Dynamically Reconfigurable Constant Multiplication on FPGAs

Konrad Möller
University of Kassel
Kassel
konrad.moeller@uni-kassel.de

Martin Kumm
University of Kassel
Kassel
kumm@uni-kassel.de

Björn Barschtipan
University of Kassel
Kassel
bjoern_barschtipan@web.de

Peter Zipf
University of Kassel
Kassel
zipf@uni-kassel.de

Abstract

Multiplication with constants is one of the most important operations in digital signal processing including digital filters and linear transformations like the fast Fourier transform. Runtime reconfiguration of the multiplied coefficients, also called reconfigurable single constant multiplication (RSCM), is an efficient method to reduce resources by time-multiplexing and resource sharing. There are several algorithms to realize RSCM on ASICs by fusing multiple single constant multipliers with multiplexers. This may lead to large multiplexers which do not fit well to the structure of FPGAs, so alternative methods are needed. There is one FPGA specific algorithm (called ReMB method) to generate RSCMs by utilizing an FPGA specific basic element based on 4-input Look-Up Tables (LUT) in Virtex 4 FPGAs. This paper investigates an extension of this heuristic algortihm utilizing the 6-input LUTs in recent FPGAs. It is shown that a reduction of the required basic elements of 18% on average can be achieved by this approach. Moreover, convergence in terms of finding a valid RSCM solution can be guaranteed. This was not the case in the original algorithm.

1. Introduction

Constant coefficient multiplication is the basic operation in many applications of digital signal processing like digital filtering and linear transformations. Thus, much research has been done to find optimal solutions for this operation often referred to as single constant multiplication (SCM). The most efficient way to implement SCM on application-specific integrated circuits (ASICs) and field-programmable gate arrays (FPGAs) is multiplierless by a so called adder graph which only consists of additions, subtractions and bit shifts. Though finding an optimal solution for such graphs is NP-complete [BH91], there are optimal solutions for constants of up to 12 bits [DM94], up to 19 bits [GDW02] and up to 32 bits [TN10]. Besides this, there are good heuristics called RAG-n, BHM [BH91] and H_{cub} [VP07] to generate SCMs. Online generators which use these

heuristics are available on the web page of the SPIRAL project [SPI13]. The focus of this paper is on the dynamic reconfiguration of SCMs. One motivation is the reuse of resources by time multiplexing to minimize the design size when the timing is not critical. Another interesting aspect is the use of reconfigurable single constant multiplier blocks (RSCMs) as basic blocks of filters with changing characteristics, e. g., for different frequency regions and adaptive filters. There are several algorithms to create such RSCMs by fusing independently optimized SCMs [THP07,CC09] which where optimized for ASICs. In [THP07] the authors suggest to fuse several optimized SCM graphs by a recursive algorithm called DAG fusion. The first step is a fusion of two SCM graphs with minimal hardware costs. Reconfiguration is realized by multiplexers to switch between the different coefficients. To include more than two coefficients the related SCMs are recursively added to the existing RSCM in the same way. Another ASIC-optimized algorithm to realize RSCMs [CC09] tries to exploit similarities between different coefficients with their canonical signed digit (CSD) representation. Common subexpression elimination (CSE) reduces the number of required adders by searching and fusing identical patterns in the CSD representation of the coefficients. The different shifts and interconnections to realize a specific coefficient are switchable through the introduction of multiplexers. These two approaches lead to large multiplexers and thus are not ideal for FPGAs. There is only one algorithm (called ReMB method) by Demirsoy et al. to generate RSCMs which is optimized for FPGAs [DDK03, DKD05a, DKD05b, DKD04, DKD07]. It is an approach that constructs an RSCM of basic structures that fit into the basic logic elements (BLE) of FPGAs. Therefore, this method was taken as a starting point of our investigation on dynamically reconfigurable constant multiplication on FPGAs. After the analysis of the original algorithm (section 2) some extensions where made to utilize larger LUTs in recent FPGAs and to solve a convergence problem found in the description of the original algorithm (section 3). The experimental results are shown in 4. In section 5 the next steps to be done are reflected and section 6 gives a conclusion.

