1)$ are integral. This algorithm will be referred to as "BERN".

2.2.4. *The BBB Algorithm*

It was found [12] that each of the BHM and BERN algorithms can perform better than the other for individual designs because they produce graphs of different basic topology. An example of this is shown in Figure 3 [12]. For integer 711, BERN's "product" graph is optimal, whereas for 707, BHM's "entangled" graph is optimal. Therefore, an algorithm that selects the "better of BHM and BERN" (BBB) algorithm was defined:

$$\text{BBB}(n) = \min(\text{BHM}(n), \text{BERN}(n)) \tag{3}$$

It is difficult to predict which of the BHM or BERN graph topology will be more appropriate for a given integer, so the BBB algorithm simply designs using both methods and chooses the better result.

2.2.5. *Comparison of Algorithms*

Figure 4 [7] clearly shows the advantage of the BHM algorithm over binary, CSD and the original BH algorithm. It can be seen that the BH algorithm does not produce noticeable improvements over CSD for wordlengths of less than 22 bits. The BHM algorithm always produces better average results than CSD, with a 26.6% reduction in adder

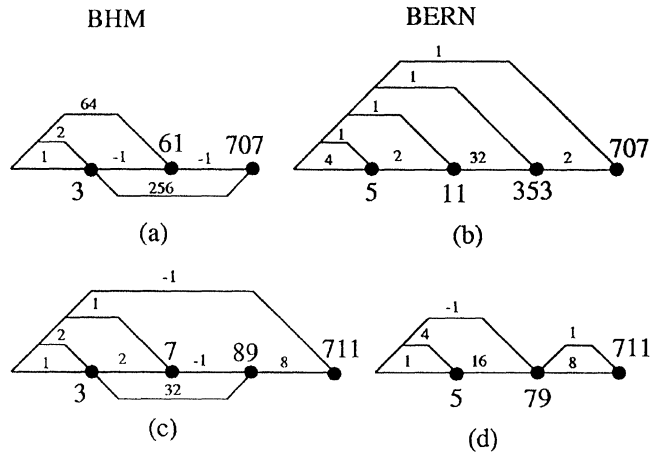


FIGURE 3 The graphs designed by BHM, (a) and (c), and BERN, (b) and (d) for integers 707 and 711. Both cost 3 designs are optimal.

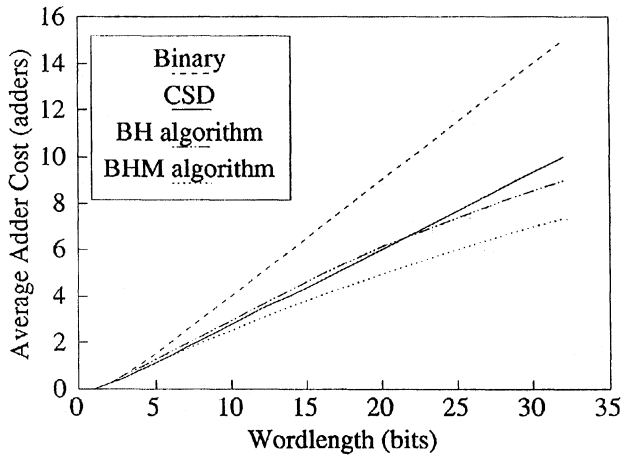


FIGURE 4 Average adder costs for multipliers designed using various techniques. For wordlengths up to 12 bits, results are exhaustive. For wordlengths above 12 bits, the averages are over 1000 uniformly distributed random integers.

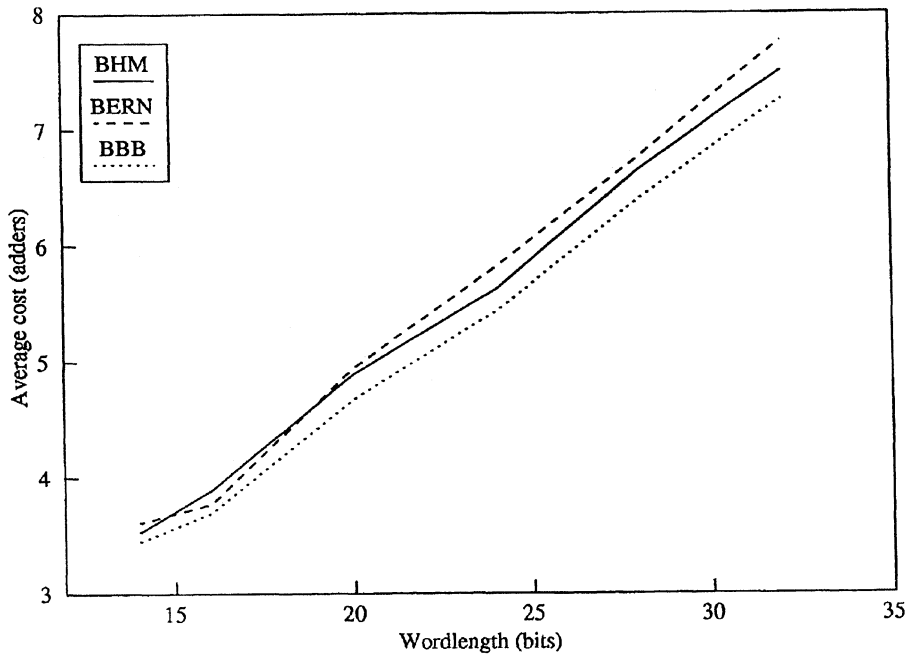


FIGURE 5 Average adder cost of 100 uniformly distributed integers of indicated wordlength, using the BHM, BERN, and BBB algorithms.

cost for 32-bit words. For wordlengths between 14 and 32, a similar comparison in Figure 5 [12] shows that despite the BERN algorithm being inferior on average, individual instances of superiority can be exploited to give a significant average cost gain for the BBB algorithm. For 32-bit words, average BERN cost is 3.5% worse than BHM cost, whereas BBB cost is 3.4% better than for BHM. Similarly, for all 12-bit words, the results in Table I indicate that the BBB algorithm provides designs much closer to optimal cost, in general, than either the BHM or BERN algorithm. The optimal costs are

TABLE I The increase in average costs of the BHM, BERN and BBB algorithms compared with the optimal costs for 12-bit integers

Algorithm	Cost increment on optimal
BHM	4.7%
BERN	4.5%
BBB	2.0%

evaluated using the MAG algorithm introduced in Section 2.3.

