Toward 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 low-level hardware cost metric, which counts the number of one-bit adders and demonstrate that it is strongly correlated with the LUT count. This new model 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 an automatic 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 synthesized hardware.

Index Terms—Multiple constant multiplication, multiplierless hardware, ILP, datapath optimization.

I. INTRODUCTION

MULTIPLICATIONS by integer constants arise in many numerical algorithms. In particular, the evaluation of FIR filters in their transposed direct form requires to multiply an input by tens or hundreds of multiple constants [1, Chapter 6.4]. This is also the case for Fast Fourier Transform or parallel deep neural network controllers, where input vectors are multiplied by constant matrices. On Field-Programmable Gate Arrays (FPGAs), these constants are usually represented with Fixed-Point (FxP) numbers which can be assimilated to integers. When implementing these applications into resource-constrained embedded systems, optimizing the multiple constant multiplication (MCM) can significantly improve area, power and delay [2], [3], [4]. In this paper we focus on improving the optimal approaches of the MCM design using high- and low-level hardware cost models.

Since the values of constants to multiply with are known a priori, dedicated multiplierless architectures can be designed, instead of relying on costly generic multipliers [5]. The shift-and-add approach is a common method to reduce hardware cost, it consists in replacing multiplications by additions/subtractions and bit-shifts. 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.

Hence, the MCM problem is implementing a multiplication of an unknown FxP input by multiple target FxP constants at the lowest cost. A typical high-level metric is the number of adders (bit-shifts can be hardwired at a negligible cost), minimizing which is the goal of what we will call the MCM-Adders problem. Its output is an adder-graph describing the shift-and-add solution as a data flow, as in Fig. 1 where labeled horizontal arrows denote shifts.

This work uses Integer Linear Programming (ILP) modeling to solve the MCM optimally. We base our model on [6] that solves the MCM-Adders problem, improve it and extend to solve other flavors of the MCM. For instance, we propose a novel bi-objective ILP formulation that permits to bound or minimize the number of cascaded adders, called the adder depth (AD), as a secondary objective.

Many adder graphs can have minimal number of adders, but their one-bit adder cost can differ drastically. Assume adder graphs in Fig. 1 requiring a 3-bit input $x$ and computing $51x$ and $49x$. While both require only three adders, the solution in Fig. 1a requires 24 one-bit adder and the one in Fig. 1b uses only 8. The adder graph in Fig. 1a also requires 9 NOT gates which come for free when targeting FPGAs. On Application-Specific Integrated Circuits (ASICs), their cost is small compared to the overall cost of the circuit, hence we leave them out of scope in cost estimation.

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. We refer to this problem as MCM-Bits. Moreover, the results of MCM do not necessarily need to be full-precision and in practice are often rounded post-MCM. We propose a new formulation, tMCM, that truncates intermediate results in adder graphs to optimize the resources while guaranteeing faithful rounding to the output format.

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 a sound error-analysis to give numerical guarantees on the computed output. The main ideas of our approach are...
presented in Section III and in the next sections we go through the details of the ILP models. In Section VII, we demonstrate the efficiency of our approach providing optimization results and obtained hardware comparisons.

II. STATE-OF-THE-ART

The multiplierless shift-and-add approach to solve the MCM problem finds its first solution from the binary representation of the constant which is literally a shift-and-add decomposition of integers. A greedy algorithm based on the Canonical Signed Digit (CSD) representation permits to reduce the number of adders [7], [8] compared to the binary representation. However, this number can be further reduced and heuristics have been proposed to enhance the results obtained with the CSD method [9], [10], [11], [12], [13]. Yet, 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 [14] and mathematical modeling [6], [15], [16], which can be solved with generic solvers; (ii) dedicated optimal algorithms based on branch and bound technique proposed and further developed by Aksoy et al. [17]. 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 [6]. This method finds optimal solutions in terms of the number of adders in reasonable time. However, we found an error in the model which makes it miss some optimal solutions. An approach to the MCM-Adders solution using SAT/SMT solvers [16] has also been developed for single and multiple constant multiplications.

Two generalizations of the MCM-Adders problem were recently tackled, the first [18] solves the Unified MCM problem and the second uses E-graphs [19] to solve generic circuit design problems. These problems are more general and harder to solve: the proposed methods do not scale well and cannot be applied to many instances we will solve in Section VII. In particular, the E-graph approach relies on a full enumeration of the search-space, as in [15], which is not tractable for solving the MCM-Adders problem.

The MCM-Bits problem has also been studied, however classical bit-level optimization approaches are heuristics [9], [20] that estimate the low-level cost of the final circuit and track the number of one-bit adders. By one-bit adders, we gather both half and full adders as logic elements having roughly the same cost compared to the other logic elements of the circuit. Moreover, on FPGAs the distinction between half and full adders cannot be made. To our knowledge, we propose the first optimal approach solving the MCM problem targeting the one-bit adders (the MCM-Bits problem).

