Towards the Multiple Constant Multiplication at Minimal Hardware Cost

Rémi Garcia, Anastasia Volkova

Towards the Multiple Constant Multiplication at Minimal Hardware Cost

Rémi Garcia and Anastasia Volkova

***Abstract*—Multiple Constant Multiplication (MCM) over integers is a frequent operation arising in embedded systems that require highly optimized hardware. An efficient way is to replace costly generic multiplication by bit-shifts and additions, *i. e.* a multiplierless circuit. In this work, we improve the state-of-the-art optimal approach for MCM, based on Integer Linear Programming (ILP). We introduce a new lower-level hardware cost, based on counting the number of one-bit adders and demonstrate that it is strongly correlated with the LUT count. This new model for the multiplierless MCM circuits permitted us to consider intermediate truncations that permit to significantly save resources when a full output precision is not required. We incorporate the error propagation rules into our ILP model to guarantee a user-given error bound on the MCM results. The proposed ILP models for multiple flavors of MCM are implemented as an open-source tool and, combined with the FloPoCo code generator, provide a complete coefficient-to-VHDL flow. We evaluate our models in extensive experiments, and propose an in-depth analysis of the impact that design metrics have on actually synthesized hardware.**

***Index Terms*—multiple constant multiplication, multiplierless hardware, ILP, datapath optimization**

I. Introduction

Multiplications by integer constants arise in many numerical algorithms and applications. In particular, algorithms that target embedded systems often involve Fixed-Point (FxP) numbers which can be assimilated to integers. These algorithms range from dot-product evaluation for deep neural networks to more complex algorithms such as digital filters.

In order to save hardware resources, the knowledge on the constants to multiply with can be used in dedicated multiplierless architectures, instead of costly generic multipliers [1]. The shift-and-add approach is the privileged method to reduce hardware cost, it consists in replacing multiplications by additions/subtractions and bit-shifts, which are multiplications of the data by a power of two that can be hardwired 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.

Given a set of target constants to multiply with, finding the implementation with the lowest cost is called the Multiple Constant Multiplication (MCM) problem. Typical way to tackle the problem is to find shift-and-add implementations represented using *adder graphs*, as in Fig. 1, that describe the multiplierless solutions with the *minimum number of adders*. In the following, this problem will be referred to as the MCM-Adders. The main objective of this work is to first improve the existing approaches for the MCM-Adders problem, then push towards finer-grained hardware cost metrics counting number of *one-bit adders* (the MCM-Bits problem), and finally, use truncations in internal data paths to considerably save resources (the tMCM problem).

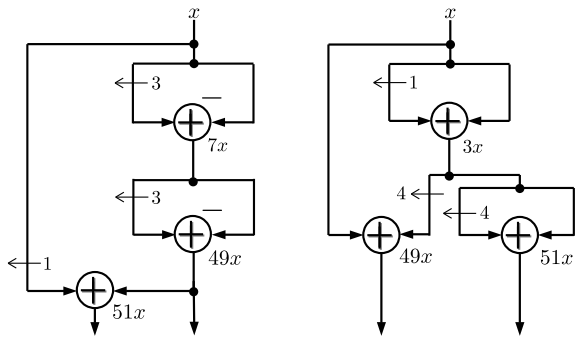
R. Garcia and A. Volkova were with Nantes Université, CNRS, LS2N, 44000 Nantes. Email: firstname.lastname@univ-nantes.fr

It is straightforward to obtain a first shift-and-add solution from the binary representation of the constant. A greedy algorithm based on the Canonical Signed Digit (CSD) representation permits to reduce the number of adders [2], [3]. Many heuristics enhance the results obtained with the CSD method [4]–[8], but heuristics do not provide any guarantee on the solution quality. Optimal approaches for the MCM-Adders can be roughly divided into two categories: (i) first approaches based on hypergraphs [9] and Integer Linear Programming (ILP) model [10], which can be solved by generic solvers; (ii) dedicated optimal algorithms based on branch and bound technique proposed and further developed by Aksoy et al. [11]. In this work, we improve and extend the results of the first category of approaches, since ILP-based modeling offers a better versatility for extensions than dedicated algorithms, and permits to rely on the efficiency of generic solvers.

The state-of-the-art ILP-based model for the MCM-Adders was proposed by Kumm in [12]. This method finds optimal solutions in terms of the number of adders in reasonable time, and has been adapted to SAT/SMT solvers [13] for single constant multiplication. However, we found an error in the model which makes it miss some optimal solutions.

In this work, we use [12] as basis, correct the modeling error for the MCM-Adders problem and build upon it to solve other flavors of the MCM. In particular, the number of cascaded adders, called the *adder depth* (AD), directly impacts the delay of the circuit, and is hence desired to be bounded. We first encode the AD count in our ILP model, and secondly propose, for the first time, a new bi-objective formulation called MCM_{AD}, which minimizes the number of adders and the AD *simultaneously*.

One of the main contributions of this paper is solving the MCM for a low-level hardware metric based on counting the one-bit adders, when the word length of the input x is known *a priori*. Assume adder graphs in Fig. 1 taking a 3-bit input x. While both require only three adders, the solution in Fig. 1a requires 22 one-bit adders and the other in Fig. 1b uses only 9. While classical bit-level optimization approaches are iterative and require synthesis and simulations [4], [14], solving the MCM with a fine-grained cost model is done only once. By one-bit adders, we gather both half and full adders as logic elements having roughly the same cost. This low-level metric has been discussed for decades, see [4] in which a heuristic for fixing the topology was presented. To our knowledge, we



(a) $AD = 3$ (b) $AD = 2$

Fig. 1: Different adder graph topologies computing the same outputs: $49x$ and $51x$

propose the first optimal approach solving the MCM problem targeting the one-bit adders (the MCM-Bits problem).

We further extend MCM-Bits model to solve the Truncated MCM (tMCM) problem. Indeed, the output results of the computations do not necessarily need to be full-precision and in practice are often rounded post-MCM. Focusing on a low-level metric allows to add intermediate truncations [15]–[17] that will save resources and not waste area and time to compute bits that will have no impact on the rounded result. At the same time, our goal is to respect a user-given absolute error on the result.

Some previous works address the tMCM but applied to a *fixed* adder-graph and either do not give a guarantee on the output error, *e. g.*, the heuristic-based approach [15], or overestimate and sometimes wrongly-compute the output error, *e. g.*, ILP-based [16]. However, as demonstrated in Fig. 4, some adder graph topologies are better suited for truncations than others. Hence, solving directly for tMCM and delegating the design exploration to an ILP solver, is a better approach. In this paper, we extend our preliminary work [17] for combining, for the first time, adder graph optimization with internal truncations. In particular, we tighten the error bounds and treat several corner cases, compared to [17].

With this paper, we aim at providing a tool that democratizes access to MCM at minimal hardware cost. We mix the optimization techniques to model the problem, have an in-depth look at the hardware addition to provide fine-grain cost metrics and provide a sound error-analysis to give numerical guarantees on the computed output. The main ideas of our approach are presented in Section II and in the next sections we go through the details of the ILP models. In Section VI, we demonstrate the efficiency of our approach providing optimization results and obtained hardware comparisons.

