Multiple Constant Multiplication: From Target Constants to Optimized Pipelined Adder Graphs

Garcia Rémi and Volkova Anastasia
Nantes Université, École Centrale Nantes,
CNRS, LS2N, UMR 6004
F-44000 Nantes, France
surname.lastname@univ-nantes.fr

***Abstract*—Multiple Constant Multiplication (MCM) is a ubiquitous problem for numerous computation-intensive applications. A standard and efficient approach is to replace generic multipliers by multiplierless architectures based on bit-shifts and additions. The adder graphs describing the multiplierless circuits can be optimized according to various metrics, in particular improving throughput by pipelining. In this paper, we improve the state-of-the-art for the design of pipelined adder graphs by searching for an optimal solution directly from target coefficients. In contrast to existing approaches, which are based on fixed adder graphs or heuristics, our solution is to describe the complete design space with Mixed-Integer Linear Programming (MILP). This results in an optimization model that can be solved to find optimal adder graphs and prove the optimality of the solutions thanks to the efficiency of modern MILP solvers. Mathematical modeling allows for an easily extendable tool which can target FPGAs and ASICs and adapt to evolving hardware models/metrics. With this work, we provide a modular framework to automate the design of adder graph-based circuits with exact optimization. The experiments demonstrate efficiency of our approach which fuses multiple design steps into a single problem.**

***Index Terms*—Adder graph, optimal design, multiplierless hardware, integer linear programming**

I. Introduction

Many numerical algorithms and applications involve multiplications by integer constants. Fixed-Point (FxP) numbers, which can be assimilated to integers, are the preferred choice for algorithms targeting embedded systems such as Field-Programmable Gate Arrays (FPGAs).

While generic multipliers perfectly handle these multiplications by constants, their cost largely exceeds multiplierless architectures that benefit from the knowledge of constants' values [1]. The shift-and-add approach is hence the privileged method to reduce hardware cost. It consists in replacing multiplications by bit-shifts, additions and subtractions where bit-shifts are multiplications by a power of two, that can be hard-wired for a negligible cost. For example, multiplying an integer variable x by the constant 7 can be rewritten as $7x = 2^3x - x$, reducing the cost to a single bit-shift by three positions to the left and a subtraction, instead of a multiplication.

The Multiple Constant Multiplication (MCM) problem consists in finding the implementation with the lowest cost that achieves the multiplication by a given set of target constants to multiply with. The ultimate goal would be to search for the lowest cost by considering the final hardware metrics. Yet, it is currently unrealistic and the typical way is to use proxy metrics, such as the number of adders, to estimate the synthesized hardware cost [2]–[7].

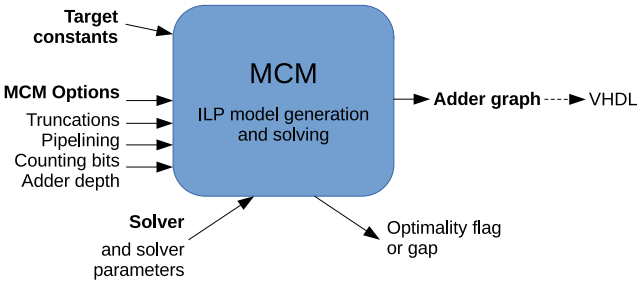


Fig. 1: Our tools takes target constants and multiple options to optimize for different MCM problem flavors

The hardware cost is actually manifold and while most higher-level metrics try to target those as a whole, in this work we put a specific focus on delay and throughput by considering pipelining. While one can simply pipeline the optimal solutions to the MCM problem [8], this is not as efficient as adding registers into the design approach and searching directly for the best multiplierless topologies minimizing the number of registers [5]. With this work, we propose a new Integer Linear Programming (ILP) based model that solves the Pipelined MCM problem (PMCM). This new model is incorporated into the `jMCM` tool providing solutions to different flavors of the MCM problem. In particular, we explore, for the first time, pipelining in the MCM targetting the low-level hardware metric that counts the number of one-bit adders.