2. ReMB Method

2.1. Basic Concept of the Algorithm and Metrics

The basic idea of the algorithm is to adapt the basic graph structure to the BLE structure of the underlying FPGA to realize a reconfigurable multiplication of an input x with a constant c_i, where i is the index of the specified coefficients in the coefficient set $C = \{c_1, c_2, ..., c_i\}$. The starting point of Demirsoy et al. is the Virtex 4 BLE structure with a 4-input LUT followed by the arithmetic carry chain. This can be used as a basic element (BE) with a 2:1 multiplexer and an adder as it is shown in Figure 1 a. The arrows denote a specific left shift l_x which equals a power of two multiplication of the corresponding input. The shown BE implements the equations

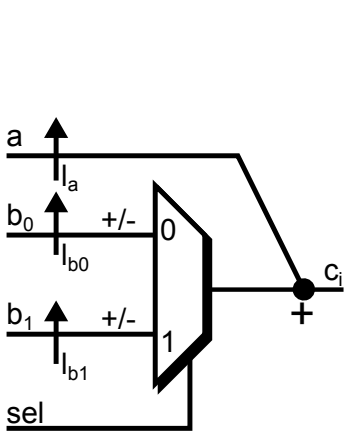
$$c_i = a2^{l_a} \pm b_0 2^{l_{b0}} \text{ or } c_i = a2^{l_a} \pm b_1 2^{l_{b1}} \tag{1}$$

respectively, depending on the selected multiplexer input sel and the sign of b_0 and b_1. In the hardware implementation the shifts can be realized by wires and the addition can be separated into the sum and $\text{carry}_{\text{out}}$ output for each output bit following the relations

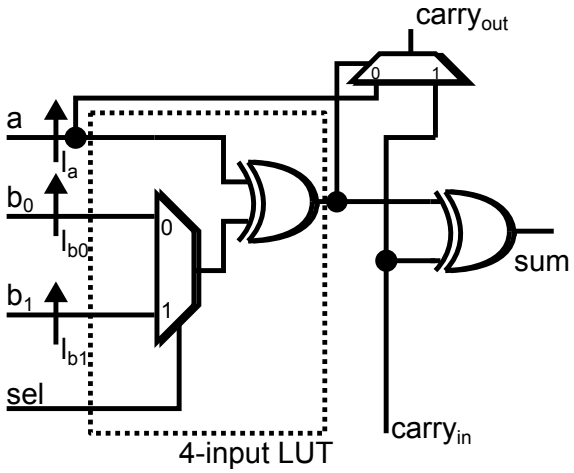
$$\text{sum} = a \oplus b \oplus \text{carry}_{\text{in}} \tag{2}$$

$$\text{carry}_{\text{out}} = (a \oplus b)\text{carry}_{\text{in}} + \overline{(a \oplus b)}a \tag{3}$$

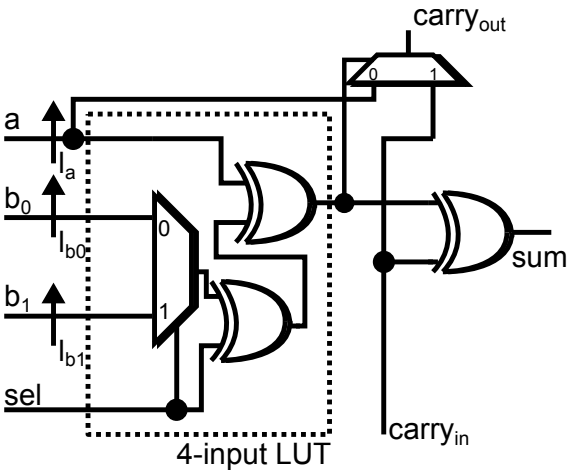
for the example in Figure 1 (a). This can be mapped to the LUT and carry chain of a single BLE per bit as it is shown for (1) in Figure 1 (b) for $b_0, b_1 > 0$, (c) for $b_0 > 0, b_1 < 0$ and (d) for $b_0, b_1 < 0$.



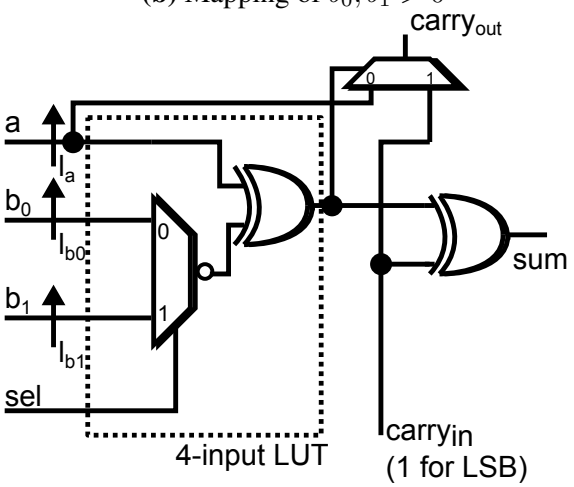
(a) Stucture of the basic element



(b) Mapping of $b_0, b_1 > 0$



(c) Mapping of $b_0 > 0, b_1 < 0$



(d) Mapping of $b_0, b_1 < 0$

Figure 1: Mapping of the basic element (a) to the Virtex 4 FPGA BLE (b-d).