II. Bird view

Our approach is based on ILP modeling, which consists in stating objectives and constraints as *linear* equations involving integer and binary variables. We chose to limit our possibilities to linear equations because it allows using efficient and robust solving approaches embedded in commercial or open-source solvers such as CPLEX, Gurobi, GLPK, etc. Our end goal is to provide a tool which, given the target constants to multiply with, the choice of a cost function and several associated parameters, builds the corresponding ILP model. Solving the model with your favorite solver results in an adder graph description, which can be passed to the fixed- and floating-point core generator FloPoCo [18] to generate the VHDL code implementing the MCM circuit.

From the modeling point of view, we search for an adder graph that computes the product of the input x by given target constants. In this work, we center the model of an adder graph around the adders and their associated integer values, called *fundamentals*. The rules of construction of an adder graph are simple:

- for every target constant, there exists one fundamental equal to its value;
- every fundamental is a sum of its signed and potentially shifted left and right inputs, which can be other fundamentals or the input x;
- every used fundamental can be traced back to the input x, meaning that the adder graph topology holds.

The search-space for the fundamentals is deduced from the target constants: a tight upper bound on the number of fundamentals can be deduced using heuristics [7], [19]; and the maximum value fundamentals can take is often restricted in practice by the word length of the largest coefficient. To solve the MCM-Adders problem correctly, we encode the above rules as linear constraints. As a result of resolution, we obtain a sequence of fundamentals, *e. g.*, for Fig. 1a this sequence is $\{1, 7, 49, 51\}$.

On top of that, for the MCM-Bits and tMCM problems, each adder has an associated one-bit adder cost which must be correctly computed depending on the values of the inputs. Furthermore, for the tMCM problem, for each adder we associate potential truncations for its left and right operands which permit to reduce the adder cost. However, as each truncation induces an error, the model of error propagation should be incorporated as a set of constraints and the output errors is bounded by a user-given parameter. The challenge is to consider all possible cases and not overestimate the one-bit adders cost and errors.

The main challenges are to translate these objective functions and constraints into *linear* equations over integer/binary variables. In the following, we give details for that process.

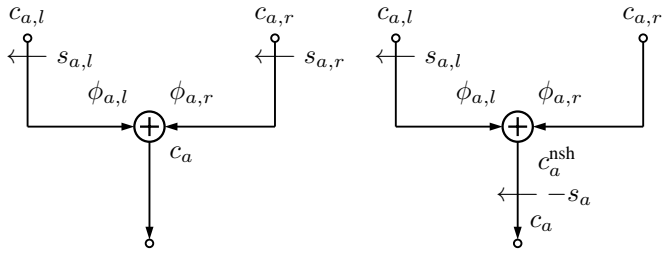
III. Optimal Multiple Constant Multiplication Counting Adders

A. The base model for MCM

The main challenge behind this model is to be able to formally state the constraints fixing the adder graph topology with linear equations. For conciseness, we also extensively use the so-called indicator constraints, which are supported by most modern MILP solvers, and have the form of:

$$ax \leq b \quad \text{if } y = 1, \tag{1}$$

where a and b are constants, x is an integer variable and y is a binary variable. These indicator constraints can be used as



(a) Adder in adder graph (b) Adder in MCM modeling

Fig. 2: Classic (left) and proposed (right) adder models

is, or linearized using big M constraints which transform (1) into

$$ax \leq b + M \times (1 - y), \tag{2}$$

where M is a constant large enough such that if $y = 0$ then (2) is similar to $ax \leq \infty$. Note that indicator constraints, when passed as is, to common MILP solvers, might slow the solving process. On the other hand, big M constraints are typically faster but could lead to numerical instabilities and should be used carefully when the value of M is order of magnitudes higher than ax [20], [21].

To fix the adder-graph topology with linear constraints, we first define a few sets of variables encoding adder information. Integer variables c_a correspond to the fundamentals for each adder and $c_{a,i}$, $\forall i \in \{l, r\}$, are the left and right inputs of the adder a. To simplify, the adder graph input is handled as the zeroth adder and its associated fundamental is one: $c_0 = 1$. Variables encoding shifts and signs are $s_{a,i}$ and $\phi_{a,i}$, respectively. Fig. 2a represents an adder and these variables which are linked together through the relation

$$c_a = (-1)^{\phi_{a,l}} 2^{s_{a,l}} c_{a,l} + (-1)^{\phi_{a,r}} 2^{s_{a,r}} c_{a,r}. \tag{3}$$

It has been proven [4] that we can limit the adder graph to odd fundamentals only, limiting the relevant shifts. Thus, it is possible to simplify the modeling by considering only positive left shifts, as illustrated by Fig. 2b, and let

$$c_a = 2^{-s_a} \left((-1)^{\phi_{a,l}} 2^{s_{a,l}} c_{a,l} + (-1)^{\phi_{a,r}} c_{a,r} \right), \tag{4}$$

where the shifts, s_a and $s_{a,l}$, take values ensuring that c_a is odd. Previous work [12], did consider positive left shift only or identical negative shifts, which is equivalent to our s_a. However, in their linearized model, integrity constraints conflicted with negative shifts dropping the latter when solving. Nevertheless, negative shifts are absolutely necessary to find optimal solution in some cases, *e. g.*, for the target constants $C = \{7, 19, 31\}$ where $19 = 2^{-1}(7 + 31)$.

Equation (4) is nonlinear, thus multiple intermediate variables are necessary to compute c_a using linear constraints only. For instance, in order to linearize

$$c_{a,l}^{\text{sh}} = 2^{s_{a,l}} c_{a,l}, \tag{5}$$

which involves a power of two and a product, a set of binary variables $\sigma_{a,s}$ is required with s taking values ranging from 0, meaning no shift, to an upper bound on the possible shift $S_{\max}$. These variables are used in (C8) where a constant, $2^0, 2^1, 2^2$, etc., is multiplied with a variable removing the power of two and product of variables issues. Variable $\sigma_{a,s}$ serves as an indicator of the shift s being used for adder a, $\forall s \in [\![0; S_{\max}]\!]$, and only one shift is selected for each adder. All the required intermediate variables are presented in TABLE I. These variables are used to produce the following linear model describing the adder graph topology and the link between fundamentals and target constants C_j:

Constants/Variables and their meaning
$\overline{N_A} \in \mathbb{N}$: bound on the number of adders;
$N_O \in \mathbb{N}$: number of outputs;
$C \in \mathbb{N}^{N_O}$: odd target constants;
$w \in \mathbb{N}$: fundamentals' word length;
$S_{\max} \in \mathbb{N}$: maximum shift, $S_{\max} = w$ by default.
$c_a \in [\![0; 2^w]\!]$, $\forall a \in [\![0; \overline{N_A}]\!]$: fundamental, or constant, obtained in adder a with c_0 fixed to the value 1, corresponding to the input;
$c_a^{\text{nsh}} \in [\![0; 2^{w+1}]\!]$, $\forall a \in [\![1; \overline{N_A}]\!]$: constant obtained in adder a before the negative shift;
$c_a^{\text{odd}} \in \mathbb{N}$, $\forall a \in [\![1; \overline{N_A}]\!]$: variable used to ensure that c_a is odd;
$c_{a,i} \in [\![0; 2^w]\!]$, $\forall a \in [\![1; \overline{N_A}]\!]$, $i \in \{l, r\}$: constant of adder from input i before adder a;
$c_{a,l}^{\text{sh}} \in [\![0; 2^{w+1}]\!]$, $\forall a \in [\![1; \overline{N_A}]\!]$: constant of adder from left input before adder a and after the left shift; for simplification $c_{a,r}^{\text{sh}}$ is an alias of $c_{a,r}$;
$c_{a,i}^{\text{sh,sg}} \in [\![-2^{w+1}; 2^{w+1}]\!]$, $\forall a \in [\![1; \overline{N_A}]\!]$, $i \in \{l, r\}$: signed constant of adder from input i before adder a and after the shift;
$\Phi_{a,i} \in \{0, 1\}$, $\forall a \in [\![1; \overline{N_A}]\!]$, $i \in \{l, r\}$: sign of i input of adder a. 0 for $+$ and 1 for $-$;
$c_{a,i,k} \in \{0, 1\}$, $\forall a \in [\![1; \overline{N_A}]\!]$, $i \in \{l, r\}$, $k \in [\![0; \overline{N_A} - 1]\!]$: 1 if input i of adder a is adder k;
$\sigma_{a,s} \in \{0, 1\}$, $\forall a \in [\![1; \overline{N_A}]\!]$, $s \in [\![0; S_{\max}]\!]$: 1 if left shift before adder a is equal to s;
$\Psi_{a,s} \in \{0, 1\}$, $\forall a \in [\![1; \overline{N_A}]\!]$, $s \in [\![S_{\min}; 0]\!]$: 1 if negative shift of adder a is equal to s;
$o_{a,j} \in \{0, 1\}$, $\forall a \in [\![1; \overline{N_A}]\!]$, $j \in [\![1; N_O]\!]$: 1 if adder a is equal to the j-th target constant.