The easy-to-use and open-source tool in Fig. 1 generates MCM solutions as adder graphs, and can be used to automatically generate the VHDL code. We demonstrate, on a large set of benchmarks, that our optimal approach provides solution that need 17.57% less adders and registers, on average, than the state-of-the-art heuristic RPAG [5].

II. State of the art

The MCM problem has been studied for decades and first solutions [9], [10] are based on the Canonical Signed Digit (CSD) representation. This consists in the rewriting of the number using a signed digit representation in a Non-Adjacent Form (NAF), *e. g.*, $23x = 0010\ \bar{1}00\bar{1}_{\text{CSD}} \times x$ replaces the

multiplication by 2 additions, as illustrated by an adder graph in Fig. 2a. In an adder graph, the number in nodes is called a fundamental and denotes a constant the input is multiplied with, and labels on edges denote either the negation, or an integer power of two the signal is multiplied with. However, CSD does not always provide optimal results and does not give any guarantee on optimality. Moreover, since CSD is applied on each constant separately, resource sharing should be explored by additional algorithms.

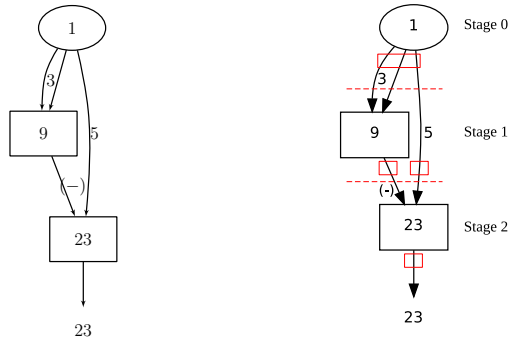
Many heuristics enhance the results obtained with the CSD method for the MCM problem [2], [11]–[14], and more recently optimal approaches have been developed [3], [4], [6], [7], [15].

The first optimal methods were based on dedicated algorithms relying on hypergraphs [3] or branch-and-bound [4]. Both methods were not easily extensible and an ILP based model [6] which could solve either the MCM or PMCM problems has been proposed to benefit from the versatility of mathematical modeling. A simpler model [7], which did not rely on heavy precomputations, was used more recently and has been adapted [16] to find solutions for the single constant multiplication using SAT/SMT solvers.

All these methods have in common a focus on the number of adders. Finer-grain metrics, such as the number of one-bit adders, the adder depth or the glitch path count, have first been tackled heuristically [2], [5]. The recent ILP-based optimal approach [15] proposes a new MCM model that efficiently encodes the adder graph topology and various metrics as ILP constraints, such that no precomputations are required and the model is easily extensible for different optimization criteria. In particular, [15] proposes a new model that minimizes the number of one-bit adders, *i. e.* solving the, as we shall call it, MCM-Bits problem. A useful functionality is the minimization of the number of cascaded adders, called the *adder depth* (AD), as a secondary objective to the minimization of the main adder/one-bit adder metric.

