Synthesis of reconfigurable multiplier blocks: part II: algorithm.

Suleyman Demirsoy[1]
Izzet Kale[1]
Andrew Dempster[2]

[1]Cavendish School of Computer Science, University of Westminster

[2]School of Surveying and Spatial Information Systems, University of New South Wales, Sydney, Australia

Synthesis of Reconfigurable Multiplier Blocks: Part II-Algorithm

Süleyman Sırrı Demirsoy, Izzet Kale
Applied DSP and VLSI Research Group
University of Westminster
London, W1W 6UW, UK
{demirss, kalei}@wmin.ac.uk

Andrew G. Dempster
School of Surveying and Spatial Information Systems
University of New South Wales
Sydney, Australia
a.dempster@unsw.edu.au

***Abstract*— Reconfigurable Multiplier Blocks (ReMB) offer significant area, delay and possibly power reduction in time-multiplexed implementation of multiple constant multiplications. This paper and its companion paper (entitled Part I- Fundamentals) together present a systematic synthesis method for Single Input Single Output (SISO) and Single Input Multiple Output (SIMO) ReMB designs. This paper illustrates the synthesis method through examples. The companion paper presents the necessary foundation and terminology needed for developing a systematic synthesis technique. The proposed method achieves reduced logic-depth and area over standard multipliers / multiplier blocks.**

I. INTRODUCTION

The companion paper [1] described new terminology and novel concepts for the design of Reconfigurable Multiplier Blocks (ReMB) for Single Input Single Output (SISO) and Single Input Multiple Output (SIMO) systems. In this paper, we present the details of a systematic synthesis method for SISO and SIMO ReMB. Section 2 will explain the flow of the method by the help of two design examples. These examples reveal the functionality of the algorithm to the maximum extent. Section 3 will conclude the paper.

The algorithm is implemented in MATLAB and is used for several applications as presented in [2] and [3]. Due to space restrictions the performance of the algorithm couldn't be discussed here. However, a full discussion on the algorithm can be found in [2].

II. DETAILS OF THE ALGORITHM

An abstract level flow diagram of the algorithm is given in Fig. 1. Each step in the diagram (after the initialization step) will be detailed with the help of design examples in the following text.

Let us consider a ReMB design with a single output node. It has four configuration stages with the coefficient set given as {39, 45, 41, 47}. After the algorithm initializes, the basic-structure-depth of the output node needs to be calculated.

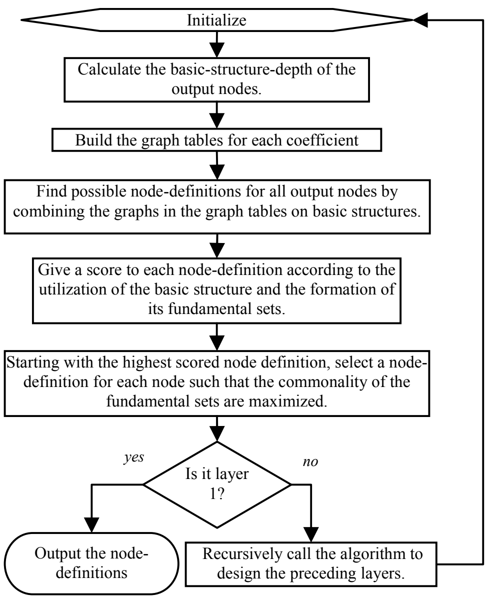

Initialize
Calculate the basic-structure-depth of the output nodes.
Build the graph tables for each coefficient
Find possible node-definitions for all output nodes by combining the graphs in the graph tables on basic structures.
Give a score to each node-definition according to the utilization of the basic structure and the formation of its fundamental sets.
Starting with the highest scored node definition, select a node-definition for each node such that the commonality of the fundamental sets are maximized.
yes
no
Is it layer 1?
Output the node-definitions
Recursively call the algorithm to design the preceding layers.

Figure 1 The flow diagram of the proposed ReMB algorithm

Since the algorithm uses the simplest basic structure for all designs, a coefficient set size of four implies a basic-structure-depth of two. Moreover, the coefficients are all cost-2. Therefore, the basic-structure-depth of the node is found as two.

Next, graph tables are formed for each coefficient. The graphs that are pre-stored in a file are searched. If the coefficient costs of the fundamentals forming the graph (a and b in Fig. 2a) are smaller than the basic-structure-depth and the magnitudes of the fundamentals are smaller than a limit number 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), they are added to the graph-tables. Fig. 3 displays the graph tables generated for the given coefficients. The number of graphs is reduced to avoid the unnecessary complexity in the explanation of the steps follow.