Following (1) one BE can produce two possible outputs. One key element of the algorithm is to determine the number of required BE stages to realize a specific coefficient set C. A horizontal cascading of k stages of BEs leads to the definition of the number of different possible constant factors N_{output_k}

$$N_{\text{output}_k} = 2^{k^2-1} \tag{4}$$

beginning with $N_{\text{output}_1} = 2$. An example to show this relation is illustrated in Figure 2. To put it another way each BE can reduce the number of required outputs in an output set that have to be realized, starting with the given coefficient set C as initial output set, until the BE with a single input (x) is reached. The minimum number of required stages in the cascade to realize one specific coefficient is called $\text{depth}_{\text{adder}}$. The number of required stages to realize a specific number of different coefficients is called $\text{depth}_{\text{mux}}$. These two definitions are necessary to classify

a coefficient set for a RSCM. This circumstance shall be justified by an example. On the one hand it is not possible to realize the coefficient set $C = \{5, 23\}$ with a single BE, because 23 requires at least two basic operations of the kind denoted in (1). On the other hand the set $C = \{5, 7, 9, 15, 31\}$ can only be realized with at least two cascaded BEs (k=2 required (4)) though each single element could be realized with only one BE. The two criteria can be summarized to the basic structure depth (BSD) with the following norm:

$$\text{BSD} = max(\text{depth}_{\text{mux}}, max(\text{depth}_{\text{adder}})) \tag{5}$$

To solve the problem of finding a valid RSCM for the aimed-at coefficient set and thus achieve convergence, the algorithm reduces the BSD with each stage of the cascade.

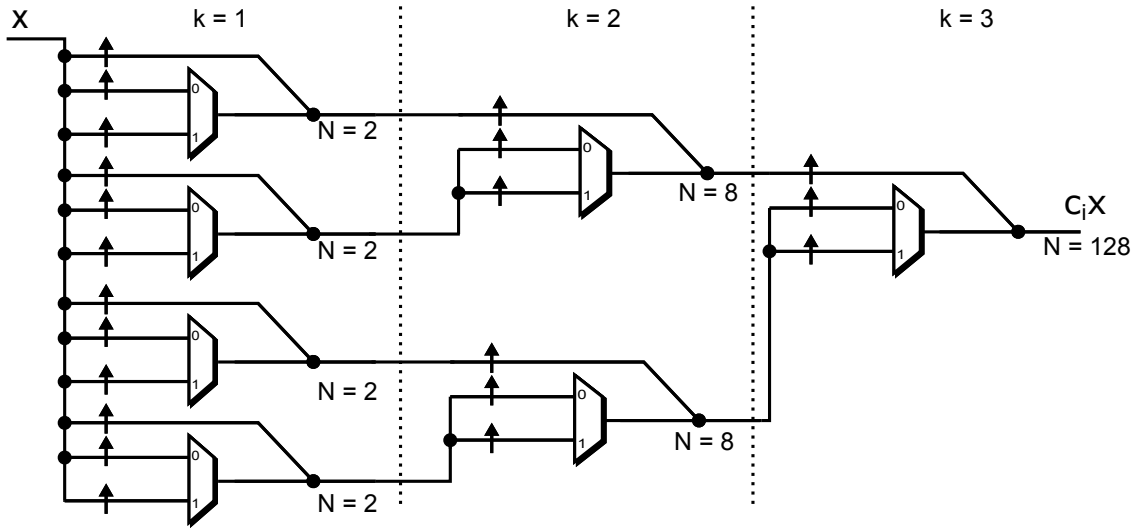


Figure 2: Example of a cascading of several basic elements, N is the number of possible output coefficients.

2.2. The Algorithm

The starting point of the algorithm as shown in Listing 1 is the coefficient set C. That means the generation of the RSCM starts from the output stage (line 1). In each iteration of the algorithm one stage of BEs is determined (lines 4-6) until an input coefficient of 1 is reached. This can be achieved by reducing the BSD in each stage. For that purpose a so called graph table for each coefficient is created (line 4). The graph table of each coefficient contains all combinations of inputs and shifts which fulfill the equation $c_i = a2^{l_a} \pm b2^{l_b}$ (cf. (1)). In the next step the graph tables of all coefficients are combined to fit into basic elements (line 5). This is done by a so called index table that lists the indices of entries in the coefficients' graph tables which can potentially be combined. These can be determined by searching for similarities (e. g. same shift value, same input) of graphs in the graph tables. Beyond that, solutions which are not valid are excluded. Non valid solutions are combinations which would not reduce the BSD or which can not be realized by the BE (an example is given below). Finally, the best combinations in the index table which cover all coefficients are selected (line 6). The choice for the best combinations is made by a score function s defined as

$$s = \sum_i \omega_1(\omega_2 N_1 - \omega_3 \sigma_i^2 + \omega_4(\text{BSD}_o - \text{BSD}_i)) \tag{6}$$