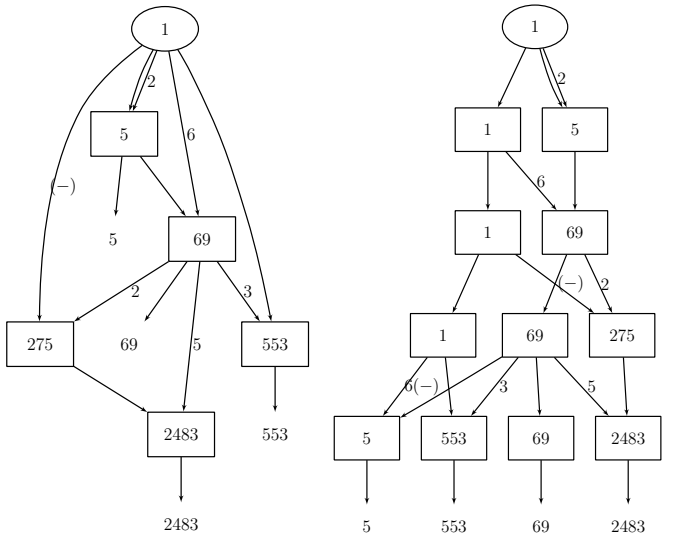
In order to obtain a higher throughput, designers often consider pipelined adder graphs, *i. e.*, adder graphs with registers before each stage [5], [6], [8], [17]. Pipelining of a fixed adder graph, for example coming from the MCM-Adders or MCM-Bits, can be obtained by performing cut-set retiming (CSR) [18]. For example, to pipeline the stages of the optimal adder graph for $23x$, shown in Fig. 2a, one must put a register at the input, one register after each adder and one register for the fundamental 1 on stage 1, as shown in Fig. 2b. When implementing on FPGAs, the registers that follow adders basically come for free, since LUTs have a register. It is worth to mention that pipelining the adders instead of the *adder stage* is possible and efficient for ASICs design [19], but we leave this option out of the scope of our paper.

As it was shown in [8], adapting the adder graph topology specifically for *pipelined* implementation yields designs with a lower cost. The RPAG tool [5] provides heuristic solutions to the problem and [6] gives first optimal ILP-based approach for PMCM. The idea of Kumm [6] is to solve a satisfaction problem, given a bound on the number of adders and on the adder depth, in which a full enumeration of the design space



(a) Before pipelining (b) With registers

Fig. 2: Pipelining adder stages



(a) Adder graph before pipelining (b) Pipelined adder graph with two adders computing the same value

Fig. 3: Increasing the number of adders to reduce the pipelined adder graph overall cost

is performed to construct the ILP problem. Pipelining is then ensured by a constraint that the adder's inputs on stage n should come exclusively from the preceding stage $n-1$, and not from earlier stages. The drawback of this approach is a strong limitation in the number of possible stages, due to heavy precomputations. Moreover, the ILP models by Kumm do not support easily combined metrics and constraints.

Hence, in the following we will use the versatile framework proposed in [15] and extend it to solve the PMCM problem. Our goal is to present a few sets of variables and constraints which permit to take the register cost into account.

III. Pipelined Multiple Constant Multiplication

In this section, we propose to extend the ILP-based models presented in [15] which permit to solve the MCM-Adders, optimizing for the adder count, and the MCM-Bits optimizing the number of one-bit adders, and add pipelining as a constraint.

Let us first make the following observation. The solution of the MCM-Adders problem is guided by the number of adders,

and naturally, each fundamental (the constant associated with the adder) should be unique in the adder graph. However, when incorporating the pipelining, it is the length of the connection between an adder and the maximum stage it interacts with, which is guiding the topology. Hence, it sometimes might be beneficial to compute the same fundamental twice, as it saves registers. Consider the adder graph in Fig. 3, computing $5x$, $69x$, $553x$ and $2483x$. Here, we need to add four registers after the adder computing $5x$ (if the adder graph is implemented in FPGAs, the register right after the adder comes for free). However, with the tool that we present below, we can find the adder graph in Fig. 3b, which computes $5x$ on stage 1, and then on stage 4. This permits to reduce the cost to 2 adders and 2 registers (or simply 2 adders, when implementing on FPGA).

Hence, we are first modifying the base model for MCM-Adders and MCM-Bits to allow for multiple occurrences of the same fundamental, if they are computed on different stages.

Now, our overall idea is to avoid modeling the registers explicitly, as entities similar to adders, but to compute the register cost for each adder in the following way. For each adder $a \in \{a_0, a_1, \ldots, a_{\overline{N_A}}\}$, where a_0 is the input and $\overline{N_A}$ is the bound on the number of used adders, we can extract the stage when they are computed, s_a, and the maximum stage they interact with as a direct input, $\overline{s}_a$. For example, in Fig. 3b, for the initial input the stage $s_{a_0} = 0$ and the maximum stage it interacts with is $\overline{s}_{a_0} = 4$.

