Towards optimized multiplierless arithmetic circuits

Rémi Garcia

COLLEGE ED MATHEMATIQUES
DOCTORAL STIC
PAYS DE LA LOIRE

Nantes Université

THÈSE DE DOCTORAT DE

NANTES UNIVERSITÉ

ÉCOLE DOCTORALE N° 641
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par
Rémi GARCIA

Towards optimized multiplierless arithmetic circuits

Thèse présentée et soutenue à Nantes, le 25 octobre 2023
Unité de recherche : LS2N – Laboratoire des Sciences du Numérique de Nantes

Rapporteurs avant soutenance :

Olivier SENTIEYS	Professeur des universités – Inria, University of Rennes
Arnaud TISSERAND	Directeur de recherche – CNRS, Lab-STICC UMR6285

Composition du Jury :

Président :	Christine SOLNON	Professeur des universités – INSA Lyon, CITI
Examinateurs :	Oscar GUSTAFSSON	Associate professor – Linköping University
	Olivier SENTIEYS	Professeur des universités – Inria, University of Rennes
	Christine SOLNON	Professeur des universités – INSA Lyon, CITI
	Arnaud TISSERAND	Directeur de recherche – CNRS, Lab-STICC UMR6285
	Lilia ZAOURAR	Chercheur expert CEA – CEA List, Saclay
Dir. de thèse :	Laurent GRANVILLIERS	Professeur des universités – Nantes Université
Co-enc. de thèse :	Anastasia VOLKOVA	Chargée de recherche – Inria, INSA Lyon, CITI

Remerciements (Acknowledgments)

Je souhaiterais adresser mes premiers remerciements à mon encadrante, Anastasia Volkova, qui m'a accompagné ces dernières années. Je n'aurais pu espérer une encadrante plus investie. Je veux également remercier Laurent Granvilliers, mon directeur de thèse, pour m'avoir suivi et avoir toujours répondu à mes questions.

Je remercie évidemment mes deux rapporteurs, Arnaud Tisserand et Olivier Sentieys, pour la relecture de mon manuscrit et leurs retours et remarques ainsi que pour avoir accepté de participer à mon jury de thèse. Merci également à Christine Solnon, Lilia Zaourar et Oscar Gustafsson de participer à mon jury. C'est un honneur pour moi que vous évaluiez mon travail.

Je souhaite également remercier les collègues avec qui j'ai travaillé ainsi que celles et ceux qui m'ont beaucoup appris du métier d'enseignant-chercheur et de chercheur : Alexandre Goldsztejn, Anthony Przybylski, Benoît Delahaye, Chantal Enguehard, Christophe Jermann, Didier Robbes, Eric Languenou, Florent de Dinechin, Jonas Kühle, Martin Kumm, Silviu Filip, Vincent Lostanlen et Xavier Gandibleux.

Je veux maintenant remercier mes collègues doctorant·e·s et ami·e·s. Ces années de thèse auraient été radicalement différentes sans votre présence chaleureuse. Au risque d'oublier des prénoms, je tiens à remercier Anna pour nos discussions qui s'éternisent, David pour notre collaboration (inachevée) sur la sensibilisation à l'hygiène numérique, Josselin pour les parties de Yahtzee, Julie pour nos fresques et ton enthousiasme, Marinna pour tes bonnes ondes, Mathieu pour nos discussions sur le bobail, Mathieu pour avoir porté LOGIN, Thibault pour nos échanges sur la psychologie humaine. La santé mentale des doctorant·e·s étant mise à rude épreuve, toutes ces interactions m'ont été extrêmement bénéfiques. Plus largement, je souhaiterais également remercier d'autres collègues et anciens collègues ainsi que les personnels administratifs et techniques qui m'ont aidé ces dernières années : Antoine, Sophie, Thomas, Charlery, Jean-Yves, Alexiane, Virginie, Christine, Lucie, Véronique et Vivian.

Au delà du cadre de la thèse, l'Aïkido m'aura permis de respirer. Ainsi, je remercie mon enseignant Yannig et son assistant Sylvain pour leur disponibilité et leur persévérance pour me pousser à progresser. Cette dernière année de thèse a aussi été l'occasion de m'investir dans d'autres activités et je tiens à remercier des personnes dont l'engagement est inspirant : Anette, Cristo, Fanny, Juliette et Matthieu.

Mes remerciements vont également à ma famille et mes ami·e·s qui m'ont donné l'énergie de travailler sur ma thèse ces dernières années et qui m'ont toujours accompagné avec bienveillance lorsque les *deadlines* ne me laissaient plus de temps. Merci à Benoit, Jocelin, Killian, Lorys, Marion, Maxime, Mélissa, Samuel, Alice et Damien. Un merci particulier à mes parents qui ont tout fait pour m'aider chaque fois que l'occasion se présentait.

Enfin, je remercie Camille sans qui il m'aurait été impossible de trouver l'énergie durant cette thèse. Ces dernières années auront été celles de notre mariage, de notre installation à la campagne et de la naissance de notre fille, de nombreux projets que tu as en grande partie portés pour nous.

Table of Contents

Introduction **1**

I Pre-requisites **5**

1 Fixed-Point Operators and Hardware Circuits **7**
1.1 Fixed-point arithmetic 8
1.2 Field-programmable gate arrays 13
1.3 Multiplication by constants 16
1.4 Linear digital filters as arithmetic operators 21

2 Mathematical Modeling **25**
2.1 Writing an MILP model 26
2.2 Improving an MILP model 31
2.3 Choosing between MILP models 35

3 Towards automatic optimization of arithmetic operators **39**

II Multiple Constant Multiplication **43**

4 Improving high-level MCM model **45**
4.1 Introduction 45
4.2 Modeling the adder graph topology 46
4.3 Bounding and minimizing the adder depth 53
4.4 Fine-tuning the MILP solver 57
4.5 Conclusion 58

5 MCM with a low-level cost metric **61**
5.1 Introduction 61
5.2 Modeling the one-bit adder count 62
5.3 Experiments 66

5.4 Conclusion 70

6 Truncations **71**
6.1 Introduction 71
6.2 Modeling truncations and error propagation 72
6.3 Optimization and hardware results 79
6.4 Conclusion 82

7 Pipelined adder graphs **83**
7.1 Introduction 83
7.2 Pipelining in a high-level model 85
7.3 Experiment results 89
7.4 Conclusion 93

III Digital Filters **97**

8 IIR Filters **99**
8.1 Introduction 99
8.2 The filter design and quantization steps 100
8.3 Incorporating the implementation step 104
8.4 Enhancing the model and wrapping up the solving process 104
8.5 Experimental results and discussion 106
8.6 Conclusion 116

Conclusion and Perspectives **119**

Appendices **123**

A MILP Models **125**

B Benchmark description **143**

C Full results **147**

D Résumé long en français **161**

List of Figures

1 Example of adder graphs for target constants 7 and 23. 2

1.1 Bit string for integer representation on w bits. 8
1.2 Bit string for two's complement fixed-point representation on w bits. . . . 10
1.3 Compilation stages. 14
1.4 Additions with full adders. 15
1.5 Simplified representation of a full adder. 15
1.6 Adder graphs computing $93x$. 18
1.7 Data sharing for MCM 19
1.8 Example of specifications. 22
1.9 Hardware implementation of FIR and IIR filters, direct form. 23
1.10 Hardware implementation of FIR and IIR filters, transposed form. 23

2.1 Adder graph and a focus on one adder. 30
2.2 Comparison process. 36

3.1 FloPoCo simplified interface. 40

4.1 Adder graphs obtained for target constants $\{7, 19, 31\}$. 47
4.2 Classic and proposed adder models. 47
4.3 Branching order. 58

5.1 Different adder graph topologies computing the same outputs: $49x$ and $51x$ 62
5.2 Counting one-bit adders. 64

6.1 Computing $49x$ and $51x$ via the fundamental 17 for a 3-bit input/output . 72
6.2 Model of an adder in presence of errors ε and truncations t. 73
6.3 Counting one-bit adders and propagating errors in presence of truncations. 74
6.4 Counting one-bit adders in case of truncations. 78
6.5 Comparing one-bit adder count w. r. t. precision. 80

7.1 Pipelining a fixed adder graph. 84
7.2 (Pipelined) adder graphs which compute $5x$, $69x$, $553x$ and $2483x$. 85
7.3 Postprocessing replaces unnecessary adders with same-cost registers 92

8.1 Second-order IIR specification, transfer function and hardware implementation. . . . 100
8.2 High-level structure of the global ILP model IIRoptim. . . . 106
8.3 Interface of the proposed tool. . . . 107
8.4 Proposed families of benchmarks. . . . 109
8.5 Frequency response of the compensator `hp0` [CVKF03, SRZ12]. . . . 109
8.6 Implementations for $\texttt{lp1}_4$ benchmark. . . . 112
8.7 Comparison of the classic and proposed structures. . . . 113
8.8 The $\texttt{lp1}_4$ benchmark can be implemented with 7 adders. . . . 114
8.9 Benchmark results of resources, power and critical path delay on FPGAs. . 115
8.10 Benchmark results of resources, power and critical path delay on ASICs. . 117

List of Tables

1.1 Error bounds according to rounding modes. 12
1.2 Full adder truth table. 15

4.1 Constants and variables used in the MILP formulation 50
4.2 Number of adders obtained with each method for MCM-Adders. 52
4.3 Number of adders and adder depth obtained with each method. 56
4.4 Solving time to prove optimality for MCM-Adders with branching priority. 58

5.1 Number of adders and one-bit adders obtained with MCM-A. vs MCM-B. . 67
5.2 Hardware results for 8-bit inputs. 69
5.3 Correlation between proxy variables and hardware metrics. 69

6.1 Number of adders and one-bit adders obtained with MCM-Bits vs tMCM. 80
6.2 Hardware results for 8-bit inputs with MCM-Bits vs tMCM. 81

7.1 Comparison of optimization results with pipelining vs RPAG. 90
7.2 A posteriori pipelining of MCM-Bits vs PMCM-Bits 91
7.3 Critical path delay for each method. 93
7.4 Comparison of hardware results for pipelined adder graphs. 94

8.1 Sets of lowpass filters used for the IIR experiments. 107
8.2 Results for our global optimization method vs applying optimal MCM upon quantized coefficients. 110

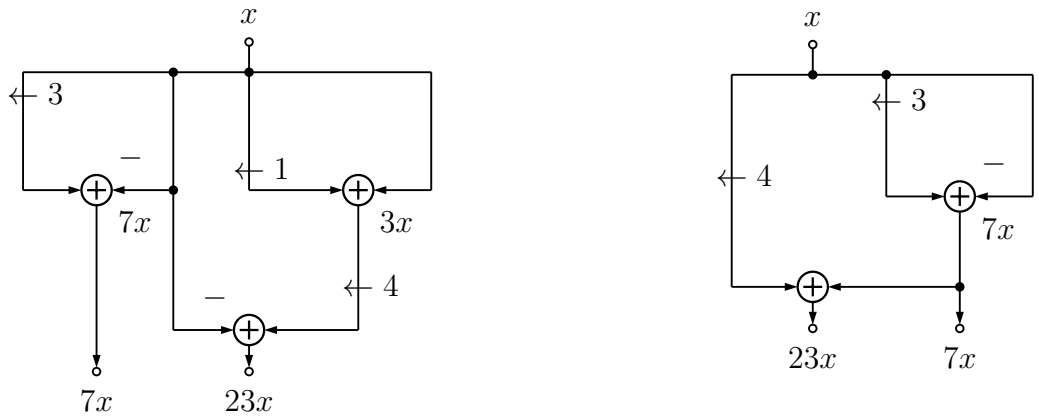
Introduction

> "In the beginning there was nothing,
> which exploded."
>
> Terry Pratchett (1948-2015),
> *Lords and Ladies (1992)*

Modern world benefits from computer science in many ways. Although we probably first think of personal computers and smartphones, electronic devices which do embedded computations are everywhere. For instance, hearing-aids or smartwatches are computers dedicated to a specific task. Embedded systems can also be used for less common tasks, such as bird voice detection from audio recording. These devices either work on batteries or on small solar panels and instantaneous results are usually needed. It is clear that we need zero delay for the hearing-aids and that their battery has to be small.

This leads to a specific need for dedicated optimized circuits and hardware designers put a lot of work and efforts in that direction. However, with more and more complex problems and systems, it is not anymore realistic to optimize by hand the circuit design. High-level tools have been developed to simplify the design, yet arithmetic cores implementation are still left to the hardware designer. Hence, digital signal processing designers, for example, still need to be integrated circuit experts. To alleviate embedded systems designers arduous hand-tuning task, automatic tools that also guide specific architectural decisions are needed. Our work is on the automation of arithmetic kernels/operators implementation which often involves solving combinatorial problems.