Listing 1: Pseudo code of the algorithm

```
1 active_output_sets = C
2 for i = BSD downto 1
3         for j = size(active_output_sets) downto 1
4                 g = graph_tables(active_output_sets(j))
5                 comb = combine(g)
6                 best_stage_sets(j) = best_combinations(comb)
7         active_output_sets = inputs(best_stage_sets)
```

It is a weighted sum depending on a factor that indicates whether all inputs are 1 (N_1), the variance of $\text{adder}_{\text{min}}$ of the inputs (σ_i^2) as well as the difference of BSD of the outputs and inputs of the basic element. In the last step the best combinations are stored as BEs for the specific stage and their inputs are the new output sets for the next iteration (line 7). The selected BEs are optimal referred to the score function for the specific stage, because all possible graph combinations are covered. The impact of this choice on the preceding stage is not considered. Thus, the algorithm does not find a globally optimal solution.

Figure 3 shows an example result of the heuristic algorithm with an exemplary coefficient set $C = \{39, 45, 41, 47\}$. In the first iteration the BE with the inputs $\{33, 31\}$ unshifted, $\{1\}$ shifted by 3 and $\{7\}$ shifted by 1 turn out to be the best choice. In the second iteration the two input sets $\{33, 31\}$ and $\{7\}$ are the new active output sets. Input set $\{1\}$ is the input of the RSCM and thus does not have to be processed in the following iterations. The new active output sets can be realized by one BE with $\{1\}$ shifted by 5, $\{1\}$, $\{1\}$ negated for the output set $\{33, 31\}$ and one BE with $\{1\}$ shifted by 3 and $\{1\}$ negated for the output set $\{7\}$. Finally each input set is $\{1\}$ (overall BSD is 0) and the algorithm stops.

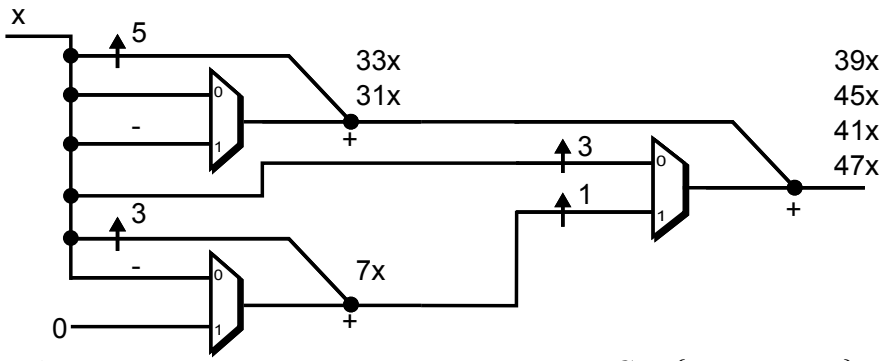


Figure 3: Example RSCM with the coefficient set $C = \{39, 45, 41, 47\}$.

The basic idea of the algorithm is very good as it considers the specific hardware structures the resulting SCM can be mapped to. One disadvantage of the implementation as it is described in [DKD07] is the fact, that not all pairs $\{c_1, c_2\}$ can be realized by a basic element and thus only reducing the BSD does not guarantee that a valid solution can be found (convergence of the algorithm). An example pair is $\{5, 7\}$ which is only realizable as $1 + 4$ and $-1 + 8$ respectively. This case is not covered by (1), because it equals the general pair $\{a2^{l_a} \pm b_0 2^{l_{b0}}, -a2^{l_a} \pm b_1 2^{l_{b1}}\}$. For such cases the authors suggest to search for alternative graphs. In cases where the only valid graph consist at least of one such element the algorithm will not find a solution. So it's a debatable point

whether the definition of the BSD (5) is valuable. Another disadvantage is, that no negative coefficients are supported by the current implementation of the original algorithm. Negative coefficients during the optimization could in some cases reduce the complexity of the resulting SCM as it was shown by examples but not included in the algorithm in [DKD07]. Further drawbacks are analyzed in section 4.