Then, the cost r_a of each adder a in terms of registers is directly computed as

$$r_a = \overline{s}_a - s_a. \tag{1}$$

Hence, for the input in Fig. 3b, the register cost is $r_{a_0} = 4 - 0$.

With these observations, we can expect that the optimal PMCM might have more adders than the solutions by MCM-Adders and MCM-Bits. In Section IV, we illustrate that it indeed occurs.

We use as basis the ILP models from [15] and below report only the modifications to them.

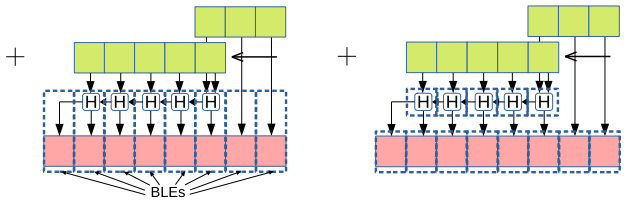
A. PMCM-Adders

The ILP from [15], given the target coefficients, first computes the upper bound on the minimum number of adders $\overline{N_A}$. Then, the solver searches the fundamentals associated with each adder $a \in \{a_0, a_1, \ldots, a_{\overline{N_A}}\}$, such that the topology abides to the rules of adder graphs and computes the target coefficients, such that the cost function is minimized.

Denote $s_a \in \mathbb{N}$ to be the stage, on which the adder a is computed. Then, for all adders a and stages s we add the binary variable $\overline{s}_{a,s}$, such that $\overline{s}_{a,s} = 1$ if adder a is used as an input of stage s. The maximum number of stages can be computed *a priori* using the bound $\overline{N_A}$ on the adder count. Then, in order to compute the maximum stage with which the adder a interacts with, $\overline{s}_a \in \mathbb{N}$, we add the constraints

$$\overline{s}_a \geq s \times \overline{s}_{a,s}, \qquad \forall a, s, \tag{2}$$

Then we simply introduce the equation (1) to count the register cost of each adder.



(a) One-bit adder outputs, carry outputs and go-through outputs are all BLEs.

(b) One-bit adders and flip-flops are independent.

Fig. 4: Counting resources on (a) FPGA and (b) ASICS

To minimize the number of registers, the objective function should be adjusted accordingly. In the original models, the objective function minimizes the sum of actually used adders, which are indicated by binary variables u_a, that are true if the adder a is used. We use this objective function as a basis and add the register cost into the weighted sum representing the total PMCM cost:

$$\min \; cost_a \sum_{a=1}^{\overline{N_A}} u_a + cost_r \sum_{a=1}^{\overline{N_A}} r_a, \tag{3}$$

where $cost_a$ and $cost_r$ are the weights of the adder and register costs, respectively.

When targeting FPGAs, we will consider that adders come with a free register, hence we have $cost_a = 0$. On ASICs, we can roughly estimate that the cost of registers is the same as the cost of adders. This objective function with the original MCM-Adders model and the new variables and constraints permits to solve the PMCM-Adders problem.

B. PMCM-Bits

When the input data word length is *a priori* known, targeting the MCM-Bits problem leads to hardware with lower cost [15]. We propose to solve the PMCM-Bits problem by building upon the existing model for MCM-Bits.

In the original model, the most significant bit (MSB) propagation of the data is already encoded by integer variables for MSB propagation while the least significant bit (LSB) is fixed to zero. For simplicity, we consider the integer variable wl_a corresponding to the data word length after adder a. Then, the number of one-bit registers, r_a^b, required for each adder is

$$r_a^b = wl_a \times r_a. \tag{4}$$

This leads to a finer-grain objective function:

$$\min \; cost_b \sum_{a=1}^{\overline{N_A}} b_a + cost_{r_b} \sum_{a=1}^{\overline{N_A}} r_a^b, \tag{5}$$

where b_a is the number of one-bit adders of adder a, and $cost_b$ and $cost_{r_b}$ are the cost of one-bit adders and one-bit registers, respectively. On ASICs, one-bit adders could be specifically half or full adders with a slightly different cost but [15] did not provide different counts for these one-bit adders.

