Circuits Syst Signal Process (2007) 26: 793–827
DOI 10.1007/s00034-007-9005-8

Reconfigurable Multiplier Blocks: Structures, Algorithm and Applications

Süleyman Sırrı Demirsoy · Izzet Kale · Andrew Dempster

Received: 19 February 2005 / Revised: 16 September 2006 / Published online: 5 January 2008

Abstract This paper presents the efficient design methodology and applications of reconfigurable multiplier blocks (ReMB). ReMB offers significant area, delay and possibly power reduction in time-multiplexed implementation of multiple constant multiplications in many application areas from fixed digital filters, adaptive filters, and filter banks to DFT, FFT and DCT. The reader will be exposed to the fundamental principles of ReMB structures coupled with a novel algorithm for their design as well as illustrative examples where appropriate that help the reader understand the technique in action. The paper also looks into the pros and cons of deploying the technique on standard FPGA platforms as well as discussing the effectiveness of the ReMB approach in custom silicon realization by means of application examples. Area, delay and power (where possible) of the ReMB designs are compared to standard implementations.

Keywords Reconfigurable multiplier blocks · Time-multiplexed multiple constant multiplications · Reduced complexity fixed multipliers · Multiplier blocks

1 Introduction

The reconfigurable multiplier blocks (ReMB) technology has been developed to efficiently implement multiple constant multiplications in time-multiplexed filters and filter banks. Its theory, applications and automated design methodology have been described and discussed in this paper.

S.S. Demirsoy now with Altera European Technology Centre, High Wycombe, UK.
A. Dempster now with the School of Surveying and Spatial Information Systems, University of New South Wales, Sydney, Australia.

S.S. Demirsoy (✉) · I. Kale · A. Dempster
Applied DSP and VLSI Research Group, University of Westminster, London, W1W 6UW, UK
e-mail: sdemirso@altera.com

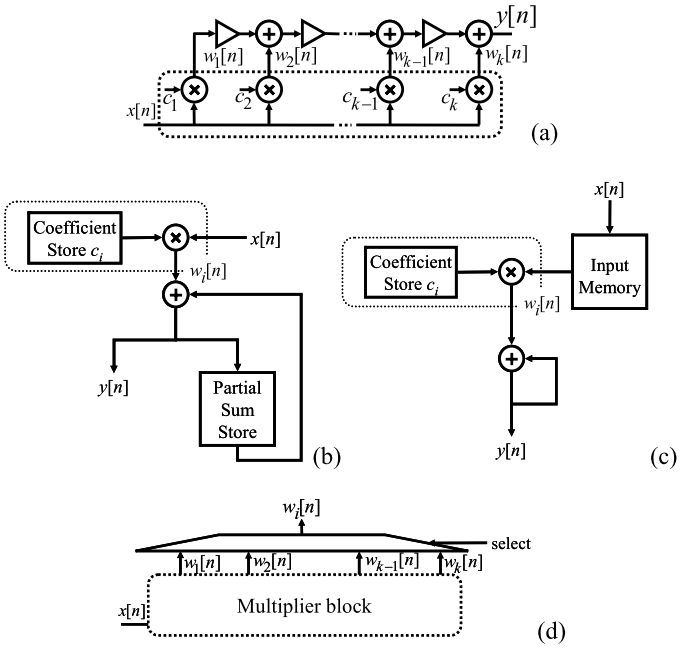


Fig. 1 **a** Fully parallel FIR time delay and accumulate (TDA) filter implementation, **b** time-multiplexed TDA filter implementation, **c** time-multiplexed tapped delay line (TDL) filter implementation, **d** substitute of general-purpose multiplier in (**b**) and (**c**) as a multiplier block with a multiplexer

In performance critical systems, concurrency of the different modules is desired to achieve higher throughput, resulting generally in the parallel implementation of the filters as separate modules. This is also the case when power is of concern, as the frequency of operation is directly proportional to the system's dynamic power consumption.

The primitive operator filter technique, as it was originally named in [3] (later, the technique was named as multiplier blocks or multiple constant multiplications) demonstrated its worth and performance mainly in the fully parallel digital filter implementations where all coefficient-sample multiplications were performed concurrently as displayed in Fig. 1(a). Instead of implementing each of the constant multiplications separately, these techniques treat the collection altogether (as a multiplier block) and realize the multiplication by means of adds, subtracts and shifts. The dotted lines in Fig. 1(a) show where a multiplier block might be applied. There are well-established techniques in the literature to deal with the redundancy that exists in the implementation of multiple constant multiplications or multiplier blocks. These techniques either use a numerical (graphical) approach where a group of coefficient products are generated using common intermediate products [3–16] or the common sub-expression elimination method that works on the signed digit (SD) representations of a group of coefficients [2–21]. Significant savings on area, speed, power and complexity have been reported using these techniques [3–14].

Multiplier blocks can also be used as a substitute to a general-purpose multiplier plus the coefficient memory in a time-multiplexed filter (Fig. 1(b) and (c)) as shown

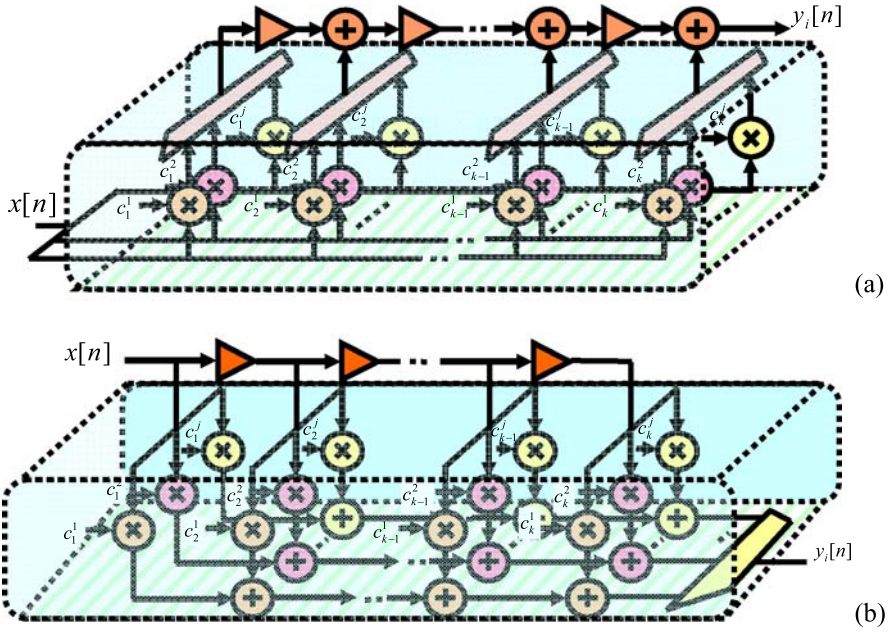


Fig. 2 Filter banks for multiplexed operation. **a** TDA form, **b** TDL form

in Fig. 1(d) for complexity and area reduction, as long as the total area of the multiplier block and the multiplexer is less than the area of the general-purpose multiplier and the coefficient memory. Instead of addressing the coefficient memory to select a different coefficient in Fig. 1(b) and (c), a different product is selected through the multiplexer in Fig. 1(d). When applied to Fig. 1(b) and (c), only one of the products that is generated by the multiplier block is required at any one time, even though the multiplier block generates all the products. The unused products incur redundancy and unnecessary power consumption.

The same kind of redundancy also occurs when using multiplier blocks in the fully parallel implementation of filter banks as shown in Fig. 2(a) and (b). These structures would be required for applications where the output of each filter in the filter bank is processed sequentially, or where the choice of filtering can change in time. The coefficients are marked as c_p^m where m denotes the index of the selected filter and p denotes the index of the coefficient or the filter tap in a particular filter. A different filter in the filter bank is selected in time by using multiplexers, and the products resulting from its coefficients are transferred to the delay line. The multiplier blocks implemented either for all of the coefficients of a filter or the coefficients from different filters of the filter bank with the same tap index would have redundancy in time because only a part of the available outputs are used. This redundancy means more power and area consumption.

Although it seems to be of no benefit due to the redundancies explained above, the conceptual idea of a multiplier block being reconfigured to output a certain coefficient-product at each cycle can be enhanced further to become very efficient for the time-multiplexed and filter bank applications (i.e. time-multiplexed implementa-

tions of the FIR, IIR filters, filter banks, polyphase filters, adaptive filters, DCT, DFT and FFT processors). This enhancement is achieved by using the reconfigurability at all stages of a multiplier block by replacing the building blocks, i.e. adders, with reconfigurable basic structures, which in simple terms consist of an adder and some form of multiplexed inputs.

Reconfigurable multiplier blocks (ReMB) were studied in [4, 6, 7] during the past few years. Its benefits in field programmable gate arrays (FPGA) and very large scale integration (VLSI) implementations of different types of filters were demonstrated in [4–7]. An algorithm was also developed in [6] to automate the ReMB design. Depending on the application it is capable of producing single input single output (SISO) or single input multiple output (SIMO) ReMB designs that can be used in time-multiplexed filters (Fig. 1(b) and (c)) or filter banks and polyphase filters (Fig. 2).

In a similar study [22], Turner reported significant savings in the area and delay of some DSP blocks by using the reduced coefficient multiplier (RCM) that uses the configurable resources of an FPGA. His design method [23, 24], which is based on common sub-expression sharing, combines the SD-encoded coefficients on to the look-up tables (LUT) that exist in FPGAs and can be used for SISO and multiple input single output (MISO) blocks.

This paper describes the ReMB theory, structures and architecture for FPGA and VLSI implementations in detail. The information already published in [4] and [5] is extended significantly. The algorithm that was developed in [6] and partially published in [8, 9] is also extended to cover more analysis and comments on its efficiency and is improved for better performance. The description of the basic algorithm is included in the paper to enhance the understanding of the subject.

Section 2 presents the theoretical and architectural description of ReMB. Section 3 discusses the implementation issues regarding ReMB structures. In this section particular attention has been paid to FPGA implementation and comparison with a prior work in this field. Section 4 investigates different approaches to the ReMB design problem and proposes a novel methodology. Section 5 establishes the foundations of a novel numerical algorithm and describes new terminology. Section 6 presents a novel ReMB design algorithm in detail with the aid of design examples. Section 7 includes further comments on the algorithm and its performance. Section 8 presents several design examples and detailed information on how the implementation is done, as well giving area and delay comparisons of the proposed technique to other approaches. Section 9 concludes the paper.

2 ReMB Architecture

2.1 Basic Structure

Referring to Fig. 3(a), we define the generalized reconfigurable basic structure as an adder represented by (•) (referring to an adder or a subtractor or an adder/subtractor), with at least two inputs, at least one of which is connected to the output of a multiplexer. All the inputs of the multiplexer(s) and all the inputs of the adder/subtractor are either the input signal to the ReMB, or the output of a similar basic structure or a shifted form of such signals. Zero can also be used as an input to the multiplexers.