3. Extended Algorithm

This section shows our extension of the original algorithm for a recent FPGA with 6-input LUTs. A Xilinx Virtex 6 FPGA is taken as an example for the recent Xilinx and Altera FPGAs which provide 6-input LUTs. In the first part new basic elements are introduced to reduce the required resources by BEs with more inputs and other functionality. In the second part the applied changes to the original algorithm are presented.

3.1. New Basic Elements

Our straight forward approach is to include a 3:1 multiplexer and an adder into one basic element BE1 which replaces the 2:1 multiplexer-adder-combination of the original algorithm. The realization of that element is shown in Figure 4 (a) and (b). Besides an addition of the inputs b_x each combination with subtracted inputs is possible by analogy with Figure 1 (b-d). The second basic element BE2 (Figure 5 (a) and (b)) uses a feature of the LUTs in Virtex 6 FPGAs. These can be used as a 6-input LUT with one output and as a 5-input LUT with two outputs. The 5-input LUT is used to realize two 2:1 multiplexers whose outputs are added. This was not possible with a Virtex 4 LUT, which offers only one output. BE2 is motivated by the example given in the previous section $C = \{5, 7\}$. This can now be implemented with BE2 by switching between $1 + 4$ and $-1 + 8$ which was not possible in the original algorithm. The basic element in the original algorithm did not support an implementation of $\pm a$ which is now possible through the use of two different a-inputs (a_1 and a_2).

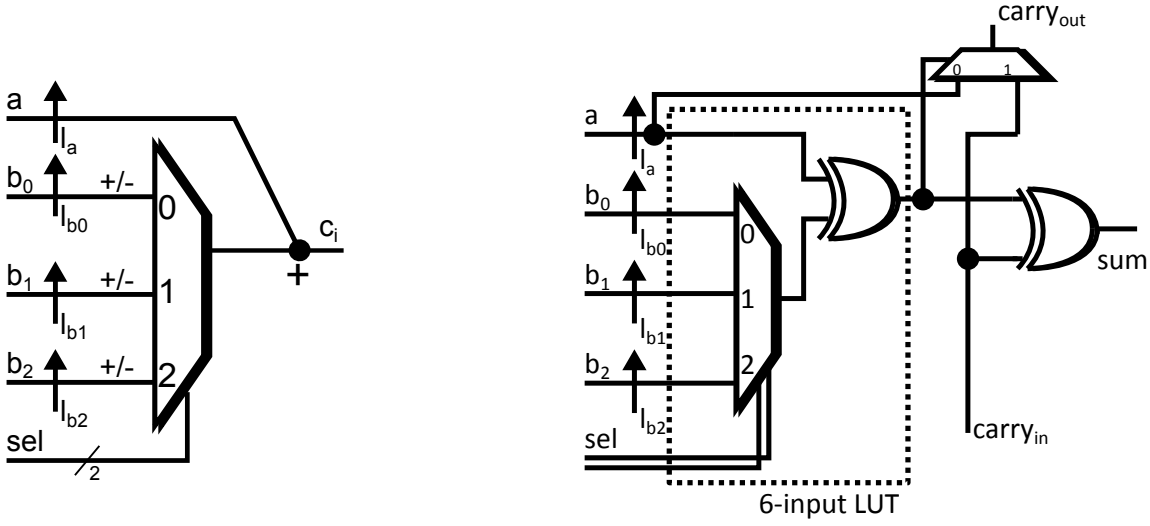


Figure 4: Structure (a) and implementation (b) with a 6-input LUT of basic element BE1.

