Optimal Shift Reassignment in Reconfigurable Constant Multiplication Circuits

Konrad Moeller, Martin Kumm, Mario Garrido Gálvez and Peter Zipf

The self-archived postprint version of this journal article is available at Linköping University Institutional Repository (DiVA):

<http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-145732>

N.B.: When citing this work, cite the original publication.

Moeller, K., Kumm, M., Garrido Gálvez, M., Zipf, P., (2018), Optimal Shift Reassignment in Reconfigurable Constant Multiplication Circuits, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(3), 710-714. <https://doi.org/10.1109/TCAD.2017.2729467>

Original publication available at:

<https://doi.org/10.1109/TCAD.2017.2729467>

Copyright: Institute of Electrical and Electronics Engineers (IEEE)

<http://www.ieee.org/index.html>

©2018 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

Optimal Shift Reassignment in Reconfigurable Constant Multiplication Circuits

Konrad Möller*, Martin Kumm*, Mario Garrido†, Peter Zipf*

*Digital Technology Group
University of Kassel, Germany

Email: {konrad.moeller, kumm, zipf}@uni-kassel.de

†Department of Electrical Engineering (ISY)
Linköping University, Sweden
Email: mario.garrido.galvez@liu.se

Abstract—This paper presents a new method called optimal shift reassignment (OSR), used for reconfigurable multiplication circuits. These circuits consist of adders, subtractors, shifts and multiplexers. They calculate the multiplication of an input number by one out of several constants which can be selected dynamically during run-time. The OSR method is based on the idea that shifts can be placed at different positions along the circuit, while the calculated output constant stays the same. This differs from previous approaches, which were limited by the fact that all constants within the constant multiplier were forced to be odd. The OSR method subsequently releases this restriction. As a result, the number of required multiplexers in the circuit can be reduced. This happens when the shift reassignment aligns the shift values of different inputs of a multiplexer. Experimental results show multiplexer savings of up to 50 % and average savings between 11 % and 16 % using the OSR method compared to previous approaches.

I. INTRODUCTION

This paper contributes to the implementation of multiplication with constant coefficients. Constant multiplication is done using additions, subtractions and bit shifts only and is a well studied research field for single and multiple constant multiplication (SCM/MCM) covered in, e.g. [1]–[4]. A special feature of the constant multipliers considered here is that the output constant c can be switched between a limited predefined set of N constants during run-time. This is enabled by multiplexers which are embedded in the arithmetic data path. The resulting reconfigurable constant multiplier (RCM) performs the multiplication $c_i x$, where x is a fixed-point input and $i = 0 \dots N - 1$ are the selectable constant's indices. The generation of RCMs was thoroughly analyzed in prior work on so called reconfigurable/time-multiplexed constant multiplication [5]–[12]. As redundant partial circuits can be reused, RCMs were shown to be more hardware efficient than using generic multipliers as long as the number of required output constants is limited. Their optimization is important to realize hardware efficient run-time adaptable filters [8], [12], DCT/FFT implementations [13] as well as multi-stage filters for decimation or interpolation like polyphase FIR filters [10]. This paper presents a post-optimization of these RCMs to further reduce the number of required multiplexers. The background for this optimization and a discussion of related work is provided in Section II-B.

Fig. 1 shows two examples of such a reconfigurable constant multiplication for $12305x$ and $20746x$. Fig. 1 (a) shows an

Fig. 1: Example of the optimization of an original reconfigurable constant multiplier solution by DAG fusion [5].

original RCM solution obtained from the DAG fusion algorithm [5]. Fig. 1 (b) shows the optimized version using the proposed OSR approach. Both circuits consist of adders, bit-shifts and multiplexers to perform a reconfigurable constant multiplication. The scaling factors of the final result and the intermediate results are given as a column vector beside the respective adder. The upper value belongs to a multiplexer selection of 0, the lower one to the selection of 1. The highlighted part of the circuit in Fig. 1 (a) calculates $3x = x + 2^1x$ or $5x = x + 2^2x$, depending on the selected multiplexer input. While both circuits in Fig. 1 calculate the same output, they show a considerable difference in the number of required multiplexers.