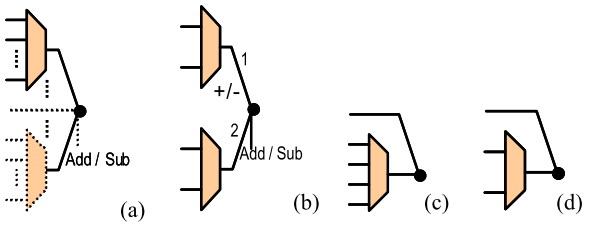


Fig. 3 **a** The general form of a basic structure, **b** basic structure with two 2-to-1 multiplexers connected to a two input adder/subtractor whose 1st input can be added or subtracted (basic structure *2-2), **c** basic structure 1-4, **d** basic structure 1-2. In all cases the select inputs of the multiplexers are not shown for clarity

A basic structure is named as 'basic structure $[*]m_1 - [*]m_2 - \cdots - [*]m_n$' where the adder/subtractor has n inputs and the size of the ith multiplexer is m_i. The input having the addition/subtraction functionality (if there is any) is distinguished by a '*' prefix before m_i.

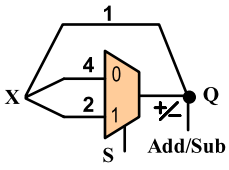
Figure 3(b), (c) and (d) display examples of different basic structures. The Add/Sub input of basic structure *2-2 in Fig. 3(b), which selects the mode of operation, increases the reconfigurability of the structure even further as the input of the 1st multiplexer can be added to or subtracted from the inputs of the 2nd multiplexer. The basic structure 1-2 displayed in Fig. 3(d) is the smallest possible basic structure with a 2-to-1 multiplexer connected to a two input adder/subtractor and is frequently used in the following text for explanation and design purposes, since its implementation on FPGA has advantages as detailed in the next section.

The realization of an adder circuit can be achieved in several ways including ripple-carry adders, carry-save adders and carry-look-ahead adders. For convenience with the number representation, and for simplicity of the structure and dedicated circuitry in FPGAs, the designs given in this paper will employ ripple-carry adders. However, the methodology is readily extendable to carry-save and carry-look-ahead adders.

The basic structure produces several fundamentals or partial products by selecting different inputs of the multiplexer(s), or by performing a different operation (i.e., adder/subtractor functionality). In a normal multiplier block, only one output would be available from the adder.

2.2 Showing Design Details on Basic Structures

An example is given in Fig. 4(a) to demonstrate how the design details are deployed on a basic structure within the actual signal flow graph (SFG) of a ReMB design. As in the multiplier block SFG, the numbers on the edges denote the achieved multiples of the signals after shifting. Shifting can be realized by hardwiring and is free as it is in multiplier blocks. When the vertex has adder/subtractor functionality, a $+/-$ is placed next to the input that is subtracted. The basic structure takes the input X and produces the output Q. The S signal is used to select one of the inputs to the multiplexer. The adder/subtractor then processes the inputs and produces 4 different outputs according to the values of the signal S and adder/subtractor mode



(a)

S	Add/sub	Q
0	0	5X
0	1	-3X
1	0	3X
1	1	-X

(b)

Fig. 4 **a** The basic structure 1-*2 with all of its inputs connected to X with the specified shift values, **b** different output (Q) values as the select signals change

Fig. 5 All five different forms of the two interconnected basic structure 1-2s

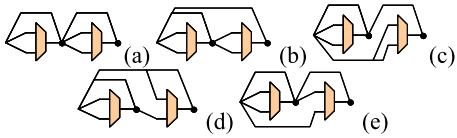
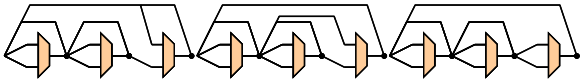


Fig. 6 Some of the sixty topologically different graphs of three interconnected basic structure 1-2s



select signal, Add/Sub. The given basic structure 1-*2, generates a set of outputs $\{5X, -3X, 3X, -1X\}$ as shown in Fig. 4(b) for the specified edge values. These outputs are available one at a time depending on the configuration of the basic structure through the S and Add/Sub signals.

2.3 Interconnection of the Basic Structures

To demonstrate the structural varieties that can be achieved by the combination of the basic structures, Fig. 5(a) to (e) shows all possible ways of interconnecting two basic structure 1-2s as given in Fig. 3(d).

These graphs are topologically all different and can generate distinct combinations of partial products that are not covered by any of the other graphs. When compared to multiplier block graphs for two adders, the number of topologically different graphs is raised from two to five for the smallest basic structure. This number would get bigger for other basic structures such as basic structure 1-3, basic structure 1-4, or basic structure 2-2.

Adding one more basic structure 1-2 which has at least one of its inputs connected to the output of the graphs in Fig. 5 would result in sixty topologically different graphs, some of which are shown in Fig. 6. As was the case for multiplier blocks, the number of interconnected basic structures has a factorial effect on the number of topologically different graphs that can be generated by different interconnections.

2.4 Output Set Size of ReMB

The output set size depends on the specification of the basic structure. In Fig. 4(a), the basic structure 1-*2 has a two-input multiplexer and two different operations for the

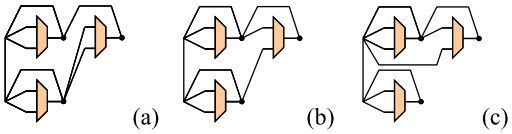


Fig. 7 Three basic structure 1-2s interconnected in a tree form

2nd input of the adder/subtractor. As a result, the output set size is four. In the same way, the basic structure *2-2 (Fig. 3(b)) has 8 different outputs, the basic structure 1-4 (Fig. 3(c)) has 4 different outputs and basic structure 1-2 (Fig. 3(d)) has two different outputs.

The reconfigurability gained by multiplexers bigger than 2-to-1 would be obvious when large designs with lots of coefficients are considered. By having a bigger output set size at the early stages of a ReMB design, the subsequent basic structures, even if they were small, would generate much larger sets of output values.

Let two basic structures, B_1 and B_2, have N and M different outputs respectively. For a ReMB design formed by interconnecting these two basic structures (i.e. all the inputs of B_2 are connected to the output of B_1 as in Fig. 5(a)), the maximum product set size of the ReMB is calculated as $N \times M$, since for each of the M different outputs on B_2, there are N different inputs coming from B_1. Applying this principle would reveal that the maximum number of outputs that can be achieved from the structures in Fig. 5 is four except for Fig. 5(d), for which it is three. This occurs because for one of the configurations of the second basic structure 1-2, it takes both of its inputs directly from the input signal, disconnecting the first basic structure from the chain.

The output set size for three basic structure 1-2s (i.e. the structures in Fig. 6) increases at most by two to become eight. For some graphs the output set size is five or six depending on the interconnection of the basic structures. The factor of increase is related to the number of inputs on the multiplexer. In other words, the maximum output set size tends to increase exponentially as the number of interconnected basic structures of the same type increases. This statement holds for basic structures interconnected as a chain (i.e. Fig. 6) and also as a tree. As an example, Fig. 7 shows three basic structure 1-2s interconnected in a tree form. In Fig. 7(a), the output of the rightmost basic structure 1-2, in other words the second stage of the ReMB, would have a maximum of eight different numbers since the non-multiplexed input of the basic structure and the multiplexed inputs do not depend on each other and therefore generate four different numbers at each configuration of the basic structure. The output set sizes for Fig. 7(b) and (c) are both six.

2.5 Depth of a ReMB Design

Although the output set sizes of the interconnected basic structures are similar for chain and tree forms, the depth of the designs are different. In Fig. 6 a maximum of eight numbers is achieved at a depth of three basic structures, whereas in Fig. 7 the depth is two. This fact is important when deciding the maximum number of outputs that can be achieved at a given depth of basic structures. The tree form interconnections should be considered as well as the type of the basic structures employed.

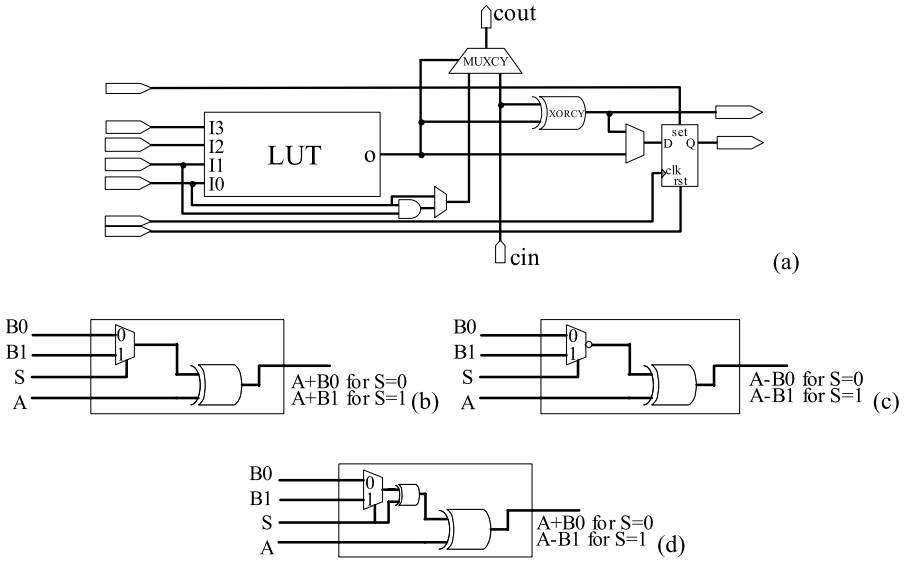


Fig. 8 The LUT configurations for implementing basic structure 1-2

3 Implementation Issues

The ReMB method can be implemented in various media. It would especially suit the goal of compact design in a fixed-resource reconfigurable environment such as an FPGA, by using the already available resources most efficiently. For FPGAs with 4-input LUT such as Xilinx Virtex, it is an advantage that connecting a 2-to-1 multiplexer to one input of an adder/subtractor does not require any extra hardware [23, 24]. This would let ReMB structures using basic structure 1-2s be very efficiently implemented on a Virtex device.

Referring to the simplified schematic of a Virtex half-slice given in Fig. 8(a), the dedicated circuitry for the carry logic and the addition is beneficial for implementing ReMB. Three alternative LUT configurations to implement the basic structure 1-2 in a single half-slice are given in Fig. 8(b), (c) and (d). It should be noted that all combinational logics that may be implemented in a LUT have the same delay and area characteristics because they are implemented as truth tables in a memory. As observed from the figure, a limited functionality of adder/subtractor is available. A single select line controls both the multiplexer and the functionality of adder/subtractor in Fig. 8(d). The displayed LUT configurations form an efficient subset basic structure 1-2s for FPGA implementation.