To solve these problems, we will use techniques from the operations research (OR) domain. In particular, we will address hardware design problems using mathematical modeling with the objective to model the hardware, specifically field-programmable gate arrays (FPGAs). We will specifically use the mixed-integer linear programming (MILP) approach to improve the state-of-the-art for the multiple constant multiplication and filter design problems. This will lead to more efficient implementations reducing the delay, power consumption and resources usage in FPGAs. Ideally, this thesis will facilitate the use of OR techniques by hardware designers in order to solve their design problems. This way, in addition to generating better hardware, our goal is to bring the OR and hardware design communities closer.



(a) Adder graph for target constants 7 and 23.

(b) Optimal for target constants 7 and 23.

Figure 1: Example of adder graphs for target constants 7 and 23.

Context and problem

The floating-point (FP) number representation, which is inspired from scientific notation, is agnostic to the application it is used for. This makes the FP arithmetic the first and simpler choice for generic applications. However, for domain-specific applications fixed-point (FxP) arithmetic is usually preferred. At the cost of some extra *a priori* work, it is possible to implement algorithms with a better efficiency and a lower hardware cost. FxP numbers can be represented with no more bits than strictly necessary. This way, each computation/operator can be specifically designed to use exactly the required resources and no more. With our work, we will take advantage of the FxP representation and accurately build every circuit.

The standard way to build a circuit consists in describing it with an hardware description language (HDL). Although high-level synthesis (HLS) tools alleviate the hardware designers from a significant part of the work, efficient arithmetic cores are still often designed in HDL. However, the required expertise to do it efficiently can be blocking, thus there is a need for automatic arithmetic cores generation. In particular, in this thesis, we will address the automation of the multiple constant multiplication (MCM) operator.

Finding the most efficient implementation of the MCM operator is an arduous task. A direct approach would simply consist in using generic multipliers. However, these are costly as they are meant to be used with variables and not constants. In our case, building dedicated circuits is preferable and we usually replace multiplications with bit-shifts, which are power-of-two multiplications, and additions/subtractions. This is called the shift-and-add approach. For example, we can compute the multiplication of a variable with the constants 7 and 23 using the adder graph represented in Figure 1a.

In this thesis, we will address the MCM problem which can be simply defined as follows: "given a set of target constants to multiply with, find the best shift-and-add implementation." We will need to define "the best implementation", as formal models for FPGAs are not available and we want to avoid performing synthesis for each adder graph to evaluate

its cost. Hence, we introduce proxy variables which will be easier to evaluate and which will be used as an estimation of the hardware cost. The most common one is the adder count within a shift-and-add solution, which includes both additions and subtractions. Using this proxy variable, we can predict that the adder graph represented in Figure 1b, which requires two adders, will be better than the one in Figure 1a, after synthesis.

Several papers address the MCM problem, some with heuristics [Ber86, DM94, ACFM12, KZFC12] and others are optimal approaches [Gus08, AGF10, Kum16, Kum18]. Despite the apparent simplicity of the problem, there still is room for improvement. Indeed, we see two main limitations of current methods. First, optimal methods require a lot of computing time and with OR tools and knowledge, this can certainly be reduced. Second, these methods rely on the number of adders and we will show that proxy variables closer to the final hardware can be defined. Moreover, in some cases, high throughput is necessary and introducing registers into adder graphs, which is called pipelining, is common [Par99, KZ11, KZFC12]. However, it increases the resource usage and we will also address this problem to minimize the cost of pipelined adder graphs.

Since MCM is a basic building block of digital filters, some works combine finite impulse response (FIR) filter design with MCM [KVF23]. However, tackling infinite impulse response (IIR) filters, whose design is highly nonlinear, has not yet been done in MILP.

Objective

Our main objective is to provide a fine-grain model of the hardware and encompass it in mathematical models. First, we will correct and enhance the state-of-the-art MILP-based model [Kum18] which solves the MCM problem w. r. t. the number of adders. This high-level metric, although useful in practice, can be refined and, in our work, we will address a lower-level metric: the one-bit adders count. This way, we propose an automatic tool to provide better adder graphs.

To further reduce the implementation cost, we will consider truncated adder graphs, *i. e.*, adder graphs in which we truncated bits in the data path to reduce the cost of the adders, at the cost of some error, which we guarantee to bound by construction with a user-given value. We will address this difficulty by proposing an error model to correctly propagate errors in the adder graph. Finally, one of the objectives of this thesis is to show the versatility of mathematical modeling. We will do so by searching for pipelined adder graphs at a minimal cost by enhancing our MILP models with new constraints.

Finally, in our work we will address the combination of MCM with the second-order IIR filter design. Our goal is to provide a hardware-aware search for filter coefficients. We propose a method to do the co-design of filter coefficients and their implementation with the MCM problem. This way, we will present a typical application of the MCM problem and that it can be tackled using the model for MCM as part of a global model.

Overall our goal is to implement all our approaches within code generator tools. The objective is to automatically provide HDL code for hardware designers to use in complete circuits.

Outline

This thesis is organized in the order of the problems presented above. Naturally, we start with high-level metrics and, step by step, we go more details with lower-level metrics. We then conclude our work with an application. The document is divided into three parts, each subdivided into chapters.

Part I. In Chapter 1, we start with main notions of fixed-point arithmetic. Then, we present our target, FPGA, and the problems we aim at solving, MCM and filter design. In Chapter 2, we provide basic knowledge over mathematical modeling and the mixed-integer linear programming approach. Then, in Chapter 3, we describe our overall goal of providing efficient code generators in its context.

Part II. This second part is dedicated to our solutions for the MCM problem, based on more and more fine-grained models of the hardware. First, in Chapter 4, we present our work on MCM-Adders. Then, in Chapter 5 and Chapter 6, we tackle the one-bit adder lower-level metric, including truncations. These chapters are more technical and include a detailed error analysis. Finally, we solve the PMCM problem in Chapter 7.

Part III. In this last part, we present an application which involves the MCM problem. With this last contribution, which chronologically came first, we present the co-design of second-order IIR filters and MCM.

Publications

The work carried out during this thesis have been published in the following international journals and peer-reviewed conferences:

[GV23a] Rémi Garcia and Anastasia Volkova. Multiple Constant Multiplication: From Target Constants to Optimized Pipelined Adder Graphs. In *33rd International Conference on Field-Programmable Logic and Applications (FPL)*, Gothenburg, Sweden, September 2023. doi: 10.1109/FPL60245.2023.00027

[GV23b] Rémi Garcia and Anastasia Volkova. Toward the Multiple Constant Multiplication at Minimal Hardware Cost. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 70(5):1976–1988, 2023. doi: 10.1109/TCSI.2023.3241859

[GVK22a] Rémi Garcia, Anastasia Volkova, and Martin Kumm. Truncated Multiple Constant Multiplication with Minimal Number of Full Adders. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, Austin, Texas, United States, May 2022. doi: 10.1109/ISCAS48785.2022.9937441

[GVK+22b] Rémi Garcia, Anastasia Volkova, Martin Kumm, Alexandre Goldsztejn, and Jonas Kühle. Hardware-aware Design of Multiplierless Second-Order IIR Filters with Minimum Adders. *IEEE Transactions on Signal Processing*, pages 70:1673–1686, 2022. doi: 10.1109/TSP.2022.3161158

Part I

Pre-requisites

CHAPTER 1

Fixed-Point Operators and Hardware Circuits

> "My programming language was solder."
>
> Terry Pratchett (1948-2015)

There is a wide range of number representations which can be used in electronic computers. In particular, we will focus on finite-precision ones, *i. e.*, representations which use a finite number of digits to encode a number. Among these, the most common are the Floating-Point (FP) and the Fixed-Point (FxP) number formats. With generic processors, FP arithmetic, inspired by normalized scientific representation, is the norm. However, when implementing for dedicated hardware, such as Application-Specific Integrated Circuit (ASIC) or Field-Programmable gate Array (FPGA), it is useful to put the extra work of using FxP arithmetic. Indeed, the FxP number format permits to precisely describe a circuit for better energy-efficiency and smaller latency, than using FP and generic processors. Note that, to get the most of FxP implementations in resource-constrained environments, knowing the hardware on which the circuit will be implemented is essential to guide the implementation choices. There is room for improvement in many operator implementation choices, even for operations like multiplication with several constraints, which are computational bottlenecks for many applications, *e. g.*, digital signal processing.

First, the FxP arithmetic will be the main topic of this chapter. Then, we will provide some details on our hardware target, FPGA, and discuss the implementation of basic operations on it, such as additions. Finally, we will present more complex operators: the Multiple Constant Multiplication (MCM) and the Digital Filters.

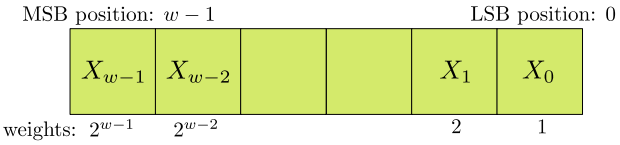


Figure 1.1: Bit string for integer representation on w bits.

1.1 Fixed-point arithmetic

1.1.1 Basics

Integers

The binary representation of a positive integer $X \in \mathbb{N}$ consists of rewriting X as a linear combination of powers of 2:

$$X = \sum_{i=0}^{w-1} 2^i X_i, \tag{1.1}$$

where w is the number of bits used to represent X and $X_i \in \{0, 1\}$ are the bits of X. We will call X_0 the Least Significant Bit (LSB) and X_{w-1} the Most Significant Bit (MSB). This representation is summarized in Figure 1.1. Using the binary representation with w bits permits to represent every positive integer between 0 and $2^w - 1$. Formally, we will write the set $[0, 2^w - 1] \cap \mathbb{Z}$ as $[\![0; 2^w - 1]\!]$ where $[\![\cdot; \cdot]\!]$ denotes an interval of integer numbers.

To extend this representation to signed numbers, $X \in \mathbb{Z}$, one could simply consider an additional bit to encode the sign:

$$X = (-1)^{X_{w-1}} \times \sum_{i=0}^{w-2} 2^i X_i. \tag{1.2}$$

An obvious drawback is that $X = 0$ has two representations, -0 and $+0$. This notation permits to represent every integer in $[\![-2^{w-1} + 1; 2^{w-1} - 1]\!]$.

Remark. Note that the MSB of a signed integer corresponds to the sign bit and that this signed notation does not lend itself to a simple hardware implementation as the sign needs to be treated separately.

With the signed notation, basic operations such as additions will differ between adding two number of the same sign or of opposite signs. However, using the two's complement notation permits to unify this [EL04, Lop14]. Using this representation, an integer $X \in \mathbb{Z}$ is written as

$$X = -2^{w-1} X_{w-1} + \sum_{i=0}^{w-2} 2^i X_i. \tag{1.3}$$

Then, the set of integers that can be represented with w bits becomes $[\![-2^{w-1}; 2^{w-1} - 1]\!]$.

It is also possible to derive from any integer the number of bits required to represent it exactly. Let $Z \in \mathbb{Z}$ be an integer. We remark that the set of representable integers is

not symmetrical, so we differentiate between positive and negative cases. If $Z \geq 0$, the number of bits w required to represent Z is:

$$w = \lceil \log_2 (Z + 1) \rceil + 1. \tag{1.4}$$

If $Z < 0$, the number of bits required to represent Z in two's complement is

$$w = \lceil \log_2 (-Z) \rceil + 1. \tag{1.5}$$

Then, given two unknown integers, X and Y, for which we only know their respective *word lengths*[1] and an operator $\diamond$, we can infer the word length of $Z = X \diamond Y$. For instance, if we denote w_X and w_Y the word lengths of X and Y respectively, then the word length of $Z = X \times Y$ is $w_Z = w_X + w_Y$ bits. In case of the addition, the word length of $Z = X + Y$ would be equal to $w_Z = \max(w_X + w_Y) + 1$ bits.

As illustrated with the "+" and "×" operators, given an operator $\diamond$, usually the word length of the result of $Z = X \diamond Y$ increases in comparison to the word length of the inputs. In Example 1, we explicitly illustrate this for the multiplication. In addition to that, we show the gap between the actual range of $X \times Y$ and the range of integers which could be represented with the necessary word length.

Example 1. Let X and Y be two integers represented using 4 bits each. Hence, we know that $X, Y \in [\![-8; 7]\!]$. It follows that $Z = X \times Y \in [\![-56; 64]\!]$. Hence, the number of bits required to represent the result of Z is

$$w = \max(\lceil \log_2 (64 + 1) \rceil, \lceil \log_2 (-56) \rceil) + 1 = 8. \tag{1.6}$$

Note that with 8 bits, every integer in the set $[\![-128; 127]\!]$ could have been represented.
Suppose we multiply the result Z by another 4-bit variables, $Z \times W \in [\![-512; 448]\!]$ can fit with 10 bits. If we computed the resulting range from the bit widths of Z and W, it would lead to 12 bits instead.