2.2.6. *Other Algorithms*

Because we are looking at the VLSI implementation of multiplication, we have always assumed that a shift operation is free. However, algorithms have been designed to cater for the case where shifts bear a cost (*e.g.*, the original Bernstein algorithm [8]) or are not available. If shifts and, further, subtractions, are assumed not to be available, the problem of designing the graph for multiplier n is equivalent to finding the shortest addition chain for n, a well-researched area in mathematics. An addition chain for n is defined as a sequence of integers

$$1 = a_0, a_1, a_2, \ldots, a_r = n$$

with the property that

$$a_i = a_j + a_k, \quad \text{for some } k \leq j < i,$$
$$\text{for all } i = 1, 2, \ldots, r.$$

An example of an addition chain represented by a graph is shown in Figure 6, illustrating the Fibonacci sequence. Knuth's "power trees" [13] optimally search these chains and hence could design an optimal graph with the above constraints. If subtractors are allowed, the problem becomes that of finding the shortest addition/subtraction chain [14]. Bull and Horrocks "add-only" and "add-subtract" algorithms [6] are general algorithms that perform suboptimal searches for the shortest addition and addition/subtraction chain, respectively. Graphs such as we have seen earlier are produced, but there is no scaling of the graph edges, and necessarily require more adders.

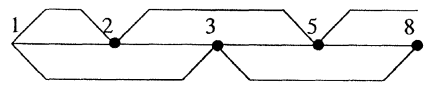


FIGURE 6 The graph to produce the Fibonacci sequence. Note that edges are not labelled – each represents a scaling of 1. This is an example of an addition chain.

The context in which Knuth introduces his "power trees" [13] is not, in fact, efficient multiplication, but efficient *exponentiation*, *i.e.*, the best way to raise an input x to a power n. This problem, which involves the best choice of *square* and *multiply* operations is a direct analogy to our choice of *shift* and *add* operations. For instance, to calculate x^{11} (note $11 = 1011$ in binary), square x, square the result, multiply by x, square, multiply by x, giving, sequentially, x^2, x^4, x^5, x^{10} and x^{11}. The algorithmic procedure is simply to replace the binary digits "1" (excepting the leading "1") with "square, multiply" and "0" with "square". Almost identically, a multiplication by 11 can be achieved by replacing binary "1" with "shift-add" and binary "0" with "shift". Thus $11x$ could be evaluated as shift x, shift the result, add x, shift, add x, giving, sequentially, $2x$, $4x$, $5x$, $10x$, $11x$. Note the product is built up identically to the power of x in the previous example. This method, when used for multiplication is almost 4000 years old, and is over 2000 years old for exponentiation [13].

Efficient exponentiation is a subject of research interest because the popular RSA cryptographic algorithm [15] requires the calculation of $m^e \pmod N$ where m and N are very large integers. Any of the algorithms already discussed could be used to perform this task, but when e is large, heuristic techniques have been used. These require certain powers to be pre-computed. The analogy with the graphical method is to force certain fundamentals into the graph prior to the graph design. Two algorithms, the k-SR algorithm [16] and the SS(1) algorithm [17] trade off the amount of work in this pre-computation (and more importantly the memory space that this requires, as the pre-computed results are stored in a look-up table) with

the speed with which the rest of the exponentiation task is completed, measured as the number of multiplications required. They could be considered useful for designing extremely long wordlength shift-and-add multipliers. For these long wordlength multipliers, it was recently shown [18] the number of adders required for a multiplication by n is $O(n/(\log n)^{\alpha})$ for any α.

2.3. Exhaustive Algorithms for Graph Design

2.3.1. Requirements

There are three important areas of consideration that ensure that an algorithm has searched all possible graph configurations:

1. All possible graph topologies are searched,
2. All possible vertex values (fundamentals) are accounted for, and
3. All possible edge values are accounted for.

These requirements place different restrictions on the algorithm. For graphs up to cost-4, all of the graphs of Figure 7 [7] must be searched. A recent algorithm presented by Li [19], which claims to be optimal does not search all of these graphs [20] and is therefore not truly exhaustive. However, because it exhaustively searches the graphs that it does recognise, it can be considered to be an algorithm of the exhaustive type. Vertex value theorems in [7] show that we need only search using positive, odd fundamentals, simplifying the search significantly. Edge value theorems in [7] indicate that there is a limit to the size of integers that can appear as edge values and fundamentals, thus ensuring that the algorithm is exhaustive and can be executed in finite time. These considerations led to the design of the MAG algorithm.