In [23, 24], a method was proposed to implement time-multiplexed multiple constant multiplications on Virtex FPGAs efficiently, called RCM. This method aims to design the multiplier by considering all of the LUT configurations as in Fig. 8 that would be useful in an RCM implementation. To this extent, sixty-five different LUT configurations were identified that are possibly useful in RCM implementation [23]. These configurations (called "cell definition" in [23]) for RCM designs can be classified as two main types. These types are shown in Fig. 9. Different cell definitions

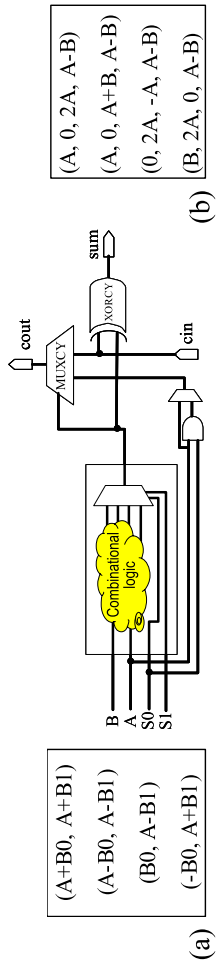


Fig. 9 Two abstract cell definitions of Virtex FPGA used in RCM [23], and some of the cell definitions

are achieved by changing the combinational logic implemented inside the LUT. In Fig. 9(a), three signal sources (A, B0 and B1) are connected to the LUT. The combinational circuit before the multiplexer can generate a variety of logical combinations to feed the 2-to-1 multiplexer. There are a total of seven different cell definitions reported of this type including all the LUT configurations given in Fig. 8. Four of these seven cell definitions are also given in Fig. 9(a).

In Fig. 9(b), up to four different logical combinations of two input sources (A and B) are entered to the 4-to-1 multiplexer. There are fifty-eight different cell definitions of this type. Some example cell definitions of this type are shown in Fig. 9(b).

The cell definitions of the RCM can be considered as a subset of the basic structure definitions of the ReMB that fits into Virtex FPGAs in the most efficient way.

Naturally, full custom VLSI implementation would have a larger degree of freedom to choose the type of basic structure and the adder/subtractor. Some of the basic structures that are very efficient in an FPGA would not be as efficient for the VLSI implementation. Any addition to the complexity of the basic structure has a hardware cost, in terms of both area and delay.

The RTL level descriptions of six basic structures for VLSI implementation are shown in Fig. 10. These basic structures all use a two-input adder, however subtractors, half-adders, half-subtractors and higher valency adders are possible. The hardware description language (HDL) description of these circuits may or may not employ full adder cells depending on the optimization and cell parameters. Although all of the focus is given to ripple-carry adders for the FPGAs because of the dedicated fast logic in them, the VLSI implementations may consist of different adder architectures like carry-save adders and carry-look ahead adders. In [1] and [14] carry-save adders were investigated for use in multiplier blocks.

The functionality of the basic structure is greatly increased with the increased number of inputs and select signals, however it has an opposite effect on the placing and routing. The wiring delay and capacitance would also increase. The more configurable the basic structures are (e.g. Fig. 10(e) and (f)), the more difficult it is to search the solution space effectively, especially for larger designs; therefore it is more difficult to justify the added complexity of basic structures vs. the utilization of the hardware involved in it.

Another difference between the FPGA and VLSI implementation is that the inputs of the full adder cells will not be balanced due to the multiplexer added to one input as shown in Fig. 10(a) and (b). This would possibly result in increased glitch activity during addition. One way to overcome this is to add some logic that would have a similar delay in front of both inputs. Figures 10(c) and (d) are examples of this.

All the cases discussed above are considered for implementation using readily available standard cell libraries with common combinational logic gates. The story is different if the basic structures are implemented at the transistor level as a single cell. Some of the disadvantages can be eliminated by specially arranging the transistor sizes. This approach would also decrease the area overhead by efficiently combining all the functionality.

Some applications may require pipelining of the multiplier structure for reducing critical path delay as well as power. Figure 11(a) shows the pipelined form of Fig. 5(a) as an example. The delay element used here can be D-type latches or flip-flops. The

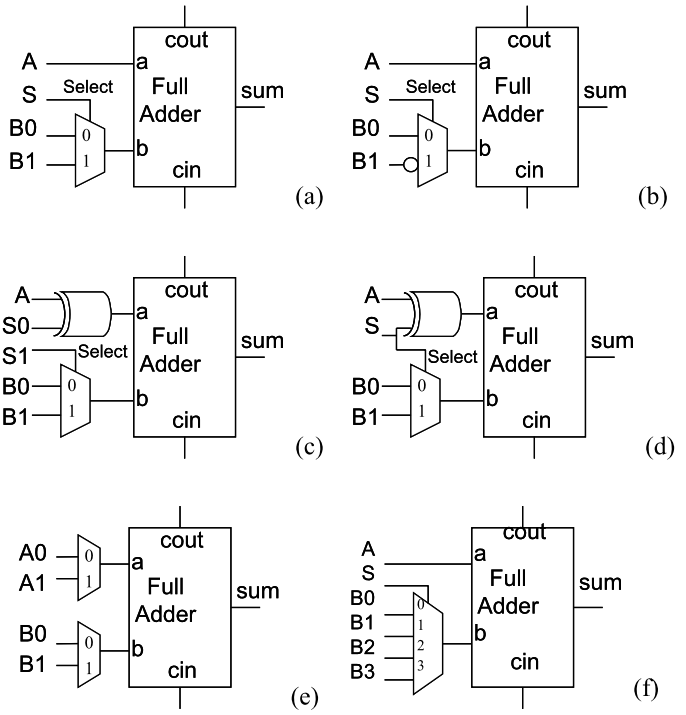


Fig. 10 RTL views of some basic structures that can be constructed for VLSI ReMB designs

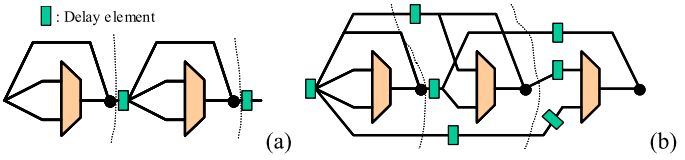


Fig. 11 **a** Pipelined form of Fig. 5(a), **b** pipelining of a more complex ReMB design

dashed lines partition the graph to indicate where to put the delays. Each intersection of the dashed line with an edge of the graph means that particular signal should be pipelined before being processed by the adder/subtractor. It can also be useful albeit expensive when pipelining ReMB structures with more complex interconnections of basic structures as shown in Fig. 11(b).

The ReMB technique can also be effectively employed in newer FPGA series like VirtexII and Virtex4 from Xilinx and Stratix II from Altera, although these devices have built-in multiplier circuitry. These dedicated circuitry can increase the circuit performance significantly when compared to the LUT implementations. However, there are applications requiring more multiplication resources than the available dedicated multipliers, which is not rare in today's very sophisticated processing circuitry. In these circumstances, the ReMB technique can help to utilize the LUT for multiple

constant multiplications and the dedicated multipliers can be used for truly general-purpose multiplications.

4 Design Considerations

For a given coefficient set, designing a ReMB with the minimal number of basic structures requires a well-defined procedure that can fully explore the capabilities of a particular basic structure. The partial product space gets extremely large as the size of the coefficient and/or coefficient set increases. The main problem is to identify the common partial products or sub-expressions that can be mapped to a single basic structure and use the minimum of such basic structures to build all the coefficient values.

Both the numerical and the sub-expression sharing methods mentioned in the introduction offer some advantages over the other for particular multiplier block applications. For example, the numerical approaches, which don't depend on a particular coefficient representation, offer solutions for SISO and SIMO type multiplier blocks, whereas Hartley's method is naturally suitable for SISO and MISO applications such as TDL filters. Therefore, different methodologies would give better results for different ReMB applications [6].

4.1 Sub-expression Sharing Approach

The case of SISO and MISO designs has been investigated for the automated RCM method in [23]. For a given set of SD-encoded coefficients, the RCM method finds sub-expression groups that fit into one of the available cell definitions. It tries to minimize the number of cells used in the design. The method is based on Hartley's common sub-expression elimination method [15]. As an example to explain the procedure of combining sub-expressions, consider two coefficient values 10 and 7. Let their SD encoding be 1010 and $100\underline{1}$ respectively. Therefore, 10 can be generated as $2^3 + 2^1$ and 7 can be generated as $2^3 - 2^0$. If 2^3 is labeled as A, 2^1 as B0 and 2^0 as B1, a cell with the set of operations (A+B0, A−B1) can be used to implement these two coefficients. For coefficients with more non-zero terms, more cells are required to cover all the sub-expressions. An automated way to build RCM has been developed in [23] to search for all alternative sub-expression combinations, and to minimize the cell usage. Although it uses all sixty-five cells, the ones with three input sources (Fig. 9(a) with inputs A, B0 and B1), are not fully explored since B0 and B1 are assumed to be coming from the same source (with different shift values). To give an analogy to ReMB, only three graphs given in Fig. 5(a), (b) and (c) would be covered by the automated RCM design out of five topologically different forms. The graphs whose multiplexer inputs are connected to different sources, i.e. Fig. 5(d) and (e), which we call hybrid forms, would not be covered. For graphs with more basic structures, the ratio of hybrid forms to the total number of graphs is much higher (i.e. for three interconnected basic structures, there are forty-five hybrid forms out of sixty topologically different graphs). Therefore, the automated RCM designs only consider a smaller subset of the available solution space. To exploit the solution

Table 1 Set sizes of the exhaustive search space for single and two interconnected basic structures

Structure	Set size	Percent of total
Figure 3(d)	471	N/A
Figure 5(a)	22 754	3.8
Figure 5(b)	62 003	10.6
Figure 5(c)	126 850	21.6
Figure 5(d)	259 295	44.4
Figure 5(e)	114 335	19.6
Total of Figure 5	585 237	N/A

space of a ReMB design effectively, hybrid forms need to be included in the design methodology.

In [4], an exhaustive search was performed to find all the coefficient sets that can be generated at the output of a single (Fig. 3(d)) and two interconnected basic structure 1-2s (Fig. 5) for the integer coefficient values up to a word-length of 10 bits. The set of basic structures given in Fig. 8 was covered to be able to compare with the RCM design. As observed in Table 1, the number of the coefficient sets dramatically increased for two interconnected basic structures. The percentage column on Table 1 indicates the contribution of a particular interconnected form to the size of the coefficient set space of two basic structures. Fig. 5(a), (b) and (c) provides 36% of the total number of coefficient sets where only a single hybrid form contributes 44.4%. As the number of basic structures increase, the percentage of the contribution by the hybrid forms will become even greater. Therefore, it is very important to incorporate to hybrid forms into the design methodology effectively.