TABLE I: Comparison of our results to RPAG [5]. Here #A, #B, #R and #R$_b$ denote the number of adders, one-bit adders, registers, and one-bit registers, respectively. C_1 =#A+#R and C_2 =#B + #R$_b$. Numbers in bold indicate best results.

Bench	RPAG						PMCM-Adders						PMCM-Bits					
	#A	#R	C_1	#B	#R$_b$	C_2	#A	#R	C_1	#B	#R$_b$	C_2	#A	#R	C_1	#B	#R$_b$	C_2
G.3	4	2	6	43	25	68	4	1	**5**	43	15	**58**	4	1	**5**	43	15	**58**
G.5	6	2	**8**	69	34	**103**	6	2	**8**	69	34	**103**	6	2	**8**	69	34	**103**
HP15	12	7	19	122	104	226	12	3	**15**	122	56	**178**	12	3	**15**	122	56	**178**
HP5	4	4	8	42	45	87	4	2	**6**	42	24	**66**	4	2	**6**	42	24	**66**
HP9	5	4	9	50	49	99	5	2	**7**	50	27	**77**	5	2	**7**	50	27	**77**
L.3	4	2	6	45	24	69	4	1	**5**	45	16	**61**	5	0	**5**	58	3	**61**
LP15	27	26	53	306	434	740	27	9	**36**	304	191	**495**	27	10	37	306	201	507
LP5	7	2	9	75	40	115	7	1	**8**	68	30	**98**	8	1	9	85	25	110
LP9	13	6	19	152	102	254	15	3	18	167	68	235	13	4	**17**	145	81	**226**
U.3-1	4	2	6	32	34	66	4	1	**5**	38	18	**56**	4	1	**5**	35	21	**56**
U.3-2	6	3	9	62	50	112	5	2	**7**	60	33	93	5	2	**7**	59	32	**91**
Total:	92	60	152	998	941	1939	94	27	**121**	1025	508	**1533**	94	29	123	1029	517	1546

In details, each basic logic element (BLE) used in modern FPGAs consists of at least one one-bit adder and an optional output register. Thus, not the whole adder output comes with free registers but only the output bits computed by a one-bit adder as illustrated in Fig. 4a. The carry outputs or the go-through output bits will have to be computed using the BLEs as well, hence the cost improvement that we obtain with PMCM-Bits might not be as interesting as for MCM-Bits whose goal is to save one-bit adders. As a consequence, we assume that the one-bit adders come for free in (5), *i.e.* $cost_b = 0$, and focus on counting register bits.

When targeting ASICs, each one-bit adder and flip-flop (FF) are independent. No register comes for free but one-bit adders do not have the same cost as BLEs, as they do not possess an integrated FF. Fig. 4b illustrates elements to consider when computing the total circuit cost. On ASICs, we will roughly estimate that the cost of registers is the same as the cost of one-bit adders.

Previous works [7], [15] have shown that the adder depth and/or the glitch path count can be considered with bounds and/or as a second objective. Yet, pipelining makes both metrics of less interest and the critical path would be a better high-level metric to consider. The critical path of the pipelined adder graph is determined by the adder with the largest number of one-bit adders. Using ILP modeling, it is straightforward to store this value into $B_{\max}$, an integer variable,

$$B_{\max} \geq b_a, \qquad \forall a \in [\![1; \overline{N_A}]\!]. \tag{6}$$

Then, the objective function can be modified as follows to minimize the critical path as a second objective,

$$\min \; M \times \left(cost_b \sum_{a=1}^{\overline{N_A}} b_a + cost_{r_b} \sum_{a=1}^{\overline{N_A}} r_a^b \right) + B_{\max}, \tag{7}$$

where M is fixed to a known upper bound of $B_{\max}$ such as the maximum data word length in the adder graph, in order to ensure that the first term of the cost function is optimized first, and $B_{\max}$ second.