2.3.2. The Minimised Adder Graph (MAG) Algorithm

The operation of the minimised adder graph (MAG) algorithm is described in detail in [7]. Here it need only to be said that it exhaustively searches

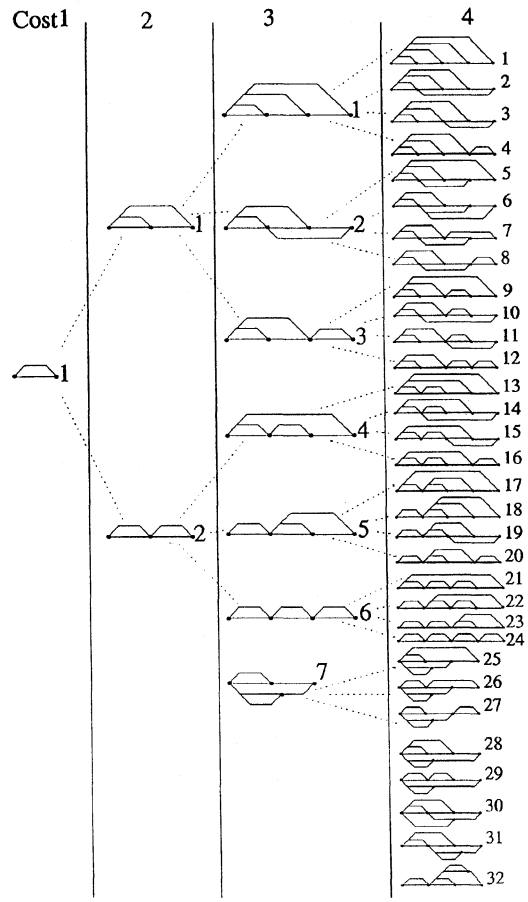


FIGURE 7 The possible graph topologies for costs 1 to 4 (Dotted lines indicate building of higher cost graphs from lower cost graphs).

all the graphs of Figure 7. The algorithm produces two lookup tables, one with the cost of the multiplier and the other with a record of the fundamentals of all the graphs used to produce the multiplier at that cost. This second "fundamentals" table grows exceedingly quickly with wordlength, and the capability of the machine used to produce the results limited the extent of those results to 12-bit wordlengths. The results of applying the algorithm to all integers up to 2^{12} are shown in Figure 8 [7]. In general, binary and CSD implementations are limited to the graph topology labelled "1" in Figure 7 for each cost, although tree structures such as cost-3 graph 7 have

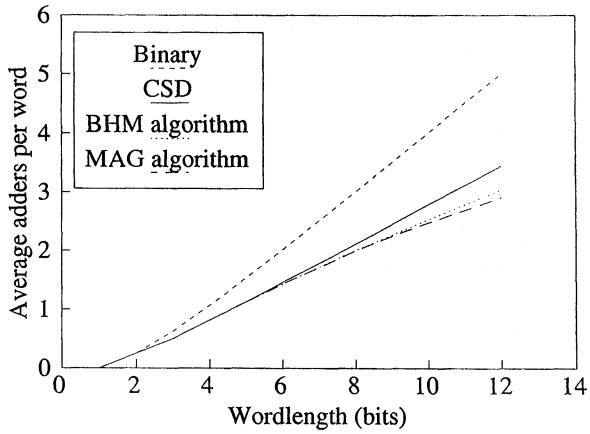


FIGURE 8 Average number of adders required against wordlength in bits for single integer multipliers.

been proposed [21] to minimise propagation delay. Not restricted by these limitations, the BHM and MAG algorithms are shown in Figure 8 to be clearly less costly. Although the advantage of MAG over BHM is only about 5%, this advantage is expected to improve for longer wordlengths.

3. MULTIPLIER BLOCKS

Multiplier blocks produce several products of a single multiplicand by exploiting redundancy in the multiplication process. For example, if the multipliers 5 and 45 are required, Figure 2b could be used to produce the multiplication by 45, with the multiplication by 5 as a "free" by-product (pun intended).

3.1. Design Algorithms

3.1.1. The Bull and Horrocks Algorithm

The algorithm mentioned earlier in Section 2.2.2 was originally designed for the multiplier block application. Again, in the multiplier block context, we denote the algorithm "BH". Modifications to the algorithm for multiplier blocks are described in [22], and the modified algorithm is again denoted "BHM".

3.1.2. The n-dimensional Reduced Adder Graph (RAG-n) Algorithm

The n-dimensional reduced adder graph (RAG-n) algorithm is currently the best algorithm for designing short-wordlength multiplier blocks. It is in two parts; the first is optimal, *i.e.*, if the set of coefficients is completely synthesised by this part of the algorithm, minimum adder cost is assured, and the second part is heuristic. It uses the two lookup tables generated by the MAG algorithm, which, at present, cover the range 1 to 4096.

The algorithm is described in detail in [22] and operates roughly as follows. A set of coefficients are input to the algorithm, and the graph is built up in stages. First, the cost-1 coefficients are added to the graph. Power-of-two multiples of those fundamentals in the graph are then added together and if another coefficient is produced, it is added to the graph and the process is repeated until no further coefficients can be added to the graph. If the whole coefficient set is synthesised by this part of the algorithm, the resulting graph is guaranteed to be optimal. If there are still some coefficients left to synthesise, heuristic methods are used using the MAG algorithm fundamentals lookup table to try and select the best coefficient to add next. Under certain conditions, the result of this process is also optimal. A hybrid algorithm has also been defined, which replaces this final heuristic stage with the BHM algorithm in order to increase speed.

3.1.3. Other "Multiplier Block" Algorithms

Nakayama's permuted differences [23], the sub-expression elimination techniques of Hartley [24, 25] and Potkonjak *et al.* [26], and the nested structures of Mahanta *et al.* [27] have been shown to produce structures that can be represented by particular types of multiplier block graphs. However, these methods have been shown [28] to be far less versatile than the two mentioned above and therefore need not be discussed further here.

The author has also defined an optimal algorithm for the design of 2-coefficient multiplier

blocks, known as MAG2 [28]. The computation time of this algorithm increases factorially, so exhaustive results have only been calculated up to 8 bits, where MAG2 produces a 27% average reduction in adder cost over CSD coefficients. For a pair of coefficients of the same wordlength, application of BHM results in a 21% reduction and RAG-*n* a 24% reduction.