4.2 Numerical Approach

As stated in [12], the RAG-n algorithm employs pre-stored MAG tables to get graphs of coefficients that utilize the already existing fundamentals in the graph. A similar approach for ReMB, utilizing an exhaustive search table, would lead to a better solution than trying to find groups of partial products that minimize the difference with the required coefficients as is the case in the BHM approach [10], as the minimization of the difference is a multi-dimensional and non-linear problem.

For a numerical ReMB design approach, the following factors need further attention when compared to the multiplier block design.

- The topology of the graph that produces a coefficient was not of importance since the primary goal was to construct the coefficients by sharing intermediate results or partial products [13]. Two different graphs generating the same coefficient were not considered separately (although in [6] they have been investigated separately for their transition activity).
- The even coefficient values were treated after finding their highest odd factor.

The basic structures are of primary importance, since the input value and the edge value on the common input line (the input line without the multiplexer) and the operation type of the basic structure affects the choice of the other inputs to the basic

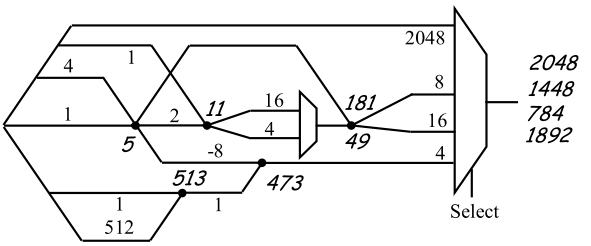


Fig. 12 An example multiplier block

structure. When an even coefficient is generated by first building its odd fundamental at the output of an adder/subtractor together with some other coefficient, as shown in Fig. 12, a separate multiplexer is required to select the output of the adder/subtractor for two different coefficients. Instead, the even numbers can be handled like the odd numbers and generated throughout the ReMB design.

The multiplier block algorithms built the coefficients in a given set one by one in an order generally defined by their costs (minimum number of interconnected adders to generate the coefficient [12]) or magnitudes. The coefficients having the same costs still needed to be built in order, by making use of all the previously generated numbers (both the fundamentals and the coefficients) in the multiplier block. The multiple-output requirement of the multiplier block to be used in the transposed direct form filters (the multiplier block in Fig. 1(d) without the multiplexer) was realized by connecting the generated partial products or coefficients to the corresponding filter taps.

The ReMB design for SIMO applications requires a different design approach than the normal multiplier blocks to avoid underutilization of the basic structures at the output stage of the ReMB (i.e. one of the output nodes is not produced using another output node). If the ReMB is designed by starting from the input, the separation of the group of fundamentals for the coefficient generation on different output nodes would be very difficult to handle. This problem has not been handled effectively in [23]. A detailed description of the problem and a solution method are presented in the following sections for SIMO designs.

4.3 Design for Multiple Outputs

A typical time-multiplexed filter bank application, as shown in Fig. 2, would have output nodes $y_1, y_2, \ldots, y_k$ of the same set size, i.e. the number of coefficients per output node is the same, which we shall assume to be M. We further assume that an output node y_1 is built using several interconnected basic structure 1-m and has M different outputs. Since the upper bound of the output set size of a ReMB design grows exponentially as the number of cascaded basic structures increases, any other output node, say y_2, built with the same type of basic structure cascaded to y_1 would typically have the capacity of $2M$ outputs even though the number of coefficients for this node remains the same.

One way to make sure that the structure is not underutilized and the output nodes are treated independently is to start designing from the output nodes and build the whole design step by step back to the input, as each output node would be a different starting point without any dependence on another.

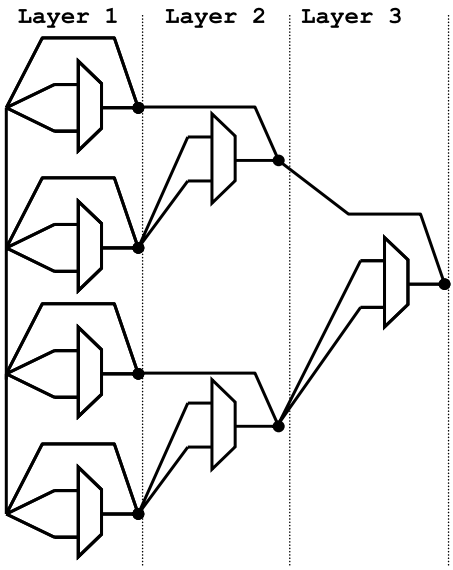


Fig. 13 A ReMB design with a basic structure depth of three, which can produce 128 different coefficients at the output of layer 3

5 Fundamental Concepts for an Algorithm

Before defining the main algorithm, new terminology will be put forward to assist explanation.

5.1 Basic Structure Depth

The depth and the number of basic structures of a ReMB design depend on the number of coefficients per output node. As shown in the previous section, eight different numbers can be generated at a depth of two basic structure 1-2s. In the same way, a ReMB design of depth three as shown in Fig. 13 would have the maximum 128 different number of outputs. The basic structures in Fig. 13 are placed in layers to indicate the depth of that node. In general, the maximum number of outputs from an output node is 2^{n_i} where i is the basic structure depth and n_i can be formulated recursively for ReMB designs comprising basic structure 1-2s as follows:

$$n_i = 2n_{i-1} + 1. \tag{1}$$

A different formula needs to be derived for a different basic structure topology.

Basic structure depth (*bsd*) is an initial and minimal indication of the depth of the design. It is important to find out the *bsd* of an output node when designing a ReMB starting from the outputs.

Consider a ReMB design with one output node with the fundamental set $\{39, 45, 41, 47, 61, 11, 27, 57, 119\}$. All of the fundamentals are cost-2, i.e. each of them requires a cascade of two adders to be generated. On the other hand, since there are nine different numbers, the *bsd* of this node would be at least three if basic structure 1-2 were to be employed, since the maximum number of outputs at depth-2 is eight. In this example, the number of coefficients at the output node dictated the *bsd*.

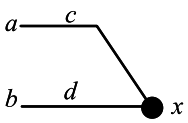


Fig. 14 A graph shows how a coefficient is formed by making use of two other numbers

Now let's consider the coefficient set {473, 181, 49}. The coefficient '49' is a cost-2 number whereas 473 and 181 are both cost-3. For a ReMB design of basic structure 1-2s, the output set size of three is possible at a depth of two. However, this time the *bsd* is dictated not by the output set size but by the cost of the coefficients, which is three although some cost-3 coefficients can be generated at depth-2 (Fig. 7). Choosing depth-3 (Fig. 6) guarantees that we cover all the different topologies that generate cost-3 coefficients.

As a rule, the *bsd* of an output node is the maximum of two values, the minimum depth ($depth_{\min}$) that can generate the required output set size (this value depends on the type of the basic structure employed in the design) and the maximum of the adder-costs of the coefficients ($coef\_cost_i$, i being the index of coefficient).

$$bsd = \max\left(depth_{\min}, \max(coef\_cost_i)\right) \tag{2}$$

It is important to know that the *bsd* only tells about the topological minimum and does not deal with the details of possible operations. Consider the set {9, 15}, which includes two cost-1 numbers. The coefficient '9' can be realized as $(8 + 1)$, whereas '15' is generated as $(16 - 1)$. For an FPGA implementation with a restricted set of basic structure 1-2s as given in Fig. 8, $(8 + 1)$ and $(16 - 1)$ cannot be combined on a single basic structure as predicted by the *bsd*.

5.2 Graph Representation

A coefficient x can be represented on a graph that consists of a set of numbers $\{a, b, c, d\}$ as shown in Fig. 14. x, a, b, c and d satisfy the following equation:

$$x = ac \pm bd, \tag{3}$$

where c and d are in the form of $\pm 2^r$, r being a natural number for integer x.

This concept is also used in graphical representation and synthesis of multiplier blocks [12].

For a predefined interval of $\{a, b, c, d\}$, the solution of (3) may yield many graphs for a coefficient x. All such graphs of a coefficient collected in a table form the graph-tables. These tables can serve as a way of implementing efficient ReMB designs in a short time as will be explained later.

5.3 Node Definition

We also need a way to fully describe the design and configuration details of a basic structure, i.e. to show a combination of graphs using a particular basic structure to produce a given coefficient set at different configurations. Such a description is given in Fig. 15 showing a "node definition" for the coefficient set $\{K, L, M\}$ on

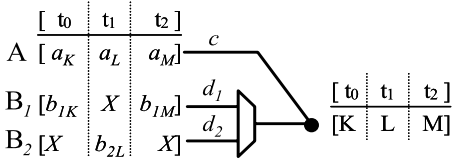


Fig. 15 A node definition includes all the details about the node: edge values, and the fundamentals required to build the coefficient set for a given basic structure

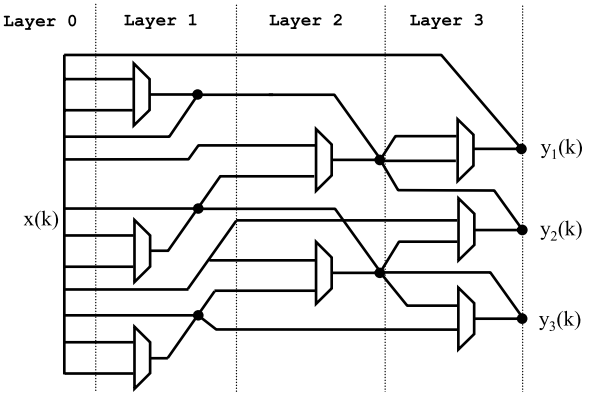


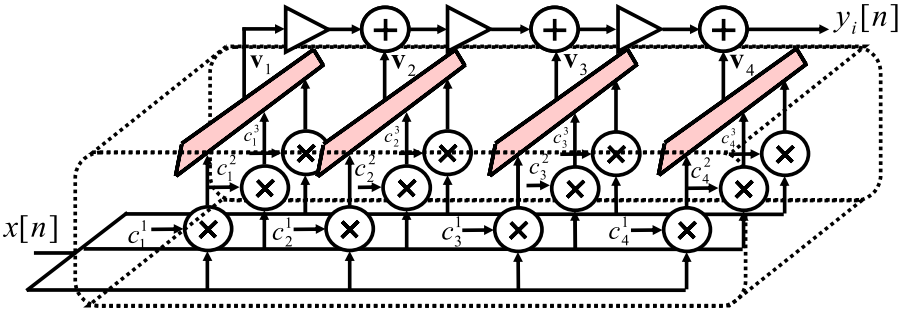
Fig. 16 A symbolic diagram for SIMO ReMB consisting of basic structure 1-2

a basic structure 1-2. Here, A, B_1 and B_2 are the inputs of the basic structure and c, d_1, and d_2 are the edge values. Each coefficient is generated at a different configuration of the set $[t_0, t_1, t_2]$. $[a_K, a_L, a_M]$, $[b_{1K}, X, b_{1M}]$ and $[X, b_{2L}, X]$ are the vectors of fundamentals—building numbers—corresponding to different configuration states. The 'X' (don't care) in $[b_{1K}, X, b_{1M}]$ means the multiplexer does not use B_1 for configuration t_1 but rather uses b_{2L} from B_2 to produce the coefficient L. At configuration t_0, this node generates K as $K = a_K \times c + b_{1K} \times d_1$.