Multiplexers are required if the shifted or unshifted inputs of adders for the different reconfigurable constants have different sources (cf. left input of second adder from top in Fig. 1 (a)). This architectural property of the RCM cannot be changed by a shift reassignment. Minimization of these multiplexers is, however, the main contribution of prior work [5]–[12]. Moreover, multiplexers are required to select different shift values if the inputs of adders for the different reconfigurable constants have the same source but different shift values (cf. right input of second adder from top in Fig. 1 (a)). This means,

it would be beneficial if most of the input shifts of the adders for the different reconfigurable constants were equal. Aligning already equal shifts is considered in prior work [5], [6]. There the shift values are adopted without modifications from odd fundamental graphs [1] as input, meaning that all intermediate constants are forced to be odd. This property is beneficial for multiplier-less single constant multiplication (SCM) and multiple constant multiplication (MCM) without reconfiguration, as it simplifies the optimization. However, it does not guarantee that the minimum number of required multiplexers is found when reconfigurable SCM or MCM is considered. Better results can be achieved by dropping this property to allow a reassignment of shift values within the circuit of each reconfigurable output constant. This can be seen in the distribution of shifts and the intermediate constants in Fig. 1.

The main contribution of this work is to show that allowing a shift reassignment regardless whether this leads to odd or even intermediate constants improves the resulting solutions. This is shown by an optimal shift reassignment using *Integer Linear Programming* (ILP) and is further discussed in Section III, followed by an experimental evaluation in Section IV. OSR is applicable subsequently to all previous solutions for multiplier-less RCM. This two-step process is still not leading to globally optimal solutions, but improves the state of the art considerably. A globally optimal solution could be achieved by including the OSR into the process of multiplier-less RCM generation. To the best of our knowledge OSR has not been considered and evaluated for the optimization of reconfigurable constant multiplication circuits so far.

II. BACKGROUND

A. Baseline for the Optimal Shift Reassignment

The proposed OSR is applied to already optimized reconfigurable constant multiplication circuits. For this paper, the solutions of Tummeltshammer et al.'s DAG fusion [5] are taken as the baseline. DAG fusion is a method to generate non-pipelined multiplier-less RCMs based on optimal SCM solutions taken from [2]. Optimal SCM means that the number of required adders in the input circuits taken from [2] is minimal. The base of this kind of multiplier-less multiplication is their composition of an addition of shifted inputs. Note that a constant shift can be hard-wired and is assumed to be realized at no cost. A formal representation of a constant is given as the so called $\mathcal{A}$ -operation [3]

$$\mathcal{A}_q(a_1, a_2) = |2^{l_1} a_1 + (-1)^\phi 2^{l_2} a_2| 2^{-r} \quad (1)$$

with $q = (l_1, l_2, r, \phi)$, where a_1 and a_2 are the input constants, and l_1 , l_2 and r are shift factors. The sign bit $\phi \in \{0, 1\}$ denotes whether an addition or subtraction is performed. Starting with an input of 1, all constant values can be built by first computing all possible outputs of the $\mathcal{A}$ -operation with input 1. Then, all these results and the input are used to determine further constants using the $\mathcal{A}$ -operation. This is repeated until the desired constant is reached. The result of this procedure can be represented as a so called adder graph G . By convention all intermediate constants of G are odd, as this reduces the search complexity and has no drawback as all

Fig. 2: Example fusion of two fragments $\text{Frag}(G_0)$ and $\text{Frag}(G_1)$.