3.2. RAG-*n* and BHM Performance

In the experiments used to test the performance of the algorithms, *uniformly* distributed random coefficients were used. The non-uniform distribution of coefficients in typical FIR filters leads to even better results [22]. For set sizes (numbers of coefficients) of 3, 5, 7, 10, 15, 25, 40 and 80, one hundred uniformly distributed random sets of coefficients were costed for even wordlengths up to 12 bits. The average adder cost is shown in Figure 9, where it can be seen that for a given wordlength, average adder cost increases roughly linearly with set size. A set of 80 coefficients of 12-bit wordlength requires fewer than one adder per coefficient on average. For the smaller wordlengths in the figure, an asymptote is reached which is the cost of the graph that can fully represent all of the coefficients of that wordlength. The value of this asymptote is the number of odd integers of wordlength w, *i.e.*, 2^{w-1}. Once this asymptote is reached, any "new" coefficient is simply a repetition of a coefficient already in the set.

For a set size of 5 (the number of coefficients that would be required for the implementation of linear phase FIR filters of order 9 or 10), the average adder cost of a multiplier block for the BH, BHM, hybrid and RAG-*n* algorithms, were compared over a range of wordlengths, as shown in Figure 10. Comparisons with individual multipliers using CSD and binary are also shown. The RAG-*n* algorithm provides a significant improvement in cost over BHM (8.4% for 12 bit words), which in turn provides a significant improvement (10.6%) over the original BH method. All the algorithms that utilise graph synthesis techniques are far superior in terms of adder cost to CSD and binary.

The computation time of these multiplier block design algorithms is an interesting subject and has been discussed at some length in [22, 28]. The optimal part of the RAG-*n* algorithm is actually very quick while the heuristic part is slow. It was also found in [22] that for a given wordlength, there is an "optimality threshold" set size above

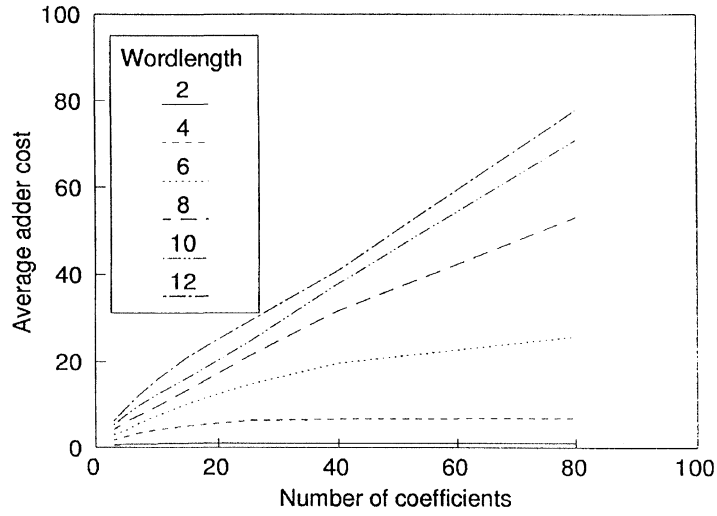


FIGURE 9 Average adder costs for the RAG-*n* algorithm for various wordlengths against uniformly-distributed coefficient set size.

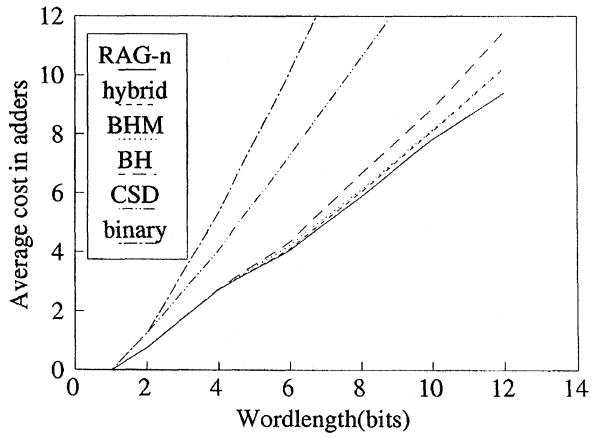


FIGURE 10 Average cost in adders evaluated for various algorithms against wordlength. Each point represents the average over one hundred uniformly distributed 5-coefficient sets.

which it is highly likely that the design has optimal cost. These are counterintuitive results, since given the potential explosive growth of complexity of the problem with set size (the problem is NP-complete [6]), optimality might seem to be less likely for large set sizes. The explanation is that the number of optimal solutions grows quickly and they are easy to find.

3.3. Application of Multiplier Blocks to Digital Filters

3.3.1. Cost Reductions

We have used multiplier blocks in the design of both finite impulse response (FIR) and infinite impulse response (IIR) filters. We measured the adder cost of these filters to be the number of the adders in the multipliers plus the adder cost of the structural elements, the adders and delays. Due to the more efficient multiplier implementation, the proportion of the total adder cost due to the multipliers is drastically reduced. In the examples examined, this proportion dropped approximately from 40% to 20% for FIR filters [22] and 80% to 50% for IIR filters [28, 29]. This has a number of important implications. In the past, the emphasis on reducing the complexity of a filter has focused on the multipliers. Multiplier blocks have been so successful in reducing that cost that there is incrementally less to be achieved by further attention to reducing multiplier complexity. In other words, elaborate schemes which select the coefficients in some "optimal" way (*e.g.* [30]) may not offer significant savings over a technique which selects the coefficients in a straightforward fashion and implements them as a multiplier block.

It is important to note that multiplier blocks are applied directly to a selected set of coefficients, and the complexity savings they offer are limited to that application. Methods that select simple coefficients can be used in conjunction with multiplier blocks, such as statistical wordlength minimisation as described by Crochiere [31] for IIR filters, Grenez [32] for FIR filters, and the author [33, 34] for average wordlengths. There are many techniques that are aimed at reducing filter wordlength (see the reference list for [22]) which can all be used in conjunction with multiplier blocks.