Focusing on a low-level metric allows to add intermediate truncations [21], [22], [23] 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, the 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., a heuristic-based approach [21], or overestimate and sometimes wrongly compute the output error, e.g., ILP-based [22]. 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. Our preliminary work [23] combined, for the first time, adder graph optimization with internal truncations. Yet, in [23] the error bounds can be tightened and a few several corner cases were not implemented. With this work, we resolve these issues and provide synthesis results on a large set of benchmarks.

III. 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 [24]
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:

- every fundamental is a sum of its signed and potentially shifted left and right inputs, which can be other fundamentals or the input $x$;
- for every target constant, there exists one fundamental equal to its value;
- 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 [12, 25]; 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 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.

On top of that, for the MCM-Bits and tMCM problems, each adder, depending on the inputs, has an associated one-bit adder cost, which should be modeled. 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.

With our approach we provide adder graphs that can multiply an arbitrary input $x$, which can be either a positive integer or a two’s complement signed integer, by an arbitrary set of integer constants. However, the ILP model we use as back-end only takes positive odd constants to multiply with, thus the sign and even/odd adjustments are done outside of the model in pre- and post-processing.

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.

IV. MCM WITH A HIGH-LEVEL COST METRIC 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:

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

where $a$ and $b$ are constants, $x$ is an integer variable and $y$ is a binary variable. These indicator constraints can be used as is, or linearized using big M constraints which transform (1) into

$$ax \leq b + M \times (1 - y),$$

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 $ax$ [26], [27].

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

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

(3)

It has been proven [9] 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

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

(4)

where the shifts, $s_a$ and $s_{a,l}$, take values ensuring that $c_a$ is odd. Previous work [6], 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

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

(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 $S_{\text{max}}$. These variables are used in (C8) where a constant,
TABLE I
CONSTANTS (TOP) AND VARIABLES (BOTTOM) USED IN
THE ILP FORMULATION

| Constants/Variables and their meaning |
|---------------------------------------|
| \( N_A \in \mathbb{N} \): bound on the number of adders; |
| \( N_O \in \mathbb{N} \): number of outputs; |
| \( C \in \mathbb{N}^N \): odd target constants; |
| \( w \in \mathbb{N} \): fundamentals’ word length; |
| \( S_{\text{max}} \in \mathbb{N} \): maximum shift. \( S_{\text{max}} = w \) by default. |

- \( c_{a,i} \in [0;2^w], \forall a \in [0;N_A] \): fundamental, or constant, obtained in adder \( a \) with \( c_0 \) fixed to the value 1, corresponding to the input; \( c_{a,i} \in [0;2^{w+1}], \forall a \in [1;N_A] \): constant obtained in adder \( a \) before the negative shift; \( c_{a,i}^{\text{odd}} \in \mathbb{N}, \forall a \in [1;N_A] \): variable used to ensure that \( c_a \) is odd; \( c_{a,i} \in [0;2^w], \forall a \in [1;N_A], i \in \{l,r\} \): constant of adder from input \( i \) before adder \( a \); \( c_{a,i} \in [0;2^{w+1}], \forall a \in [1;N_A] \): constant of adder from left input before adder \( a \) and after the left shift; \( c_{a,i} \in [-2^{w+1},2^{w+1}], \forall a \in [1;N_A], i \in \{l,r\} \): signed constant of adder from input \( i \) before adder \( a \) and after the shift; \( \Phi_{a,i} \in [0,1], \forall a \in [1;N_A], i \in \{l,r\} \): sign of input of adder \( a \). 0 for + and 1 for −; \( c_{a,i} \in [0,1], \forall a \in [1;N_A], i \in \{l,r\}, k \in [0;N_A-1] \): if input \( i \) of adder \( a \) is adder \( k \); \( c_{a,s} \in \{0,1\}, \forall a \in [1;N_A], s \in [0;S_{\text{max}}] \): if left shift before adder \( a \) is equal to \( s \); \( \Psi_{a,s} \in \{0,1\}, \forall a \in [1;N_A], s \in [-S_{\text{max}},0] \): if negative shift of adder \( a \) is equal to \( s \); \( u_a \in \{0,1\}, \forall a \in [1;N_A], j \in [1;N_O] \): if adder \( a \) is equal to the \( j \)-th target constant.

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 \( c_{a,i} \) 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 \( N_A \) 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 \( N_A \) 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, \( N_A \) can be fixed to a known upper bound, obtained using a heuristic solution [12], [25] or a greedy algorithm [8], and then the objective function is to minimize the number of effectively used adders. A binary variable, \( u_a \in [0,1], \forall a \in [1;N_A] \), permits to deactivate an adder if not used: \( c_a = 0 \) if \( u_a = 0 \). Then, adding to the model the objective function

\[
\min \sum_{a=0}^{N_A} u_a, \tag{6}
\]

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 [14], [15] 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 \( ad_{\text{max}} \) and/or to minimize it as a second objective.