TABLE II: A posteriori pipelining of MCM-Bits solution vs. one-step optimization with PMCM-Bits

Bench	MCM-Bits			PMCM-Bits		
	#B	#R$_b$	#B + #R$_b$	#B	#R$_b$	#B + #R$_b$
G.3	40	28	68	43	15	**58**
G.5	57	55	**112**	84	32	116
HP15	105	230	335	122	56	**178**
HP5	39	77	116	42	24	**66**
HP9	47	41	88	50	27	**77**
L.3	31	59	90	58	3	**61**
LP15	250	688	938	306	201	**507**
LP5	60	89	149	85	25	**110**
LP9	137	218	355	145	81	**226**
U.3-1	32	34	66	35	21	**56**
U.3-2	49	85	134	59	32	**91**
Total:	847	1604	2451	1029	517	**1546**

IV. Tool and experiment results

The flow of the jMCM tool is summarized in Fig. 1 and can be easily extended to include more high-level options similarly to our approach with the critical path. This tool is implemented in julia using the modeling language JuMP [20], so any generic MILP solver supported by JuMP can be used for the solving process as backend. The tool is open-source and freely available on git[1].

With our tool, embedded system designers can provide target coefficients together with a just few high-level options, such as the input/output word length. Then, an ILP model, adjusted accordingly to the options, will be automatically generated and solved. This generates adder graphs that can be used to generate the VHDL code, for example using the FloPoCo [21] code generator.

We applied our tool to benchmarks from image processing (11 instances) [22] and from the whole FIRsuite project (75 instances) [23], which is a collection of finite impulse response digital filters. Generated models were solved with

[1]The link is blinded for the review.

Gurobi [24], with a time limit of 30 minutes and 4 threads from 2.1 GHz CPUs. When solving the PMCM-Bits problem, input word length is fixed at 8 bits as it is a logical choice for the image-processing benchmarks. Detailed results are given only for the image processing benchmark while statistics integrate all 86 benchmarks.

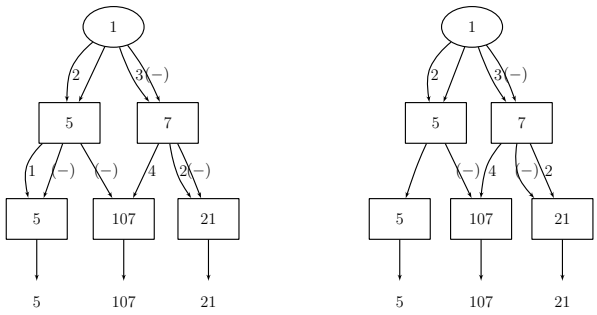
In [5] it was shown that solutions obtained with the RPAG heuristic require, on average, a lower number of adders plus registers than pipelining fixed adder graphs. In TABLE I, the state-of-the-art RPAG results are compared with our optimal approach for the PMCM-Adders problem showing that `jMCM` leads to adder graphs with a lower high-level cost than RPAG, by 17.57% on average. In details, the number of adders is increased by 1.86% allowing to reduce the number of registers that are not associated to an adder by 49.26%.

We observed a trade-off between the number of adders and the number of registers which discourages using tight bounds on the number of adders when solving the PMCM problem. We solved the PMCM-Adders problem with the smallest upper bound on the number of adders we could obtain and with this upper bound arbitrarily increased by two. For 11 instances, the larger upper bound permitted to obtain lower cost adder graphs.