TABLE I: Constants (top) and variables (bottom) used in the ILP formulation

(C1) $c_a^{\text{nsh}} = c_{a,l}^{\text{sh,sg}} + c_{a,r}^{\text{sh,sg}}$ $\quad \forall a \in [\![1; N_A]\!]$

(C2) $c_a^{\text{nsh}} = 2^{-s} c_a$ if $\Psi_{a,s} = 1$ $\quad \forall a \in [\![1; N_A]\!], s \in [\![S_{\min}; 0]\!]$

(C3) $\sum_{s=S_{\min}}^{0} \Psi_{a,s} = 1$ $\quad \forall a \in [\![1; N_A]\!]$

(C4) $\sigma_{a,0} = \sum_{s=S_{\min}}^{-1} \Psi_{a,s}$ $\quad \forall a \in [\![1; N_A]\!]$

(C5) $c_a = 2c_a^{\text{odd}} + 1$ $\quad \forall a \in [\![1; N_A]\!]$

(C6) $c_{a,i} = c_k$ if $c_{a,i,k} = 1$ $\quad \forall a \in [\![1; N_A]\!], i \in \{l, r\}, \forall k \in [\![0; a-1]\!]$

(C7) $\sum_{k=0}^{a-1} c_{a,i,k} = 1$ $\quad \forall a \in [\![1; N_A]\!], i \in \{l, r\}$

(C8) $c_{a,l}^{\text{sh}} = 2^s c_{a,l}$ if $\sigma_{a,s} = 1$ $\quad \forall a \in [\![1; N_A]\!], s \in [\![0; S_{\max}]\!]$

(C9) $\sum_{s=0}^{S_{\max}} \sigma_{a,s} = 1$ $\quad \forall a \in [\![1; N_A]\!]$

(C10) $c_{a,i}^{\text{sh,sg}} = -c_{a,i}^{\text{sh}}$ if $\Phi_{a,i} = 1$ $\quad \forall a \in [\![1; N_A]\!], i \in \{l, r\}$

(C11) $c_{a,i}^{\text{sh,sg}} = c_{a,i}^{\text{sh}}$ if $\Phi_{a,i} = 0$ $\quad \forall a \in [\![1; N_A]\!], i \in \{l, r\}$

(C12) $c_a = C_j$ if $o_{a,j} = 1$ $\quad \forall a \in [\![0; N_A]\!], j \in [\![1; N_O]\!]$

(C13) $\sum_{a=0}^{N_A} o_{a,j} = 1$ $\quad \forall j \in [\![1; N_O]\!]$

In details, constraint (C1) states that the value of an adder before a potential negative shift is equal to the sum of its shifted and signed inputs. Constraints (C2), (C3) and (C5) apply the negative shift to variables c_a and ensure that the computed fundamental is odd. It can be noticed that a potential negative shift after c_a^{nsh} only makes sense if the sum of the inputs is even, which can only happen if no left shift is applied to the left input: constraint (C4) specifically states this in order to speed up the solving process. The link between an adder and its inputs is enforced by constraints (C6) and (C7). Constraints (C8) and (C9) permit to apply the shift to the left input of an adder while constraints (C10) and (C11) apply the sign. Finally, every target constant C_j is computed once and only once thanks to constraints (C12) and (C13).

These constraints are sufficient to define a model for the following decision problem: is there an adder graph with $\overline{N_A}$ adders where all the target constants C_j are computed as fundamentals? Either the model is "infeasible", meaning that no adder graph exists for computing the target constants with just $\overline{N_A}$ adders, or "optimal" returning values c_a, that encode fundamentals, and corresponding shifts and signs that permit to produce a valid adder graph as in Fig. 1.

To tackle MCM as minimization problem, which is desirable in order to use the full potential of MILP solvers, $\overline{N_A}$ can be fixed to a known upper bound, obtained using a heuristic solution [7], [19] or a greedy algorithm [3], and then the objective function is to minimize the number of *effectively used* adders. A binary variable, $u_a \in \{0,1\}$, $\forall a \in [\![1;\overline{N_A}]\!]$, permits to deactivate an adder if not used: $c_a = 0$ if $u_a = 0$. Then, adding to the model the objective function

$$\min \sum u_a, \tag{6}$$

permits to solve the MCM problem as a minimization problem.

B. Bounding and/or Minimizing Adder Depth

The latency of the circuit is directly related to the number of cascaded adders or adder depth (AD). It is often preferable to have a bound on the AD to ensure a bound on the latency, or at least to have the minimal AD possible as a second objective, while simultaneously minimizing the number of adders. Works [9], [10] that rely on the search space enumeration solve the MCM problem with an AD bound up to 3 or 4 in best case. In this section we propose a new simple way to propagate the AD and to optionally bound it by a user-given constant $\overline{ad}$ and/or to minimize it as a second objective.

In order to track the AD, we introduce the variable $ad_{\max}$, and two sets of integer variables: ad_a and $ad_{a,i}, \forall a \in [\![1;\overline{N_A}]\!]$, $i \in \{l,r\}$, representing the AD of the adder a and the AD of its left and right inputs, respectively. Naturally, the adder depth of the input is set at zero, $ad_0 = 0$, and the AD propagation and the bound are handled by the following constraints:

$$ad_a = \max\left(ad_{a,l}+1, ad_{a,r}+1\right), \quad \forall a \in [\![1;\overline{N_A}]\!], \tag{7}$$

$$ad_{a,i} = ad_k \quad \text{if } c_{a,i,k} = 1, \quad \forall a \in [\![1;\overline{N_A}]\!], i \in \{l,r\}, \forall k \in [\![0;a-1]\!], \tag{8}$$

$$ad_{\max} \geq ad_a, \quad \forall a \in [\![1;\overline{N_A}]\!], \tag{9}$$

$$ad_{\max} \leq \overline{ad}. \tag{10}$$

Note that the $\max$ in (7) can be linearized adding a set of binary variables to the model. Constraint (9) ensures that the integer variable $ad_{\max}$ will be equal or greater than actual adder depth. With (10), we guarantee that the adder depth is bounded by $\overline{ad}$. In contrast to existing approaches [9] and [10], this encoding of AD is performed automatically by the solver, and does not require precomputations or a large number of variables and constraints.