In order to track the AD, we introduce the variable \( ad_{\text{max}} \), and two sets of integer variables: \( ad_a, ad_{a,j}, \forall a \in [1;N_A] \), \( a \in [1;N_A] \), \( j \in [1;N_O] \).
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:

$$ad_a = \max(a_{d_{a, l}} + 1, a_{d_{a, r}} + 1), \quad \forall a \in \{1; \overline{N_A}\}, \quad (7)$$

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

$$\forall k \in \{0; a - 1\}, \quad (8)$$

$$ad_{\text{max}} \geq ad_a, \quad \forall a \in \{1; \overline{N_A}\}, \quad (9)$$

$$ad_{\text{max}} \leq ad. \quad (10)$$

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 $ad_{\text{max}}$ will be equal or greater than actual adder depth. With (10), we guarantee that the adder depth is bounded by $ad$. In contrast to existing approaches [14] and [15], 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 $\text{MCM}_{AD}$ in order to optimize for both adder count and AD. The variable $ad_{\text{max}}$, which encodes the AD of the adder graph, is used to replace the objective function (6) by

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

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 $\overline{N_A} \geq ad_{\text{max}}$.

Solving $\text{MCM}_{AD}$ 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 $\text{MCM}_{AD}$ 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 [28] proposed some lower bounds on the AD for the target constants which we can use in the following way:

$$ad_a \geq o_{a, j} \times ad_j, \quad \forall a \in \{1; \overline{N_A}\}, \quad j \in \{1; \overline{N_O}\}, \quad (12)$$

where $ad_j$ 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 $ad_1 = 2$, hence the constraint becomes:

$$ad_a \geq 2 o_{a, 1}, \quad \forall a \in \{1; \overline{N_A}\}, \quad (13)$$

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 VII, we will show the advantages of $\text{MCM}_{AD}$ over the classic single-objective MCM-Adders. Optimal solutions are obtained in reasonable time when solving either problems, and solving $\text{MCM}_{AD}$ 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 [29]. In [6], Kumm showed how to include such metric into an ILP model and this can be adopted for our solution.

V. MCM WITH A LOW-LEVEL COST METRIC 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., ASICs or FPGAs, and when the word length of the input is a priori known, it is possible to utilize window grain metrics: 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 24 one-bit adders, while the one in Fig. 1b needs only 8 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 $\text{MCM}_{AD}$ 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 [20], 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, $X$, it is possible to compute the required MSB after any adder:

$$\text{msb}_a = \left\lceil \log_2 (\overline{X_{c_a}}) \right\rceil \quad (14)$$

where $c_a$ is the adder fundamental. Incorporating this non-linear 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

$$2^{\text{msb}_a} \geq \overline{X_{c_a}} \quad (15)$$

The expression $2^{\text{msb}_a}$ can be linearized similarly to shifts with the challenge of the wider range of possible values for $\text{msb}_a$. 

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
This permits to propagate the data word length, which is, in the worst case, equal to the number of one-bit adders, $B_a$:

$$B_a \leq \text{msb}_a + 1. \quad (16)$$

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. Both cases are illustrated in Fig. 1b, where the MSB of 3x is done by the carry and the MSB of 51x requires a dedicated one-bit adder. This can be deduced from the 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 $\psi_a$.

$$B_a = \text{msb}_a + 1 - g_a - \psi_a. \quad (17)$$

where $\psi_a$ 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:

$$g_a = \begin{cases} 
\text{msb}_a & \text{if } s_{a,l} > \text{msb}_{a,r} \land \Phi_{a,l} = \Phi_{a,r} \\
0 & \text{if } s_{a,l} \leq \text{msb}_{a,r} \land \Phi_{a,l} = \Phi_{a,r} \\
0 & \text{if } \Phi_{a,l} = 0 \land \Phi_{a,r} = 0 \\
1 & \text{otherwise}
\end{cases} \quad (18)$$

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.
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 VII, 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, $t_{a,l}$, encoding the position of the truncated bits from the left and right inputs of adder $a$. Then, we update (18) in order to include the truncations in the one-bit adders gain count:

$g_a = \begin{cases} 
\text{msb}_a & \text{if } s_{a,l} > \text{msb}_{a,r} \land \Phi_{a,l} = \Phi_{a,r}, \\
\max (t_{a,l}, s_{a,l}, t_{a,r}) & \text{if } s_{a,l} \leq \text{msb}_{a,r} \land \Phi_{a,l} = \Phi_{a,r}, \\
\max (t_{a,l}, s_{a,l}) & \text{if } \Phi_{a,l} = 1 \land \Phi_{a,r} = 0, \\
0 & \text{if } \Phi_{a,l} = 0 \land \Phi_{a,r} = 1.
\end{cases}$

The second case is illustrated in Fig. 5b where $t_{a,l} = 3$, $s_{a,l} = 2$ and $t_{a,r} = 0$ leading to $g_a = 3$. The third case roughly corresponds to the third case of (18) where the left input truncation could surpass the shift. The last case corresponds to the gain that could be obtained by adding a truncation in Fig. 3c.