6 Description of the Algorithm

Figure 16 shows the symbolic diagram of a SIMO ReMB design. There are three output nodes, y_1, y_2, and y_3. As observed from the figure, all output nodes are at depth-3. This design is a typical example that can be generated by the algorithm explained below.

The design is partitioned into smaller units by layers to be systematically handled by the algorithm. Each layer has output nodes and fundamental sets that feed the basic structures. For an intermediate layer, the fundamental sets are the output nodes generated in the preceding layers. For the first layer, the fundamental set is formed of either the input signal, which is shown as '1' or the logical zero, which is '0'. Starting from the last layer of the design, the algorithm recursively calls itself at each layer until the fundamental sets are formed out of only '1's and '0's. At each call, the algorithm takes a number of coefficient sets or output nodes as inputs and creates node definitions that can produce the required coefficient sets. The fundamental sets that are required by these node definitions are designed by the following recursive calls of the algorithm.



$$F^1 = \{c_1^1, c_2^1, c_3^1, c_4^1\} \qquad \mathbf{v}_1 = \begin{bmatrix} c_1^1 & c_1^2 & c_1^3 \end{bmatrix}$$
$$F^2 = \{c_1^2, c_2^2, c_3^2, c_4^2\} \qquad \mathbf{v}_2 = \begin{bmatrix} c_2^1 & c_2^2 & c_2^3 \end{bmatrix}$$
$$F^3 = \{c_1^3, c_2^3, c_3^3, c_4^3\} \qquad \mathbf{v}_3 = \begin{bmatrix} c_3^1 & c_3^2 & c_3^3 \end{bmatrix}$$
$$\mathbf{v}_4 = \begin{bmatrix} c_4^1 & c_4^2 & c_4^3 \end{bmatrix}$$

Fig. 17 A TDA filter bank consisting of three filters F^1, F^2, and F^3. The dotted block can be replaced by a SIMO ReMB design

To make sure that the fundamental sets converge to '1', the graphs of the coefficients should be combined in such a way that the *bsd* of the resulting fundamental set vectors at A, B_1 and B_2 should be less than the *bsd* of the coefficient set. To satisfy this condition, the number of different fundamentals (fundamental set size) at an input and the cost of the fundamentals have to be reduced with the chosen graphs. For example, if the coefficient set has a *bsd* of three, the fundamentals at the input sets should be at most cost-2, and the fundamental set sizes can be at most eight (i.e. the maximum number of outputs allowed at that particular depth, see (1)). The node definitions satisfying these requirements can be found by processing the combinations of graphs that exist in the graph-tables.

In the initialization step the coefficient sets defined for each output node are processed to identify the double entries and any negative coefficients. To better explain the format of the coefficient sets, consider a typical SIMO ReMB design that can be replaced with the multiplications taking place in a TDA filter bank application as shown in Fig. 17. The filter bank has three different filters, F^1, F^2, and F^3 each of which consists of four coefficients. Therefore there are four outputs of the ReMB design that would replace the block shown with dotted lines. Each output is assigned with an array of coefficients $\mathbf{v}_1$ to $\mathbf{v}_4$. Each of these arrays or the output node has three coefficients, each of which has to be produced in a different configuration of the ReMB design. These arrays can include the same number as a coefficient for two or more distinct filters.

Figure 18 shows the abstract level flow diagram of the algorithm. Each step in the diagram is detailed below with the help of design examples.

Only positive coefficients are generated in the current form of the algorithm. The negative sign needs to be handled outside the ReMB block by modifying the proceeding design unit, i.e. using a subtractor instead of an adder at the output of the

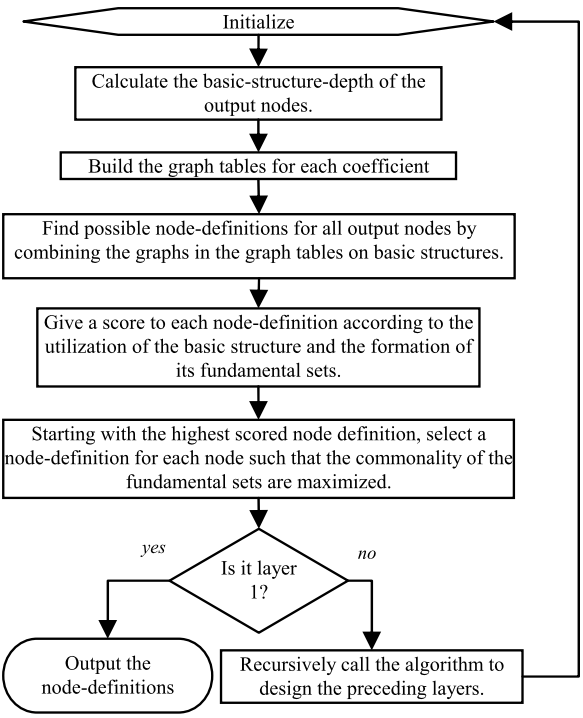


Fig. 18 The flow diagram of the proposed ReMB algorithm

multiplier. The super table holding all the graphs of all numbers up to a limit (for example, for 10-bit coefficients, the limit is 512) is utilized to gather the graphs of the coefficients faster during the algorithm run, as we are merely reading the graph from a pre-generated and stored table.

To explain the rest of the algorithm steps, we first build a ReMB design with single output node and then extend it to a design with multiple output nodes.

Consider the coefficient set $\{39, 45, 41, 47\}$ which has four configuration stages. After the algorithm initializes, the *bsd* of the output node needs to be calculated.

By using (2), the *bsd* can be calculated as 2. Next, the graph tables are formed for each coefficient. All the graphs of a coefficient that are stored in a super table are checked for the costs of their fundamentals—coefficient cost of a and b in Fig. 19(a) should be smaller than the *bsd*—and the magnitudes of the fundamentals—again, a and b should be smaller than a limit number, which is defined as the smallest power of two that is larger than all the coefficients in the node vector (for the current design, the limit number is 64). Figure 20 displays the graph tables generated for the given coefficients. The number of graphs is reduced to avoid unnecessary complexity in the explanation of the following steps.

Before building the node definitions, we form an index space that consists of the indexes of the graphs with matching edge values. Table 2 shows the index space for the graph-tables in Fig. 20 for node definitions on a basic structure 1-2 as given in Fig. 19(b). The graphs of the first coefficient in the coefficient set are compared with the graphs of the other coefficients to find out whether they can be combined.

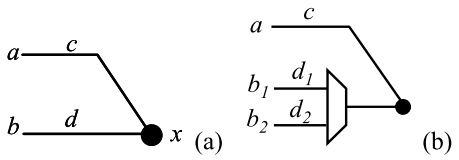


Fig. 19 **a** An example graph, **b** the basic structure 1-2 used in the algorithm to generate the index space

index	1st coef. (39)	2nd coef. (45)	3rd coef. (41)	4th coef. (47)
1	1 -1 5 8	3 -1 3 16	5 1 9 4	9 -1 7 8
2	9 1 15 2	5 1 5 8	9 1 1 32	15 1 1 32
3	17 -1 7 8	15 1 15 2	15 -1 7 8	15 -1 31 2
4	31 1 1 8	17 -1 31 2	31 1 5 2	31 1 1 16
5	33 1 3 2	33 1 3 4	33 1 1 8	33 1 7 2
6	15 1 3 8	31 1 7 2	17 1 3 8	17 1 15 2

Fig. 20 The graph-tables generated for the coefficient set {39, 45, 41, 47}. The number of possible graphs is reduced for easier explanation of the algorithm

The indexes are displayed in two separate columns per coefficient depending on the number of matching edges with the reference graph.

Each node definition will have one graph per coefficient. A node definition of the current design would have four indexes from (1st), (2nd or 3rd), (4th or 5th), (6th or 7th) columns on any given row.

Figure 21 demonstrates how a node definition is formed out of the index space. For each coefficient, the first index on the first row of Table 2 is chosen, and their corresponding graphs are combined. The first graph is placed on the inputs a and b_1 of the basic structure 1-2 for coefficient '39' in configuration state t_0. The unused input b_2 is assigned an 'X' as a "don't care". The second graph uses the other available input b_2 since one of its edge values is different from the edge values of the first graph. In the same way the third and the fourth graphs are added for the remaining coefficients '41' and '47' respectively. Each graph is placed at a different configuration state of the node definition (t_0–t_3). The algorithm systematically scans through all the indexes to find all possible node definitions. During and after this process, all node definitions are examined for two criteria:

Table 2 The index space for the graph tables given in Fig. 20

Graph index for the 1st coefficient (39)	Graph index for the 2nd coefficient (45)		Graph index for the 3rd coefficient (41)		Graph index for the 4th coefficient (47)	
	c & d same	c same, d different	c & d same	c same, d different	c & d same	c same, d different
1	[]	1, 4	3	[]	1	3
2	3, 6	2, 5	4	1, 2, 5, 6	5, 6	2, 4
3	[]	1, 4	1	[]	1	3
4	2	3, 5, 6	5, 6	1, 2, 3	[]	2, 4, 5, 6
5	3, 6	2, 5	4	1, 2, 5, 6	5, 6	2, 4
6	2	3, 5, 6	5, 6	1, 2, 3	[]	2, 4, 5, 6

- The usage of the basic structure (whether all the inputs on the basic structure are used or not)
- The *bsd* of the fundamental sets on the node definition are tested.

If the *bsd* of the fundamental sets becomes equal to larger than the current *bsd*, the generated node definition is discarded.

The node definition generated in Fig. 21 would have been discarded since one of its fundamental sets $\{1, 3, 15, 9\}$ would have a *bsd* of two after the second step, which is the same as the current *bsd*.

Each of the valid node definitions is given a score by the use of a score cost function (SCF), which can depend on several parameters. The priorities of the algorithm can be adapted to new design conditions by simply changing the SCF.

A typical SCF can employ the following parameters:

- σ_i^2: The variance of the cost of the fundamentals in the input sets a, b_1 and b_2, i being the input set index.
- λ_i: Number of different fundamentals in an input set, and
- $x1_i$: The existence of input sets comprising only '1's as a fundamental.

σ_i^2 is zero when the costs of the fundamentals in the set are all the same, and increases as the cost changes. Minimal variance of the fundamental costs is desired to effectively generate each coefficient at its minimal cost and not more.