On average, compared to solving the PMCM-Adders problem, no cost reduction is achieved by solving the PMCM-Bits problem. We observed that solving PMCM-Bits is more challenging for the solver, which translates into obtaining, in rare cases, worse solutions than PMCM-Adders. Our hypothesis is that, for hard instances, solving the PMCM-Adders problem leads to an adder and register reduction that is not reached by solving the PMCM-Bits within the available time. On the other hand, this might be explained in the following way: the interest of MCM-Bits is to count the number of one-bit adders and look for corner cases that permit to avoid using them, but it is somewhat leveled by the register bit count, which always depends on the word length of the adder output. Though we did observe one case, benchmark `U.3-2`, where PMCM-Bits found a better adder graph than PMCM-Adders.

The state-of-the-art RPAG heuristic demonstrated [5] that solving the PMCM as one step is better, w.r.t. the number of adders and registers, than *a posteriori* pipelining an optimal adder graph. The results presented in TABLE II show that this principle holds for the fine-grained metric of counting one-bit adders and one-bit registers. By solving the PMCM-Bits problem, instead of pipelining *a posteriori* the MCM-Bits solution, we achieve a 33.15% cost reduction in terms of number of one-bit registers, on average.

We observed that sometimes the solver provides solutions in which adders could have been replaced by registers such as in Fig. 5a. When targeting FPGA, this has the same cost as the register used in Fig. 5b since the adder comes for free. Hence, the solver will not differentiate between these solutions. Hardware costs, such as power consumption, would however differ. These unnecessary adders can be replaced by registers with a trivial postprocessing that `jMCM` supports.



(a) Adder is used instead of a register

(b) Register can be used

Fig. 5: Postprocessing replaces unnecessary adders with same-cost registers

V. Conclusion and Perspectives

With this work, we proposed an optimal approach to incorporating pipelining into the ILP-based design of multiplierless MCM. Our results, evaluated on a large set of benchmarks, demonstrate the superiority of *optimal* approach compared to the state-of-the-art heuristic RPAG. In contrast to previous approaches, we also incorporate the pipelining into a model that targets a low-level metric, based on counting one-bit adders. Our investigation reveals that an optimizing in one step, *i. e.* PMCM-Bits, is better than *a posteriori* pipelining of MCM-Bits solutions, confirming the same trend as with counting adders. However, in the allocated time, PMCM-Bits design exploration is rarely revealing better results than the high-level metric-based PMCM-Adders.

With this work we demonstrate the versatility and power of the ILP-based approach: one can rely on the `jMCM` tool and, by simple modification of constraints/cost function, automatically design multiplierless constant multiplication circuits that target different metrics. This new design automation tool permits to alleviate time and effort embedded system designers put into the low-level data path optimization and fine-tuning, and let them focus on more high-level tasks related to their implemented algorithm.

In future work, we plan to either provide a custom VHDL code-generator for the target MCM/PMCM circuits, or extend the FloPoCo tool. We also plan on making the last adjustments to permit to tackle truncated pipelined adder graphs. More extensions to the `jMCM` tool are awaited, in particular by including DSPs into solving the MCM problem when targeting FPGAs. In particular, we plan to automate the (pipelined) MCM design in presence of a certain budget of DSP blocks.

References

[1] K. Wiatr and E. Jamro, "Constant coefficient multiplication in FPGA structures," in *Proceedings of the 26th Euromicro Conference. EUROMICRO 2000. Informatics: Inventing the Future*. IEEE Comput. Soc, Sep. 2000.

[2] A. G. Dempster and M. D. Macleod, "Constant integer multiplication using minimum adders," *IEE Proceedings - Circuits, Devices and Systems*, vol. 141, no. 5, pp. 407–413, Oct. 1994.

[3] O. Gustafsson, "Towards optimal multiple constant multiplication: A hypergraph approach," in *2008 42nd Asilomar Conference on Signals, Systems and Computers*, Oct. 2008, pp. 1805–1809.

[4] L. Aksoy, E. O. Güneş, and P. Flores, "Search algorithms for the multiple constant multiplications problem: Exact and approximate," *Microprocessors and Microsystems*, vol. 34, no. 5, pp. 151–162, Aug. 2010.