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 [22], [23] underestimate the truncation-induced errors resulting in an output error exceeding the bound $\varepsilon$. 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:

$\tilde{y} = y + \Delta$, \hspace{1cm} where $|\Delta| < \varepsilon$.

This equation holds for positive quantities and the following error-analysis assume that all the quantities are positives. Minor sign adjustments of $\varepsilon$‘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 $\tilde{y}$ is straightforward and as it increases both error bounds:

$2^s \cdot y - 2^s \varepsilon^{\inf} \leq \tilde{y} \leq 2^s y + 2^s \varepsilon^{\sup}$.

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

$-y - \varepsilon^{\sup} \leq -\tilde{y} \leq -y + \varepsilon^{\inf}$.

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

$y - \varepsilon^{\inf} - \varepsilon_t \leq \Theta_t(\tilde{y}) \leq y + \varepsilon^{\sup} + \varepsilon_t$,

where $\varepsilon_t$ is bounded by the quantity it removes from $\tilde{y}$:

$\varepsilon_t \leq 2^t - 1$.

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:

$\varepsilon_t \leq \max (2^t - 2^t, 0)$.

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, \( y_1 \) and \( y_2 \), adds error bounds together:
\[
y_1 + y_2 - \varepsilon_{y_2}^\inf - \varepsilon_{y_2}^\sup \leq y_1 + y_2 \leq y_1 + y_2 + \varepsilon_{y_1}^\sup + \varepsilon_{y_2}^\sup
\] (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 defined variables for error bounds \( (\varepsilon_{a}^\inf, \varepsilon_{a}^\sup) \) which need to be bounded by the user-given acceptable output error \( \varepsilon \):
\[
|\varepsilon_{a}^\inf| \leq \varepsilon \quad \text{and} \quad |\varepsilon_{a}^\sup| \leq \varepsilon.
\] (29)
Together with left and right truncations \( t_{a,i} \), we propagate \( (\varepsilon_{a}^\inf, \varepsilon_{a}^\sup) \), and Fig. 5a sums up the interconnections between these variables.

The propagation rule for each adder \( a \) is as follows:
\[
\varepsilon_{a}^\inf = \begin{cases} 
2\varepsilon_{a}^\inf + \varepsilon_{t_{a,i}}^\inf + (\varepsilon_{a}^\inf + \varepsilon_{t_{a,i}}^\inf), & \text{if } \Phi_{a,i} = \Phi_{a,r}, \\
2\varepsilon_{a}^\inf + \varepsilon_{t_{a,i}}^\inf + (\varepsilon_{a}^\sup + \varepsilon_{t_{a,i}}^\inf), & \text{if } \Phi_{a,i} = 1,
\end{cases}
\] (30)
\[
\varepsilon_{a}^\sup = \begin{cases} 
2\varepsilon_{a}^\sup + \varepsilon_{t_{a,i}}^\sup + (\varepsilon_{a}^\inf + \varepsilon_{t_{a,i}}^\sup), & \text{if } \Phi_{a,i} = \Phi_{a,r}, \\
2\varepsilon_{a}^\sup + \varepsilon_{t_{a,i}}^\sup + (\varepsilon_{a}^\inf + \varepsilon_{t_{a,i}}^\sup), & \text{if } \Phi_{a,i} = 1.
\end{cases}
\] (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:
\[
2^{\text{msb}_{a}} \geq \bar{x}_{a} + |\varepsilon_{a}^\sup|,
\] (32)
where \( \bar{x} \) 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.

VII. EXPERIMENTS AND DISCUSSION

We have implemented the ILP model-generation as an open-source tool called J\textsuperscript{MC}M.\textsuperscript{1} We used the modeling language JuMP [30] 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 [31] 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 [24], 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 five 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?
(RQ5) How does our approach scale w.r.t. the number of coefficients, their word length, etc.?

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 [15, 32]. We extend our tests on the whole FIRsuite project (75 instances) [33], 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.

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 [25], if available, or the CSD greedy algorithm [8].

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\textsubscript{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.

Minimizing the AD is an empirical approach to lower the delay of resulting hardware, its area and power. In Fig. 6 we observe that minimizing the AD but, by default, incorporate the automatic AD minimization as a second objective in MCM-Bits and tMCM.

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

### C. RQ2: Hardware Impact of the One-Bit Adder Metric

The synthesis results over all 86 benchmarks demonstrate that our approach MCM-Bits, based on counting the one-bit adders, reduces the number of LUTs by 7.59% on average, and 19.1% at most, compared to the state-of-the-art MCM-Adders. However, some outliers exist and in one case the number of LUTs, the delay and the power consumption actually increased by 11%, 41% and 19%, respectively. This is due to our test setting, which starts with a heuristic number of adders in the beginning, and has 30 minutes to find the optimal solution. Allowing for more resolution time and/or giving a good warmstart resolves the problem.