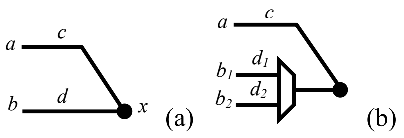


Figure 2 (a) An example graph (b) The basic structure used in the algorithm to generate the index space.

The graphs of different coefficients on the same node are combined on a basic structure, when at least one of the edge values (c and d in Fig. 2a) is same (for the basic structure in Fig. 2b). The procedure of creating a node-definition will be explained in Fig. 4. However, before generating a node-definition, an overall investigation of the edge values has to be done to identify which graphs can be combined. For this purpose, an index space consisting of the indexes of the graphs with matching edge values is built. Table 1 shows the index space for the graph-tables in Fig. 3 for node-definitions on a basic structure as given in Fig. 2(b). The graphs for the first coefficient in the coefficient set are taken as reference and the graphs of the other coefficients that can be combined with these graphs are identified. The indexes are displayed in two separate columns per coefficient depending on the number of matching edges with the reference graph.

The index space makes the search of node-definitions easier. Each node-definition will have one graph for each coefficient. Therefore, for the current design, four indexes from 1st, 2nd or 3rd, 4th or 5th, 6th or 7th columns have to be picked up on any given row. The algorithm automatically goes over all the combinations of indexes to find all possible node-definitions. To demonstrate how a node-definition is formed out of the index space, the first index for each coefficient in Table 1 is chosen, and their corresponding graphs are combined in Fig. 4 step by step. The first graph is placed on the inputs *a* and b_1 of the basic structure 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 to the edge values of the first graph. In the same way the third and the forth 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). All node-definitions built in this way are examined for two criteria.

- First, the usage of the basic-structure (whether all the inputs on the basic structure are used or not)
- Secondly the basic-structure-depth of the fundamental sets on the node-definition are tested.

If the basic-structure-depth of the fundamental sets is not smaller than the current basic-structure-depth, the generated node-definition is discarded.

The node-definition generated in Fig. 4 has to be discarded since one of its fundamental sets {1, 3, 15, 9} would have a basic-structure-depth of two, which is the same as the current basic-structure-depth.

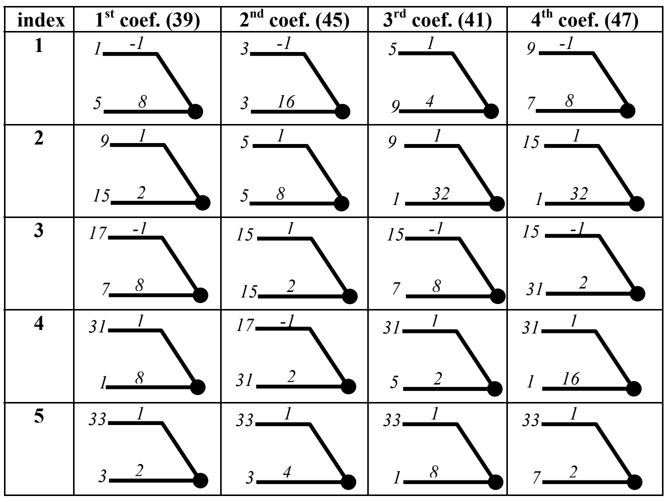


Figure 3 The graph tables generated for the coefficient set {39, 45, 41, 47}. The number of possible graphs is reduced for easier explanation of the algorithm.

TABLE I. THE INDEX SPACE FOR THE GRAPH TABLES GIVEN IN FIG. 3

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	2, 5	4	1, 2, 5	5	2, 4
3	[]	1, 4	1	[]	1	3
4	2	3, 5	5	1, 2, 3	[]	2, 4, 5
5	3	2, 5	4	1, 2, 5	5	2, 4

By going over all the combinations of the indexes in the index-space, all the node-definitions that suit the validity criteria are generated. Five of these are displayed in Fig. 5.

Next, 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 output of the algorithm can be adapted to new design conditions by changing the SCF.

A typical SCF can employ the following parameters:

- The variance of the cost of the fundamentals in the input sets *a*, b_1 and b_2, denoted as σ_i^2, *i* being the input set index.
- Number of different fundamentals in an input set, denoted as ℓ_i,
- And the existence of input sets comprising only ‘1’ as a fundamental, denoted as $x1_i$.

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.