When possible, the range of the variables should be provided and range propagation should be preferred over word length propagation as it is more precise. Word length propagation gives a simple upper bound on the actual word length of the result. In particular, in Section 1.3, we will discuss the multiplication by constant where the constant could be seen as a variable whose range is equal to a single value. In that case, the growth of the bit width can be evaluated more precisely than simply working with word lengths. We illustrate this in Example 2 where two 6-bit signed integers are multiplied together. We could expect the result to require up to 12 bits. However, thanks to knowledge on the range, we can deduce that a smaller word length will be enough.

[1] sometime referred to as the *bit width*.

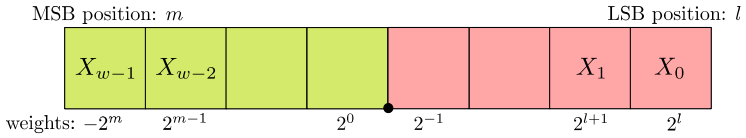


Figure 1.2: Bit string for two's complement fixed-point representation on w bits.

Example 2. Let X be an integer represented using 6 bits. Then, we know that $X \in [\![-32; 31]\!]$. We consider the multiplication $17 \times X$ where 17 requires 6 bits to be represented using the two's complement notation. It follows that $17 \times X \in [\![-544; 527]\!]$, hence 11 bits are enough to represent the result of $17 \times X$, instead of 12.

If one tries to represent a number outside of the representable range, then an overflow could arise. If not considered, the behavior of operators, and consequently of algorithms, could become completely unforeseeable. Hence, we have two possibilities: either we deal with the result – saturation to max value, wrap around, etc. – or we extend the bit width as necessary. In this thesis, we guarantee, by construction, that no overflow occurs by adjusting the word lengths as necessary.

Approximating real numbers

In fixed-point arithmetic, a real number is represented by the closest fixed-point number, x, as a scaled integer:

$$x = X \times 2^l, \tag{1.7}$$

where integer X is called the mantissa, l is the LSB position and 2^l the implicit scaling factor, not stored in memory.

In two's complement FxP notation, (1.3) becomes

$$x = \left(-2^{w-1} X_{w-1} + \sum_{i=0}^{w-2} 2^i X_i \right) \times 2^l, \tag{1.8}$$

which can be simplified as

$$x = -2^m X_{w-1} + \sum_{i=l}^{m-1} 2^i X_{i-l}, \tag{1.9}$$

where m is the MSB position. Note that we have the following relationship between the word length and the MSB/LSB positions:

$$w = m - l + 1. \tag{1.10}$$

In Figure 1.2, we summarize the format definition which just depends on the MSB, m, and LSB, l, where we assumed that the LSB position is negative. In the following we will refer to the format of a FxP number with the notation (m, l) [Lop14]. Beside the word

length propagation, after each operation, we need to evaluate the MSB and LSB positions. This differs for every algorithm and increasing the word length to do exact arithmetic comes with a cost which we could mitigate by rounding intermediate results.

1.1.2 Finite-precision error analysis for FxP arithmetic

Due to the finiteness of the FxP representation, most numbers cannot be represented exactly, regardless of the word length available. Hence, one solution is to fix a word length w and to represent the number with the closest w-bit FxP number. This leads to an error between the exact number and its representation. In practice, rounding is inevitable, as we usually work with quantities that are not exactly representable. Moreover, even when it is representable, we might want to save resources and round nevertheless. A software-inspired approach could consist in fixing a format for intermediate quantities. However, this is not efficient in hardware as this probably overestimates the format in comparison to the one strictly necessary. What we will do instead, is fixing an accuracy for the output and determine the intermediate word lengths.

Given an accuracy to meet, to approximate a real number, x, to the smallest FxP representation, $\widetilde{x}$, such that $|x - \widetilde{x}| \leq \overline{\varepsilon}$, we fix the MSB position

$$m = \begin{cases} 1 & \text{if } x = 0, \\ \lceil \log_2 (c+1) \rceil & \text{if } x > 0, \\ \lceil \log_2 (-c) \rceil & \text{if } x < 0, \end{cases} \tag{1.11}$$

to ensure that the FxP number will not overflow. Then, starting from $l = m - 1$, we decrease the LSB until $\widetilde{x}$ is at distance at most $\overline{\varepsilon}$ to x. This way we find a rounded value for x which fits the error bound.

Rounding and error modeling

Rounding can be done in various ways and *round-to-nearest* is not always the right choice as it involves an addition [BdDI+13]. *Faithful* rounding is usually preferred as it still enforces a tight bound on the error:

$$|\widetilde{y} - y| \leq 2^{l_{\widetilde{y}}}, \tag{1.12}$$

where $\widetilde{y}$ is the inexact quantity which approximates y and $l_{\widetilde{y}}$ corresponds to the LSBposition of $\widetilde{y}$ [Lop14]. This error is smaller than the LSB of the result, thus this rounding mode is also called *last-bit accuracy* [VIDDH19].

In this work, our goal will be to ensure faithful rounding of the operator output. This way, the precision specification becomes the accuracy constraint. Hence, we will need to consider rounding of basic operators and the propagation of errors we induced with intermediate rounding. We could perform a static error analysis of the circuit [GP11]. However, in our case, we want to define the circuit, taking errors into account *on-the-fly*. Hence, we need to model the errors *a priori*.

Table 1.1: Error bounds according to rounding modes where $\widetilde{y}$ is the approximation of y.

Rounding mode	Error bounds
Round-to-nearest	$-2^{l-1} \leq y - \widetilde{y} \leq 2^{l-1}$
Faithful	$-2^{l} \leq y - \widetilde{y} \leq 2^{l}$
Truncation	$0 \leq y - \widetilde{y} \leq 2^{l}$

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

$$\widetilde{y} = y + \Delta, \quad \text{where } |\Delta| \leq \overline{\varepsilon}, \tag{1.13}$$

where $\widetilde{y}$ is the actually computed quantity, y is the exact quantity, and $\overline{\varepsilon}$ is a bound on the computational error it bears.

In Table 1.1, we note that round-to-nearest and faithful rounding mode induce symmetric errors. However, truncations, which we will heavily use in Chapter 6, induce nonsymmetric ones. To obtain a model that closely links the computed quantity with the exact quantity, instead of the classical error model, we track errors using two positive bounds:

$$y - \varepsilon^{\text{inf}} \leq \widetilde{y} \leq y + \varepsilon^{\text{sup}}, \tag{1.14}$$

where $\varepsilon^{\text{inf}} \in \mathbb{N}$ and $\varepsilon^{\text{sup}} \in \mathbb{N}$ correspond, respectively, to the negative deviation and the positive deviation from the exact quantity.

Error propagation through operators

In the following, we will propagate errors through three unary operators: shift, which corresponds to multiplying a number by a power of two, negation and truncation. Bit-shifts can be right and left, however we will only consider left-shift as right-shift will only be used to shift zeros. Given our error model, if we apply a shift s to the inexact quantity $\widetilde{y}$, we obtain the following error bounds:

$$2^s y - 2^s \varepsilon^{\text{inf}} \leq \widetilde{2^s y} \leq 2^s y + 2^s \varepsilon^{\text{sup}}. \tag{1.15}$$

With the above equation, it is clear that the error of the shifted inexact quantity, $\widetilde{2^s y}$ is equal to the shifted error increasing the error by 2^s on both sides.

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:

$$-y - \varepsilon^{\text{sup}} \leq -\widetilde{y} \leq -y + \varepsilon^{\text{inf}}. \tag{1.16}$$

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 ε^{inf} but not the positive deviation ε^{sup}, thus the error bounds increase asymmetrically:

$$y - \varepsilon^{\text{inf}} - \varepsilon_t \leq \diamond_t(\widetilde{y}) \leq y + \varepsilon^{\text{sup}}, \tag{1.17}$$

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

$$\varepsilon_t \leq 2^t - 2^{\text{LSB}}. \tag{1.18}$$

The bound is reached when all truncated bits are 1's.

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

$$y_1 + y_2 - \varepsilon_{y_2}^{\text{inf}} - \varepsilon_{y_2}^{\text{inf}} \leq \widetilde{y_1} + \widetilde{y_2} \leq y_1 + y_2 + \varepsilon_{y_1}^{\text{sup}} + \varepsilon_{y_2}^{\text{sup}}. \tag{1.19}$$

These error propagation rules will be particularly useful for the evaluation of errors in adder graphs, topic that we will discuss in Section 1.3.1.

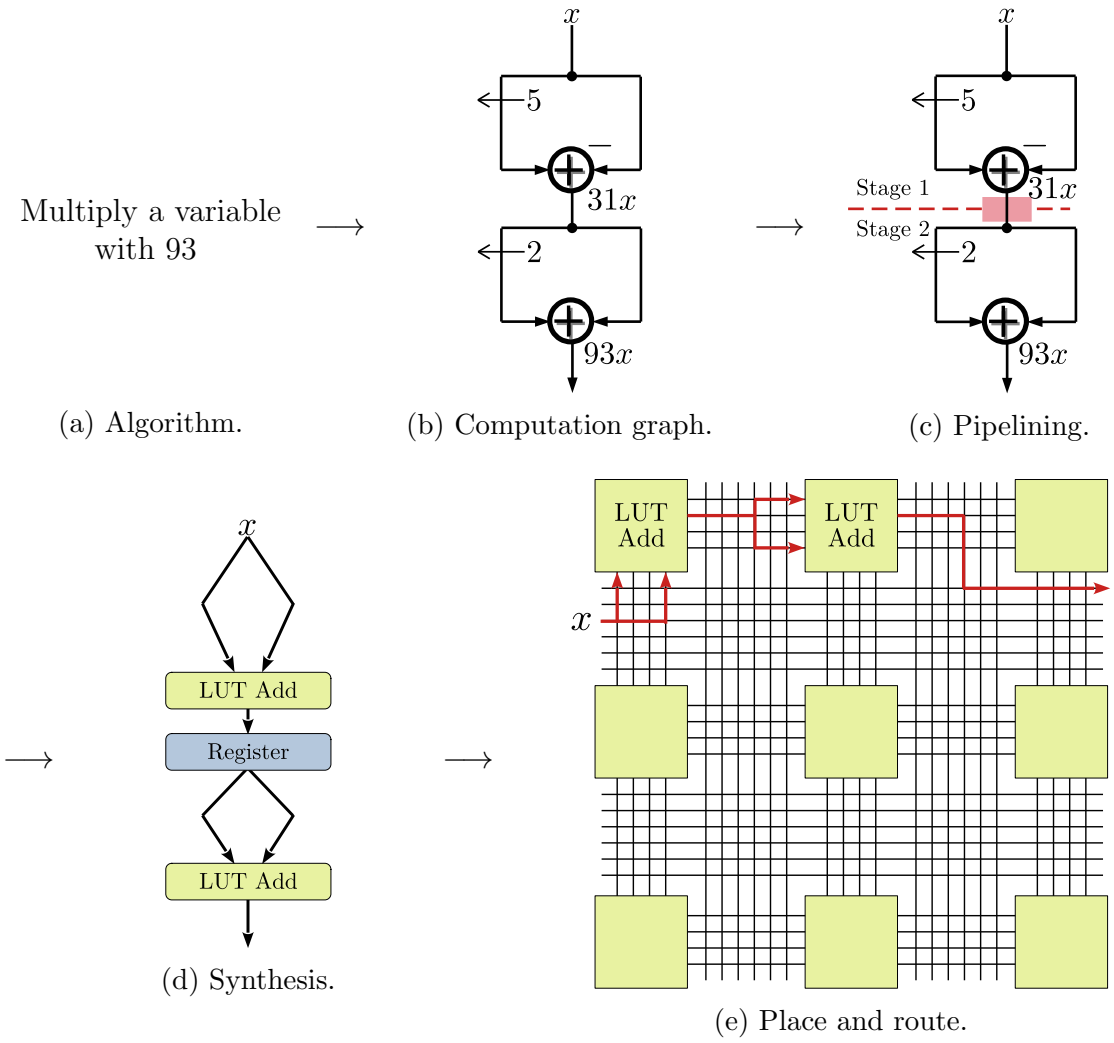
1.2 Field-programmable gate arrays

Usually, algorithms are not aware of the hardware they run on. In this case, generic processors are often the right choice as they are meant to work fast and efficiently independently of the algorithm. Dedicated circuit can be implemented as Application-Specific Integrated Circuits (ASICs), however, in many cases, applications are not meant to last for years nor be implemented on thousands of copies. Hence, reprogrammable integrated circuit, such as Field-Programmable Gate Arrays (FPGAs), are often the preferred choice. There are no formal models of FPGAs, thus the exact hardware cost will not be directly evaluated but proxy variables will be used instead. These proxy variables will permit to efficiently guide the design of hardware arithmetic operators and are chosen with hardware functioning in mind.