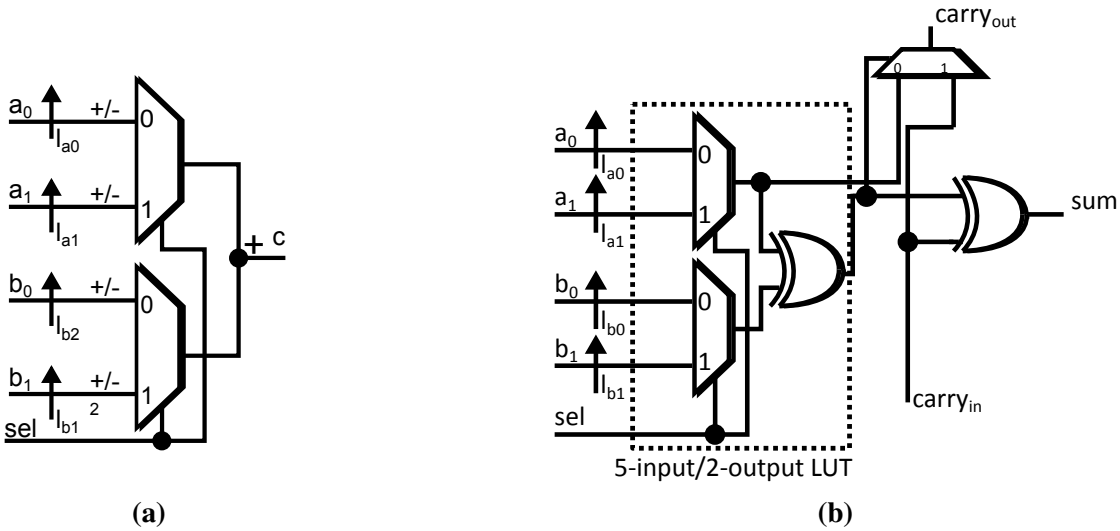


Figure 5: Structure (a) and implementation (b) with a 2-output/5-input LUT of BE2.

3.2. Changes to the Original Algorithm

The main advantage of our extension of the BE used in the original algorithm to BE1 is that now not two but three coefficients can be realized by one BE1. Thus the number of required basic elements is expected to be reduced by a factor of $\frac{2}{3}$ in the best cases. To include the new BE1 into the algorithm steps 5 and 6 in Listing 1 were changed. More combinations are possible and valid in the extended algorithm, because the definition of the N_{output_k} (4) and thus the BSD calculation were adapted to the larger multiplexer size.

Besides the implementation and integration of the new basic elements BE1 into the algorithm some other changes were made. These are necessary to guarantee that the algorithm finds a valid solution. The problem with the coefficient sets of the kind $C = \{5, 7\}$ that could not be integrated into the SCM graph is solved by splitting C into two sets $C_1 = \{5\}$, $C_2 = \{7\}$ which are processed separately (called splitting in the following). This leads to the insertion of an additional BE1 for the seperate processing but guarantees convergence of the algorithm.

In order to compensate the drawback of the additional BE1 an operation called fusion, which is possible through the introduction of BE2, was attached at the end of the algorithm. The fusion searches for underutilized BE1s. These are BE1s which result from splitting and have only one used multiplexer input. A pair of those BE1s can be fused to one BE2, like it is shown in Figure 6. This leads to a decrease in the BE overhead caused by splitting. To see the effect of the fusion we generated 92 random graphs with 2-7 coefficients and 6-8 bit word size. In 42 cases splitting was required to get a valid solution with an overall BE overhead of 40%. This means that in the original algorithm no valid solution could be guaranteed for these cases. The BE reduction by fusion was successful in all example cases and lead to a complete compensation of the BEs which were inserted by splitting.

Thus, the improvements of the new algorithm are that it can reduce the number of required basic elements and that a valid solution for all positive input coefficients can be found by splitting and fusion, without additionally required BE resources for our random examples.

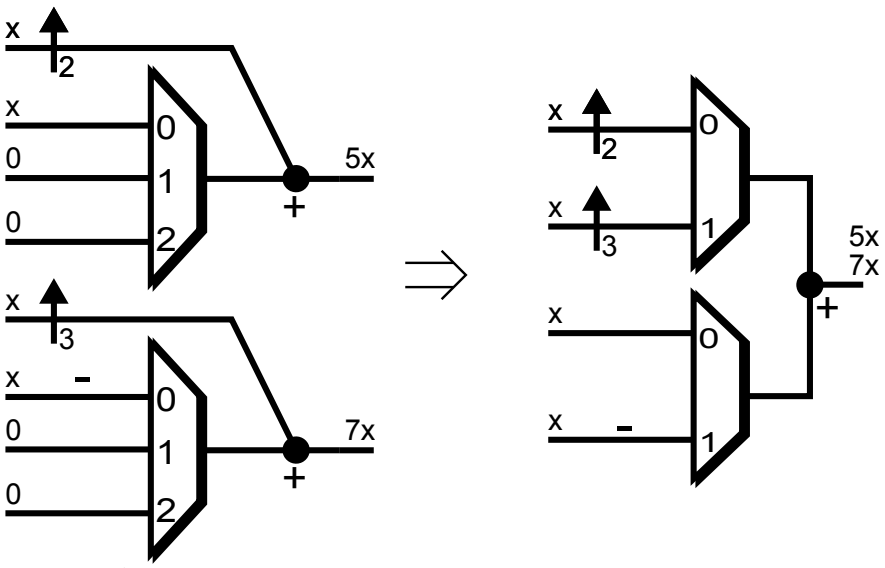


Figure 6: Example for the fusion of two BE1s to one BE2

4. Experimental Results

As there is no analysis of the complexity of the algorithm in previous work we analyse in this section the complexity in terms of resulting graph combinations for different numbers of coefficients per set. Besides this we compare the number of required basic elements for the original algorithm to our extended algorithm to show the impact of the new basic elements to the resulting design size. Finally we compare results of mapping our extended algortihm to mapping a state of the art RSCM method for ASICs (DAG fusion) to a Virtex 6 FPGA to justify specific optimization of RSCM graphs for FPGAs.