even constants can be generated by left-shifting odd constants. Further details on actually generating optimal SCM and MCM circuits can be found in [1]-[4]. DAG fusion sequentially fuses these SCM circuits to a circuit computing $c_i x$ by inserting multiplexers. In this context, x is a fixed-point input value of a specified bit-width and c_i is one of N given fixed-point constants $\{c_0, c_1, \dots, c_{N-1}\}$ selected according to the $\lceil \log_2 N \rceil$ -bit control input i [5]. DAG fusion starts fusing two input adder graphs, G_0 and G_1 , optimally in terms of required multiplexers in an initial step to a RCM graph G_* . In the next step, the remaining input adder graphs G_i , if any, are fused optimally one after the other into G_* until all input graphs are included. An example of such a fusion for fragments of two adder graphs can be found in Fig. 2. The two fragments $\text{frag}(G_0)$ in Fig. 2 (a) and $\text{frag}(G_1)$ in Fig. 2 (b) can be fused in two ways. This is denoted as solution 1 in Fig. 2 (c) and solution 2 in Fig. 2 (d). Clearly, solution 1 is the best choice in this example.

For a sequential fusion of more than two input adder graphs, the order of consideration for fusion has an effect on the results [5]. Hence, all permutations of including the input circuits into the optimization have to be evaluated to find the best sequential fusion solution. As stated before, using the solutions of [2] for the fusion includes their nature of having odd intermediate constants only. This is a desired property for SCM circuits, which is also present in RCMs generated using DAG fusion. Optimality of the fusion steps is thus limited to this property, too. This is why the shift reassignment which is described in Section III can improve these results.

B. Further Related Work

Prior work on multiplier-less reconfigurable constant multiplication is separated into methods targeting field programmable gate arrays (FPGAs) and methods targeting application-specific integrated circuits (ASICs). While all methods result in adder graphs in which intermediate results are reused and reconfiguration is done by multiplexers (see Fig. 1), the construction of the RCM follows different methodologies. In [7], [8], [11] a basic computation kernel was defined which perfectly fits into the FPGA's logic. This kernel was used to generate larger RCM circuits. This was done by cascading the basic computation kernels based on given target output constants. The goal was to find the solution with the least number of required basic computation kernels. In [6] the idea of Tummeltshammer et al. [5] was adopted for pipelined RCMs on FPGAs. Introducing pipelining is essential to overcome long routing delays in FPGAs. Pipelined adder

graphs generated with RPAG [4] were fused optimally and by a heuristic [6] to reduce the number of reconfiguration multiplexers. It could be shown that this method is beneficial for the implementation of RCMs on FPGAs compared to pipelining the results of Tummelshammer et al. [5] as pipelining is considered already during optimization. Moreover, solutions for single and multiple output RCM were provided.

The principle of fusing multiplier-less constant multipliers using multiplexers was also applied for ASICs. The basic idea proposed by Chen and Chang [9] was to reduce the hardware costs for adders and multiplexers by using identical patterns in the canonical signed digit representation of the target constants when creating single output RCMs. A similar approach to Tummelshammer et al. [5] was used by Faust et al. [10], but special care was taken on keeping a minimal logic depth. Moreover, this approach can be used to generate RCMs with more than one output, which was a limitation in [5]. Another algorithm which is able to generate multiple output RCMs is ORPHEUS proposed by Aksoy et al. [12]. Their heuristic constructs the set of reconfigurable constants by a stepwise realization of reconfigurable intermediate constants beginning from the input.

All presented approaches have their main focus on minimizing adder costs in the construction of RCMs and multiplexer costs to do reconfiguration, but no special focus was given to the shift value selection. The results in Section IV show that a special focus on the shift values themselves can reduce the number of required multiplexers. Hence, this work contributes to the current state of the art of multiplier-less reconfigurable multiplication circuits.