λ_i will be maximum when it is equal to the output set size defined by the *bsd* (l_{bsd}) to increase the utilization of the basic structures.

$x1_i$ is either zero or one depending on having a fundamental set comprising only '1'. As '1' does not require a basic structure to be generated, having fundamental sets comprising only '1' is desirable.

These parameters can be put into a generalized SCF as follows:

$$score = \sum_i w_i^0 \left(w_i^1 \times x1_i - w_i^2 \times \sigma_i^2 - w_i^3 \times (\lambda_{bsd} - \lambda_i) \right), \tag{4}$$

where w_i^1–w_i^3 are weights for each parameter specified above, and w_i^0 is the weight for different input sets. These weights can be changed to prioritize any of the parameters. Furthermore, new parameters can be added to the function very easily. For

Fig. 21 The steps for generating a node definition out of the index space

example the variance of the magnitudes of the numbers in a set can be another small priority parameter. If there is more than one type of basic structure used in the algorithm, the SCF can prioritize one among the others by having another parameter.

The valid node definitions in our example design are scored with the weights w_i^0–w_i^3 as $\{1, 2, 1, -1\}$ and displayed in descending order of their score in Fig. 22. The weights are chosen to reveal more about the functioning of the algorithm in our design example. They are not necessarily optimal values for an efficient design.

Out of the ten available node definitions, the highest scored one will be chosen as the solution in the following steps. In general, if no node definitions can be built for any of the output nodes, we cannot proceed any further to the solution. If the algorithm exits during its top-level call, a higher *bsd* should be forced into the algorithm for searching the solution in a bigger graph space. If this situation occurs during one of the recursive calls in an intermediate layer, the algorithm overcomes the problem by automatically changing the proposed solution for the initiating layer and recursively calls the algorithm with the new fundamental sets of the next available node definition.

In the example explained until now, there is only one node vector, hence the procedure involving two or more nodes is not required. The highest scored node definition (the top-left node in Fig. 22) is chosen as the winner and its fundamental sets {31, 33}, {7} and {1} are defined as the new inputs of the algorithm for the next recursive call.

The algorithm classifies the fundamental sets that have to be designed in the preceding layers of the design (the following recursive runs of the algorithm) in two groups. The fundamental sets that have to be generated in the preceding layer are defined as the new output nodes of the preceding layer, and the sets that will be generated in the deeper layers are defined as the feed-through sets. This decision is based on the *bsd* of the fundamental sets. Feed-through sets are not designed until the algorithm reaches down to their *bsd* in the design. Let's assume that a node definition in layer R required a fundamental set F with a *bsd* of (R-3). The design of set F wouldn't be considered in layers (R-1) and (R-2). However, because the set F is already required by a node definition in the design and necessarily would be designed in layer (R-3), the algorithm prioritizes the node definitions that can make use of set F in the layer (R-1) and (R-2) to minimize the number of nodes in the design.

In our current design example, {31, 33} and {7} have a *bsd* of one, and hence they are assigned as output nodes in the next recursive run of the algorithm. The set {1} has a *bsd* of zero, therefore it is assigned as a feed-through set although really no design is needed for it.

The next run of the algorithm for a *bsd* of one concludes the design since the fundamental sets of this layer consist of '1' only. The resulting ReMB structure is given in Fig. 23. Since the set {7} has a single number, a basic structure is not required and 7 is formed with only a subtractor. The resulting design is composed of two basic structures and a subtractor as opposed to five adders and a multiplexer stage if designed as a multiplier block by the RAG-n algorithm [12]. In a Virtex FPGA implementation it would occupy three half-slices per bit of data word-length where as the RAG-n design would occupy seven half-slices per bit. Figure 23 does not show all the details about the configuration states and the select signals in order to simply show the complexity and the signal flow of the design.

Fig. 22 The node definitions generated by the algorithm for the set {39, 45, 41, 47}. They are displayed in descending order of score from left to right row-wise

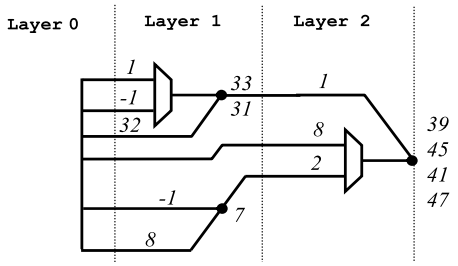


Fig. 23 The ReMB generated by the algorithm for the set {39, 45, 41, 47}

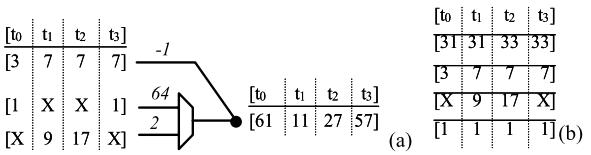


Fig. 24 **a** The winner node definition for the 2nd node, **b** the input vectors for layer 2 of the design example

Now, we extend our initial example to include one more coefficient set {61, 11, 27, 57} to show how the algorithm handles multiple outputs. The coefficients in the set are ordered according to the configuration state required, i.e. '61' and '39' will be produced at the same time.

The *bsd* of this second node is two. The algorithm generates graph-tables, index space and node definitions for both nodes independently. Therefore Fig. 20, Table 2 and Fig. 22 remain the same for the first node.

After generating the scored node definitions for all output nodes, the algorithm picks the highest scored node definition to start forming the design. If there was a feed-through set declared in the algorithm call, then the highest scored node definition, which has the feed-through vector as one of the fundamental sets, would be selected. The highest scored node in our design example is the first node definition of the first node given in Fig. 22 with the input vectors [31 31 33 33], [$X7X7$] and [$1X1X$]. The fundamental set [$1X1X$] can be turned into [1 1 1 1] since the input signal would be available all the time.

We then search the fundamental vectors of the winning node definition on the generated node definitions of the other output nodes. The node definition having the highest number of matching input vectors and the highest score is then selected as the solution for that particular node. This process is repeated until all the nodes are covered by the set of node definitions.

The search of fundamental vectors is a complex task that checks the 'X' states of the fundamental set vectors and identifies whether the vectors are subsets of one another to minimize the number of fundamental sets. For our design, the highest scored node definition with matching input vectors for the second node is shown in Fig. 24(a) and is designated as the solution of the second node. The vector [$1X1X$] is a subset of [1 1 1 1]. Moreover, the input vector [$X7X7$] for the first node definition is covered by the new vector [3 7 7 7].

The input vectors required for the 2nd layer of the design (since the *bsd* is two and possible solutions are found for the output nodes) are given in Fig. 24(b). Their corresponding fundamental sets {31, 33}, {3, 7} and {9, 17} have to be designed in the

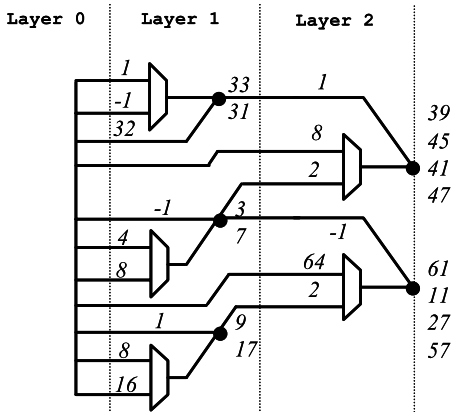


Fig. 25 The ReMB design generated by the algorithm for the coefficient sets {39, 45, 41, 47} and {61, 11, 27, 57}

consecutive call of the algorithm. Therefore, these sets are defined as output nodes, and {1} is again defined as a feed-through set for the next run.

Figure 25 shows the resulting ReMB design. It consists of two more basic structures than the design given in Fig. 23, which contained a single output node. The algorithm efficiently added the second node by making use of the fundamentals that were already required for the first node.

The design generated by the algorithm is readily translatable to an HDL definition for implementation. Its output provides all the configuration details for the control of select inputs of the multiplexers.

The algorithm has been implemented in MATLAB because of its flexibility in handling arrays and matrices.

7 Comments on the Algorithm

7.1 Effect of Super Table, and the Negative Fundamentals

The efficiency of the algorithm greatly depends on the set of graphs available in the super table. The availability of graphs with a variety of fundamentals and edge values is crucial to increasing the number of possible node definitions for a given coefficient set. The more alternative node definitions that exist for a node, the more probable it is that common input vectors will be found for a multiple output design. Therefore, it is essential to have the maximum coverage of graphs for any given number.

Inclusion of the graphs of negative numbers would increase the design efficiency and can reduce the number of basic structures required for a design. As an example, consider the run of the coefficients with a super table consisting of only graphs of positive numbers for the coefficient set {137, 119, 113, 143}. When the algorithm builds the node definitions for this set, the highest scored node definition includes an input vector of {9, 15} as shown in Fig. 26. However, {9, 15} cannot be fit onto a basic structure at a *bsd* of one. Therefore the algorithm picks up another node definition from the available ones and generates the design with four basic structures.

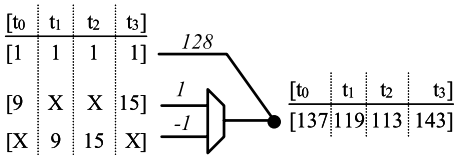


Fig. 26 The highest scored node definition for the coefficient set {137, 119, 113, 143}

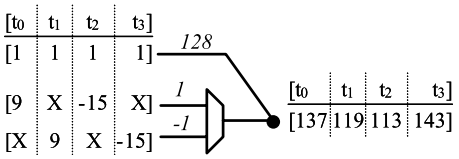


Fig. 27 An alternative node definition to Fig. 26 utilizing graphs of negative numbers

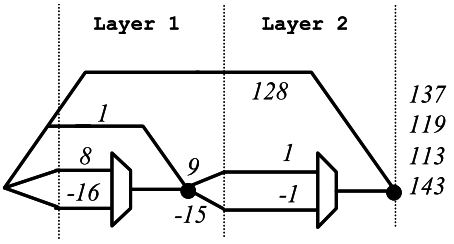


Fig. 28 The ReMB design for the coefficient set {137, 119, 113, 143} utilizing negative numbers in the graph generation

If the graphs of negative numbers are considered, the size of the design shrinks by half for the given coefficient set. As observed on Fig. 26, the edge values of the multiplexed inputs are the negative of each other. Therefore instead of having the input value of 15, we can easily use -15 on the same basic structure to output the same coefficients. The resulting node definition would be as shown in Fig. 27. The advantage we get from this node definition is that it is possible to put 9 and -15 on the same basic structure for a *bsd* of one, leading to the successful generation of a ReMB design with a total of two basic structures as displayed in Fig. 28.

Another issue related to the inclusion of negative numbers into the generation of ReMB designs is that it reduces the necessity to correct the sign of the output for the negative coefficients at the proceeding stages of the actual top-level circuit.