We further extend our ILP model towards the bi-objective problem MCM_{AD} in order to optimize for both adder count and AD. The variable $ad_{\max}$, which encodes the AD of the adder graph, is used to replace the objective function (6) by

$$\min \sum_{a=1}^{\overline{N_A}} \left(\overline{N_A} u_a\right) + ad_{\max}. \tag{11}$$

This new objective function is a weighted sum that enforces a lexicographic optimization with the number of adders as first objective and the AD as second. Indeed, reducing the number of adders is unconditionally stronger than increasing the AD because $\overline{N_A} \geq ad_{\max}$.

Solving MCM_{AD} permits to select from the set of solutions with minimal number of adders those which yield the smallest delay in a hardware implementation. For example, in Fig. 1, we propose two optimal solutions in terms of the number of adders, $N_A = 3$, for the target constants $\{49, 51\}$, but different adder depths: AD = 3 in Fig. 1a and AD = 2 in Fig. 1b. Solving MCM_{AD} would directly find the adder graph with the lowest adder depth among the optimal ones in terms of number of adders.

This problem can be difficult for the solver, thus, in order to speed up the solving process, we provide redundant constraints that should help to reduce the search space. Gustafsson [22] proposed some lower bounds on the AD for the target constants which we can use in the following way:

$$ad_a \geq o_{a,j} \times \underline{ad}_j \quad \forall a \in [\![1;\overline{N_A}]\!], j \in [\![1;N_O]\!], \tag{12}$$

where $\underline{ad}_j$ is the lower bound on the adder depth of the target constant indexed by j. For our $\{49, 51\}$ target set, the first constant 49 has a minimal adder depth $\underline{ad}_1 = 2$, hence the constraint becomes:

$$ad_a \geq 2o_{a,1}, \quad \forall a \in [\![1;\overline{N_A}]\!], \tag{13}$$

enforcing the adder a to have an AD of at least 2 before considering it as a potential output for the first target constant 49.

In Section VI, we will show the advantages of MCM_{AD} over the classic single-objective MCM-Adders. Optimal solutions are obtained in reasonable time when solving either problems, and solving MCM_{AD} provides better solutions. For that reason, we will also consider the AD minimization as second objective when tackling MCM-Bits and tMCM.

Similarly to bounding and/or minimizing the adder depth, the *Glitch Path Count* (GPC) metric could be targeted in order to control the power consumption [23]. In [12], Kumm showed how to include such metric into an ILP model and this can be adopted for our solution.

IV. Optimal Multiple Constant Multiplication Counting One-Bit Adders

Minimizing the number of adders and the AD leads to efficient hardware but can also be used in software. In many cases, optimizing w. r. t. these metrics is the best we can provide and they already lead to a significant cost reduction. However, when specifically targeting hardware, *e. g.*, Application-Specific Integrated Circuits (ASICs) or Field Programmable Gate Arrays (FPGAs), and when the word length of the input is *a priori* known, it is possible to optimize using a finer-grain metric: the number of one-bit adders. Indeed, two optimal adder graphs, w. r. t. the number of adders, can have a significantly different number of one-bit adders. This is illustrated by Fig. 1: for a 3-bit input x, solution in Fig. 1a requires 22 one-bit adders, while the one in Fig. 1b needs only 9 one-bit adders. These large differences motivate the choice to specifically target the one-bit adder metric.

To solve the MCM-Bits problem, we enhance our MCM_{AD} model. In addition to the constraints fixing the adder graph topology and keeping track of the adder depth, we need to be able to count the number of one-bit adders. This additional need comes with two main difficulties: first, we have to be able to propagate the data word length which directly impacts the number of one-bit adders required for a given adder; second, counting the number of one-bit adders requires to consider many cases, as presented in [14], *e. g.*, the cost in terms of one-bit adders differs between two adders because of shifts and subtractions.

The former problem of word length propagation can be decomposed into tracking the Most Significant Bit (MSB) and Least Significant Bit (LSB) positions. Integer additions and subtractions impact the MSB but do not change the LSB, hence without loss of generality we can consider that the LSB is always equal to zero. Given an upper bound on the input, $\overline{x}$, it is possible to compute the required MSB after any adder:

$$\text{msb}_a = \lceil \log_2 (\overline{x} c_a) \rceil \tag{14}$$

where c_a is the adder fundamental. Incorporating this nonlinear constraint into our ILP model requires a few adjustments. First, the rounding can be removed by relaxing the equality. Second, exponentiation of both sides to remove the $\log_2$ leads to

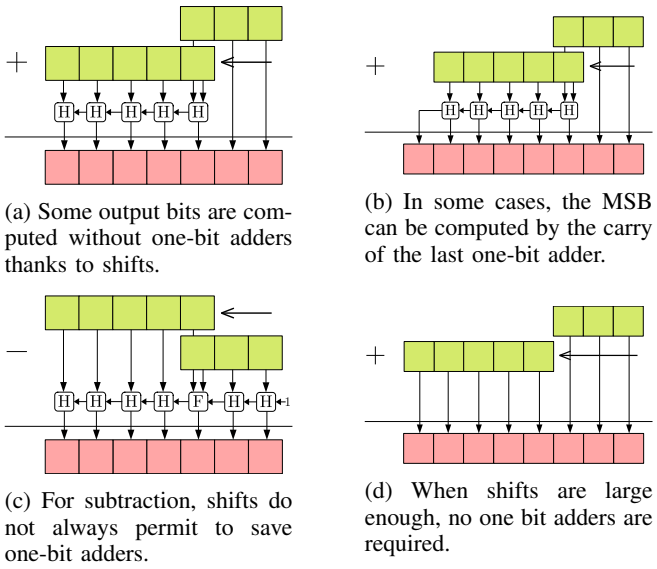
$$2^{\text{msb}_a} \geq \overline{x} c_a. \tag{15}$$

The expression 2^{msb_a} can be linearized similarly to shifts with the challenge of the wider range of possible values for msb_a. This permits to propagate the data word length, which is, in the worst case, equal to the number of one-bit adders, B_a:

$$B_a \leq \text{msb}_a + 1. \tag{16}$$

However, multiple corner cases permit to save one-bit adders.

As illustrated in Fig. 3a, shifts can lead to output bits computed without one-bit adders. Yet, one should note that it is not always the case with subtractions, see Fig. 3c. The MSB computation is either done by a dedicated one-bit adder, as represented in Fig. 3a, or obtained from the carry of the last one-bit adder, as in Fig. 3b. This can be deduced from the



(a) Some output bits are computed without one-bit adders thanks to shifts.

(b) In some cases, the MSB can be computed by the carry of the last one-bit adder.

(c) For subtraction, shifts do not always permit to save one-bit adders.

(d) When shifts are large enough, no one bit adders are required.

Fig. 3: Counting one-bit adders

MSB positions which depend on the fundamental values, see (14), allowing for the gain of a one-bit adder. One-bit adders might even not be necessary at all, as illustrated in Fig. 3d.