TABLE III demonstrates detailed hardware results for the 11 image-processing instances, for which MCM-Bits always reduces the number of LUTs, though in rare cases the results coincide with the MCM-Adders solution, e.g., for U.3–1. We observe that minimizing the one-bit adders can sometimes lead to a larger adder count, as in the case Lp15, nevertheless 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. 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 0.5996 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 [6].

### Table II

|                   | #adders | #one-bit adders | adder depth |
|-------------------|---------|----------------|-------------|
| #LUTs             | 0.9602  | 0.9967         | 0.2367      |
| Delay             | 0.6623  | 0.5996         | 0.0915      |
| Power             | 0.9812  | 0.9659         | 0.2592      |
### D. RQ3 and RQ4: Advantages of Truncated Adder Graphs

In [23] 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 one-bit-adders reduction by 24% and 31% compared to MCM-Bits problem. 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.

### E. RQ5: Scalability of Our Approach

a) Number of target constants: We tested our approach on the 75 instances from FIRSuite and 11 from image-processing. These instances have from 3 and up to 121 target constants. During pre-processing, we modified the constants to make them positive and odd, leading to the maximum instance size reduced to 51 unique target constants. We observed that increasing the number of target constants does not necessarily increase the difficulty, e.g., one of the FIRSuite instances with 19 target constants is solved in less than a second. On the contrary, more target constants might lead to a more constrained design-space, and hence faster resolution time.

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.

### TABLE III

| 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     | 24 | 45 | 3.771 | 111  |       | 4     | 24 | 45 | 3.719 | 108  |       | 4     | 24 | 45 | 3.658 | 85   |       |
| G.5   | 5     | 58 | 60 | 4.484 | 168  |       | 5     | 58 | 60 | 4.369 | 175  |       | 5     | 58 | 60 | 4.348 | 139  |       |
| HP5   | 4     | 42 | 42 | 3.876 | 121  |       | 4     | 42 | 42 | 3.914 | 134  |       | 4     | 42 | 42 | 3.774 | 89   |       |
| HP9   | 5     | 50 | 50 | 4.056 | 158  |       | 5     | 50 | 50 | 4.035 | 165  |       | 5     | 50 | 50 | 4.002 | 146  |       |
| HP15  | 12    | 122| 122| 6.317 | 401  |       | 12    | 122| 122| 6.174 | 421  |       | 12    | 122| 122| 5.757 | 270  |       |
| L.3   | 3     | 34 | 36 | 3.842 | 116  |       | 3     | 34 | 36 | 3.833 | 112  |       | 3     | 34 | 36 | 3.826 | 102  |       |
| LP5   | 6     | 66 | 68 | 4.236 | 213  |       | 6     | 66 | 68 | 4.335 | 209  |       | 6     | 66 | 68 | 4.346 | 173  |       |
| LP9   | 12    | 157| 157| 7.266 | 537  |       | 12    | 157| 157| 7.743 | 516  |       | 13    | 157| 157| 6.164 | 361  |       |
| LP15  | 26    | 313| 315| 15.312| 1359 |       | 27    | 250| 258| 9.055 | 1218 |       | 27    | 250| 258| 16.179| 959  |       |
| U.3-1 | 4     | 32 | 33 | 3.817 | 104  |       | 4     | 32 | 33 | 3.817 | 104  |       | 4     | 32 | 33 | 3.714 | 63   |       |
| U.3-2 | 5     | 61 | 58 | 4.049 | 169  |       | 5     | 49 | 49 | 4.436 | 167  |       | 5     | 49 | 49 | 3.997 | 102  |       |

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 significantly 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.