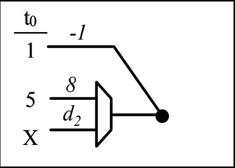
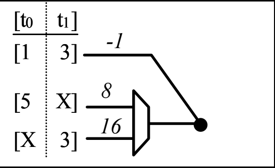
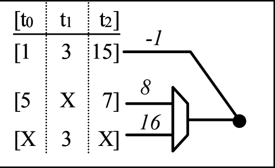
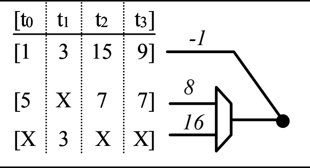
Step 1	Step 2	Step 3	Step 4
		 [t0 t1 t2] [1 3 15] [5 X 7] [X 3 X] -1 8 16	 [t0 t1 t2 t3] [1 3 15 9] [5 X 7 7] [X 3 X X] -1 8 16
First graph of the first coefficient (39) is implemented on the basic structure	First graph of the second coefficient(45) is placed on the available input of the multiplexer.	Third graph of the third coefficient (41) is added to the node-definition.	First graph of the forth coefficient (47) is added to the graph.

Figure 4 The steps for generating a node-definition out of the index space.

ℓ_i will be maximum when it is equal to the output set size defined by the basic-structure-depth (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 combined in a generalized SCF as follows:

$$score = \sum_i w_i^0 (w_i^1 \times x1_i - w_i^2 \times \sigma_i^2 - w_i^3 \times (\ell_{bsd} - \ell_i)) \quad (1)$$

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 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 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. 5. These weights are not necessarily optimal values for an efficient design. They are chosen to reveal more about the functioning of the algorithm in our design example.

In the example design given here, there are in total five alternative 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, the algorithm cannot proceed any further to the solution and it terminates. If the algorithm exits during its top-level call, a higher basic-structure-depth should be forced into the algorithm for searching the solution in a bigger 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.

After scoring all the node definitions, the next task is to choose node-definitions for each node such that, the fundamental sets required in total will be of minimal size. In the example explained until now, there is only one node vector hence the procedure involving two or more nodes are not required. The highest scored node-definition (the 1st node in Fig. 5) 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 fundamental sets that have to be designed in the preceding layers of the design are classified in two groups for the next recursive run of the algorithm. The fundamental sets that have to be generated in the preceding layer are defined as the new output nodes, and the sets that will be generated in the deeper layers are defined as the feed-through sets. This decision is based on the basic-structure-depth of the fundamental sets. Feed-through sets are not designed until the algorithm reaches down to their basic-structure-depth in the design. Assume that a node-definition in layer R required a fundamental set F from layer (R-3). The algorithm does not consider designing that particular fundamental set F in layers (R-1) and (R-2). However, because the set F is already required by a node-definition in the design, the algorithm prioritize 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 the current design example, {31, 33} and {7} have basic-structure-depth of one, and hence they are assigned as output nodes in the next recursive run of the algorithm. The

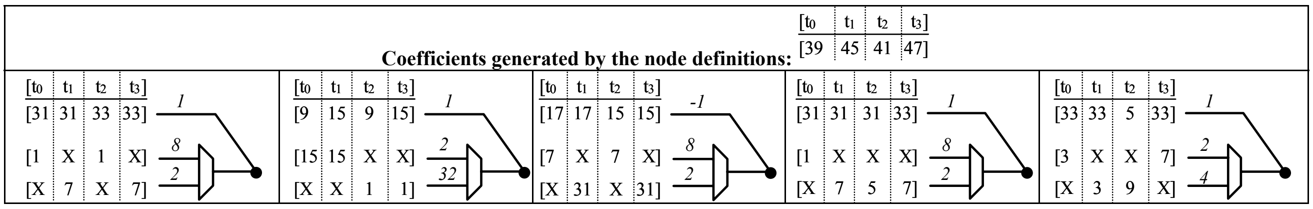

Coefficients generated by the node definitions:
[t0 t1 t2 t3]
[39 45 41 47]
[t0 t1 t2 t3] [31 31 33 33] [1 X 1 X] [X 7 X 7] 1 8 2
[t0 t1 t2 t3] [9 15 9 15] [15 15 X X] [X X 1 1] 1 2 32
[t0 t1 t2 t3] [17 17 15 15] [7 X 7 X] [X 31 X 31] -1 8 2
[t0 t1 t2 t3] [31 31 31 33] [1 X X X] [X 7 5 7] 1 8 2
[t0 t1 t2 t3] [33 33 5 33] [3 X X 7] [X 3 9 X] 1 2 4

Figure 5 The node-definitions generated by the algorithm for the set {39, 45, 41, 47}. They are displayed in descending order of score.

set {1} has a basic-structure-depth of zero, therefore it is assigned as a feed-through set.