To wrap up, counting the number of one-bit adders, B_a, consists in computing the word length and deducting the gains g_a and ψ_a,

$$\text{B}_a = \text{msb}_a + 1 - g_a - \psi_a, \tag{17}$$

where ψ_a is equal to one if the MSB is computed by the carry of the last one-bit adder. The gain g_a is dependent on the signs, shifts and MSBs and can be summarized as follows:

$$g_a = \begin{cases} \text{msb}_a & \text{if } s_{a,l} > \text{msb}_{a,r} \\ s_{a,l} & \text{if } \Phi_{a,r} = 0 \\ 0 & \text{otherwise} \end{cases}, \tag{18}$$

formalizing cases illustrated in Fig. 3a, 3c and 3d. To include the computation of B_a and g_a in an ILP model, we add several binary variables to handle each condition.

As each adder a has an associated cost B_a, it is straightforward to formulate the objective as:

$$\min \sum_{a=1}^{\overline{N_A}} \overline{N_A} \text{B}_a + a d_{\max}, \tag{19}$$

where $\overline{N_A}$ is the number of adders the adder graph can use. This solves the bi-objective problem minimizing the number of one-bit adders first and the adder depth second.

For a given instance, we make the following conjecture: if a solution of that instance requires N_A adders, then the optimal solution in terms of number of one-bit adders also requires at most N_A adders. As a consequence, we will use a bound on $\overline{N_A}$ and assume we do not exclude optimal solutions with respect to one-bit adders.

In Section VI, our benchmarks show that solving MCM-Bits, on average, reduces the number of LUTs by 7.59% compared to our MCM-Adders.

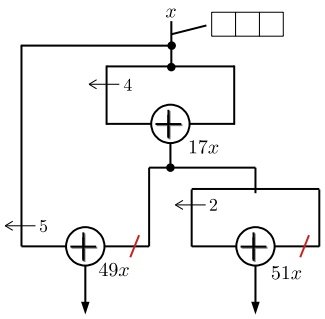


Fig. 4: Computing $49x$ and $51x$ via the fundamental 17 leads to only 4 one-bit adders for a 3-bit input/output.

V. Truncated Multiple Constant Multiplication

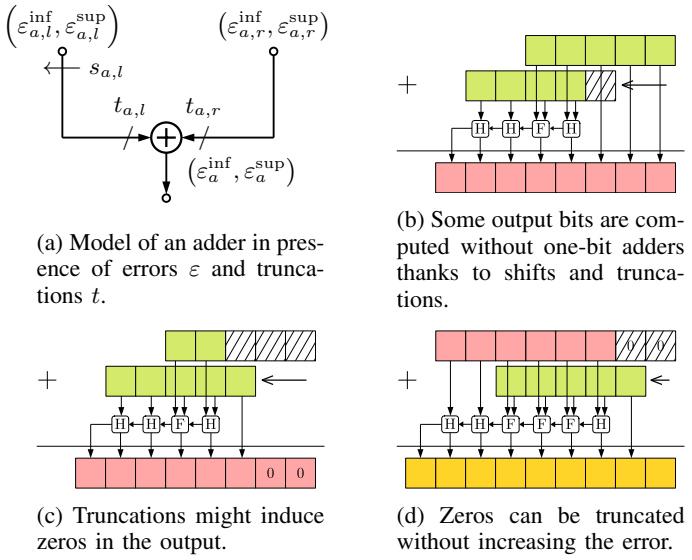
Arithmetic operation usually increase the size of the data paths but the full-precision result is often not required. For instance, consider recursive digital filters, in which the result of MCM is typically rounded to some internal word length at each iteration. In these cases, hardware designers can often provide an absolute error bound, $\overline{\varepsilon}$, that the computed products should respect. In order to avoid wasting resources to compute the bits that will not impact the rounded result, we propose to introduce truncations inside the adder graphs. Fig. 5b illustrates a case when truncating a bit in one of the adder's operands saves a one-bit adder. Of course, a truncation can generate an error that should be bounded and propagated, and the truncated adder graph should respect the bound $\overline{\varepsilon}$. Incorporating truncations the complexity of the problem, however it has been shown that it can significantly reduce the final hardware cost [15], [16].

In contrast to previous works, which focus on truncations of a *fixed* adder graph, we postulate that truncations should guide the topology, potentially leading to a different adder graph than the one obtained solving the MCM-Bits problem. Consider for instance the multiplication by 49 and 51 in Fig. 1, where the adder graph Fig. 1b is the MCM-Bits solution. Applying the truncations, s.t. only 3 output bits are faithful, yields a design requiring at least 5 one-bit adders. However, there exists another adder graph topology, shown in Fig. 4, that requires only 4 one-bit adders and is hence better suited for a target truncated result.

In this Section we outline the ILP model permitting to solve the tMCM as *one* optimization problem, covering the whole design space. In Section VI, our experiments demonstrate that when the goal is to keep only half of the output bits, our tMCM solutions reduce the number of LUTs by 28.87% compared to our new MCM-Bits, and by 35.93% compared to the state of the art MCM-Adders, on average.

A. Modeling truncations and error propagation

Truncations permit to gain one-bit adders and should be incorporated into a model in order to benefit from this possibility. To do so, we add integer variables, $t_{a,i}$, encoding the position of the truncated bits from the left and right inputs



(a) Model of an adder in presence of errors ε and truncations t.

(b) Some output bits are computed without one-bit adders thanks to shifts and truncations.

(c) Truncations might induce zeros in the output.

(d) Zeros can be truncated without increasing the error.

Fig. 5: Counting one-bit adders and propagating errors in presence of truncations

of adder a. Then, we update (18) in order to include the truncations in the one-bit adders gain count:

$$g_a = \begin{cases} \mathrm{msb}_a & \text{if } s_{a,l} > \mathrm{msb}_{a,r}, \\ \max\left(t_{a,l}, s_{a,l}, t_{a,r}\right) & \text{if } s_{a,l} \geq 0 \wedge \Phi_{a,l} = \Phi_{a,r}, \\ \max\left(t_{a,l}, s_{a,l}\right) & \text{if } s_{a,l} \geq 0 \wedge \Phi_{a,l} = 1, \\ t_{a,r} & \text{if } s_{a,l} \geq 0 \wedge \Phi_{a,r} = 1, \\ 0 & \text{otherwise.} \end{cases} \tag{20}$$

Typically, this gain is maximized in order to decrease the number of one-bit adders in the adder graph. Obviously, truncations should be constrained by the error they induce. These errors must be propagated and respect a user-given bound for each output.

Previous works [16], [17] underestimate the truncation-induced errors resulting in an output error exceeding the bound $\overline{\varepsilon}$. This underestimation was partly mitigated by an overestimation of the propagation of errors that we discuss here.

Classically, in computer arithmetic we consider the following model to link the exact quantities and errors:

$$\widetilde{y} = y + \Delta, \quad \text{where } |\Delta| \leq \varepsilon, \tag{21}$$

where $\widetilde{y}$ is the actually computed quantity, y is the exact quantity, and ε is a bound on the error. This fits well for symmetric errors, which is the case when rounding to nearest, but overestimates nonsymmetric truncation-induced errors. We propose to track errors using two positive bounds, $\varepsilon^{\inf}$ and $\varepsilon^{\sup}$, the former for the negative deviation from the exact quantity and the latter for the positive deviation. That way, it is possible to closely link the computed quantity with the exact quantity:

$$y - \varepsilon^{\inf} \leq \widetilde{y} \leq y + \varepsilon^{\sup}. \tag{22}$$

This equation holds for positive quantities and the following error-analysis assume that all the quantities are positives. Minor sign adjustments of ε's and reversing inequality relations are necessary to extend our analysis to negative quantities.