### TABLE III

| 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     | 24 | 45 | 3.771 | 111  |       | 4     | 24 | 45 | 3.719 | 108  |       | 4     | 24 | 45 | 3.658 | 85   |       |
| G.5   | 5     | 58 | 60 | 4.484 | 168  |       | 5     | 58 | 60 | 4.369 | 175  |       | 5     | 58 | 60 | 4.348 | 139  |       |
| HP5   | 4     | 42 | 42 | 3.876 | 121  |       | 4     | 42 | 42 | 3.914 | 134  |       | 4     | 42 | 42 | 3.774 | 89   |       |
| HP9   | 5     | 50 | 50 | 4.056 | 158  |       | 5     | 50 | 50 | 4.035 | 165  |       | 5     | 50 | 50 | 4.002 | 146  |       |
| HP15  | 12    | 122| 122| 6.317 | 401  |       | 12    | 122| 122| 6.174 | 421  |       | 12    | 122| 122| 5.757 | 270  |       |
| L.3   | 3     | 34 | 36 | 3.842 | 116  |       | 3     | 34 | 36 | 3.833 | 112  |       | 3     | 34 | 36 | 3.826 | 102  |       |
| LP5   | 6     | 66 | 68 | 4.236 | 213  |       | 6     | 66 | 68 | 4.335 | 209  |       | 6     | 66 | 68 | 4.346 | 173  |       |
| LP9   | 12    | 157| 157| 7.266 | 537  |       | 12    | 157| 157| 7.743 | 516  |       | 13    | 157| 157| 6.164 | 361  |       |
| LP15  | 26    | 313| 315| 15.312| 1359 |       | 27    | 250| 258| 9.055 | 1218 |       | 27    | 250| 258| 16.179| 959  |       |
| U.3-1 | 4     | 32 | 33 | 3.817 | 104  |       | 4     | 32 | 33 | 3.817 | 104  |       | 4     | 32 | 33 | 3.714 | 63   |       |
| U.3-2 | 5     | 61 | 58 | 4.049 | 169  |       | 5     | 49 | 49 | 4.436 | 167  |       | 5     | 49 | 49 | 3.997 | 102  |       |
b) Target constants word length: The target constants from the FIRSuite project are represented with word lengths up to 19 bits. Due to the solvers’ integrality tolerance, our model is sensitive to numerical instabilities when a target constant requires more than 12 bits to be represented. This tolerance can be adjusted at the cost of higher solving time and the issue is common to ILP models. At the current state of the MILP solvers, the largest target coefficient word length that can be reasonably handled is around 20 bits, otherwise the solver cannot ensure the correctness of the solution. In any event, we ensure that the returned solution is a correct adder graph, verifying the topology a posteriori. For larger constants, a workaround involving a multiple-step process should be developed. For example, constants used in cryptography are usually hundreds of bits and could be split into multiple smaller constants, for which an MCM instance is solved and whose result is collected back, as recently proposed by Aksoy et al. [37].

c) Input word length: On every instance, we solved the MCM-Bits considering an 8-bit input word length. For image processing benchmarks we additionally experimented for 16-bit and 32-bit input word lengths. Interestingly, proving optimality is easier when increasing the input word length: LP9 benchmark timed out for 8- and 16-bit input word length but for the 32-bit input word length the solver proved optimality in less than a minute. For higher data word lengths, the solver-related numerical instabilities might lead to errors in the one-bit adder count, hence to an overflow in the one-bit adder metric into half-adders and full-adders could further improve the results for ASICs. We did not consider the use of DSPs in this work but we believe that combining our one-bit adder approach with, for example, [38] would permit to take maximum advantage of the built-in component of FPGAs.

Given enough time, our tool permits to find optimal solutions and prove optimality with an exhaustive search, or quickly propose good candidate solutions. Overall, we proposed a push-button tool with various options allowing for tackling the MCM problem with respect to many end-user needs.

APPENDIX
APPLICATIONS

A. High-Level Vs Low-Level Metrics for Fully Parallel Neural Network Implementation

Another pertinent application for multiplierless MCM are neural network controllers (NNCs) implemented in embedded systems running in real-time. NNCs are neural networks typically implemented with just a few hidden layers and their goal is to serve as a cheaper, faster and more robust substitute to classic numerical control algorithms [39], [40], [41]. A typical approach is to use High Level Synthesis (HLS) to generate FPGA implementations, which do not necessarily take advantage of the structured nature of DNNs or optimize the dense layer computation. Hence, it is pertinent to explore the possibility of applying our MCM techniques to replace the dense layer matrix computation by fully-parallel adder-graphs, one for each neuron. Some work in combining MCM and NNCs has recently been done [4] in the context of quantization-aware training, where the float32 model was retrained for low-cost MCM solutions. However, as retraining is not always possible, here we explore a direct application of MCM to quantized NNC weights.

Our goal here is to test the scalability of the proposed MCM formulations and evaluate possible gains of the low-level MCM-Bits model w.r.t. MCM-Adders on real-life NNCs. Testing the tMCM model would require an in-depth error analysis of the NNC to deduce the accuracy requirement for each layer. As the scalability of tMCM model is closely correlated with the one of MCM-Bits, we leave this research question out of scope of this paper.

a) Benchmarks: For our experiments, we consider two neural networks from the ARCH competition [42]: ACC3, which is an adaptive cruise control system of a car and Tora benchmark, which controls transitional oscillations. The ACC3 network has 5 inputs, passing through 3 hidden layers with 20 neurons each, and has an output layer of dimension 1. The Tora benchmark has 4 inputs, 100 neurons in each of 3 hidden layers and an output layer of dimension 1.

b) Word lengths: The weight matrices from the ARCH competition are given in float32 format, hence the question of quantization to fixed-point arises. The difficulty here is that for NNCs weight quantization impacts not only the mean square error (MSE) of the network, but also the safety of the controller. In [4], it was shown that using the NNV tool [43] which performs a formal verification for NNCs, and some retraining, 8-bit FxP coefficients yield a safe controller with an MSE similar to the baseline float32 model. For the input-output word length of each layer, which is necessary for...
TABLE IV

| NNC  | Model       | #parameters | \(\sum N_A\) | \(\sum B\) | R   |
|------|-------------|-------------|--------------|------------|-----|
| ACC3 | MCM-Adders  | 920         | 18.6         | 995        | 20119 | 1.16 |
|      | MCM-Bits    | 1022        | 18148        | 1.19       |
| Tora | MCM-Adders  | 20500       | 68.5         | 14460      | 293439 | 1.03 |
|      | MCM-Bits    | 14462       | 242539       | 1.03       |