The multiplier block method does not in itself attempt to minimise the number of adders that are required to meet a given filter specification; instead it aims to minimise the number of adders required to produce the products for a given set of coefficients. Some methods have been described that attempt to minimise the number of adders in the filter directly. The earliest technique of this type, described by Jain *et al.* [35, 36], aimed to minimise the number of CSD "bits" required by the coefficients (without using redundancy between the coefficient multipliers). Another method, of Wade *et al.* [37, 38], tries to reduce the number of adders in a cascade of primitive sections that meets the filter specification. The cost functions associated with this type of optimisation are badly behaved so non-gradient searches such as genetic algorithms have been devised for this task by Roberts [39] and Suckley [40] and for the relationship between this adder cost function and the filter error specification by Wilson and Macleod [41]. These various optimisation algorithms could be

modified to operate with multiplier blocks producing the cost function. However, this cost function has been shown to be relatively flat in a local region [28, 29], providing further discouragement for optimisation in addition to the reasons discussed earlier in this section.

3.3.2. The Complexity Hierarchy

Multiplier blocks apply where several products of a single multiplicand are required. The larger the block, the more that the cost of the multipliers can be reduced. Therefore, structures that allow the use of large blocks, such as direct-form FIR and IIR filters, can be expected to gain more from using multiplier blocks than structures with isolated multipliers, such as the lattice wave structure. In fact, early results [42, 43] showed that using multiplier blocks can make the direct form structure more efficient than the wave structure! In these studies, we found that whereas traditional methods favour the lattice wave structure for filters, multiplier block implementation so drastically reduces the cost of cascaded second-order forms that they become significantly less costly. The cost of the direct structure is reduced to less than that of the wave structure, despite having more coefficients and requiring a much longer coefficient wordlength. Even when the data wordlength noise effects are taken into account [28], the direct structure is still competitive with the wave structure. However, the direct form has always had poor limit-cycle (instability due to nonlinear feedback) performance and a more recent study [44] shows that although cascaded second-order forms still outperforms the lattice wave structure when limit cycles are eliminated, the direct form no longer competes.

3.3.3. The Order-complexity Trade-off

In Section 3.3.1, reference was made to the flattening of the cost function in coefficient space due to the use of multiplier blocks. This means that the total cost of a set of coefficients in a block does not vary very much if the values of the coefficients are varied in value. This slow variation in cost has also been observed when the number of coefficients in the block is varied, corresponding for example to a variation in filter order for an FIR filter. This slow variation means that there may be an incentive to increase the order of the filter in order to reduce the complexity. Studies of both FIR [45] and IIR [28] filters indeed show that there is an incentive to increasing the order and that multiplier blocks flatten the cost of FIR filters such that any order up to 10% above the estimate produced by the usual order estimators may produce the most efficient design.

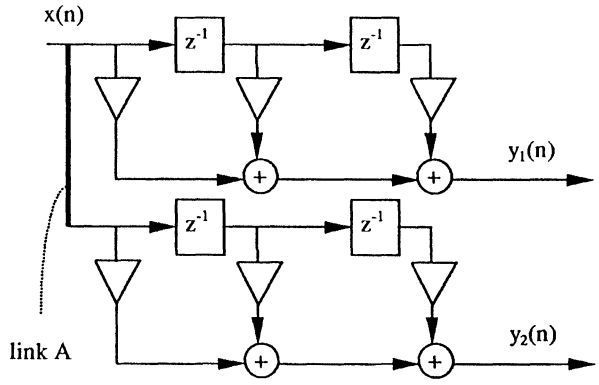
3.4. Comparison with Other Efficient Filter Design Methods

We applied multiplier blocks to some of the filters published as examples of advanced techniques of designing low-complexity filters. These methods include Powell and Chau's CSD delta-modulation of the coefficients [46] and the cascade of primitive sections due to Wade *et al.* [37, 38]. For the multiplier block filters used in the comparison, the output of the Remez exchange algorithm design was simply quantised prior to application of multiplier blocks, *i.e.*, no special technique was used to select the coefficients. For all the examples, it was shown [28, 47] that multiplier blocks produced a filter that required fewer adders. Where a recursive running sum (RRS) prefilter was not used, the cost of the multiplier block filter ranged from 48% to 78% of the cost of the other design. Where an RRS prefilter was used, the advantage of the multiplier block design was less significant.

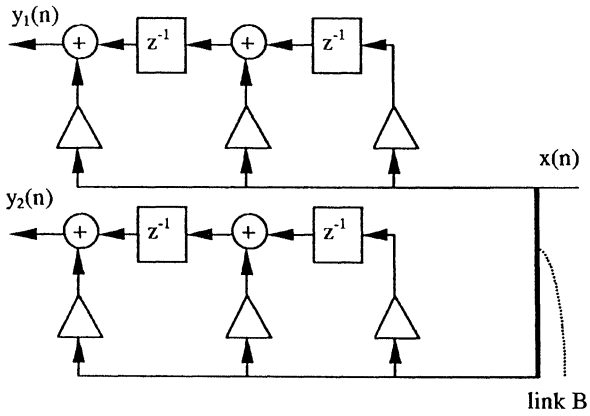
Jones [48] has proposed a distributed arithmetic method, which extends the idea originally described by Peled and Liu [49]. He shows that this method requires more adders than the Bull and Horrocks method [6], and also uses RAM, ROM and control circuitry. It is therefore also less efficient than the RAG-*n* algorithm, but it is not coefficient-dependent.