FPGAs are in-between generic processors and are fully customizable re-programmable boards. They come with an important flexibility, yet they still have a few constraints: instead of programming a circuit, we configure their basic logic elements (BLEs) and the routing between these elements. The FPGA configuration starts with the algorithm from which we produce a computation graph. Then, the computation graph can be split into pipeline stages, increasing the throughput. This leads to the register transfer level (RTL) description of the computation graph. The next step is the synthesis which converts the RTL description into FPGA primitives. Finally, the final stage is the place and route which determines (i) the physical location of each primitive on the FPGA and (ii) the routing between these locations. In Figure 1.3, we provide a simplified representation of the different compilation stages. The synthesis and the routing and placement on the chip are out of scope of our work and we let the synthesis tool[2] optimize it.

Overall, our goal is to minimize the cost of the circuit which we consider through the number of BLEs required to implement it. In hardware experiments, we will also discuss the power consumption and the delay of the circuit.

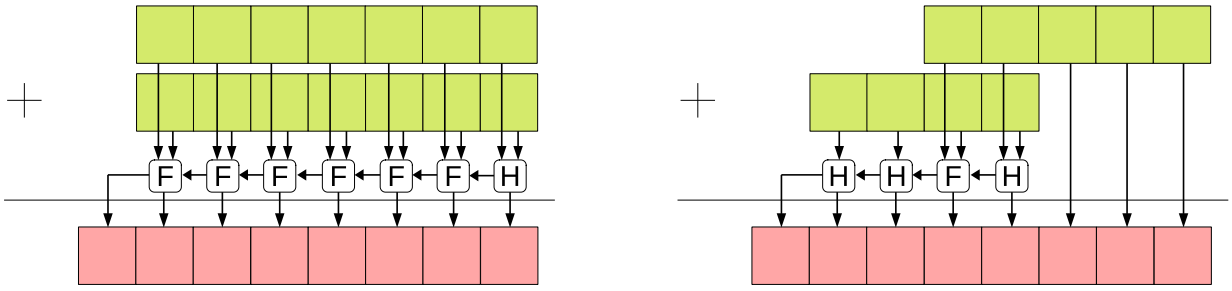
[2]In our case we will use Vivado: `https://www.xilinx.com/products/design-tools/vivado.html`



(a) Algorithm. (b) Computation graph. (c) Pipelining.

(d) Synthesis. (e) Place and route.

Figure 1.3: Compilation stages.



(a) Addition with same LSB and MSB.

(b) Addition with different LSB and MSB.

Figure 1.4: Additions with full adders.

Table 1.2: Full adder truth table where i_1 and i_2 correspond to the input bits, *sum* is the output. Columns c_{in} and c_{out} are the carry *in* and carry *out*, respectively.

i_1	i_2	c_{in}	*sum*	c_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

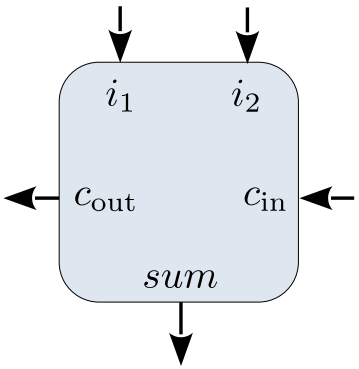


Figure 1.5: Simplified representation of a full adder. i_1 and i_2 correspond to the input bits, *sum* is the output. c_{in} and c_{out} are the carry *in* and carry *out*, respectively

Components of these BLEs are:

- *lookup tables* (LUTs) which are logic tables. They take a few inputs and generate an output. Their size varies from manufacturer to model. Often, a same LUT can be used to produce 1 output from 6 inputs or 2 outputs from 5 inputs.
- *flip-flops* (FFs) are data storage elements. At each clock cycle, each FF outputs the value which is stored and then stores its input.
- the *carry chain* is actually a local routing which goes through multiple BLEs.

Note that this is a rough approximation of the programmable content of FPGA. Among other things, we ignore here that BLEs are packed in slices, that wires are most of the FPGAs, and that, in addition to BLEs, other elements such as Digital Signal Processing (DSP) blocks are present on FPGAs. Roughly, we can consider that everything is connected to everything and in the end, implementation is not programming but configuring a board.

Tables can be used to perform arithmetic operations, in particular, full adders can be encoded with truth tables as illustrated in Table 1.2. Hence, by packing LUTs which

are linked together with the carry chain, it is possible to perform additions over FxP numbers [Ped19] as illustrated with full and half adders in Figure 1.4a. This way, it seems that the more additions we perform in a circuit, the higher will be its cost and this reasonable assumption motivates the content of the Chapter 4.

In Figure 1.4a, we performed an addition using 7 LUTs or FAs. Obviously, with larger input word lengths, we need more FAs to perform the addition. Hence, the cost of additions is directly impacted by the word lengths of its inputs. In Chapters 5 and 6, we will dissect additions to determine the precise cost of the additions. In particular, in these chapters, inputs with different LSB and MSB will be added together and some outputs bits will be obtained without using any FA, as illustrated in Figure 1.4b where left output bits are just passed through.

In some cases, high throughput is required and this can be obtained by pipelining the hardware circuit. This means that we slice the circuit in multiple steps, each step being performed in a clock cycle instead of having a unique clock cycle for the whole circuit. This way, the frequency can be increased, hence the throughput. However, this requires to store the output of each step in FFs partly increasing the number of BLEs required to implement the circuit. In Chapter 7, we will consider the minimization of the hardware cost while pipelining.

In FPGAs, each BLE embed at least one FF and one LUT [Ped19]. Hence, the extra cost in BLEs usage does not fully follow the number of FFs in the circuit. Finally, note that using FF, (i) decreases the power consumption by reducing glitches, but, (ii), increases the power consumption because it allows for higher frequencies [RDJ99].

In this thesis, we will study operators and applications with the goal of an implementation on FPGAs. Hence, the above knowledge on BLEs will guide our choices for defining good proxy variables.

1.3 Multiplication by constants

Multiplication algorithms, despite their simplicity, represent a real performance trade-off opportunity, in particular when one side of the multiplication is a constant. Numerical kernels such as matrix multiplications or scalar-vector multiplications involve *Multiple Constant Multiplications* (MCM). These kernels are essential building blocks of many algorithms such as the digital filter evaluation. Although the approaches used for multiple constants can also be used to solve the single-constant instances, there exist dedicated methods for this specific case [Ber86, Lef01, TN10, TN11, KGGZ16]. In any case, this problem is conjectured NP-hard for any number of constants.

Multiple Constant Multiplication problem

Input: A set of K target constants $C \in \mathbb{F}^K$ and a variable $x \in \mathbb{F}$, where $\mathbb{F}$ is the set of FxP numbers.

Problem: Find an implementation of $C \times x$ at the minimal cost.

In the following, without loss of generality, we will only consider $C \in \mathbb{N}^K$, as FxP number are integers scaled by an *implicit* scaling factor which sign can be adjusted the MCM part. The simplest way of performing several multiplications is to use a generic multiplier for each product. We could also use a single generic multiplier multiple times to reduce the cost in terms of resources, however, this will directly increase the delay of the operator. Furthermore, the gain in terms of resources is not guaranteed as this latter implementation would require many registers. This directly raises the question of the "minimal cost" which needs to be defined for each case.

A commonly used approach is to exploit the knowledge on binary representation of constants and rewrite the products by a sequence of additions and bit-shifts. In hardware, bit-shifts can be hardwired and are basically free. Additions, however, have a cost which we can provide an estimation for. On FPGAs, generic multiplications have a higher cost than additions as they usually require compressor trees. Thus, in the following, we will use the shift-and-add approach and we will focus on improving it.

1.3.1 The shift-and-add approach

Basic idea

The binary representation of a constant directly leads to a first possible way to replace a multiplication by shifts and additions.

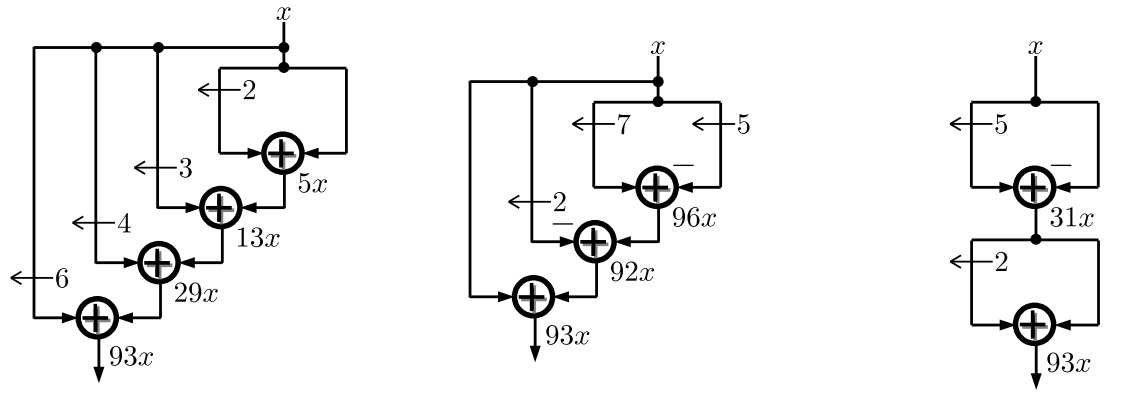
Example 3. The binary representation of $93 = 2^6 + 2^4 + 2^3 + 2^2 + 2^0$ is 1011101_2 and can be interpreted as the following shift-and-add chain:

$$\begin{aligned} 5x &= 2^2x + 2^0x, \\ 13x &= 5x + 2^3x, \\ 29x &= 13x + 2^4x, \\ 93x &= 29x + 2^6x. \end{aligned}$$

Here, only 4 additions and 4 bit-shifts are necessary to compute the multiplication by 93.

With this approach, the "length" of the shift-and-add chain is directly determined by the binary representation of the constant, *i. e.*, the cost of the multiplication in terms of the number of additions is equal to the number of ones in the binary representation minus 1. This corresponds to the Hamming weight [Ham50].

To reduce the Hamming weight of the representation, we can use signed digit (SD) representation using the digits $\left\{\overline{1}, 0, 1\right\}$, where $\overline{1} = -1$. In particular, we consider the Canonical Signed Digit (CSD) representation in which nonzero bits are not adjacent [Boo51, Ber86]. Here, we have subtractions, but their cost is not exceeding that of additions. From here on, we will only talk about adders, but allow for addition of opposite-sign values. Then, rewrit-



(a) Obtained with exponentiation-like adder chain.

(b) Obtained with CSD.

(c) Optimal adder graph.

Figure 1.6: Adder graphs computing $93x$.

ing in CSD, we can reduce the cost of the shift-and-add implementation. A polynomial-time algorithm permits to rewrite a binary number into its CSD representation [Boo51].

Example 4. The CSD representation of 93 is $10\bar{1}0\,0\bar{1}01_{\text{CSD}}$ and leads to a shift-and-add solution with 3 adders:

$$\begin{aligned} 96x &= 2^7x - 2^5x, \\ 92x &= 96x - 2^2x, \\ 93x &= 92x + x. \end{aligned}$$

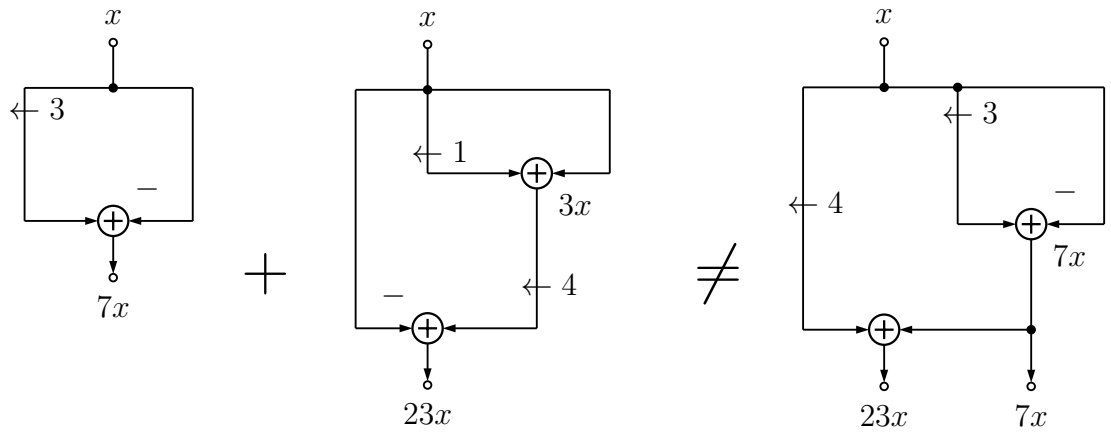
Here, CSD reduced the constant multiplication cost to 3 adders compared to Example 3. The CSD approach gives interesting results in polynomial time, however the CSD representation does not lead to the minimal number of adders required to perform the multiplication. For instance, an optimal solution for $93x$ would be 2 adders:

$$31x = 2^5x - 1, \tag{1.20}$$
$$93x = 2^1 \times 31x + 31x. \tag{1.21}$$

Finding such optimal shift-and-add solution with the minimal number of adders is conjectured to be an NP-hard problem [TN11].