The MCM-Bits model, we chose 16 bits, which is typical for the application. This is a naive yet sensible baseline choice, and better results can be obtained with additional error analysis of the NNC.

c) Target coefficient set sizes: For each layer we replace an \(N \times M\) matrix multiplication with \(N\) adder graphs for \(M\) constants each. For example, the first hidden layer of Tora controller requires solving 4 MCM problems for at most 100 constant weights. As both controllers have only one output, the output layer actually requires only single constant multiplications (SCM). In total, we solved 45 instances of MCM and 20 instances of the SCM problem for ACC3, and 204 MCM and 100 SCM instances for Tora. In Tora, each instance of the hidden layer, after removing the duplicates during pre-processing, had on average 65.8 target coefficients, and ACC3 had 18.6 unique coefficients, on average.

d) Results: In our experiments we observed that with MCM-Adders, the number of adders per adder graph is roughly equal to the number of target constants plus a few overhead adders. This number of overhead adders tends to decrease with the number of target constants: larger instances are easier to solve when counting the number of adders. The adder depth ranged from 2 to 10, independently of the target coefficients. For ACC3, we observe that 1.16 adders are required per target constant, while 1.03 adders per target constant is enough for the larger NNC Tora. In general, in both the NNCs and digital filter applications, we observe a positive trend that increasing the number of unique target coefficients does not represent a bottleneck for our MCM approach.

These experiments again confirm that when we consider the lower-level metric in MCM-Bits, the number of one-bit adders is decreased by 10% for the smaller network, ACC3, and by 17% for the larger network, Tora. The detailed values are presented in Table IV.

To conclude, application of our MCM models for safety-critical NNCs yields promising results. Using a static analysis for rounding error propagation, e. g. Daisy [34], will permit to better decide on the input/output word lengths for our model. Moreover, solving a (mixed-)precision assignment problem between layers will permit to also apply the tMCM model and further reduce the resource while providing precision guarantees.

B. Very Large Constant Multiplication for Cryptography

Multiplierless implementation of MCM can also be useful for cryptographic applications. These applications usually involve multiplications with very large constants. A single constant which is represented with hundreds of bits can be split into multiple smaller word length constants. Then, the multiplication can be rewritten as

\[
C \times x = (c_0 + 2^{t_1}c_1 + \cdots + 2^{t_n}c_n) \times x
\]

\[
= c_0 \times x + 2^{t_1} (c_1 \times x) + \cdots + 2^{t_n} (c_n \times x).
\]

The multiplication of the input \(x\) by the target constants \(c_i\) can be solved using an MCM and the overall result then can be computed using shifts and \(n\) adders. The splitting part can be optimized and Aksoy et al. [37] recently proposed a heuristic to solve this problem and used various techniques to solve the MCM part.

Our intention is to illustrate that our proposed MCM-Adders model can be used as well. Hence, we leave the use of heuristic out of scope and consider a naive split every \(p\) bits to generate an MCM instance.

For our experiment, we took a 204-bit constant \(C\) from the definition of Anomalous elliptic curve described in [44]

\[
C = 153478980553715805908905767153478980553715805908905767213143188232075319630356737503096292.
\]

We test our MCM-Adders in two settings: we split the constant into 8- and 12-bit chunks. The only heuristic we apply is to skip consequent zeros between chunks.

As a result, in the 8-bit experiment, we obtained 25 target constants which led to 10 unique odd coefficients. Our MCM-Adders proved that exactly 10 adders are required to perform the 10 8-bit multiplications. To collect these results together as in (34), 24 more structural adders are necessary leading to a total of 34 adders to compute \(C \times x\).

For the 12-bit experiment, we obtained a target set of 16 constants, 10 out of which are unique. Within the allocated time, 30 minutes, our tool found a solution requiring 18 adders. Adding the 15 structural adders to recompute the \(C \times x\) yields the total cost of 33 adders.

We observe a trade-off between the MCM cost and the recollection cost which should be explored. To get an optimal result given a target constant, all combinations of the size and the uniformity of the chunks could be tested with an external loop which can take a considerable time. Instead, it might be pertinent to model these constraints into our versatile ILP model and to rely on the solver to efficiently explore the design space.

REFERENCES

[1] A. Oppenheim and R. W. Schäfer, *Discrete-Time Signal Processing*, Englewood Cliffs, NJ, USA: Prentice-Hall, 1989.

[2] M. Kumm, A. Volkova, and S.-I. Filip, “Design of optimal multiplierless FIR filters with minimal number of adders,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 2, pp. 658–671, Feb. 2023.