4.1. Complexity

Finding an optimal RSCM is a generalization of the SCM problem and therefore also NP-complete. In both versions of the algorithm, the original one and our extended version, all possible graph tables for each coefficient as well as all combinations of these tables are included in the search for the best basic structures of each stage which seems to be very inefficient. The complexity of this search was quantified by 92 graphs already used for the experiment in the section 3.2. The average number of possible graph combinations that have to be evaluated by the score function are shown in Figure 7. An exponential growth of complexity with the number of coefficients in C can be seen. This leads to a memory bottleneck for larger coefficient sets. Putting it on a level with the execution time of the algorithm, the average execution time for seven-constant sets was 155 seconds for an Intel Core i7-3610QM, 2,3 GHz . The memory bottlenecks for set sizes larger than seven in our Matlab implementation confirmes that the exhaustive search in each stage is very inefficient and the realization of larger coefficient sets seems to be impracticable.

4.2. Comparison to the Original Algorithm

The extension of the basic element with a 2:1 multiplexer to a basic element with a 3:1 multiplexer is expected to reduce the number of required basic elements (and thus the required basic logic

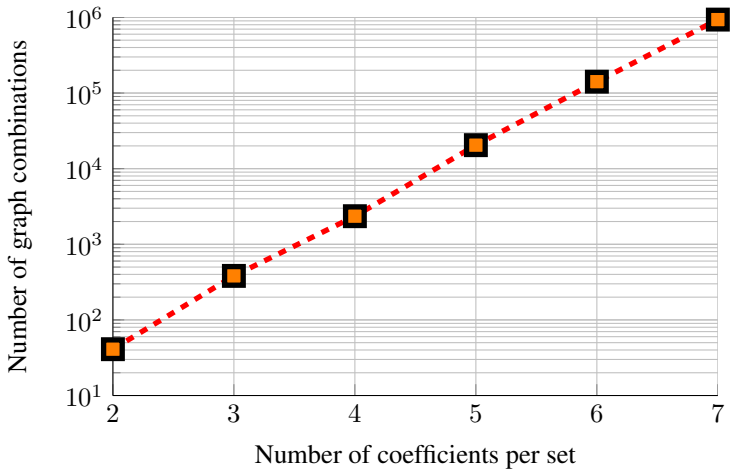


Figure 7: Complexity for different numbers of coefficients per set

elements in the FPGA). This reduction is shown by examples (Table 1) taken from [DKD07] and amounts to 18% reduction on average for our proposed algorithm in these three cases. It can be observed that even for these small examples, considerable reductions can be achieved.

Table 1: Required basic elements of the ReMB method and its extension

coef. set C from [DKD07]	BEs ReMB	BEs ext. ReMB (proposed)
{137,119,113,143}	4	**3**
{1,2,4,8,15,26,50,162,256}	7	**5**
{39,45,41,47}	**3**	**3**

4.3. Comparison to DAG Fusion

As already stated in the introduction, the evaluated algorithm is the only algorithm which has been designed for the RSCM optimization for FPGAs. To see whether it is really necessary to have a specialized FPGA optimization algorithm the results of the proposed algorithm are compared to the results of the DAG fusion algorithm for some examples taken from [THP07] and both synthesized on a Virtex 6 FPGA. The realizations for DAG fusion were created with the SPIRAL online tool [SPI13] that outputs synthesizable Verilog code. It can be seen that less resources (81% of the resources of DAG fusion on average) are required for the result of the specialized FPGA optimization algorithm. These results justify to put some effort into FPGA specific optimization of RSCM graphs.
A general observation of the experiments in 4.1, 4.2 and this section is that the complexity, execution time and resulting design size is heavily dependent on the particular coefficient set.

5. Next Steps to be Done

Some optimization ideas to reduce the number of evaluated graph combinations are already stated in [DKD07]. These inlcude the reduction of the search space by omitting some graphs of the graph table. But there are no strict statements in the text which parts of the search space can be neglected.