III. OPTIMAL SHIFT REASSIGNMENT

A. Example for Shift Reassignment

In the example in Fig. 1, a reduction of 50% of required multiplexers can be seen. Obviously, this example is exceptionally good, therefore it is particularly suitable to explain the effects of a shift reassignment. The necessary steps to get the optimized circuit of Fig. 1(b) out of Fig. 1(a), which is repeated in Fig. 3(a), are described in the following. First, the left shift of 1 at the input of the output multiplexer of Fig. 3(a) is moved to the inputs of the previous adder's multiplexers within the circuit of the reconfigurable constant output $c_1 = 20746$. This changes the intermediate constant of adder 3 from 10373 to 20746 and eliminates the output multiplexer. To retain a valid circuit, the shift at the left input of adder 3 is changed from 7 to 8 and the shift at the right input of adder 3 is changed from 0 to 1 in the circuit of c_1 . The resulting circuit after this step can be seen in Fig. 3(b). The changes discussed so far are highlighted. In a second step, a left-shift of 3 (out of 4) at the output of adder 2 is moved to the inputs of the adder's multiplexers within the circuit of $c_0 = 12305$. The shift at the left input of adder 2 is changed from 0 to 3 in the circuit of c_0 . As another result, a shift of 1 remains after adder 2, but is now identical to the shift reassigned in the first step and no multiplexer is needed. At the same time, both shifts at the right input of adder 2 are 11 and the multiplexer can be eliminated. The resulting circuit is the one shown in Fig. 1(b).

Fig. 3: Original and intermediate result after step one for the optimization of the reconfigurable multiplier shown in Fig. 1.

B. Basic Rules for the Reassignment

The example in the last section showed that changing a shift of one multiplexer has an effect on other shifts in the circuit to keep the output valid. This results from a special property of addition, subtraction and bit-shift based constant multiplications: The sums of shifts on each path from the input to the output determine the resulting output constant.

More formally, let S_{ivk} be the shift in the circuit of constant c_i at input k of an adder with index v . In the following figures we use $k = 0$ for the left and $k = 1$ for the right input of the adder. In addition, the output shift is a special case in which k can be ignored and v is equal to the output y . Now, let $\mathcal{P}_{ip}$ be the set of all shifts on path p for constant c_i . The sum

$$\sigma_{ip} = \sum_{S \in \mathcal{P}_{ip}} S_{ivk} \quad (2)$$

of these shifts is a constant for each path p for constant c_i .

To give an example of our notation, Fig. 4 shows two realizations of the adder graph of c_0 in Fig. 1, with 4 paths from the input x to the output. Note that the shift of input 1 (right input) of adder 3 S_{031} is 4 in Fig. 4 (a) and 1 in Fig. 4 (b). However, the sums of shifts on each path are the constants $\sigma_{00} = 0$, $\sigma_{01} = 4$, $\sigma_{02} = 12$ and $\sigma_{03} = 13$ in both variants. The relation to the computed constant is

$$c_i = \sum_{p=0}^{P_i} \phi_{ip} 2^{\sigma_{ip}}, \quad (3)$$

where P_i is the total number of paths for constant c_i . The variable $\phi_{ip} \in \{-1, 1\}$ is the sign on path p , which is the product of all signs on that path. In the example of Fig. 4 this results in $c_0 = 2^0 + 2^4 + 2^{12} + 2^{13} = 12305$. This means, all distributions of shifts having the given path sums are valid solutions for c_i . That is why a shift distribution is possible. This is taken as the basis for the ILP formulation presented in the next section.

The topology of underlying adder graphs is not changed by the OSR. However, a change in shift values could increase the word size of adders and multiplexers. For the adders zeros are added, when intermediate results get even. Therefore, this does

Fig. 4: Two different adder graph representations of constant $c_0 = 12305$.

not increase the adder's implementation costs. This is also true for multiplexers in which all input shifts are equally increased. In case the difference between shift values in a multiplexer is changed by the OSR, the multiplexer has to switch between the signal input or zero for some additional bits. This case can be handled by a simple bitwise AND instead of a multiplexer for those bits [10]. So, the resulting hardware overhead in this case is very small.