[5] M. Kumm, P. Zipf, M. Faust, and C.-H. Chang, "Pipelined adder graph optimization for high speed multiple constant multiplication," in *2012 IEEE International Symposium on Circuits and Systems*. IEEE, May 2012.

[6] M. Kumm, *Multiple Constant Multiplication Optimizations for Field Programmable Gate Arrays*. Wiesbaden: Springer Fachmedien Wiesbaden, 2016.

[7] ——, "Optimal Constant Multiplication Using Integer Linear Programming," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 5, pp. 567–571, May 2018.

[8] M. Kumm and P. Zipf, "High speed low complexity FPGA-based FIR filters using pipelined adder graphs," in *2011 International Conference on Field-Programmable Technology*. IEEE, Dec. 2011.

[9] A. D. Booth, "A Signed Binary Multiplication Technique," *The Quarterly Journal of Mechanics and Applied Mathematics*, vol. 4, no. 2, pp. 236–240, 1951.

[10] R. Bernstein, "Multiplication by integer constants," *Software: Practice and Experience*, vol. 16, no. 7, pp. 641–652, Jul. 1986.

[11] M. Potkonjak, M. B. Srivastava, and A. P. Chandrakasan, "Multiple constant multiplications: efficient and versatile framework and algorithms for exploring common subexpression elimination," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 2, pp. 151–165, Feb. 1996.

[12] V. Lefèvre, "Multiplication by an Integer Constant," INRIA, Research Report RR-4192, May 2001. [Online]. Available: https://hal.inria.fr/inria-00072430

[13] Y. Voronenko and M. Püschel, "Multiplierless multiple constant multiplication," *ACM Transactions on Algorithms*, vol. 3, no. 2, p. 11, May 2007.

[14] J. Thong and N. Nicolici, "Combined optimal and heuristic approaches for multiple constant multiplication," in *2010 IEEE International Conference on Computer Design*, Oct. 2010, pp. 266–273.

[15] R. Garcia and A. Volkova, "Towards the Multiple Constant Multiplication at Minimal Hardware Cost," Sep. 2022, working paper or preprint. [Online]. Available: https://hal.archives-ouvertes.fr/hal-03784625

[16] V. Lagoon and A. Metodi, "Deriving Optimal Multiplication-by-Constant Circuits With A SAT-based Constraint Engine," ModRef 2020 – The 19th workshop on Constraint Modelling and Reformulation, Sep. 2020.

[17] U. Meyer-Baese, J. Chen, C. H. Chang, and A. G. Dempster, "A Comparison of Pipelined RAG-n and DA FPGA-based Multiplierless Filters," in *APCCAS 2006 - 2006 IEEE Asia Pacific Conference on Circuits and Systems*. IEEE, dec 2006.

[18] K. K. Parhi, *VLSI digital signal processing systems*. Wiley, 1999.

[19] X. Lou, P. K. Meher, and Y. J. Yu, "Fine-grained pipelining for multiple constant multiplications," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, May 2015.

[20] I. Dunning, J. Huchette, and M. Lubin, "JuMP: A Modeling Language for Mathematical Optimization," *SIAM Review*, vol. 59, no. 2, pp. 295–320, May 2017.

[21] F. de Dinechin and B. Pasca, "Designing Custom Arithmetic Data Paths with FloPoCo," *IEEE Design and Test of Computers*, vol. 28, no. 4, pp. 18–27, Jul. 2011.

[22] M. Kumm, D. Fanghänel, K. Möller, P. Zipf, and U. Meyer-Baese, "FIR filter optimization for video processing on FPGAs," *EURASIP Journal On Advances in Signal Processing*, vol. 2013, no. 1, May 2013.

[23] FIRsuite, "Suite of Constant Coefficient FIR Filters," 2021, Accessed: Jan., 2022. [Online]. Available: http://www.firsuite.net

[24] Gurobi Optimization, "Gurobi Optimizer Reference Manual," 2020. [Online]. Available: https://www.gurobi.com/