Table 2: Comparison of the slices required by DAG fusion and extended ReMB algorithm results

coef. set C from [THP07]	slices DAG fusion	slices ext. ReMB (proposed)
{362,392,473}	25	**20** (80%)
{1,2,4,8,15,26,50,162,256}	28	**16** (57%)
{39,150,196}	21	**18** (86%)
{39,45,41,47}	**11**	**11** (100%)

Another idea is to discard non-valid graph combinations. In the basic algorithm all possible graph combinations are generated and thrown out of the set of possible solutions at the evaluation of the score function. The proposed optimizations for the original algorithm have to be included in our extended algorithm, with some changes. Our idea is to reduce the search space by already rating the specific graphs in the graph tables of the coefficients with weighting functions. Graph tables which bring about non-valid graph combinations or have a low rating will not be included in the graph table of the specific coefficient and thus are not combined with graphs of other coefficients. This will lead to considerable reductions in the number of possible graph combinations which have to be stored and rated by the score function. Moreover the integration of negative coefficients will be included in our algorithm. The current version of our algorithm is written and executed in MATLAB. It generates the RSCM graph and outputs synthesizable VHDL code. The generation of the graph tables as well as the generation and evaluation of the graph combinations results in long runtimes. To obtain better runtime results the algorithm will be implemented in C++.

6. Conclusion

In this paper an algorithm from literature to generate FPGA optimized RSCMs was analysed and extended. The extension of the basic element used in the original algorithm to two basic elements with larger input sizes leads to a reduction of required basic elements of 18% on average. The new basic elements are based on 6-input LUTs which are available in recent FPGAs. Furthermore the convergence in terms of finding a valid solution could be ensured, which was not the case in the original algorithm. Besides this, the analysis of the complexity revealed some general drawbacks of the algorithm: For larger coefficient sets, there are too many graph combinations which have to be stored and scored. This characteristic already existed in the original algorithm and has to be eliminated by future work. The experimental results have shown that the basic element approach is an interesting method for FPGA specific optimization of RSCMs. The FPGA specific approach needed only 81% of the FPGA resources on average compared to the ASIC optimized RSCM algorithm DAG fusion. This reduction has to be confirmed by a larger set of examples. A future requirement is to reduce the drawbacks of the presented approach to make the algorithm also available for larger coefficient sets.

References

[BH91] Bull, D. R. and D. H. Horrocks: *Primitive Operator Digital Filters*. Circuits, Devices and Systems, IEE Proceedings G, 1991.

[CC09] Chen, J. and C. H. Chang: *High-Level Synthesis Algorithm for the Design of Reconfigurable Constant Multiplier*. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2009.

[DDK03] Demirsoy, S. S., A. G. Dempster, and I. Kale: *Design Guidelines for Reconfigurable Multiplier Blocks*. Circuits and Systems (ISCAS) Proceedings of the International Symposium on, 2003.

[DKD04] Demirsoy, S. S., I. Kale, and A. G. Dempster: *Efficient Implementation of Digital Filters Using Novel Reconfigurable Multiplier Blocks*. In *Signals, Systems and Computers. Conference Record of the Thirty-Eighth Asilomar Conference on*, 2004.

[DKD05a] Demirsoy, S. S., I. Kale, and A. G. Dempster: *Synthesis of Reconfigurable Multiplier Blocks: Part - II Algorithm*. Circuits and Systems (ISCAS) IEEE International Symposium on, 2005.

[DKD05b] Demirsoy, S. S., I. Kale, and A. G. Dempster: *Synthesis of Reconfigurable Multiplier Blocks: Part I - Fundamentals*. Circuits and Systems (ISCAS) IEEE International Symposium on, 2005.

[DKD07] Demirsoy, S. S., I. Kale, and A. G. Dempster: *Reconfigurable Multiplier Blocks: Structures, Algorithm and Applications*. Circuits, Systems and Signal Processing, 2007.

[DM94] Dempster, A. G. and M. D. Macleod: *Constant Integer Multiplication Using Minimum Adders*. In *Circuits, Devices and Systems, IEE Proceedings*, 1994.

[GDW02] Gustafsson, O., A. G. Dempster, and L. Wanhammar: *Extended Results for Minimum-Adder Constant Integer Multipliers*. Circuits and Systems (ISCAS). IEEE International Symposium on, 2002.

[SPI13] SPIRAL-Project: *http://www.spiral.net*, 2013.

[THP07] Tummeltshammer, P., J. C. Hoe, and M. Puschel: *Time-Multiplexed Multiple-Constant Multiplication*. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 2007.

[TN10] Thong, J. and N. Nicolici: *A Novel Optimal Single Constant Multiplication Algorithm*. In *Design Automation Conference (DAC), 47th ACM/IEEE*, 2010.

[VP07] Voronenko, Y. and M. Puschel: *Multiplierless Multiple Constant Multiplication*. Transactions on Algorithms (TALG), 2007.