Adder graph representation

Every shift-and-add solution can be conveniently represented with a shift-and-add tree, or as we will call it adder graph. In Figure 1.6, we provide the adder graphs obtained from the binary representation, the CSD and the solution with the minimal number of adders for the target constant 93. The left arrows correspond to the bit-shifts and adders have an



(a) Optimal for target constant 7.

(b) Optimal for target constant 23.

(c) Optimal for target constants 7 and 23.

Figure 1.7: Combining adder graphs is better than one adder graph per output.

optional minus sign to indicate subtractions. The value below each adder corresponds to the intermediate result at the current step and is called a fundamental [DM94]. We order adders into stages and, for simplicity of notation, we consider that the input is the adder 0 and is at the stage 0. For example, in Figure 1.6b, the first adder's fundamental is 96, it is obtained by subtracting the right input, shifted by 5 bits, from the left input, which has been shifted by 7 bits. This adder, which is computed at the first stage, has both its input coming from the "adder 0".

Definition 1 (Adder depth)**.** Let a, b and c be three adders such that a and b are the inputs of c. We denote ad_a and ad_b the adder depths, or stage, of adders a and b, respectively. Then, the adder depth of c is defined as

$$ad_c = \max(ad_a, ad_b) + 1.$$

The adder depth of an adder is the stage at which its value is computed. By definition, the adder depth of the input of the adder graph is equal to 0 and the adder depth of the adder graph is the maximum of the adder depth of its adders. For example, in Figure 1.6a, c could correspond to the adder computing $13x$, its inputs a and b are the adder graph input and the adder $5x$, respectively. Then, we have $ad_a = 0$, $ad_b = 1$ which leads to $ad_c = 2$.

Shift-and-add for MCM

It is not enough to decompose the set of target constants into multiple single constant problems solved separately and to combine all solutions, we can do better. In Figure 1.7, we see that we can share intermediate constants between multiple outputs. It permits to

reduce the overall cost when we tackle multiple constants at once.

The CSD algorithm is not meant for multiple constants and a common way to improve this approach is to search for common patterns in the CSD representation, so a same pattern can be computed once then reused for multiple target constants. This approach is called Common Subexpression Elimination (CSE) [Har96, PSC96] and is the idea behind multiple heuristics [DM04, ACFM12].

The approaches presented above rely on number representation. However, it is possible to directly build the adder graph, fundamental by fundamental. Starting from the input, adder 0, the first question to answer is: "which adder/fundamental can be computed from the input?" Then, once one adder has been added to the adder graph, and at every next step, the question becomes "which fundamental can we add to the adder graph by adding an adder to it?" This permits to build an adder graph, step by step, choosing in priority fundamentals which are either target constants or have the best chance to lead to target constants at the next steps.

Based on this idea, it is possible to perform exhaustive searches [Gus08, Kum16] where, given a fixed number of adders, all the possible adder graph topologies and fundamentals are considered [DM94, VP07]. However, the size of the search space quickly becomes far too large even for small adder graphs and it needs to be reduced it by considering a few theoretical results.

For instance, we can remove from the search space even-valued fundamentals [DM94, Theorem 2] and construct adder graphs which only involve odd fundamentals, the so-called odd adder graphs. Thus, in the following we will consider only odd adder graphs and that all our target constants are positive and odd, if necessary the sign is adjusted later and a shift can be applied to retrieve the original even-valued constant. To build a new odd target constant set C_{odd} from the original one C, we first take the absolute value of each target constant and second use the odd function which evaluates to the odd part [SP95, Sequence M2222] of its input:

$$\text{odd}(n) = \frac{n}{\gcd\left(2^n, n\right)}. \tag{1.22}$$

Then, we define C_{odd} as:

$$C_{\text{odd}} = \left\{\text{odd}(|C_j|) \mid C_j \in C\right\}. \tag{1.23}$$

For example, for the target constant 186 we simply search for the best adder graph for the target constant $\text{odd}\left(186\right) = 93$, as in Figure 1.6c, and shift the output by one bit to retrieve 186.

Dempster and Macleod also demonstrate that it is not necessary to consider very large shifts [DM94, Theorem 4] and we can consider that the left and the right input cannot be left-shifted simultaneously [DM94, Theorem 3]. This drastically reduce the search space but the number of possible adder graphs still grows exponentially and a complete enumeration is not realistically feasible in most cases. To overcome this issue, a possible approach consists in adding more bounds on the problem to reduce the search space to an acceptable size [Kum16]. Another common approach is to explore the search space

progressively, instead of generating it upfront, either heuristically, *i. e.*, only enumerate fundamentals that seem promising [DM94, VP07, KZFC12], or with a dedicated branch and bound [AGF10]. Although the branch and bound approach should provide optimal results and proof of optimality, it is tedious to implement and to maintain and extend.

Remark. There exist a few constructive heuristics such as Hcub [VP07] or RPAG [KZFC12] to solve the MCM problem. However, to our knowledge, no local search algorithms were proposed to improve heuristic solutions. Local search are heuristics that aim at improving a solution by making small modifications, called moves, to it. Proposing a dedicated local search could be interesting to obtain better solutions in limited time.

The last approach we think of consists in solving the describing the MCM problem using mathematical modeling and relying on solvers to explore the search space as they build it. In 2018, Kumm [Kum18] provided an MILP-based model to find an adder graph which minimizes the number of adders. With his model, Kumm found and proved the optimal solution for 10 instances out of the 11 instances from image processing he used for comparison [Kum18]. In Chapter 2, we will present this model in details and in Chapter 4 we will identify a flaw which we will correct in this thesis. Based on the same idea of using mathematical modeling, Lagoon and Metodi proposed an SMT-based approach [LM20] and demonstrate for the first time an important result for single constant multiplications: the smallest constant that cannot be computed with strictly less than 6 adders is 171 398 453. These approaches, which use mathematical modeling, have proven their efficiency and we will base our work on the same principle.

1.4 Linear digital filters as arithmetic operators

An example of application benefiting from MCM are linear time-invariant (LTI) digital filters. These basic building blocks of DSP algorithms can be seen as arithmetic operators, optimized and then seen as black boxes. The theory of LTI and filter design is a rich topic but out of scope of this thesis. Our interest is in the improvement that optimized MCM can bring to design of hardware filters, and how they can be co-designed. Hence, we keep the definitions to minimum and refer the interested reader to [PV08, Ant18] and to [MB14] for a textbook which specifically presents DSP for FPGA implementation.

A filter is a transformation of the input signal to the output which reduces or enhances certain aspects of the original signal. For example, low pass filters can be used to reduce high-frequencies in order to remove noise. Finite and infinite impulse response (FIR and IIR) digital filters are fully defined by one or two vectors of coefficients, respectively. These vectors of coefficients, $h_i \in \mathbb{R}$ for FIR filters, and $a_i, b_i \in \mathbb{R}$ for IIR filters, intervene in their transfer functions:

$$\text{FIR:} \quad H(z) = \sum_{i=0}^{N} h_i z^{-i}, \qquad (1.24) \qquad\qquad \text{IIR:} \quad H(z) = \frac{\sum_{i=0}^{N_b} b_i z^{-i}}{1 + \sum_{i=1}^{N_a} a_i z^{-i}}, \qquad (1.25)$$

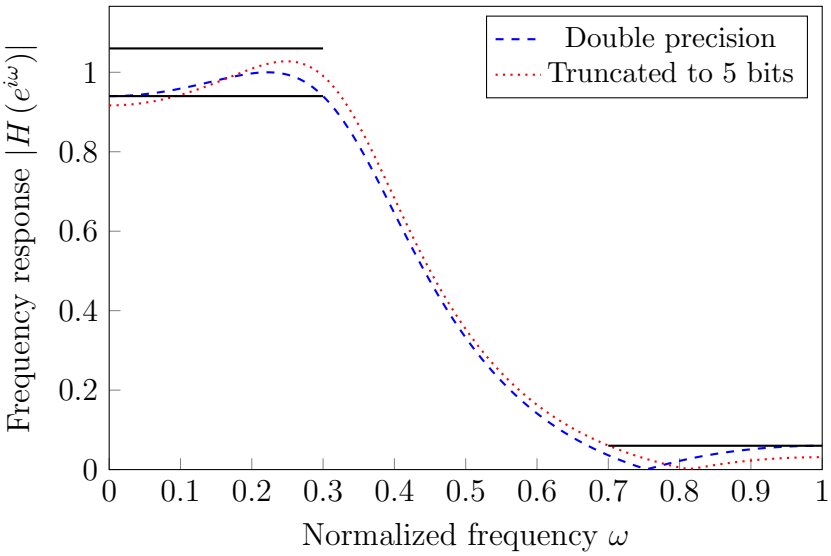


Figure 1.8: Example of specifications.

where $z \in \mathbb{C}$. Note that, in the following, we will limit the discussion to second-order IIR filters, $N_b = N_a = 2$, as it is common to decompose higher-order filters into a cascade of second-order sections [JN86, OS89, DM95, Lei97, WLL10].

Obviously, the values of the coefficients of the filter define the effect of the filter. To obtain a filter which fits a wanted set of characteristics, we usually constrain the filter's frequency response,

$$\underline{\beta}(\omega) \leq \left|H\left(e^{j\omega}\right)\right| \leq \overline{\beta}(\omega)\,, \qquad \forall \omega \in [0;\pi]\,, \tag{1.26}$$

with specifications $\underline{\beta}$ and $\overline{\beta}$ which are the lower and upper bounds. They typically encode constant bounds valid inside frequency intervals, but our definition (1.26) allows to model them as functions of ω. For example, in Figure 1.8, we see black lines which correspond to the specifications. In the top left corner is the passband and in the bottom right corner the stopband, these bands correspond to the frequencies which should be left unmodified or attenuated, respectively.

In our case, in addition to specifications, we want stable IIR filters: necessary and sufficient stability conditions for second-order filters are well-known [Ant18, Section 16.8] to be

$$-2 < a_1 < 2, \tag{1.27}$$

$$|a_1| - 1 < a_2 < 1. \tag{1.28}$$

Once the specifications are fixed, various methods [OS89, Ant18] permit to find a set of coefficients which fits. This is a first step is called *filter design* (FD). Using the filter design and analysis tool (FDATool) in Matlab, we generate double-precision coefficients for a second-order IIR filter fitting the specifications in Figure 1.8. The second step comes

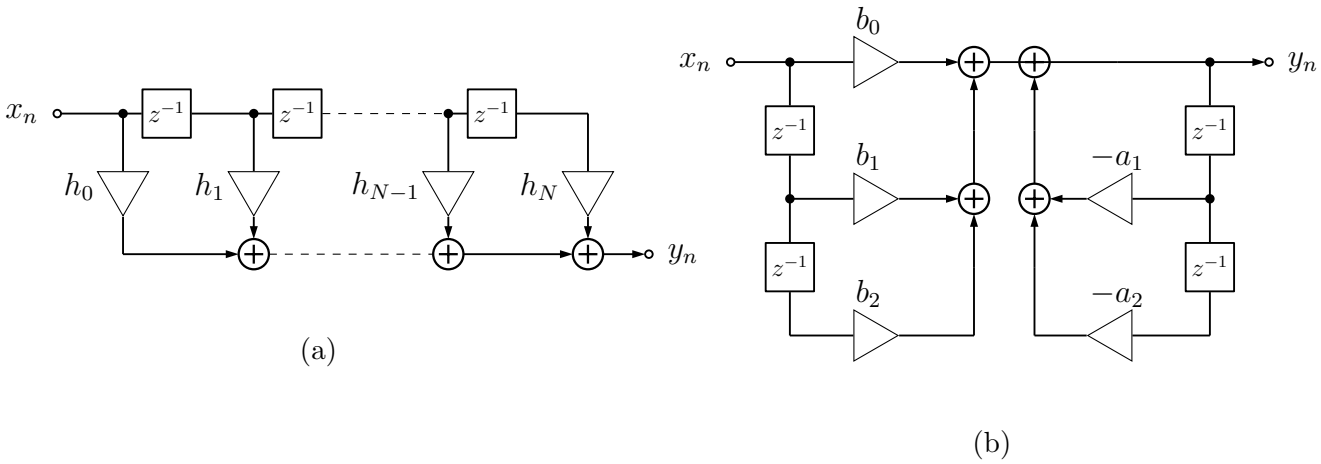


Figure 1.9: Hardware implementation of FIR (left) and IIR (right) filters, direct form.