In order to correctly propagate the errors, we start by investigating the evolution of errors when applying unary operators used in truncated adder graphs. These unary operators are the shift, the negation and the truncation. First, applying a shift s to an inexact quantity $\widetilde{y}$ is straightforward and as it increases both error bounds:

$$2^s y - 2^s \varepsilon^{\inf} \leq \widetilde{2^s y} \leq 2^s y + 2^s \varepsilon^{\sup}. \tag{23}$$

Second, the negation applied on an inexact quantity $\widetilde{y}$ does not increase the overall error but swaps the deviations from the exact quantity y like so:

$$-y - \varepsilon^{\sup} \leq -\widetilde{y} \leq -y + \varepsilon^{\inf}. \tag{24}$$

Finally, when applied to a quantity $\widetilde{y}$, truncation up to the t-th bit, $\diamond_t(\cdot)$, removes information. This can increase the negative deviation $\varepsilon^{\inf}$ but not the positive deviation $\varepsilon^{\sup}$, thus the error bounds increase asymmetrically:

$$y - \varepsilon^{\inf} - \varepsilon_t \leq \diamond_t(\widetilde{y}) \leq y + \varepsilon^{\sup} + 0, \tag{25}$$

where ε_t is bounded by the quantity it removes from $\widetilde{y}$:

$$\varepsilon_t \leq 2^t - 1. \tag{26}$$

The bound is reached when all truncated bits are 1's. However, truncations could induce bits equal to 0 in the data path, as illustrated in Fig. 5c, and keeping track of these trailing 0's, denoted z, allows for tighter bound on the truncation errors:

$$\varepsilon_t \leq \max\left(2^t - 2^z, 0\right). \tag{27}$$

To be able to use this bound, we add an integer variable z_a, for each adder a, that keeps track of the number of trailing 0's induced by truncations or propagated from the inputs. In some cases where truncated bits are all zeros, as in Fig. 5d, we can even have error-free truncations. And with our approach, the ILP solver will automatically privilege those.

Finally, we need to propagate errors through adders which, for two inexact inputs, $\widetilde{y_1}$ and $\widetilde{y_2}$, adds error bounds together:

$$y_1 + y_2 - \varepsilon_{y_2}^{\inf} - \varepsilon_{y_2}^{\inf} \leq \widetilde{y_1} + \widetilde{y_2} \leq y_1 + y_2 + \varepsilon_{y_1}^{\sup} + \varepsilon_{y_2}^{\sup} \tag{28}$$

For each operand, we defined a dedicated propagation rule. Our end goal is to bring them all and to bind them in a single error propagation rule. For each adder, a, we define variables for error bounds $\left(\varepsilon_a^{\inf}, \varepsilon_a^{\sup}\right)$ which need to be bounded by the user-given acceptable output error $\overline{\varepsilon}$:

$$\left|\varepsilon_a^{\inf}\right| \leq \overline{\varepsilon} \quad \text{and} \quad \left|\varepsilon_a^{\sup}\right| \leq \overline{\varepsilon}. \tag{29}$$

Together with left and right truncations $t_{a,i}$, we propagate $\left(\varepsilon_{a,i}^{\inf}, \varepsilon_{a,i}^{\sup}\right)$, and Fig. 5a sums up the interconnections between these variables.

The propagation rule for each adder a is as follows:

$$\varepsilon_a^{\inf} = \begin{cases} \left(2^{s_{a,l}}\varepsilon_{a,l}^{\inf} + \varepsilon_{t_{a,l}}\right) + \left(\varepsilon_{a,r}^{\inf} + \varepsilon_{t_{a,r}}\right), \text{ if } \Phi_{a,l} = \Phi_{a,r}, \\ \left(2^{s_{a,l}}\varepsilon_{a,l}^{\inf} + \varepsilon_{t_{a,l}}\right) + \left(\varepsilon_{a,r}^{\sup} + \varepsilon_{t_{a,r}}\right), \text{ if } \Phi_{a,r} = 1, \\ \left(2^{s_{a,l}}\varepsilon_{a,l}^{\sup} + \varepsilon_{t_{a,l}}\right) + \left(\varepsilon_{a,r}^{\inf} + \varepsilon_{t_{a,r}}\right), \text{if } \Phi_{a,l} = 1, \end{cases} \tag{30}$$

$$\varepsilon_a^{\sup} = \begin{cases} 2^{s_{a,l}}\varepsilon_{a,l}^{\sup} + \varepsilon_{a,r}^{\sup}, & \text{if } \Phi_{a,l} = \Phi_{a,r}, \\ 2^{s_{a,l}}\varepsilon_{a,l}^{\sup} + \varepsilon_{a,r}^{\inf}, & \text{if } \Phi_{a,r} = 1, \\ 2^{s_{a,l}}\varepsilon_{a,l}^{\inf} + \varepsilon_{a,r}^{\sup}, & \text{if } \Phi_{a,l} = 1. \end{cases} \tag{31}$$

Finally, the propagated errors can force the result of the adder to the next binade w. r. t. the one deduced by (15). To prevent the risk of overflow, we adjust the MSB taking the error into account:

$$2^{\text{msb}_a} \geq \overline{x} c_a + \left|\varepsilon_a^{\sup}\right|, \tag{32}$$

where $\overline{x}$ is an upper bound on the maximum of the absolute values of lower and upper bounds of x. The solver will chose itself if it is more interesting to increase the MSB in order to gain more from truncations or not.

We do not give the full mathematical model in this paper but it is available alongside the open-source implementation.

VI. Experiments

We have implemented the ILP model-generation as an open-source tool called `jMCM`[1]. We used the modeling language JuMP [24] to implement the model generator, which can then use any generic MILP solver supported by JuMP as backend. We chose the Gurobi 9.5.1 [25] solver executed with 4 threads on an Intel® Core™ i9-11950H CPU at 2.60GHz and a time limit of 30 minutes. We also fully automate a tool-chain with FloPoCo [18], a state-of-the-art hardware code generator for arithmetic operators, to produce VHDL which we then synthesized for FPGA using Vivado v2018.2 for the xc7v585tffg1761-3 Kintex 7 device.

In this section, we will compare the impact of our models on different metrics, especially hardware metrics that are targeted only indirectly. With these experiments, we want to confirm that minimizing the adder depth does positively impact the final hardware delay or that a focus on one-bit adders permits to reduce the hardware cost. The research questions we answer in this section are summarized into four main questions:

(RQ1) Does taking the adder depth into account lead to better hardware?
(RQ2) Do adder graphs with less one-bit adders permit cost reduction in the final hardware?
(RQ3) What is the impact of intermediate truncations?
(RQ4) How increasing error bound can decrease the synthesized hardware costs?

To answer these questions, we solve all our models on benchmarks from image processing (11 instances) that have already been used to compare MCM algorithms [10], [26]. We extend our tests on the whole FIRsuite project (75 instances) [27], which is a collection of digital filter designs and is a direct application of the MCM problem. We solved multiple flavors of the MCM problem, four to be precise, producing hundreds of adder graphs. Using FloPoCo and Vivado, we synthesized all these adder graphs and extracted the number of LUTs, the delay and power (we present the sum of the logic, dynamic and signal powers). For conciseness, detailed results in TABLE III are shown only for the image processing benchmark, while the statistics are discussed over the full set of 86 benchmarks. The detailed report is available on git and results are reproducible in an automated way.