[3] M. Garrido, K. Möller, and M. Kumm, “World’s fastest FFT architectures: Breaking the barrier of 100 GS/s,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 4, pp. 1507–1516, Apr. 2019.
[4] T. Hubermann, J. Kühle, M. Kumm, and A. Volkova, “Hardware-aware quantization for multiplierless neural network controllers,” in Proc. IEEE Asia Pacific Conf. Circuits Syst., Shenzhen, China, Nov. 2022. Paper AP2025. [Online]. Available: https://ieeexplore.ieee.org/xpl/conhome/1000091/all-proceedings and http://apccas2022.org/fullprogram.pdf

[5] K. Watt and E. Jamro, “Constant coefficient multiplication in FPGA structures,” in Proc. 26th Euromicro Conf. EUROMICRO Inform., Inventing Future, Sep. 2000, pp. 252–259.

[6] M. Kumm, “Optimal constant multiplication using integer linear programming,” IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 65, no. 5, pp. 567–571, May 2018.

[7] A. D. Booth, “A signed binary multiplication technique,” Quart. J. Mech. Appl. Math., vol. 4, pp. 236–240, Jan. 1951.

[8] R. Bernstein, “Multiplication by integer constants,” J. Softw. Pract. Exp., vol. 16, pp. 641–652, Jul. 1986.

[9] A. G. Dempster and M. D. Macleod, “Constant integer multiplication using minimum adders,” IEEE Proc.—Circuits, Devices Syst., vol. 141, pp. 407–413, Oct. 1994.

[10] M. Potkonjak, M. B. Srivastava, and A. P. Chandrakasan, “Multiple constant multiplications: Efficient and versatile framework and algorithms for exploring common subexpression elimination,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 15, no. 2, pp. 151–165, Feb. 1996.

[11] V. Lefèvre, “Multiplication by an integer constant,” INRIA, Res., Paris, France, Tech. Rep. RR-4192, May 2001. [Online]. Available: https://hal.inria.fr/inria-00072430

[12] Y. Voronenko and M. Püschel, “Multiplierless multiple constant multiplication,” ACM Trans. Algorithms, vol. 3, no. 2, p. 11, 2007.

[13] J. Thong and N. Nicolici, “Combined optimal and heuristic approaches for multiple constant multiplication,” in Proc. IEEE Int. Conf. Comput. Design, Oct. 2010, pp. 266–273.

[14] O. Gustafsson, “Towards optimal multiple constant multiplication: A hypergraph approach,” in Proc. 42nd Asilomar Conf. Signals, Syst. Comput., Oct. 2008, pp. 1805–1809.

[15] M. Kumm, Multiple Constant Multiplication Optimizations for Field Program. Gate Arrays. Wiesbaden, Germany: Springer Fachmedien Wiesbaden, 2016.

[16] V. Lagoon and A. Metodi, “Deriving optimal multiplication-by-constant circuits with a SAT-based constraint engine,” in Proc. ModRef 19th Workshop Constraint Modelling Reformulation, Sep. 2020, pp. 1–6.

[17] L. Aksoy, E. Günes, and P. Flores, “Search algorithms for the multiple constant multiplications problem: Exact and approximate,” Microprocess. Microsyst., vol. 34, no. 5, pp. 151–162, Aug. 2010.

[18] C.-C. Yen, M.-Y. Yeh, and M.-S. Chen, “Unified multiple constant multipliers for dynamic exchange of low-precision kernels on FPGAs,” IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 69, no. 5, pp. 1617–1621, Mar. 2022.

[19] S. Coward, G. A. Constantinides, and T. Drane, “Automatic datapath optimization using E-graphs,” in Proc. IEEE 29th Symp. Comput. Arithmetic (ARITH), Sep. 2022, pp. 43–50.

[20] K. Johansson, O. Gustafsson, and L. Wanhammar, “Bit-level optimization of shift-and-add based FIR filters,” in Proc. IEEE 14th Int. Conf. Electron., Circuits Syst., Dec. 2007, pp. 713–716.

[21] R. Guo, L. S. DeBrummer, and K. Johansson, “Truncated MCM using pattern modification for FIR filter implementation,” in Proc. IEEE Int. Symp. Circuits Syst., May 2010, pp. 3881–3884.

[22] F. de Dinechin, S.-I. Filip, M. Kumm, and L. Forget, “Table-based versus shift-and-add constant multipliers for FPGAs,” in Proc. IEEE 26th Symp. Comput. Arithmetic (ARITH), Jun. 2019, pp. 151–158.

[23] R. Garcia, A. Volkova, and M. Kumm, “Truncated multiple constant multiplication with minimal number of full adders,” in Proc. IEEE Int. Symp. Circuits Syst. (ISCAS), May 2022, pp. 263–267. [Online]. Available: https://hal.archives-ouvertes.fr/hal-03582935

[24] F. de Dinechin and B. Pasca, “Designing custom arithmetic data paths with FLoPoCo,” IEEE Design Test Comput., vol. 28, no. 4, pp. 18–27, Jul. 2011.