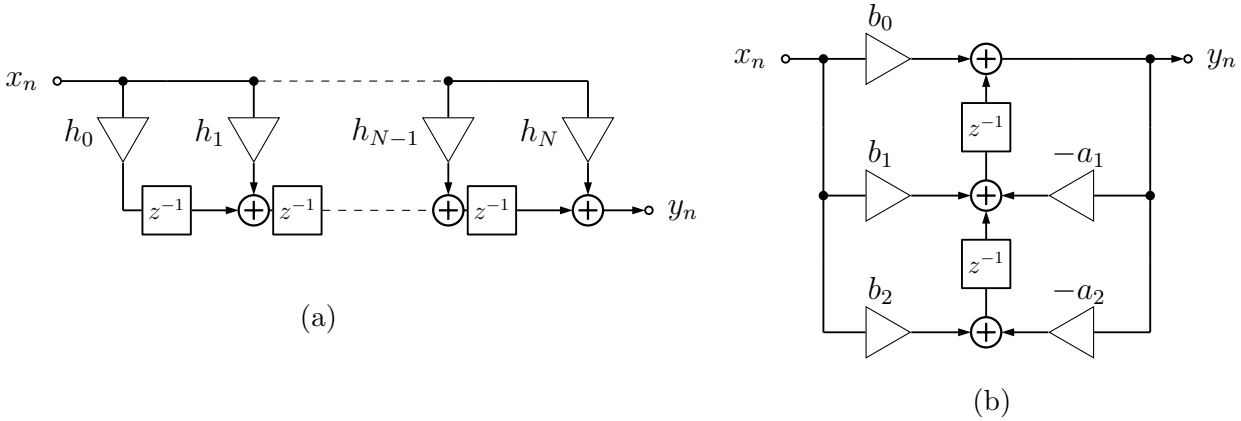


Figure 1.10: Hardware implementation of FIR (left) and IIR (right) filters, transposed form.

precisely from a hardware constraint. We target hardware implementation which uses FxP arithmetic. However, the coefficients we obtained, with the FD, are usually represented with double precision, thus, the coefficients need to be *quantized* (Q) and still fit the specifications and stability constraints, which is not always the case. For instance, as illustrated in Figure 1.8, the coefficients obtained with Matlab and quantized to 5 bits, lead to a filter which does not fit the specifications anymore, see the red dotted line.

Finally, we want to *implement* (I) filters and, to do so, we need to evaluate their output, $y_m \in \mathbb{R}$, given the input signal $x_m \in \mathbb{R}, \forall m \in \mathbb{Z}$ in the time-domain:

$$\text{FIR:} \quad y_m = \sum_{i=0}^{N} h_i x_{m-i}, \qquad (1.29) \qquad \text{IIR:} \quad y_m = \sum_{i=0}^{N_b} b_i x_{m-i} - \sum_{i=1}^{N_a} a_i y_{m-i}. \quad (1.30)$$

Basically, samples are passed through the filters, partly stored in an internal memory, and then used to compute the outputs. In the case of IIR filters, we identify a recursive part in which outputs are multiplied with coefficients a_i to impact the next outputs.

The implementation can precisely follow the difference equations (1.29) and (1.30) as illustrated in Figure 1.9. In Figure 1.9a, registers z^{-1} delay the signal, thus we indeed have h_N which is multiplied with the input signal delayed N times, x_{n-N}, while x_n is directly multiplied with h_0. This implementation is called the Direct Form and, at each step, we see that $N+1$ different inputs are multiplied with $N+1$ different constants. Instead, we can implement the Transposed Direct Form, which is represented in Figure 1.10, and, in this case, a same variable is multiplied with multiple constants.

Due to the heterogeneous nature of the different steps, dedicated algorithms that tackle the global problem, FD+Q+I, have specificities which are hard to merge together into a single global algorithm. There exists many approaches [Ant18] to find coefficients which fits given specifications, such as Butterworth, Chebyshev, etc. These algorithms are specific and although the combination of FD and Q steps has been studied extensively [SV99, VBVBT02, WLL10], it is not easy to adapt algorithms dedicated to the FD step to include the Q step as well.

A large body of work exists for the I step [BT05, JGW07, GWD09], in particular for FIR filters. Hardware filters involve multiplications with constants and could be done with MCM based on shift-and-add Another constant multiplication method is based on pre-computed tables and is called Ken Chapman multiplier (KCM), after its inventor [Cha94]. It has also been successfully applied to digital filtering [KFM$^+$13, dDFKF19] and using this method, an approach for optimization of combined Q & I steps has been proposed [VIDDH19].

Despite efficient methods for the I step and Q+I combined steps, the resource cost still strongly depends on the coefficient values. However, the coefficients are typically fixed in the first FD step with very little knowledge over their implementation cost. There are exceptions and the filter design step, aware of the implementation cost, have been proposed [LP83, ZT88]. But in this case the search space for the coefficient values is way more constrained than strictly necessary as they usually limit the coefficients to power-of-two's only. In any case, the obtained implementations are optimized only for one filter instance, or a small sub-set of the overall design space, not permitting overall optimal solution.

Merging all the steps has been done successfully for the design of FIR filters [KVF23] using an MILP-based model. Our ambition is to solve the combined FD+Q+I steps for the second-order IIR filters within one global optimization, and, in Chapter 8, we will see that hardware-aware IIR filter design is a difficult nonlinear combinatorial problem.

CHAPTER 2

Mathematical Modeling

"Real stupidity beats artificial intelligence every time."

Terry Pratchett (1948-2015),
Hogfather (1996)

"We fail more often because we solve the wrong problem than because we get the wrong solution to the right problem."

Russell L. Ackoff (1919-2009)

Russell L. Ackoff said that "we fail more often because we solve the wrong problem than because we get the wrong solution to the right problem." Mathematical modeling permits to solve the *right* problem by focusing on its description and not on the algorithm to solve it. However, we are not yet close to be able to model exactly most real-life problems and we usually need to find an abstraction which we consider good enough. In this thesis, we will specifically use Mixed-Integer Linear Programming (MILP) which fits well for hardware operator design. Choosing building blocks and basic operators is a combinatorial problem and MILP solvers are perfectly suited for these kind of problems.

In this chapter we recall the concept of mathematical modeling, and MILP in particular, and its applications to hardware operator design. The MILP approach benefits from the advances in academic and commercial solvers year after year. Hence, even instances which are currently out of reach with this approach might soon be solved without any effort, simply with progress in generic solving [KBPV22].

Despite the importance and interest of the algorithms used to solve MILP models, we will present the modeling aspects and not the algorithms themselves in the following. For more details on the algorithms, we refer the reader to textbooks on the topic [BT97, Wol20].

In this chapter however, we expose important modeling methods and tools, such as the linearization of nonlinear expressions or the cutting planes approach, which permit to write and improve models. Finally, we present the *performance variability* problem which is an additional challenge when a choice between two different models rises.

2.1 Writing an MILP model

When we approach a problem with MILP, the goal is to express the problem to solve using equations. It should be noted that the more constraints we put on our declarative programming language, the more specialized, and hopefully efficient, algorithms can be used. Hence, in this paradigm we only allow ourselves to use linear equations.

Definition 2. Linear programming problems can be formalized as follows:

$$\begin{aligned} &\min && c^T x \\ &\text{subject to} && Ax \leq b, \end{aligned}$$

where $x \in \mathbb{R}^n$ are continuous variables, and $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{n \times m}$ and b^m the parameters of the model.

In Definition 2, linear programming (LP) problem is defined as the minimization of the objective function $c^T x$, while satisfying a set of constraints. One of the most widely used algorithms to solve LP models is the simplex method. Although the worst-case complexity of simplex method is exponential time, in practice this method is very efficient and is often preferred over interior point methods which have a polynomial-time complexity [NN94].

It is important to note that, in most solvers, computations are performed in double-precision floating-point format. This sometime leads to numerical instabilities, despite dedicated methods in the simplex algorithm to mitigate this issue. There also exist rational and arbitrary-precision simplex implementations such as QSopt or SoPlex. As we will see, solving linear programs is often a subroutine which needs to be fast and the overhead of computing exact solutions is usually not acceptable and we use FP ones. To prevent numerical instabilities we avoid parameters which differ by too many orders of magnitude: we should be fine if we model a needle in a haystack but modeling a drop in the ocean is not an option. For the MCM problem, the word length of the target constants will have a direct impact on the size of the parameters.

In MILP, the difference with LP is that we can impose integrality constraints on the variables. This way, a subset of the variables x are integers. In particular, integer variables inside $[0, 1]$, *i. e.* $x_i \in \{0, 1\}$, will be denoted binary variables. If all the variables x are integers or binary, dedicated algorithms or subroutines can be activated during the solving process.

Note that, in the general case, these integrality constraints cannot just be removed to solve an LP model before rounding the solution, hoping that it stays optimal. However,

removing integrality constraints, *i. e.* generating a *linear relaxation* of the problem is still useful in the intermediate steps. By solving the relaxed problem, we obtain a lower bound of the objective function as adding constraints can only worsen the solution. This is particularly important to remember since good lower bounds will guide the algorithms used to solve MILP problems. As stated above, we will not get into many details on the solving methods but we will give a simple idea of how solvers work in Section 2.2.3.

Remark. The objective function of a linear programming problem can be rewritten as a "max" problem just by taking the opposite of c and, similarly, constraints can also be expressed as $\geq$ or $=$ equations.

For our experiments throughout this thesis, we will use two main commercial MILP solvers which are Gurobi [Gur20] and CPLEX [CPL20]. Note that there also are well-known open-source MILP solvers such as GLPK and SCIP [BBC+21]. Models are usually stored in files in the `MPS` or `LP` formats, for example, and passed to a solver, which eventually produces a solution file in the `SOL` format. Yet, writing a model into a file and reading a solution file is tedious. Hence, modeling languages which then call the solver APIs are usually preferred. In C++, we can use ScaLP [SSKZ18] and Pyomo is the standard choice[BHH+21] in Python. In our work, we have used JuMP [DHL17] which is a domain-specific modeling language for mathematical optimization in Julia. This greatly facilitates modeling and any common solver can be used as a back-end. Using such a modeling language also permits to easily switch between paradigms, as JuMP also allows for nonlinear models or constraint programming. Finally, multiple solver parameters, such as the time limit or the verbosity level, are unified and do not necessitate to specifically know the parameter name of each solver.

Remark. JuMP [DHL17] is an *open-source* modeling language in which it is easy to dive and contribute.

2.1.1 Big-M and indicator constraints

It is common to need implications or optional constraints in mathematical modeling and indicator constraints are a direct transcription of this which is usually well-supported by MILP solvers. The general case of indicator constraints have the form

$$z = y \quad \text{if } x = 1, \tag{2.1}$$

where x is a binary variable and y and z two variables, continuous or not. These constraints have several drawbacks, which will be discussed, and can be advantageously substituted by the so-called big-M constraints.

Indeed, Equation (2.1) can be replaced with the following two constraints:

$$z \leq y + (1 - x)\, M_1, \tag{2.2}$$
$$z \geq y - (1 - x)\, M_2, \tag{2.3}$$

for sufficiently large values of constant M. If $x = 1$, then we have $z \leq y$ and $z \geq y$, hence $z = y$. If $x = 0$, then we have $z \leq y + M_1$ and $z \geq y - M_2$, which, in practice, should be equivalent the absence of constraints on z. This equivalence between Equation (2.1) and Equations (2.2) and (2.3) can only be true if y and z are bounded. Otherwise, M's cannot be large enough to ensure the absence of constraint on y and z if $x = 0$.

Assuming that $y \in \left[\underline{y}, \overline{y}\right]$ and $z \in [\underline{z}, \overline{z}]$, we can deduce appropriate lower bounds for M_1 and M_2:

$$M_1 \geq \overline{z} - \underline{y}, \tag{2.4}$$

$$M_2 \geq \overline{y} - \underline{z}. \tag{2.5}$$

Actually, M_1 and M_2 values should be exactly these bounds, since higher values would only deteriorate the linear relaxation or even lead to numerical instabilities [KN13, Esp18]. When indicator constraints are used, the risk of numerical instability is completely avoided. However, these constraints are usually dropped in the linear relaxation, or replaced by unnecessarily large big-M constraints, leading to models really hard to solve [KN13, BBF+16]. Hence, indicator constraints are usually slower than fine-tuned big-M constraints.

Overall, when possible, we will prefer to use big-M. Yet, it requires to find the smallest value that fits and to verify the robustness of the model with respect to numerical instabilities. In particular, a common issue is that in solvers integrality constraints on variables are not strict: a variable is considered to be an integer if it is close enough to an integer, 10^{-5} is the default tolerance for CPLEX and Gurobi[1]. For example, if x is a binary variable, $x = 10^{-6}$ would be within the tolerance to assume that $x = 0$. But the following constraint

$$z \leq 1 + 10^6 x, \tag{2.6}$$

would actually permit $z = 2$.