7.2 Application of the Algorithm to the Traditional Multiplier Blocks

The algorithm was originally developed to handle SIMO ReMB designs. It takes multiple coefficient sets and multiple feed-through sets as inputs. When there is only one coefficient at each node, the design is a traditional multiplier block. Therefore it can be used as a traditional multiplier block design algorithm. Instead of using basic structures, the algorithm uses adders and subtractors to build the design when there is one coefficient per node. The *bsd* defines the depth of the adders in this case.

The algorithm described in this paper checks the minimum cost of each coefficient and tries to generate them at their minimal adder cost. Since it builds all the coefficients of the same cost concurrently, it tends to generate them by using fundamentals with lower cost only. By default, it acts like the step-limiting algorithms [16] where

the number of cascade stages of adders (adder-cost) can be defined. The *bsd* can be forced on the algorithm to tell the number of steps (number of cascaded adders), however in its default form, it produces designs with a minimum number of steps. Therefore, it is expected to reduce the transition activity of the design by minimizing the paths that the glitches propagate, hence minimizing the power consumption. On the other hand, the number of adders used in the design can be more than the design generated by RAG-n or BHM. This is because the score function is suited for distinguishing node definitions having multiple coefficients per node. For multiplier block generation, a pre-defined number of different designs can be compared to find the one having the least number of adders.

The feed-through input of the algorithm facilitates the forcing of any intermediate fundamentals to the designs whereby a solution using a certain fundamental vector would be prioritized.

7.3 Run-Time and Intelligence Improvements to the Fully Exhaustive Search of a Solution

The selection of a node definition as a solution depends on its score. Starting with the highest scored node definition and building the rest of the design on the basis of the input vectors of the highest scored node definition may lead to sub-optimal designs in terms of the number of basic structures. This is because there is no guarantee that there will be node definitions for other nodes such that their input vectors match with the ones that are already selected as a solution.

Therefore, an exhaustive search of designs that can be generated by using different node definitions would be required to find the minimum sized design. However, generating all the possible designs is very inefficient if not impossible when there are many alternative node definitions and/or many nodes. Moreover, the probability of finding the minimum sized design by starting from a highly scored node definition is more than that for a lower scored node definition due to the nature of the score function. It is therefore logical to generate a pre-defined number of alternative designs and pick the minimum sized one.

The implemented algorithm already collects a pre-defined number of highly scored node definitions among all nodes into a start-up table to facilitate the generation of an alternative design when a recursive run of the algorithm fails to find a solution. Moreover, it generates a pre-defined number of solutions and outputs the minimal design out of this collection.

Most of the algorithm run-time is spent on searching and collecting node definitions. It is possible to restrict it to a certain percentage of the search space or certain amount of node definitions; however, it decreases the possibility of identifying an optimal solution. The search space gets factorially bigger as the coefficient size and set size increase. However, by checking the validity of the fundamental sets while generating a node definition and skipping the generation whenever it becomes invalid, we were further able to decrease the unnecessary search time.

Due to the nature of the node definition search, bigger graph spaces can be searched concurrently by many processors and then the solutions can be combined and compared.

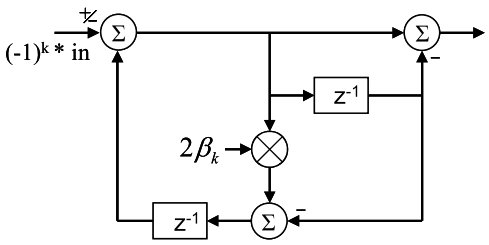


Fig. 29 Recursive part of a Goertzel DCT implementation

8 Applications

The ReMB technique can be applied to any time-multiplexed multiple-constant-multiplication problem. In this section, we will give two examples for its application to a DCT processor [7] and FIR filter implementation [5, 19] to demonstrate its effectiveness both on the FPGA and ASIC platforms. Both examples require single output ReMB structures.

8.1 DCT

A recursive loop of a Goertzel filter that computes an 8-point DCT for video processing is shown in Fig. 29 [7]. In this filter, the multiplier has 8 different coefficient values, characterized as

$$2\beta_k = 2\cos(k\pi/8) \quad \text{for } k \in [0, 7], \tag{5}$$

where k is the DCT coefficient index. For a 12-bit coefficient word-length, the β_k values can be found as $\{2048, 1892, 1448, 784, 0, -784, -1448, -1892\}$ by converting the quantized decimal coefficients from (5) to integers. As observed from the coefficient set, the coefficient values for $k = 5, 6$ and 7 are the negatives of the coefficients for $k = 3, 2$ and 1 respectively. It is therefore possible to implement a ReMB that can generate the multiplications for $k \in [0, 3]$ and its output can be negated afterward at the next adder for $k \in [5, 7]$. The output of the ReMB is shifted right by 10 bits to make the multiplication correct with respect to the decimal coefficient.

Figure 30 shows the ReMB design. It consists of four basic structures that can be efficiently implemented on a Virtex FPGA. This design is highly efficient when compared to a classical multiplier block implementation by the RAG-n algorithm [12], which is known to give the smallest area. It occupies seven adders and extra multiplexer stages at the output as shown in Fig. 31.

A more detailed form of the area and delay comparison of the actual circuit implementations reveals more about the efficiency of the ReMB in Table 3. Assuming an input data word-length of 16-bits, it would cost 221 LUT to use a general purpose multiplier together with a coefficient store. The conventional multiplier block given in Fig. 31 reduced the multiplier area by 21% (48 LUT) to 173 LUT. On the other hand, our ReMB design occupies only 91 LUT for the same word-length achieving a massive 58% saving. The delay values reveal that although the multiplier block implementation is smaller, it has a higher delay. The ReMB design demonstrated the smallest delay amongst the three implementations.

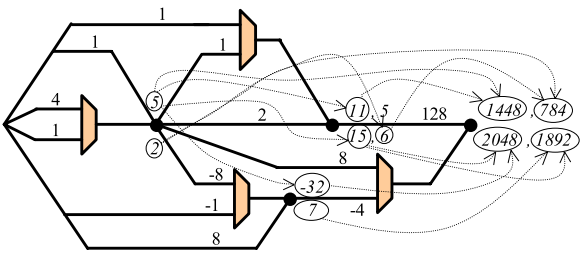


Fig. 30 ReMB design of the DCT loop multiplier. The italic numbers are the partial products generated at each node

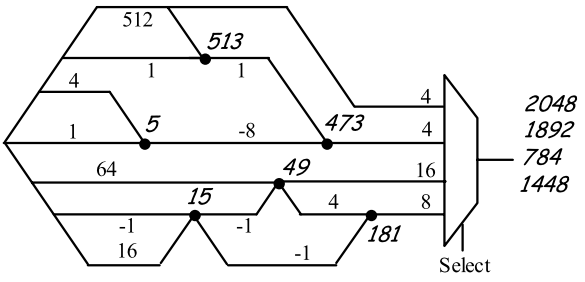


Fig. 31 The corresponding multiplier block design of the ReMB block given in Fig. 30

Table 3 Area comparison of the Virtex FPGA implementations of different multiplier designs for the DCT

Designs	General-purpose multiplier with coefficient store	Multiplier block produced by the RAG-n algorithm	ReMB design
Area (LUT)	221	173	91
Delay (ns)	10.3	10.94	7.61

8.2 A Half-Band FIR Filter

A 32-tap half-band FIR filter was implemented using ReMB in [5]. The coefficients of a half-band filter are symmetric and every other coefficient is zero except the middle coefficient. The coefficient word-length was chosen to be 10 bits and the data word-length was assumed to be 16 bits for the fixed-point implementation.

Three filter implementations were realized in [5] to compare conventional approaches with the ReMB technique. Figure 32(a) shows one of the reference designs, a typical time-multiplexed TDL filter architecture. All the coefficients are stored in a coefficient memory and the incoming input samples are stored in an input memory. With the help of a simple controller one filter tap per cycle is processed.

Figure 32(b) shows the proposed implementation of the filter using ReMB. The coefficient store and the general-purpose multiplier in Fig. 32(a) are replaced with a ReMB structure that performs multiplication for the distinct coefficients stored in the coefficient memory (only the absolute values of the distinct non-zero coefficients are needed). The controller in this case needs to be more intelligent to address the correct coefficient for each tap. With the help of a multiplexer either the coefficient-product or its complement or zero is accumulated at each cycle.

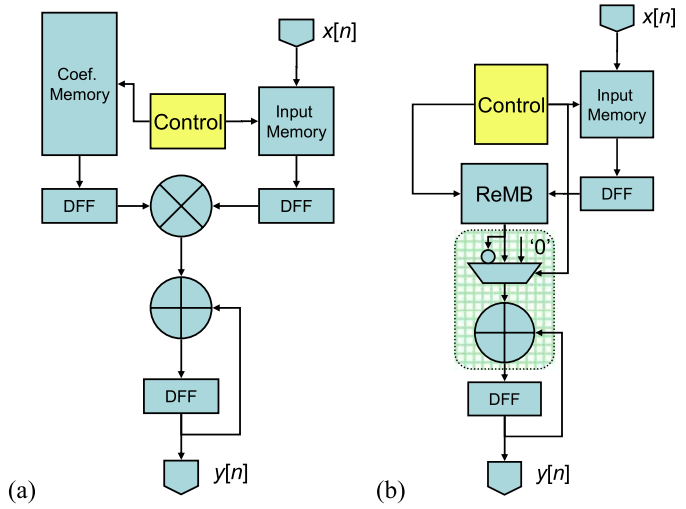


Fig. 32 **a** Time-multiplexed TDL structure implemented as reference filter, **b** the proposed filter implementation using ReMB technique

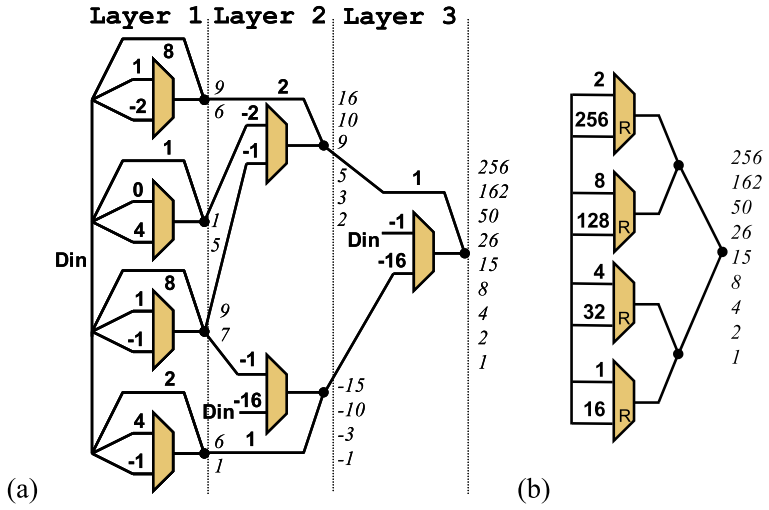


Fig. 33 The ReMB design used in the proposed filter implementation. Din is the input signal to the block

There were nine quantized distinct numbers in the coefficient set: 256, 162, −50, 26, −15, 8, −4, 2, −1. Figure 33 shows two ReMB designs used in the filter. Figure 33(a) is generated by the algorithm described in this paper in [5]. It comprises seven basic structure 1-2s. Figure 33(b) is designed in [19] specifically for efficient custom VLSI implementation. The 'R' letter on the multiplexers indicates that they can be reset to '0' if none of the inputs are desired. This design uses two basic structures 2-2 and an adder.