3.5. Multiplier Blocks and Filter Banks

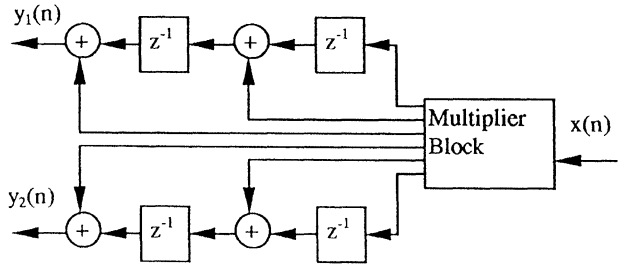
Filter banks (parallel connections of digital filters) are used in many signal processing applications including design of analysis and synthesis filters for multirate signal processing [50], time-frequency analysis [51], wavelets [52] and for fractional delay filter design [53]. Figure 11 [54] shows that for a simple filter bank of two second-order FIR filters, all of the coefficients in the filter bank can be



(a) Filter bank



(b) Filter bank: transposed



(c) Filter bank: all multipliers in one block

FIGURE 11 How to incorporate all coefficients of an FIR filter bank into a single block.

incorporated into a single block. Figure 11a is the usual direct-form interconnection of the filter bank. If we consider the filter banks as separate filters, each accepting the same input (*i.e.*, remove link A), and then transpose this structure, we get the structure in Figure 11b (without link B). With link B in place, however, we see that all of the coefficients multiply a single data input, and can be placed in a single multiplier block as in Figure 11c.

We have examined the application of multiplier blocks to filter banks and found [54] that once again, costs of the multiplier elements can be reduced significantly. Hence, the cost of the structural components becomes even more significant than if each filter was built separately. In designing a filter bank, there are a variety of structures that can be used, including, for the interpolation application we examined, the Farrow structure [53]. If multiplier blocks are used for multiplication, this choice of structure then dominates the overall cost of the filter bank.

4. MATRIX MULTIPLICATION

The problem of performing matrix multiplication using graphical techniques increases the complexity of the problem by one dimension. If the single multiplier case of Section 2 has zero dimensionality, and the single-input, multiple-output case of Section 3 has dimension 1 (a vector multiplied by a scalar), then the multiple-input, multiple-output task of multiplying a matrix by a vector has dimension 2. The algorithms described already can be used to design the multipliers, but there is no guarantee of an optimal result. The multiple-input nature of the problem means that an optimal graph will be even more "entangled" than some

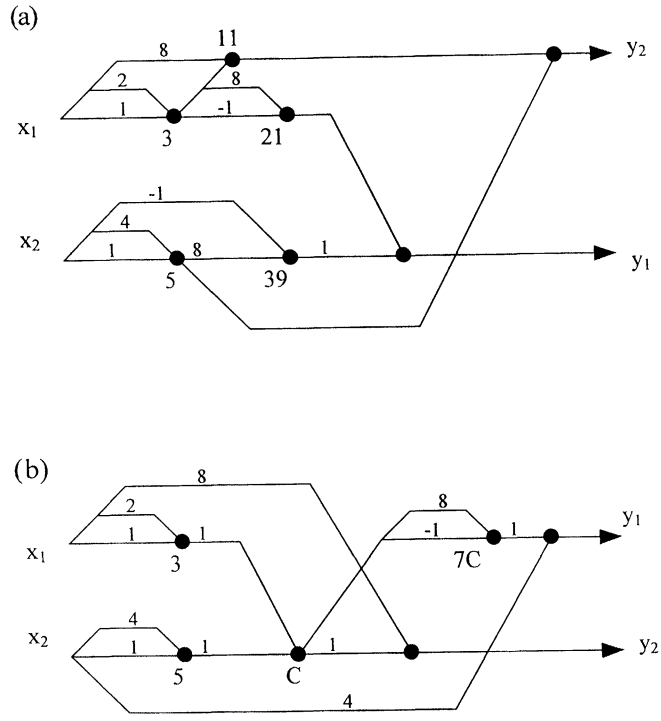


FIGURE 12 Implementing a matrix multiplication using graphs. (a) Using the RAG-n design and combining separate outputs requires 8 adders. (b) A better method, using only 6 adders, that uses an intermediate vertex C ($=3x_1+5x_2$) that uses both inputs and feeds both outputs.

of the complex-looking graphs produced by RAG-*n* or BHM. Take, for example, the matrix equation:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 21 & 39 \\ 11 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Using the RAG-*n* or BHM algorithm to design the matrix multiplier would result in the structure of Figure 12a, where the various products of the inputs are produced and then combined at the end. This method uses 7 adders. A more efficient graph is shown in Figure 12b, which requires only 6 adders. The outputs are synthesised using the equations

$$y_1 = 7(3x_1 + 5x_2) + 4x_2$$
$$y_2 = 3x_1 + 5x_2 + 8x_1$$

using an intermediate result, $C = 3x_1 + 5x_2$, which uses both inputs and supplies both outputs. It would appear that algorithms of the type used for the 0- and 1-dimensional designs are not appropriate for matrix multiplication. The search for an appropriate algorithm remains the subject of ongoing research.

5. DISCUSSION AND CONCLUSIONS

Graph multipliers and multiplier blocks have many advantages as we have discussed above. However, there are also some limitations that affect their application. First, they are only of use where constant multipliers are required. If the filter coefficients need to be programmable or variable, another technique should be used. Second, when synthesised, they do not produce regular structures. It is believed that the gains they make will outweigh the inefficiency due to irregular layout, but this conjecture has yet to be tested. The comparisons made herein and throughout this work have been at an adder level in an attempt to make the comparisons as independent of technology as possible. Technology-dependent comparisons will be explored in the near future. These comparisons will extend to serial arithmetic, where all the comparisons here effectively apply to parallel arithmetic. Third, the products produced from a multiplier block do not necessarily have the same latency, so for pipelined applications, extra pipelining registers will be required.