Remark. As a rule of thumb, big-M values should not exceed 10^4 and should not be used at all when working with continuous variables. In any case, every big-M constraint should be scrutinized to ensure it constrains as expected.

2.1.2 Linearization

In MILP formulations, we can only use linear equations as constraints. However, it is common to naturally describe a problem with nonlinear constraints. When this is the case, different approaches are possible:

1. Use another mathematical modeling approach, instead of MILP;
2. Find a different, but equivalent, formulation of the problem which is *linear*;
3. Rewrite nonlinear constraints as linear, usually adding new variables.

[1] this can be changed with the parameter is `CPXPARAM_MIP_Tolerances_Integrality` or `IntFeasTol` with CPLEX or Gurobi, respectively.

As our goal is to benefit from MILP solvers, we need to rely on the second and third solutions, and linearization is often the only solution.

Let us illustrate the process of linearization with an example. Suppose we need to constrain an integer variable $z \in \mathbb{N}$ to be equal to the product of two other integer variables $x \in [\![0; \overline{x}]\!]$ and $y \in [\![0; \overline{y}]\!]$:

$$z = x \times y. \tag{2.7}$$

To be able to incorporate this constraint into the model, we need a linear reformulation.

Remark. Not every constraint can be linearized, in particular the linearization of the product of *continuous* variables is currently out of reach.

The following linearization was proposed in 2008 [BEL08] and basically consists in rewriting one of the positive integers into its binary representation:

$$x = \sum_{i=0}^{\lfloor \log_2 \overline{x} \rfloor} 2^i \, t_{x,i}, \tag{2.8}$$

where $t_{x,i}$ are $\lfloor \log_2 \overline{x} \rfloor + 1$ binary auxiliary variables. This constraint ensures that the bits $t_{x,i}$ encode the value of x. This simplifies the product $z = xy$ to a sum of products between the binary variables $t_{x,i}$ and the positive integer y:

$$z = \sum_{i=0}^{\lfloor \log_2 \overline{x} \rfloor} 2^i z_i, \quad \text{where } z_i = t_{x,i} \times y. \tag{2.9}$$

The linearization of a binary-by-integer product is common and involves indicator or big-M constraints [Glo75, OK92]. Basically, $z_i = t_{x,i} \times y$, where $t_{x,i} \in \{0, 1\}$, is equivalent to

$$z_i = 0 \quad \text{if } t_{x,i} = 0, \tag{2.10}$$

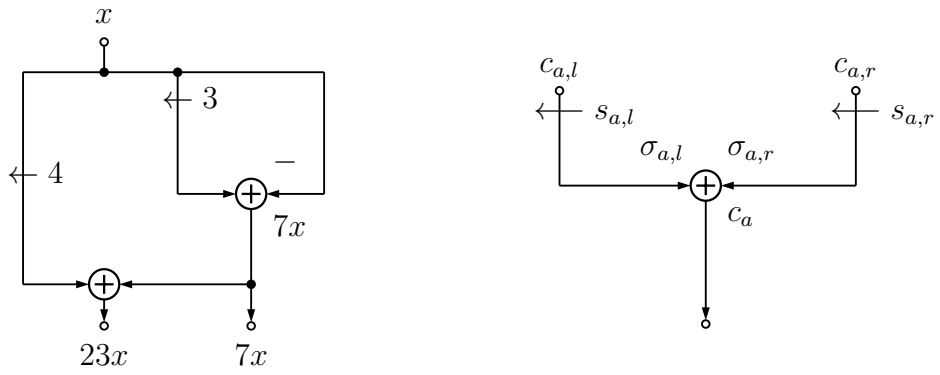
$$z_i = y \quad \text{if } t_{x,i} = 1. \tag{2.11}$$

This adds again $\lfloor \log_2 \overline{x} \rfloor + 1$ variables to the model, along with twice as many constraints.

> *Example* 5. The linearization of $z = x \times y$ where $x, y \in [\![0; 255]\!]$ are two 8-bit positive integers requires to add to the model 16 variables, 17 constraints and as many places to make a mistake.

Although the numbers in Example 5 are quite reasonable, one must acknowledge that we do not usually simply have $x \times y$ but we would rather deal with $x_i \times y_i$ for i in a large set. Thus, the number of additional variables and constraints is actually a multiplier factor.

Remark. We provided a way to linearize the product between two positive integer variables assuming x and y are bounded by $\overline{x}$ and $\overline{y}$, respectively. The bound on x is necessary to rewrite x as its binary representation. Although we did not explicitly use the bound on y, it is required to replace Equations (2.10) and (2.11) with big-M constraints.



(a) Example of an adder graph.

(b) Adder in adder graph.

Figure 2.1: Adder graph and a focus on one adder.

2.1.3 Writing an MILP model: step-by-step example with MCM

In the following, we will demonstrate how MILP can be used for arithmetic operator design and unfold the model for MCM proposed by Kumm in 2018 [Kum18]. The idea behind this model is, given the target constants, to build an adder graph, as represented in Figure 2.1a, adder per adder. The number of adders is fixed *a priori* to a value N and the objective of the model is to answer the question: "could we find an adder graph which outputs the target constants with N adders?". In this case we can already determine that there is no obvious objective function to optimize, we solve a satisfiability problem instead.

The first step to write an MILP model requires to define the variables of the problem we solve. To do so, we formalize what a solutions looks like and build upon it. For the MCM problem, a solution is characterized by its fundamentals, the shifts and signs of the inputs, as well as the link between adders. Thus, the variables of the model could be:

- $c_a \in \mathbb{N}$ for the fundamentals for each adder $a \in [\![0; N]\!]$ where the input, $c_0 = 1$, is treated as an adder for simplicity;
- $s_{a,i} \in \mathbb{Z}$ for the shifts of both inputs, $i \in \{l, r\}$;
- $\sigma_{a,i} \in \{0, 1\}$ for the sign of each input;
- $c_{a,i} \in \mathbb{Z}$ for the input of the adder.

In Figure 2.1b, we have represented these variables on an adder to have a better idea of their role. Then, we need to precise the exact domains of each variable. For instance, the fundamentals c_a range from 1 to the power of two directly after the largest target constant, $\max(\log_2(C_i))$ where C is the set of target constants.

Finally, we add constraints to link the variables together. For the MCM problem such constraints could be

$$c_a = (-1)^{\sigma_{a,l}} 2^{s_{a,l}} \times c_{a,l} + (-1)^{\sigma_{a,r}} 2^{s_{a,r}} \times c_{a,r}, \tag{2.12}$$

which means that the fundamental c_a is equal to the sum of the shifted and signed inputs. We note that $c_{a,l}$ and $c_{a,r}$ could take any random value as there are currently no links between adders. Ultimately, the process of writing a model consists in toing and froing between defining variables and adding constraints, and in that respect, we need to add binary variables, $c_{a,i,k} \in \{0,1\}$, to link the input of adder a with previous adders k. For example, to link c_a, $c_{a,i}$ and $c_{a,i,k}$ together, we need the following set of constraints:

$$c_{a,i} = c_k \quad \text{if } c_{a,i,k} = 1, \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, k \in [\![0; a-1]\!], \tag{2.13}$$

which ensures that $c_{a,i}$ takes the value of the adder a if the binary variable $c_{a,i,k}$ is equal to one. To ensure that at least one $c_{a,i,k}$ is equal to one, we add this last set of constraints:

$$\sum_{k=0}^{a-1} c_{a,i,k} = 1 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}. \tag{2.14}$$

Eventually, new variables could be necessary to rewrite constraints which might initially be nonlinear, like (2.12). In particular, in [Kum18], this led to additional variables,

$$c_a = \underbrace{(-1)^{\sigma_{a,l}} \underbrace{2^{s_{a,l}} c_{a,l}}_{c^{\text{sh}}_{a,l}}}_{c^{\text{sh,sg}}_{a,l}} + \underbrace{(-1)^{\sigma_{a,r}} \underbrace{2^{s_{a,r}} c_{a,r}}_{c^{\text{sh}}_{a,r}}}_{c^{\text{sh,sg}}_{a,r}}, \tag{2.15}$$

which permit to store intermediate results:

$$c^{\text{sh}}_{a,l} = 2^{s_{a,l}} c_{a,l}, \tag{2.16}$$
$$c^{\text{sh,sg}}_{a,l} = (-1)^{\sigma_{a,l}} c^{\text{sh}}_{a,l}, \tag{2.17}$$
$$c^{\text{sh}}_{a,r} = 2^{s_{a,r}} c_{a,r}, \tag{2.18}$$
$$c^{\text{sh,sg}}_{a,r} = (-1)^{\sigma_{a,r}} c^{\text{sh}}_{a,r}. \tag{2.19}$$

Then, it is possible to linearize each constraint. For instance, knowing that the shifts $s_{a,l}$ live in $[\![S_{\text{min}}; S_{\text{max}}]\!]$ [DM94], we can enumerate all the possible values they can take and add binary variable, $\Phi_{a,i,s} \in \{0,1\}$, which is equal to 1 if the shift is equal to s. This way, the set of constraints

$$c^{\text{sh}}_{a,i} = 2^s c_{a,i} \quad \text{if } \Phi_{a,i,s} = 1 \qquad \forall a \in [\![1; N_A]\!], i \in \{l, r\}, s \in [\![S_{\text{min}}; S_{\text{max}}]\!], \tag{2.20}$$

is a linearization of (2.16). Kumm [Kum18] applied this process until (2.12) has been fully linearized, obtaining a first model for the MCM problem. Note that, for the linearization of (2.16) only, we need to introduce $N \times 2 \times (S_{\text{max}} - S_{\text{min}})$ variables and constraints.

2.2 Improving an MILP model

2.2.1 Warm start

An initial solution is a key ingredient to solvers, as it usually permits to speed up the solving process. Most solvers, if not all, rely on primal heuristics to find initial solutions.

It is also possible to provide a first solution, called a warm start, to the solver. This is *harder* than solely relying on the solver and, at first glance, this seems flawless. Yet, if internal heuristics of the solver are able to obtain a *better* initial solution, and *faster* than the time needed to obtain the warm start, then the solver is *stronger* than using a warm start.

In the case of MCM, the internal primal heuristics work poorly and providing a warm start obtained using a greedy algorithm or a dedicated heuristic is interesting. However, it requires to translate a solution into variables values, including into all the intermediate variables needed for linearization, while we usually do the other way around, reading the variable values after the solving process to build a solution. It is also possible to only provide a warm start for a subset of the variables and let the solver complete the solution. However, it might give up and start from scratch if building the rest of the solution consumes too much time. Note that, even in the MCM case where internal primal heuristics are not efficient, providing a warm start changes the behavior of the solver which can result in unexpectedly worse performance in the long run. This cannot be anticipated and we expect warm starts to help, on average.

2.2.2 Valid inequalities and symmetry breaking

Valid inequalities. A valid inequality is a constraint that can be added to a model without removing any valid solution. In other words, adding a valid inequality to a model does not constrain the model more than before adding the constraint, which naturally raises the question of their use case. Yet, valid inequalities can remove valid solutions from the relaxed problem. These constraints, the so-called cutting planes, should help the solving process and are the essence of useful valid inequalities.

Remark. There exist ILP solving approaches which solely rely on solving linear programming problems and cutting planes such as the Gomory Cutting Plane Algorithm [Gom58].

Adding constraints has a cost and one has to find the right trade-off between a heavier model and a finely-constrained one. In some cases, it is possible to drastically reduce the search space by removing *equivalent* or *symmetric* valid solutions with new constraints. These are not valid inequalities but permit to achieve a similar goal: speeding up the solving process.

Symmetry breaking. Among the constraints which remove valid solutions, we have a particular interest in symmetry breaking constraints [Wal06]. In many problems, one can go from one solution to an equivalent one just by interchanging some variables values. Additional constraints that remove some symmetric solutions but keep at least one of them are called symmetry breaking constraints. Often enough, these constraints drastically reduce the search space and looking for symmetries in a problem is usually not vain.

Remark. Given a solution obtained by the solver, symmetric solutions that were removed from the search space can always be built afterwards, in post-processing.

For example, shift-and-add solutions to the MCM problem have symmetries such as the order of the adders at a given depth. In the MILP models proposed in this thesis, we will add constraints to break these and other symmetries.

Equivalent solutions. Finally, we can also remove equivalent solutions which are not symmetries. For example, for MCM, to obtain an optimal solution it is not necessary to explore all the possible shifts.

Theorem 1: Shifts search space [DM94, Theorem 3]

In odd fundamental graphs, we can consider only the following shift values:

- either $s_{a,l} > 0$ and $s_{a,r} = 0$, *i. e.*, if the left input shift is positive, there is no right shift;
- or $s_{a,l} = s_{a,r} < 0$, *i. e.*, if the left input shift is negative, the right input shift is negative and equal to the left input shift.