Table 4 Select values required for some coefficients

Coef.	Select signals for each basic structure						
	BS0	BS1	BS2	BS3	BS4	BS5	BS6
256	0	0	1	X	0	0	0
162	1	1	0	X	0	0	0
50	1	1	0	0	0	1	0
26	1	0	0	1	0	1	0

Select signals were not shown on the ReMB diagrams for simplicity. Table 4 displays the values of select signals in Fig. 33(a) to generate some of the coefficients. They were controlled by the main controller in the same way as the coefficient memory was addressed in Fig. 32(a). In the table, basic structures are indexed from BS0 to BS6 starting from the top of layer 1 downward and then layer 2 and layer 3. The select value of '0' means that the top branch of the multiplexer is selected. An 'X' value means that particular basic structure is not involved in generating the coefficient.

The filters given in Fig. 32 were implemented in both FPGA and VLSI [5] where Fig. 32(b) employed the design of Fig. 33(a). The FPGA implementations were realized on a Virtex FPGA. Area and delay figures for the FPGA designs were obtained after place and route (PAR).

The ASIC implementation targeted the UMC 0.18um CMOS technology and was not placed and routed. The results reported were obtained after synthesis.

Table 5 displays the area and delay figures for all implementations. The coefficient memory and input memory were not included in the area figures. No pipelining has been applied to the filters. Critical path delays are reported for the full combinational logic in the multiply-and-accumulate circuits. An extended version of this comparison with more designs can be found in [5].

The multiplexer in the multiply-and-accumulate path in Fig. 32(b) only contributes to the area for the ASIC implementations. For the Virtex implementation, the components inside the dashed line in Fig. 32(b) were fitted into one LUT, which in turn means the multiplexer comes free.

The area savings achieved by the ReMB technique for the FPGA and the ASIC implementations of this particular example is around 20% compared to the conventional reference design. The ReMB block given in Fig. 33 is not optimal in the sense of the number of basic structures because we did not perform a full search of the graph space. Therefore it may be possible to reduce the area more with a more efficient ReMB structure. Moreover, the use of basic structure 1-2s in VLSI design doesn't provide the best area results as proved by Fig. 33(b).

The decrease in the critical path delay for the FPGA implementations was due to the reduced logic depth of the multiplier. Here, the extra multiplexer stages between adders did not contribute to the delay. However, for ASIC implementations, the critical path delay is increased a little due to the multiplexers. The reduced logic depth of the adder network in the multiplier avoided a large increase in the delay. Reduced logic depth also contributes to lower power since fewer glitches are produced.

If pipelining was considered, the delays associated with all implementations would be similar. Even then, the area of the ReMB filter would be the smallest because of the addition of the same amount of latches or flip-flops to all of the filters.

Table 5 Area and delay figures from the filter implementations

Filters	FPGA implementations (Virtex XCV300)		ASIC implementations (UMC 0.18um CMOS)	
	Figure 32(a)	Figure 32(b)	Figure 32(a)	Figure 32(b)
Area (comb. Logic)	223 LUT	190 LUT	1637 gates	1394 gates
Area (D Flip-Fops)	61 FF	55 FF	380 gates	339 gates
Delay	27.3 ns	23.6 ns	6.52 ns	7.17 ns

Table 6 Area and delay figures for custom implementations

Filters	Multiplier area (transistor count)	Delay (ns)	Power (μW at 62.5 Msamples/sec)
Reference filter Figure 32(a)	5138	4.1	340.9
ReMB filter Figure 33(a)	4058	4.3	447.8
ReMB filter Figure 33(b)	2342	2.1	208.1

Another set of implementations has been compared in [19] where the filters in Fig. 32 were implemented in custom VLSI design using 0.18um UMC technology. In this experiment, all the components including the memory elements were designed in custom. The general-purpose multiplier used in Fig. 32(a) was a Pezaris multiplier. Both of the ReMB designs given in Fig. 33 are implemented in Fig. 32(b). Table 6 compares the three implementations for their area, power, and delay. The area and delay values for the reference filter and the ReMB filter using Fig. 33(a) shows the same trend as in Table 5. The power consumption figures reveal that although the area figure is a bit smaller, the ReMB block that has been designed to efficiently suit an FPGA does not provide power reduction when implemented as ASIC. The second ReMB implementation, on the other hand, which was designed for VLSI by using the balanced basic structure 2-2s, has the best area, delay and power figures.

It should be noted that the above power consumption figures for the custom VLSI design do not necessarily reflect the situation in the FPGA implementation, because the multiplexer is invisible in the LUT and the transitions that occur at the output of the multiplexer do not exist in the FPGA implementations.

9 Conclusion

All types of filters and filter banks where multiple constant multiplications are deployed in a time-multiplexed fashion can benefit from the ReMB technique to reduce the complexity of the multiplication and hence delay and possibly power.

In this paper, we mainly concentrated on the area and delay efficiency ReMB can offer; however, it is also highly likely that fewer operations and simplified hardware would lead to lower power consumption. This is a topic of further investigation.

We have also laid out the theoretical concepts to enable the use of the technique in the most effective way and to expose the pros and cons associated with different implementation platforms. The algorithm presented forms a solid basis for further design automation.

Although the ReMB technique is suitable for implementation using the currently available standard cell libraries, the design and development of dedicated basic structures for custom VLSI designs resulted in significantly better area/power/delay figures.

In conclusion, we would like to state that ReMB can be of great benefit to many real-time specialized DSP applications where area, delay and power are of importance.

References

1. Bartlett, V., Dempster, A.G.: Using carry-save adders in low-power multiplier blocks. In: IEEE Int. Symp. on Circuits and Systems (ISCAS'2001), vol. 4, pp. 222–225. Sydney (May 2001)
2. Bernstein, R.: Multiplication by integer constants. Softw.—Pract. Exp. 16(7), 641–652 (July 1986)
3. Bull, D.R., Horrocks, D.H.: Primitive operator digital filters. IEE Proc.-G 138(3), 401–412 (June 1991)
4. Demirsoy, S.S., Dempster, A.G., Kale, I.: Design guidelines for reconfigurable multiplier blocks. In: IEEE ISCAS'03, vol. 4, pp. 293–296. Thailand (May 2003)
5. Demirsoy, S.S., Dempster, A.G., Kale, I.: Efficient implementation of digital filters using reconfigurable multiplier blocks. In: Asilomar Conf. on Signals, Systems and Computers, November 2004, CA
6. Demirsoy, S.S.: Complexity reduction in digital filters and filter banks, Ph.D. Thesis, University of Westminster (October 2003)
7. Demirsoy, S.S., Beck, R., Dempster, A.G., Kale, I.: Reconfigurable implementation of recursive DCT kernels with reduced quantization noise. In: IEEE ISCAS'2003, vol. 4, pp. 289–292. Thailand (May 2003)
8. Demirsoy, S.S., Kale, I., Dempster, A.G.: Synthesis of reconfigurable multiplier blocks: Part I—fundamentals. In: IEEE ISCAS'05, pp. 536–539, Kobe, Japan
9. Demirsoy, S.S., Kale, I., Dempster, A.G.: Synthesis of reconfigurable multiplier blocks: Part II—algorithm. In: IEEE ISCAS'05, pp. 540–543, Kobe, Japan
10. Dempster, A.G., Macleod, M.D.: Constant integer multiplication using minimum adders. IEE Proc. Circuits Devices Syst. 141(5), 407–413 (October 1994)
11. Dempster, A.G., Macleod, M.D.: General algorithms for reduced-adder integer multiplier design. Electron. Lett. 31(21), 1800–1802 (October 1995)
12. Dempster, A.G., Macleod, M.D.: Use of minimum-adder multiplier-blocks in FIR digital filters. IEEE Trans. CAS-II 42(9), 569–577 (November 1995)
13. Gustafsson, O., Dempster, A., Wanhammar, L.: Extended results for minimum-adder constant integer multipliers. In: IEEE ISCAS'2002, vol. 1, pp. 73–76 (May 2002)
14. Gustafsson, O., Ohlsson, H., Wanhammar, L.: Minimum-adder integer multipliers using carry-save adders. In: IEEE ISCAS'01, vol. 2, pp. 709–712 (2001)
15. Hartley, R.: Subexpression sharing in filters using canonic signed digit multipliers. IEEE Trans. CAS-II 43(10), 677–688 (1996)
16. Kang, H.J., Park, I.C.: Multiplier-less IIR filter synthesis algorithms to trade-off the delay and the number of adders. In: Proceedings of IEEE ISCAS'01, vol. 2, pp. 693–696. Australia (2001)
17. Li, D.: Minimum number of adders for implementing a multiplier and its application to the design of multiplierless digital filters. IEEE Trans. CAS-II 42(7), 453–460 (July 1995)

18. Martinez-Peiro, M., Boemo, E.I., Wanhammar, L.: Design of high-speed multiplierless filters using a nonrecursive signed common subexpression algorithm. IEEE Trans. CAS-II 49(3), 196–203 (March 2002)
19. Morris, P.: Transition and power analysis of multiplier blocks implemented as custom VLSI designs, M.Sc. Thesis, University of Westminster (2005)
20. Pasko, R., et al.: A new algorithm for elimination of common sub-expressions. IEEE Trans. CAD ICS 18, 58–68 (January 1999)
21. Potkonjak, M., Srivastava, M.B., Chandrakasan, A.P.: Multiple constant multiplications: efficient and versatile framework and algorithms for exploring common subexpression elimination. IEEE Trans. CAD ICS 15(2), 151–165 (February 1996)
22. Turner, R.H., Courtney, T., Woods, R.: Implementation of fixed DSP functions using the reduced coefficient multiplier. In: IEEE Proc. of ICASSP'2001, vol. 2, pp. 881–884. USA (May 2001)
23. Turner, R.H.: Functionally diverse programmable logic implementations of digital signal processing algorithms, Ph.D. Thesis, Queen's University of Belfast (August 2002)
24. Turner, R.H., Woods, R.F.: Highly efficient, limited range multipliers for LUT-based FPGA architectures. IEEE Trans. VLSI Syst. 12(10), 1113–1117 (October 2004)