To summarise the findings of the multiplier work:

1. For single coefficients, the MAG algorithm guarantees minimum adders for a given multiplier. Due to memory use, the MAG algorithm has been limited to a given wordlength. Above this wordlength, the BBB algorithm, the better of BHM and BERN, is the best available. For extremely long wordlengths, the exponentiation algorithms k-SR and SS(1) are worth considering. In addition to the VLSI application of primary interest here, all of these algorithms can also be used for reducing the number of ADD (and SHIFT) instructions a software compiler assigns to a multiply, and may assist in reducing the exponentiation overhead in cryptograhic algorithms.
2. When multipliers can be blocked, *i.e.*, where several products of a single multiplicand are required, the RAG-*n* algorithm is the best. It is often optimal, but its use of the MAG algorithm also limits its maximum wordlength. The BHM algorithm is the best to use above that wordlength. These algorithms design filters that are more efficient in terms of adders than any other method to which they have been compared.
3. For both FIR and IIR filters, the use of multiplier blocks substantially reduces the contribution to overall complexity made by the multipliers, reducing the imperative to optimise the multiplier contribution. The remaining elements (adders and delays) are intrinsic to the structure of the filter and cannot be optimised. Attention must then turn to the selection of structure and order.
4. This choice of structure should not be made without examination of the effects of the use of

multiplier blocks. Without the use of multiplier blocks, wave structures are the most efficient choice. Application of multiplier blocks so dramatically reduces the cost of cascade structures that they are then least costly.

References

[1] Booth, A. D. (1951). "A signed binary multiplication technique", *Quarterly Journal of Mechanical and Applied Mathematics*, **4**(2), 236–240.

[2] Algirdas Avizienis, "Signed-digit representation for fast parallel arithmetic", *IRE Trans. Electronic Computers*, **10**, 389–400, Sep., 1961.

[3] George W. Reitweisner (1960). "Binary Arithmetic", *Advances in Computers*, **1**, 232–308.

[4] Kai Hwang, "*Computer Arithmetic: Principles, Architecture and Design*", Wiley, 1979.

[5] Harvey L. Garner (1965). "Number Systems and Arithmetic", *Advances in Computers*, **6**, 131–194.

[6] Bull, D. R. and Horrocks, D. H. (1991). "Primitive operator digital filters", *IEE Proceedings G*, **138**(3), 401–412.

[7] Dempster, A. G. and Macleod, M. D., "Constant integer multiplication using minimum adders", *IEE Proceedings – Circuits, Devices and Systems*, **141**(5), 407–413, Oct., 1994.

[8] Robert Bernstein, "Multiplication by Integer Constants", *Software-Practice and Experience*, **16**(7), 641–652, Jul., 1986.

[9] Bacon, D. F., Graham, S. L. and Sharp, O. J. (1994). "Compiler Transformations for High-Performance Computing", *ACM Computing Surveys*, **26**(4), 345–420.

[10] Wall, D. W. (1992). "Experience with a software-defined machine architecture, "*ACM Transactions on Programming Languages and Systems*", **14**(3), 299–338.

[11] Wu, Y. F. (1995). "Strength Reduction of multiplications by integer constants", *SIGPLAN Notices*, **30**(2), 42–48.

[12] Dempster, A. G. and Macleod, M. D., "General algorithms for reduced-adder integer multiplier design", *Electronics Letters*, **31**(21), 1800–1802, Oct., 1995.

[13] Donald E. Knuth, "*The Art of Computer Programming*", Vol. 2, 2nd edn., Addison-Wesley, 1981.

[14] Hugo Volger, "Some results on addition/subtraction chains", *Information Processing Letters*, **20**, 155–160, Apr., 1985.

[15] Rivest, R. L., Shamir, A. and Adleman, L. (1978). "A method for obtaining digital signatures and public-key cryptosystems", *Communications of the ACM*, **21**, 120–126.

[16] Mitchell, C. J., "*Another improvement to square-and-multiply exponentiation*", Technical report CSD-TR-93-cc, Royal Holloway, University of London, Aug., 1993.

[17] Hui, L. C. K. and Lam, K.-Y., "Fast square-and-multiply exponentiation for RSA", *Electronics Letters*, **30**(17), 1396–1397, Aug., 1994.

[18] Pinch, R. G. E., "Asymptotic upper bound for multiplier design", *Electronics Letters*, **32**(5), 420–421, Feb., 1996.

[19] Dongning Li, "Minimum number of adders for implementing a multiplier and its application to the design of multiplierless digital filters", *IEEE Trans. Circuits and Systems II*, **42**(7), 453–460, July, 1995.

[20] Dempster, A. G. and Macleod, M. D., "Comments on 'Minimum number of adders for implementing a multiplier and its application to the design of multiplier-less digital filters", *IEEE Trans. Circuits and Systems II*, **45**(2), 242–243, Feb., 1998.

[21] Kingsbury, N. G. (1971). "High-speed binary multiplier", *Electronics Letters*, **7**(10), 277–278.

[22] Dempster, A. G. and Macleod, M. D., "Use of minimum-adder multiplier blocks in FIR digital filters", *IEEE Trans. Circuits and Systems II*, **42**(9), 569–577, September, 1995.

[23] Kenji Nakayama, "Permuted difference coefficient realisation of FIR digital filters", *IEEE Trans. ASSP*, **30**(2), 269–278, Apr., 1982.

[24] Richard Hartley (1991). "Optimization of canonic signed digit multipliers for filter design", In: *ISCAS91*, pp. 1992–1995.

[25] Hartley, R. I. (1996). "Subexpression Sharing in Filters Using Canonic Signed-Digit Multipliers", *IEEE Trans. Circuits and Systems II*, **43**(10), 677–688.

[26] Potkonjak, M., Srivastava, M. B. and Chandrakasan, A. (1994). "Efficient substitution of multiple constant multiplications by shifts and additions using iterative pairwise matching", In: *Proc. 31st ACM/IEEE Design Automation Conference*, pp. 189–194.

[27] Anil Mahanta, Ramesh C. Agarwal and Suhash C. Dutta Roy, "FIR filter structures having low sensitivity and roundoff noise", *IEEE Trans. ASSP*, **30**(6), 930–937, Dec., 1982.