C. ILP Formulation for Optimal Distribution of Shifts

The objective of the ILP formulation is the minimization of multiplexers for a given reconfigurable constant multiplier by selecting the best distribution of shifts. Optimization of multiplexers means optimizing the number of 2:1 multiplexers. A $k:1$ multiplexer can be realized by a tree of $k-1$ 2:1 multiplexers. This leads to a linear consideration of multiplexers during optimization, which was also used in previous work [5], [6], [9]. Therefore, the S_{ivk} variables defined in the last section are directly used as integer variables in the ILP formulation. The observation of the previous section was that the sum of shifts on each path σ_{ip} is a constant for a specific circuit computing c_i . Hence, instead of using the non-linear relation in (3), relation (2) can be used. This is directly represented in constraints C1 in the ILP formulation in Listing 1. To link the described multiplexer input usage to the shifts, binary variables s_{ivkb} are defined, which are 1 when $S_{ivk} = b$. Constraints C2 in Listing 1 define the relation between the binary variables for the shift of b and the integer shift value S_{ivk} . At the same time constraints C3 assure that only one of the shift binaries s_{ivkb} is 1 in the final solution to prevent ambiguities at the definition of the integer shift value in C2.

The next step is to link the corresponding multiplexer costs to a given shift distribution. As already introduced, multiplexers appear if the shifted input values for the different constants have different sources, or if the input shifts for the different constants have the same source but different shifts. Only the second case can be influenced by shift reassignment. Therefore, binary variables M_{uvkb} are defined, which are 1 in case a bit shift of b is set for the edge from adder u to input k of adder v . Note that the variables M_{uvkb} are independent from i . Thus, if the same bit shift b can be used at a specific input in several graphs G_i , less M_{uvkb} variables are 1. Constraints C4

Listing 1: ILP formulation for the optimal shift reassignment

$$\begin{aligned}
 & \min \sum_{u \rightarrow v \in G_*} \sum_{k=0}^1 \sum_{b=0}^{B_{\max}} M_{uvkb} \\
 & \text{subject to} \\
 & \text{C1: } \sum_{S \in \mathcal{P}_{ip}} S_{ivk} = \sigma_{ip} \quad \text{for all } v \text{ in } G_*, p = 0 \dots P_i, \\
 & \quad \quad \quad i = 0 \dots N-1, k \in \{0, 1\} \\
 & \text{C2: } \sum_{b=0}^{B_{\max}} s_{ivkb} b = S_{ivk} \quad \text{for all } v \text{ in } G_*, \\
 & \quad \quad \quad i = 0 \dots N-1, k \in \{0, 1\} \\
 & \text{C3: } \sum_{b=0}^{B_{\max}} s_{ivkb} = 1 \quad \text{for all } v \text{ in } G_*, \\
 & \quad \quad \quad i = 0 \dots N-1, k \in \{0, 1\} \\
 & \text{C4: } M_{uvkb} \geq s_{ivkb} \quad \text{for all edges } u \rightarrow v \text{ in } G_*, \\
 & \quad \quad \quad i = 0 \dots N-1, k \in \{0, 1\}, \\
 & \quad \quad \quad b = 0 \dots B_{\max}
 \end{aligned}$$

link the different shift values and different sources for different constants to the multiplexer input usage.

Following the definitions before, the sum of all M_{uvkb} set to 1 is identical to the sum of required multiplexer inputs. To minimize this sum is the objective of the ILP formulation in Listing 1. All existing edges in the fused graph G_* have to be considered, which includes the edges from the input to the first adder and the edges to the output.

IV. EXPERIMENTAL EVALUATION

The proposed OSR has been applied to solutions generated with DAG fusion [5]. We used their open source code [14] to produce our results. The 1500 analyzed benchmark constant sets were already used in [6] and can be found online [15] as constant set and text representation. It is a composition of 100 random constant sets each for 2 to 16 reconfigurable output constants. The results for the required 2:1 multiplexers (MUX) can be found in Table I. Each value is the average of 100 test cases. The comparison shows that on average 11-16 % less 2:1 multiplexers are required after the OSR.