The next run of the algorithm for basic-structure-depth of one concludes the design since the fundamental sets consist of '1' only. The resulting ReMB structure is given in Fig. 6. Since {7} includes only one number, a basic structure is not required and 7 is formed with only a subtractor. The 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 [4]. In a Virtex FPGA implementation it would occupy three half slices per bit where as the RAG-n design would occupy seven half slices per bit.

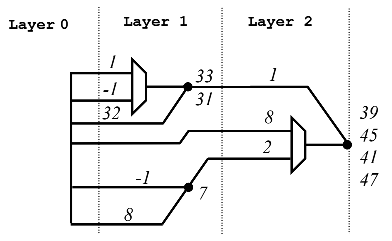


Figure 6 The ReMB generated by the algorithm for the set {39,45,41,47}

To explain the decision procedure of the node-definitions in the existence of the multiple nodes, we extend our initial example to include one more coefficient set, {61, 11, 27, 57}. This new node also has a basic-structure-depth of two. The order of the coefficients are decided according to the configuration state they are required. For example, '39' and '61' are required at the same configuration. The algorithm is run with both of the sets declared as output nodes. It generates graph tables, index space and node-definitions for both nodes separately. Fig. 3, Table 1 and Fig. 5 are not affected because of the additional coefficient set.

Having the scored node-definitions at hand 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 set as an input, would be selected. In the design example, the highest scored node is the first node-definition of the first node given in Fig. 5 with the input vectors [31 31 33 33], [X 7 X 7] and [1 X 1 X]. The input vector [1 X 1 X] is transformed into [1 1 1 1] since the input signal is available at all configurations. The input vectors of the selected node-definition are then searched on the node-definitions of the remaining nodes. The node-definition having the highest number of matching input vectors and having the highest score is then selected as the solution for that particular node. The matching process is a complex task that checks the 'X' states of the vectors and identifies the vectors that are subsets of another to minimize the number of fundamental sets.

The highest scored node-definition with matching input vectors for the 2nd node is shown in Fig. 7(a) and is designated as the solution. The vector [1 X 1 X] is a subset of [1 1 1 1]. Moreover, the input vector [X 7 X 7] for the first node-definition is covered by the new vector [3 7 7 7].

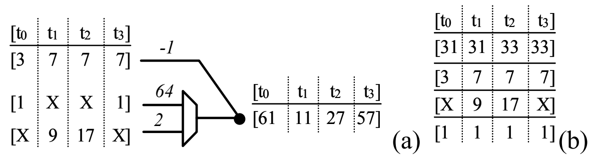


Figure 7 (a) The winner node-definition for the 2nd node. (b) The input vectors for the layer 2 of the design example

The input vectors required in the current layer of the design (layer 2, since the basic-structure-depth is two) are given in Fig. 7(b). Their corresponding fundamental sets {31, 33}, {3, 7} and {9, 17} have to be designed in the 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.

The resulting ReMB design is shown in Fig. 8. It consists of two more basic structures than the design given in Fig. 6. The algorithm efficiently added the second coefficient set for the second node by making use of the fundamentals that were already required for the first node.

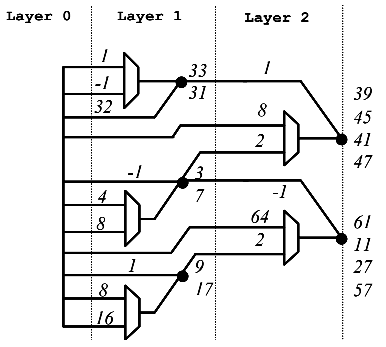


Figure 8 The ReMB design generated by the algorithm for the coefficient sets {39, 45, 41, 47} and {61, 11, 27, 57}.

III. Conclusion

A novel method for designing SISO and SIMO ReMB is presented. The method can handle different basic structures and with the aid of an SCF, the decision mechanism can be easily changed to address different design constraints and conditions.

References

[1] Demirsoy S. S., I. Kale, A. G. Dempster, "Synthesis of reconfigurable multiplier blocks:Part I-Fundamentals", to be published IEEE ISCAS'05.

[2] Demirsoy S. S., "Complexity Reduction in Digital Filters and Filter Banks", Ph.D. Thesis, University of Westminster, October 2003

[3] Demirsoy S. S., A.G. Dempster and I. Kale, "Efficient Implementation of Digital Filters using Reconfigurable Multiplier Blocks", .Asilomar Conf. on Signal, Systems and Computers,November 2004, CA

[4] Dempster A.G. and Macleod M.D., "Use of minimum-adder multiplier-blocks in FIR digital filters", *IEEE Trans. CAS-II,* vol. 42, no. 9, pp. 569-577, November 1995.