[28] Andrew G. Dempster, "Digital Filter Design for Low-Complexity Implementation", *Ph.D. Thesis*, Cambridge University, June, 1995.

[29] Dempster, A. G. and Macleod, M. D., "IIR digital filter design using minimum-adder multiplier blocks", *IEEE Trans. Circuits and Systems II*, **45**(6), 761–763, June, 1998.

[30] Christakis Charalambous and Michael J. Best (1974). "Optimization of Recursive Digital Filters with Finite Word Lengths", *IEEE Trans. ASSP*, **22**(6), 424–431, Dec., 1974.

[31] Crochiere, R. E., "A new statistical approach to the coefficient wordlength problem for digital filters", *IEEE Trans. Circuits and Systems*, **22**, 190–196, Mar., 1975.

[32] Grenez, F., "Design of FIR linear phase digital filters to minimise the statistical wordlength of the coefficients", *IEE J. Electronic Circuits and Systems*, **1**(5), 181–185, Sep., 1977.

[33] Dempster, A. G. and Macleod, M. D., "Variable statistical wordlength in digital filters", *IEE Proc: Vision, Image and Signal Processing*, **143**(1), 62–66, Feb., 1996.

[34] Dempster, A. G. and Macleod, M. D., "Variable Wordlengths in Filters with Least-Squares Error Criterion", *Electronics Letters*, **33**(3), 201–202, Jan., 1997.

[35] Jain, R., Vandewalle, J. and De Man, H. (1984). "Efficient CAD tools for the coefficient optimisation of arbitrary integrated digital filters", In: *ICASSP84*, pp. 30.11.1–4.

[36] Rajeev Jain, Joos Vandewalle and Hugo De Man, "Efficient and accurate multiparameter analysis of linear digital filters using a multivariable feedback representation", *IEEE Trans. Circuits and Systems*, **32**(3), 225–235, Mar., 1985.

[37] Wade, G., Van_Eetvelt, P. and Darwen, H., "Synthesis of efficient low-order FIR filters from primitive sections", *IEE Proceedings G*, **137**(5), 367–372, Oct., 1990.

[38] Wade, G., Roberts, A. and Williams, G., "Multiplier-less FIR filter design using agenetic algorithm", *IEE Proceedings – Vision, Image and Signal Processing*, **141**(3), 175–180, Jun., 1994.

[39] Roberts, A. and Wade, G., "A structured GA for FIR filter design", In: *Natural Algorithms in Signal Processing Workshop*, Chelmsford Essex, Nov., 1993, IEE.

[40] Suckley, D., "Genetic algorithm in the design of digital filters", *IEE Proceedings G*, **138**(2), 234–238, Apr., 1991.
[41] Wilson, P. B. and Macleod, M. D., "Low implementation cost IIR digital filter design using genetic algorithms", In: *Natural Algorithms in Signal Processing Workshop*, Chelmsford Essex, Nov., 1993, IEE.
[42] Dempster, A. G. and Macleod, M. D., "Multiplier blocks and the complexity of IIR structures", *Electronics Letters*, **30**(22), 1841–1842, Oct., 1994.
[43] Dempster, A. G. and Macleod, M. D., "Comparison of IIR filter structure complexities using multiplier blocks", In: *ISCAS95*, **2**, 858–861, IEEE.
[44] Dempster, A. G., "The Cost of Limit-Cycle Elimination in IIR Digital Filters Using Multiplier Blocks", *Proc. International Symposium on Circuits and Systems (ISCAS97)*, **4**, 2204–2207, June, 1997.
[45] Dempster, A. G. and Macleod, M. D., "Variation of FIR filter complexity with order", In: *MWSCAS95*, Rio de Janeiro, IEEE, Aug., 1995.
[46] Scott R. Powell and Paul M. Chau, "Efficient narrowband FIR and IFIR filters based on powers-of-two sigm-delta coefficient truncation", *IEEE Trans. Circuits and Systems II*, **41**(8), 497–505, Aug., 1994.
[47] Dempster, A. G. and Macleod, M. D., "Comparison of fixed point FIR filter design techniques", *IEEE Trans. Circuits and Systems II – Digital and Analog Signal Processing*, **44**(7), 591–593, July, 1997.
[48] Douglas L. Jones, "Efficient computation of time-varying and adaptive filters", *IEEE Trans. Signal Processing*, **41**(3), 1077–1086, Mar., 1993.
[49] Abraham Peled and Bede Liu, "A new hardware realization of digital filters", *IEEE Trans. ASSP*, **22**(6), 456–462, Dec., 1974.
[50] Vaidyanathan, P. P., "*Multirate Systems and Filter Banks*", Prentice-Hall, 1993.
[51] Cohen, L., "*Time-frequency Analysis*", Prentice-Hall, 1995.
[52] Rioul, O. and Vitterli, M., "Wavelets and Signal Processing", *IEEE SP Magazine*, pp. 14–38, Oct., 1991.
[53] Farrow, C. W., "A Continuously Variable Digital Delay Element", *Proc. International Symposium on Circuits and Systems (ISCAS88)*, pp. 2641–2645.
[54] Dempster, A. G. and Murphy, N. P., "Efficient Interpolators and Filter Banks using Multiplier Blocks", *IEEE Trans. Signal Processing*, **48**(1), 257–261, Jan., 2000.

Authors' Biography

Andrew Dempster received the BE and M.Eng.Sc degrees from the University of New South Wales in Sydney and Canberra in 1984 and 1992 respectively. He received the Ph.D. from the University of Cambridge in 1995. He worked for eight years as a design engineer and project manager for STC in Sydney and Auspace Limited in Canberra. Since 1995 he has been with the Department of Electronic Systems at the University of Westminster in London. His research interests are in the areas of low-complexity arithmetic, especially for digital filters, image processing and satellite navigation.