Considering the fact that an average value can always contain outliers, which could obscure the real situation, we provide a detailed analysis of the achieved savings for the proposed OSR in Table II. This table shows the sum of the cases for which a certain number of 2:1 multiplexers can be saved compared to the original DAG fusion solution. Each column can be seen as a histogram of 100 values for a specific number of reconfigurable constants. The last column is the sum of each row of the benchmark. The sum of savings with the largest occurrence is marked in bold-face. For the cases with only few reconfigurable constants, the cases with large savings are rare, compared to the cases with no savings. This results from the fact that these rather small cases have little flexibility to reassign shifts. In cases with no savings, the original solution was optimal already or had equal multiplexer costs. For the more complex cases, it can be seen that with increasing number of reconfigurable constants, 2 to 5 multiplexers can be saved in the majority of cases and

TABLE I: Average number of 2:1 multiplexers (MUXs) for DAG Fusion [5] RSCM before and after the proposed optimal shift reassignment. Each value is the average of 100 test cases of a 1500 case benchmark taken from [6].

# sw. out. const.	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DAG fusion [5]	4.12	8.44	11.8	14.77	17.29	20.24	22.51	24.71	27.00	28.12	29.86	31.86	33.59	34.83	36.25
Proposed OSR	3.66	7.06	10.1	12.87	15.10	17.49	19.63	21.43	23.48	24.78	26.32	27.97	29.57	30.34	31.62
% decrease in 2:1 MUXs used	11.17	16.35	14.41	12.86	12.67	13.59	12.79	13.27	13.04	11.88	11.86	12.21	11.97	12.89	12.77

TABLE II: Number of cases in which a certain number of 2:1 multiplexers can be saved compared to the original DAG fusion solution using the proposed optimal shift reassignment.

		number of reconfigurable output constants																
		2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	sum	
saved multiplexers	0	64	26	20	13	20	7	5	6	5	6	2	2	1	1	-	178	
	1	27	35	26	31	12	12	15	11	8	9	10	6	8	2	7	219	
	2	8	22	29	29	28	26	23	20	17	20	17	18	13	13	7	290	
	3	1	11	16	10	23	23	25	16	24	22	25	20	17	14	15	262	
	4	-	4	8	15	8	19	18	24	19	17	16	16	23	22	18	227	
	5	-	2	0	1	5	12	6	11	12	10	16	17	13	23	24	152	
	6	-	-	1	1	3	1	6	6	6	14	10	15	18	10	12	103	
	7	-	-	-	-	1	-	2	6	7	0	1	2	4	8	10	41	
	8	-	-	-	-	-	-	-	-	1	1	3	2	3	3	3	16	
	9	-	-	-	-	-	-	-	-	1	1	-	2	-	4	2	10	
	10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	1	
	11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	1	

TABLE III: Relative resulting logic depth after OSR, considering multiplexers as tree of 2:1 multiplexers.

Relative resulting depth	+1	+0	-1	-2	-3	-4	-5	-6
number of cases	15	539	315	353	187	71	19	1

there are few cases without savings. This confirms that the absolute average numbers in Table I are meaningful. A saving of up to 11 out of 41 multiplexers is possible (see last row in Table II). At the same time, there was no 16 output case without savings. The run-time of the proposed optimization using the ILP solver Gurobi 7.0.1 [16] was below one second even for the largest cases in single thread mode on a 2.2 GHz Intel Core i7-4770HQ CPU. A speedup is possible by using more than one thread, which is supported by Gurobi.

Finally, the logic depth of the original and the optimized solutions was analyzed. While the adder depth is not affected by the OSR, the reassignment of 2:1 multiplexers could change the overall logic depth. In 63 % of the analyzed cases, OSR yields to a reduction in depth (-1 to -6 in TABLE III). The logic depth was increased in only 1 % of the cases after OSR. These cases result from an unfavorable distribution of 2:1 multiplexers at different inputs of the same adder.