[1]Available on git: https://github.com/remi-garcia/jMCM

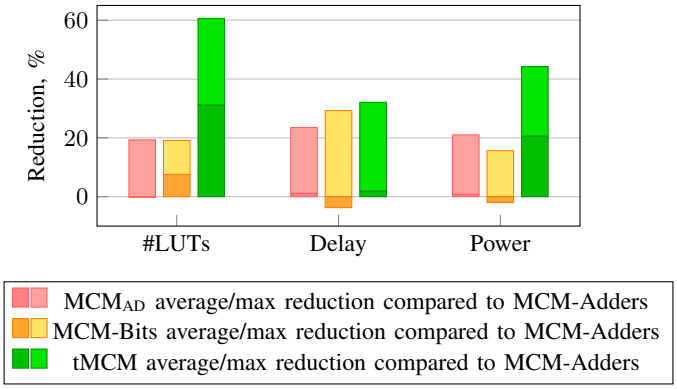


Fig. 6: The cost reduction of proposed models compared to the corrected [12]. Here tMCM is for 8-bit inputs and faithful rounding to 50% of the output word length.

A. Optimization settings and timings

One of the advantages of ILP-based approach is that a "good" candidate solution is available after the first several second of resolution, even if optimality over a huge design space is too difficult to prove by a generic solver. And in most cases that we observed, the solver either proves the optimality within first 10 seconds, or timeouts. To be precise, for the metrics based on number of adders, we could prove optimality for two thirds of benchmarks, and for models targeting one-bit adders it decreases to only a third of proven optimal results. With the ILP modeling for MCM, the complexity of the problem is not necessarily in the number of target coefficients, as we could quickly solve large instances and struggle with smaller ones, but in the coefficients themselves. One limitation is, however, data representation in the ILP solvers and potential numerical instabilities that can arise. In our benchmarks, we detected that starting from 12 bits, optimality could not be proven, but good candidate solutions were found for coefficients up to 19 bits.

Since solving the MCM-Adders problem permits to quickly obtain adder graphs with a low number of adders, we use it as a warm-start for the MCM-Bits or tMCM in order to speed up the resolution of complex models. And for any model, we give a bound on the number of adders by either using the RPAG library [19], if available, or the CSD greedy algorithm [3].

In many cases the MCM circuits are used in an iterative way, hence the output result is truncated to some intermediate format and then re-injected in the circuit, so that the input/output word lengths are the same. For the image-processing benchmarks an 8-bit input/output word length is a reasonable choice, hence we limited the error to a single unit of least precision (ulp) of the output precision. When taking the error of last rounding into account, we obtain that the error bound for internal truncations is actually half an ulp.

B. RQ1: the impact of simultaneous adder count and adder depth minimization

We start by analyzing the first improvement to the corrected MCM-Adders model, which is the simultaneous minimization of AD, as a secondary objective to the adder count. In 13 out of 86 benchmarks we exhibited a decrease of the AD by 1 and up to 4 stages, for the same minimal number of adders. Solving the MCM-Adders led to many adder graphs with AD greater than 4 and up to 6, while MCM$_{\text{min AD}}$ shows that for every instance of our benchmarks there exists an adder graph with AD ≤ 4. The running times were similar to our MCM-Adders, though some solutions were not proven optimal anymore.

TABLE II: Correlation between different metrics and actual hardware cost

	#adders	#one-bit adders	adder depth
#LUTs	0.9602	0.9967	0.2367
Delay	0.9549	0.913	0.3481
Power	0.9812	0.9659	0.2592

Minimizing the AD is an empirical approach to lower the delay of resulting hardware, its area and power. In Fig. 6 we exhibit, with the red bars, the reductions in each metric compared to the corrected state-of-the-art MCM-Adders. Over the 13 instances impacted by the simultaneous AD minimization, the average improvements are small, around 1% but can go up to roughly 20% for all metrics. We conclude that while MCM-Adders is already doing a good job w.r.t. the adder depth, it can be quite beneficial to automatically search the design space of adder graphs with minimal number of adders for the ones with least AD.

We also investigate the correlation between adder depth and the delay. We computed the correlation between different metrics and the hardware cost for all our benchmarks, each implemented in four flavours (MCM-Adders, MCM$_{\text{min AD}}$, MCM-Bits, tMCM). As illustrated in TABLE II, the correlation between the delay and the AD is $r = 0.3481$. We find this quite interesting, since in previous works [10], [12] bounding the adder depth to some low value was motivated by supposedly lower delay, which is refuted by our analysis. Indeed, as we observed, bounding the adder depth to a value lower than optimal, *i. e.*, obtained with our MCM$_{\text{min AD}}$, always leads to a larger number of adders that have much bigger correlation with delay: $r = 0.9549$.

As a result, we do not further investigate artificially bounding the AD but, by default, incorporate the AD minimization as a second objective in MCM-Bits and tMCM.

C. RQ2: hardware impact of the one-bit adder metric

The detailed hardware results in TABLE III demonstrate that minimizing the one-bit adders always reduces the number of LUTs, though in rare cases the results coincide with the MCM-Adders solution, *e. g.*, for `U.3-1`. As illustrated by Fig. 6, we reduced the number of LUTs by 7.59% on average, and 19.1% at most for the full benchmark set.

As illustrated in TABLE III, minimizing the one-bit adders can sometimes lead to a larger adder count, as in the case `LP15`, while significantly improving all hardware metrics. Also, in some cases, even if minimizing the AD is a default secondary objective, the optimal one-bit adder solutions require more stages.

TABLE III: Detailed results for the image-processing benchmarks, where Delay is in (ns), Power is in (mW)

Bench	MCM-Adders						MCM-Bits						tMCM					
	N_A	AD	#B	#LUTs	Delay	Power	N_A	AD	#B	#LUTs	Delay	Power	N_A	AD	#B	#LUTs	Delay	Power
G.3	4	2	43	45	7.385	111	4	2	40	41	7.305	108	4	2	24	27	7.205	85
G.5	5	4	58	60	9.700	168	5	4	57	58	8.937	175	5	4	36	44	9.267	139
HP5	4	2	42	42	7.506	121	4	3	39	41	8.574	134	4	2	25	26	7.678	89
HP9	5	2	50	50	7.859	158	5	2	47	48	7.675	165	5	2	34	40	7.826	146
HP15	12	2	122	122	10.207	401	12	3	105	108	10.377	421	12	2	60	62	9.842	270
L.3	3	3	34	36	7.868	116	3	3	31	33	8.377	112	3	3	26	29	7.925	102
LP5	6	3	66	68	9.099	213	6	3	60	65	9.142	209	6	3	47	52	8.994	173
LP9	12	5	157	154	11.915	537	12	3	137	138	12.439	516	13	3	88	95	11.379	361
LP15	26	6	313	315	20.979	1359	27	4	250	258	15.617	1218	27	3	150	182	14.632	959
U.3-1	4	2	32	33	7.372	104	4	2	32	33	7.372	104	4	2	22	17	6.479	63
U.3-2	5	3	61	58	8.937	169	5	4	49	49	9.625	167	5	4	33	32	8.967	102

To compare the impact of the one-bit adder metric vs. the adder count, we analyzed the correlation between each of them and the actual hardware cost. TABLE II clearly demonstrates that the one-bit adders are most certainly in a linear relationship with the LUT count, with a correlation factor $r = 0.9967$, which is stronger than $r = 0.9602$ for the number of adders. This does not mean that minimizing the number of one-bit adders will surely minimize the number of LUTs, yet it is reasonable to expect it.

The one-bit adder correlation factors for the delay and power are 0.9129 and 0.9659, respectively, which indicates a less strong relationship, which is also worse than for the number of adders. This analysis also confirms the detailed results in TABLE III, where sporadic increase in delay and power for seemingly better MCM-Bits solutions can be observed. In general, the increase is small and could be neglected but is probably due to a different glitch path count, which can also be modeled and used to guide the adder topology [12].

With the above, we can conclude that using the one-bit adder metric is more beneficial for the LUT count, regardless of occasional increase in the adder depth and number of adders.

D. RQ3 and RQ4: advantages of truncated adder graphs

In [17] we provide a preliminary analysis and comparison with the state-of-the-art models and our tMCM by counting the number of one-bit-adders. On the same set of benchmarks, we deduced that the number of one-bit adders, was on average reduced by 25.31% compared to MCM-Bits problem.

In this work we go further and synthesize all adder graphs for hardware. TABLE III confirms that tMCM provides major hardware cost reductions: for 8-bit inputs/outputs, the number of LUTs is decreased by 21.55%, the delay by 5.15% and the power by 18.68%.

These results are also expected due to the strong correlation factor between the LUT count and the number of one-bit adders, which are minimized in tMCM. Similarly to the case of MCM-Bits, truncated adder graphs in some cases require more adders, which is natural, since there are many different corner cases that permit to reduce the one-bit adder count with a drawback of larger number of fundamentals. Interestingly, in presence of truncations the adder depth might be, on the contrary, smaller than for the MCM-Bits solutions, significantly improving power.

Of course, this significant performance improvement requires the embedded system designers to be able to provide an *a priori* error bound for the outputs. However, we do not find it unreasonable, since analyzing the finite-precision behavior of the implemented system is expected for resource-constrained applications. Generic static analysis tools can be used to compute such bounds, such as [28], [29], or they can be manually deduced for specific applications, such as has been done for digital filters [30].

In the above discussion we fixed the accuracy of the MCM outputs by setting the output word length to 8 bits, which is reasonable for an image-processing benchmark. In general, as the coefficients have different magnitudes, this meant that for some instances 8-bit output contained the exact result, and for some only a fraction of it. In order to present a fair general comparison, we now vary the error bound for each instance and first remove a quarter, and then a half of the *exact* output word length. For example, if the full-precision result requires 16 bits, we solve tMCM to faithfully round to 12 bits and then to 8 bits. In cases when the coefficient magnitude is close to that of the input, keeping half of the result bits is quite reasonable, as it basically corresponds to maintaining the same input/output size. This experiment also permits us to see how varying the error bound impacts the hardware cost and might help to find the sweet spot between accuracy and resources.

On average, reducing the output word length by 25% results in 6.2% one-bit adders reduction, compared to full precision. Leaving only 50% of the output bits permits to significanly change the topology of the adder graphs (as illustrated by the adder depth and adder count change in) and obtain, on average, the reduction of 28.87% in terms of one-bit adder count. This significant improvement is also reflected in the actual hardware metrics, as illustrated in Fig. 6, reaching in some cases a 61% reduction in LUTs.

VII. Conclusion

Many approaches exist for solving the MCM problem but they mostly rely on solving the MCM-Adders problem, *i. e.*, with a high-level metric. In this work we propose a low-level metric based on counting the number of one-bit

adders, and tackle the MCM problem in different flavors: from minimization of the adder depth as a secondary objective, to adding intermediate truncations whilst computing a faithfully rounded output. With this work, the non-trivial MCM design-space exploration for embedded system designers is automated and delegated to powerful ILP solvers, liberating the designers to study high-level questions, such as definition of the input/output word lengths for their application.

With the ILP-based approach, we showed that extending the MCM-Adders problem does not require too much effort, when the cost/constraint modeling is done, which opens an easy way for more extensions such as the glitch path count. Moreover, our MCM model can be incorporated into the design of more complex algorithms that use MCM, such as digital filters [31].

We kept our focus on FPGA design but most of this work directly applies to ASICs. As a perspective, refining the one-bit adder metric into half-adders and full-adders could further improve the results for ASICs.

Overall, we proposed a tool with options allowing for tackling the MCM problem with respect to many end-user needs. Given enough time and numerical robustness of the MILP solver, the solving process permits to find optimal solutions and prove optimality with an exhaustive search, or propose candidate solutions if a timeout is fixed.

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

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

[4] 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.

[5] 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.

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

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

[8] 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.

[9] 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.

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

[11] 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.

[12] M. Kumm, "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.

[13] 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.

[14] K. Johansson, O. Gustafsson, and L. Wanhammar, "Bit-Level Optimization of Shift-and-Add Based FIR Filters," in *2007 14th IEEE International Conference on Electronics, Circuits and Systems*. IEEE, Dec. 2007, pp. 713–716.

[15] R. Guo, L. S. DeBrunner, and K. Johansson, "Truncated MCM using pattern modification for FIR filter implementation," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. IEEE, May 2010, pp. 3881–3884.

[16] F. de Dinechin, S.-I. Filip, M. Kumm, and L. Forget, "Table-Based versus Shift-And-Add Constant Multipliers for FPGAs," in *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*. IEEE, Jun. 2019, pp. 151–158.

[17] R. Garcia, A. Volkova, and M. Kumm, "Truncated Multiple Constant Multiplication with Minimal Number of Full Adders," in *ISCAS 2022*, Austin, Texas, United States, May 2022. [Online]. Available: https://hal.archives-ouvertes.fr/hal-03582935

[18] 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.

[19] 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.

[20] E. Klotz and A. M. Newman, "Practical guidelines for solving difficult mixed integer linear programs," *Surveys in Operations Research and Management Science*, vol. 18, no. 1, pp. 18–32, Oct. 2013.

[21] P. Belotti, P. Bonami, M. Fischetti, A. Lodi, M. Monaci, A. Nogales-Gómez, and D. Salvagnin, "On handling indicator constraints in mixed integer programming," *Computational Optimization and Applications*, vol. 65, no. 3, pp. 545–566, May 2016.

[22] O. Gustafsson, "Lower Bounds for Constant Multiplication Problems," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 54, no. 11, pp. 974–978, Nov. 2007.

[23] S. S. Demirsoy, A. G. Dempster, and I. Kale, "Power analysis of multiplier blocks," in *2002 IEEE International Symposium on Circuits and Systems. Proceedings (Cat. No.02CH37353)*, vol. 1, 2002, pp. 297–300.

[24] 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.

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

[26] 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.

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

[28] E. Darulova, A. Izycheva, F. Nasir, F. Ritter, H. Becker, and R. Bastian, "Daisy - Framework for Analysis and Optimization of Numerical Programs (Tool Paper)," in *Tools and Algorithms for the Construction and Analysis of Systems*, D. Beyer and M. Huisman, Eds. Cham: Springer International Publishing, 2018, pp. 270–287.

[29] E. Goubault and S. Putot, "Static Analysis of Finite Precision Computations," in *Verification, Model Checking, and Abstract Interpretation*, R. Jhala and D. Schmidt, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 232–247.

[30] A. Volkova, M. Istoan, F. De Dinechin, and T. Hilaire, "Towards Hardware IIR Filters Computing Just Right: Direct Form I Case Study," *IEEE Transactions on Computers*, vol. 68, no. 4, pp. 597–608, Apr. 2019.

[31] M. Kumm, A. Volkova, and S.-I. Filip, "Design of Optimal Multiplierless FIR Filters with Minimal Number of Adders," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2022.