V. CONCLUSION

We presented the optimal reassignment of shifts in reconfigurable constant multipliers using *Integer Linear Programming*. This was done to save additional multiplexer resources compared to previous work. It was shown that even though the original multiplier-less reconfigurable multiplication circuits were generated in an optimal way while preserving an odd

fundamental representation, improvements can be achieved by a redistribution of shifts within the original solutions. This results from the fact that changing the shifts to save further multiplexers has not been considered before. In doing so, this makes large absolute and average multiplexer savings between 11 % and 16 % possible. The shown post-optimization is applicable to all previous solutions for multiplier-less RCM [5]–[12]. The benchmarks used and the source code of the proposed approach are available as open source [15] to enhance reproducibility of the presented results and encourage future research.

REFERENCES

- [1] A. G. Dempster and M. D. Macleod, "Constant Integer Multiplication Using Minimum Adders," *Circuits, Devices and Systems, IEE Proceedings*, vol. 141, no. 5, pp. 407–413, Oct 1994.
- [2] O. Gustafsson, A. G. Dempster, and L. Wanhammar, "Extended Results for Minimum-Adder Constant Integer Multipliers," *IEEE Int. Symp. on Circuits and Systems*, vol. 1, pp. 1–73–1–76, Aug 2002.
- [3] Y. Voronenko and M. Püschel, "Multiplierless Multiple Constant Multiplication," *ACM Trans. Algorithms*, vol. 3, no. 2, May 2007.
- [4] M. Kumm, P. Zipf, M. Faust, and C.-H. Chang, "Pipelined Adder Graph Optimization for High Speed Multiple Constant Multiplication," in *IEEE Int. Symp. on Circuits and Systems*, May 2012, pp. 49–52.
- [5] P. Tummeltshammer, J. C. Hoe, and M. Püschel, "Time-Multiplexed Multiple-Constant Multiplication," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 9, pp. 1551–1563, Sep. 2007.
- [6] K. Möller, M. Kumm, M. Kleinlein, and P. Zipf, "Reconfigurable Constant Multiplication for FPGAs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 36, no. 6, pp. 927–937, Jun. 2017.
- [7] R. H. Turner and R. F. Woods, "Highly Efficient, Limited Range Multipliers for LUT-Based FPGA Architectures," *IEEE Trans. VLSI Syst.*, vol. 12, no. 10, pp. 1113–1118, Oct 2004.
- [8] S. S. Demirsoy, I. Kale, and A. G. Dempster, "Reconfigurable Multiplier Blocks: Structures, Algorithm and Applications," *Circuits, Systems and Signal Processing*, vol. 26, no. 6, pp. 793–827, Jan 2008.
- [9] J. Chen and C. H. Chang, "High-Level Synthesis Algorithm for the Design of Reconfigurable Constant Multiplier," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 12, pp. 1844–1856, Dec 2009.
- [10] M. Faust, O. Gustafsson, and C.-H. Chang, "Reconfigurable Multiple Constant Multiplication Using Minimum Adder Depth," in *Conf. Rec. 44th Asilomar Conf. Sign., Syst. and Comput.*, Nov 2010, pp. 1297–1301.
- [11] K. Möller, M. Kumm, B. Barschtipan, and P. Zipf, "Dynamically Reconfigurable Constant Multiplication on FPGAs," in *Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV)*, Mar 2014, pp. 159–169.
- [12] L. Aksoy, P. Flores, and J. Monteiro, "Optimization of Design Complexity in Time-Multiplexed Constant Multiplications," in *IEEE Design Automation and Test in Europe*, March 2014, pp. 1–4.
- [13] M. Garrido, F. Qureshi, and O. Gustafsson, "Low-Complexity Multiplierless Constant Rotators Based on Combined Coefficient Selection and Shift-and-Add Implementation (CCSSI)," *IEEE Trans. Circuits Syst. I*, vol. 61, no. 7, pp. 2002–2012, July 2014.
- [14] SPIRAL-Project. (2016) <http://www.spiral.net>.
- [15] M. Kumm, K. Möller, and P. Zipf, "PAGSuite Project Website," 2016. [Online]. Available: <http://www.uni-kassel.de/go/pagsuite>
- [16] Gurobi Optimization, Inc., "Gurobi Optimizer Reference Manual," 2016. [Online]. Available: <http://www.gurobi.com>