Based on Theorem 1, Kumm [Kum18] added constraints on the variables encoding shifts:

$$\Phi_{a,r,s} = 0, \qquad \forall a \in [\![1; N]\!], s \in [\![1; S_{\max}]\!], \tag{2.21}$$

$$\Phi_{a,l,s} = \Phi_{a,r,s}, \qquad \forall a \in [\![1; N]\!], s \in [\![S_{\min}; -1]\!]. \tag{2.22}$$

However, as we will detail in Chapter 4, adding these constraints within his model actually removed optimal solutions.

2.2.3 Tweaking solver parameters

Solver parameters can be used to: (i) decrease the solving times and (ii) hopefully increase the robustness of the model by decreasing the randomness. In the following, we will present a few parameters that will be used in the different MILP models of this thesis. Fixing some of these parameters to a sensible value requires basic knowledge of the solving process.

Essentially, the solver will use multiple primal heuristics to find an initial solution, and a branch-and-bound-based approach with many subroutines. Roughly, the better the solution is, the faster the branch-and-bound will finish. For its part, the branch-and-bound algorithm will reduce the size of the problem by splitting it into multiple sub-nodes. These nodes have some variables fixed and the solver tries to only explore nodes that could improve the current solution. To determine the nodes to explore in priority, multiple subroutines run in parallel. For instance, the linear relaxation of a node is computed to obtain a first bound on the best solution which can be obtained, now that some variables are fixed. If this lower bound is worse than the current best known solution, the node is bounded and can be removed. When the node is kept, it means that it might still lead to interesting solutions, if not to the optimal one. Then, heuristics on the subproblem are used to find a valid solution which is used to guide the solver towards the best node to split.

Therefore, parameters that influence the primal heuristics, the branching options and the subroutines are of interest, as they can speed up the solving process. Although default parameter values are well-suited to most cases, knowledge of the problem that is being solved permits to tweak them better.

Primal heuristics

We recall that primal heuristics are meant to obtain an initial solution. Ideally, we would like the best initial solution possible. However, any solution is already interesting as it allows for a local search along with the branch-and-bound. As we will usually use a warm start for our models, we will already have a feasible solution so we mostly need to know if primal heuristics work well on our problem in the sense that they permit to obtain a better solution than the one we provide. If they do not, then we probably can even deactivate them.

Specifically, the feasibility pump [FGL05, BLS18] heuristic which permits to find first feasible solutions to hard problems can be activated or deactivated, using the parameters `PumpPasses` on Gurobi and `CPXPARAM_MIP_Strategy_FPHeur` on CPLEX. This primal heuristic is not meant to obtain *good* solutions hence it could certainly be deactivated as we already have a solution.

Branch-and-bound

One of the subroutines of branch-and-bound consists in using heuristics to find and improve solutions at each node. The heuristic effort is determined using `Heuristics` with Gurobi and `CPXPARAM_MIP_Strategy_HeuristicEffort` with CPLEX. The harder the problem is, the worse heuristics will perform close to the root node. In that case, we could be tempted to reduce the heuristic effort. However, if the problem involves a very large number of variables, branching might not be a good option either. Hence, to set this parameter it seems difficult to avoid trying different values to the see solver behavior on each specific problem.

Knowing the problem that we try to solve, we can have an intuition on which variables should be fixed *first* in the branch-and-bound, to quickly reduce the overall complexity of the problem. This can be precisely defined with `BranchPriority` on Gurobi or with `CPXstrongbranch` on CPLEX by assigning a branching priority to each variable. A more general branching rule order can be selected using parameters such as `CPXPARAM_MIP_Strategy_VariableSelect` on CPLEX or `VarBranch` on Gurobi. However, we believe that knowing which option is best between the available general branching rules probably already requires an expertise which permits to precisely specify the branching order anyway.

Several other options exist on branching strategy, such as the branching direction. Yet, besides the knowledge that it requires on the model, in our understanding, it is necessary to rewrite the whole model to correctly use this option. Indeed, all the variables need to be rewritten so the branching direction is common between them. Otherwise, setting a

branching direction would work for some variables but not all.

Finally, other options exist like specifying the presolving effort, the symmetry detection effort or the integrality tolerance. Especially, as exposed in Section 2.1.1, big-M can induce numerical instability and the integrality tolerance parameter might need to be adjusted. Depending on the problem, most of these parameters should have an impact on run-time and, when correctly set, we can use them to improve a model by reducing solving times, on average.

Remark. In the following, we will consider the sets of parameters to be a part of the models and, when not specifically mentioned, these parameters are supposed to be set to their default values.

2.3 Choosing between MILP models

There usually exists multiple MILP models to solve the same problem and this must raise the question of choosing the best one. On the one hand, the smallest model, in terms of the number of variables and constraints, might seem preferable. On the other hand, the model with the smallest search space might appear interesting too, though it usually involves more variables and constraints than strictly necessary. In addition to that, theoretically comparing solver parameters sets on a same model does not seem possible.

Overall, the theoretical "best" model probably does not exist as this seems to be a multi-criteria problem with equivalent solutions. That is why, in addition to these theoretical criteria, we will rely on comparison of the model's performance on the same set of benchmarks. As we will see in the following, performance is very variable and identifying a significant difference is an arduous task.

Performance variability. Benign changes in the model, such as different order in the constraints have a significant impact on the solving times. This phenomenon is called the *performance variability* [LT13, KBPV22] and needs to be acknowledged when comparing models and sets of parameters. According to [KBPV22], "The term performance variability, loosely speaking, comprises unexpected changes in performance that are triggered by seemingly performance-neutral changes."

From mathematical point of view, the order in which the constraints of an MILP model are fed to a solver does not impact the model itself and no performance variation is expected. By reordering the constraints, we did not modify the number of constraints and variables of the model, nor its linear relaxation and optimal solutions. However, these modifications naturally change the order in which the variables and constraints are passed to solvers. Unless the solver takes the time to reorder all data, breaking every tie in a deterministic way, a small difference in terms of solving times is expected.

In practice, it has been shown that this has a *significant* effect on the performance [LT13]. To measure this phenomenon, we have developed a Julia package which permits

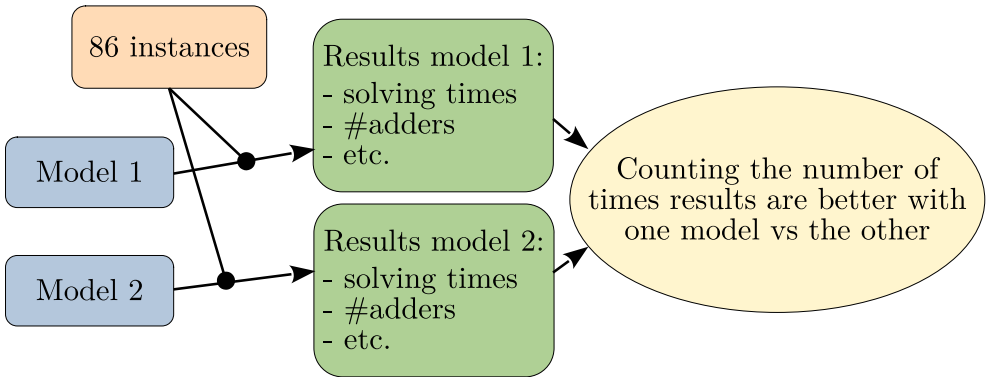


Figure 2.2: Comparison process.

to shuffle the constraints of a model, `ShuffleConstraints`[2]. Then, we compared a model for the MCM problem using multiple orders for the constraints and encounter a very large variability: on a given instance, one constraints order led to the optimal solution in less than a second, while, with a different constraints order, the solver timed out after 10 minutes. Note that we believe it is possible to mitigate this unexpected behavior by specifying solving parameters of the solver. However, we will not specifically explore this effect on performance variability in this thesis.

2.3.1 Statistical comparison

Solving times can range from unexpectedly small to unexpectedly large due to the performance variability phenomenon. Ideally, we would like to tweak our models, add symmetry breaking constraints and valid inequalities up to the point where the solver becomes completely deterministic without compromising solving time. Obviously, this is inaccessible and we will still need to compare nondeterministic models one with another.

Naturally, we could rely on statistical tests such as the Student's t-test [Stu08] to test the null hypothesis of equal average solving time. However, this test requires a large amount of data that we cannot obtain in a reasonable time: in our case, the time limit for the models is set to 8 hours, thus for an instance which would time out every time it is unrealistic to obtain enough data. To overcome this issue we could use the Welch's t-test [Wel47] which works on small samples.

However, Welch's t-test expects input samples to be normally distributed and, although it has been shown that Student's t-test also works for distributions that are not normally distributed [EN84], this has not been demonstrated for Welch's t-test as well. We do not know whether performance variability induces normally distributed solving times or not. This can be verified with Spiegelhalter's normality test [Spi80] or Shapiro-Wilk test [SW65] but non-rejection would not be a clear guarantee for the former and sample size might be too small for the latter. Nevertheless, we could still try to use Welch's t-test and expect it

[2] `https://github.com/remi-garcia/ShuffleConstraints`

to be statistically significant but this seems far-fetched and an in-depth analysis is needed, hence we prefer not to claim that our comparisons are statistically rigorous while there actually are multiple gray areas.

In this thesis, we will use "eyeball statistics", as referred to by Emery Berger[3], and compare our models on one run over 86 instances[4] from digital signal processing. When comparing two models, we start by running the optimization, on each instance, and then we count the number of times the first model outperforms the second one in terms of objective metrics and solving times, and vice-versa. This way, we can observe a trend and we expect that it holds despite the performance variability phenomenon. In Figure 2.2, we present the flow of our comparison process.

[3] `https://www.thestrangeloop.com/2019/performance-matters.html` – StrangeLoop 2019 talk, "Performance Matters" by Emery Berger.

[4] Details on the instances are provided in Appendix B.

CHAPTER 3

Towards automatic optimization of arithmetic operators

> "The wizards' automatic response to any problem was to see if there was a book about it."
>
> Terry Pratchett (1948-2015),
> *The Globe (2002)*

Implementation on FPGAs

Any hardware implementation starts with an algorithm's description, usually as a computational graph. This description is usually done using the Verilog [IEE05] or VHDL [IEE19] syntax which are two well-known hardware description languages (HDL). Using HDL, hardware designers must translate algorithms into hardware dataflows. This is not trivial and requires a deep understanding of each involved operator which needs to be optimized in order to obtain efficient implementations. Indeed, when implementing algorithms in resource constrained environments, such as FPGAs, even basic operations could be studied to provide a substantial gain in comparison with the straightforward implementation.

To alleviate this effort from hardware designers, high level synthesis (HLS) tools, such as Vitis HLS by AMD (formerly Xilinx), have been developed and are widely used. They consist in writing the algorithm in a high-level language, in C++ for example, and rely on the HLS tool to produce hardware code. This permits to write code closer to the algorithm to benefit from a significant abstraction level. The tools can usually be parameterized extensively to still have a good control over the synthesis process.

This higher level of abstraction provided by HLS, in comparison to HDL, comes with a few setbacks. In particular, the FxP arithmetic support is still on a basic level and HLS tools usually only efficiently optimize the implementation of basic arithmetic operations. Meaning that more complex operators, such as MCM or mathematical functions, are left

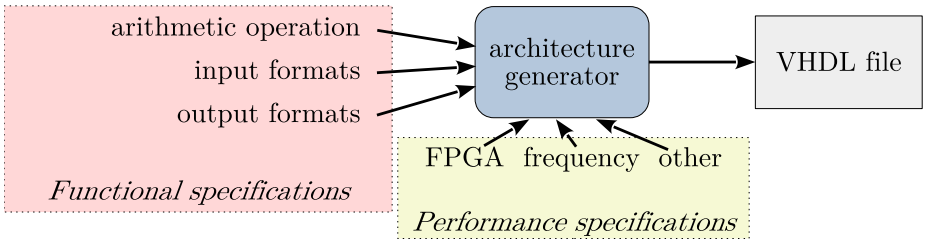


Figure 3.1: FloPoCo simplified interface.

to the designer and need to be implemented specifically for each application. All this is nontrivial and time-consuming for hardware designers. However, it is possible to specifically provide to the HLS tools the operators in HDL code. When doing so, in order to keep the accessibility of HLS, this HDL code is usually obtained using libraries and code generators such as Spiral [FLP+18] for fast-Fourier transforms or FloPoCo [dDP11] for various arithmetic operators.

With a few inputs, these tools aim at producing optimized hardware code. Basically, either we provide a hardware-agnostic representation of the operator, then the arithmetic core generator simply translates it into HDL code, or we rely on the arithmetic core generator for finding an efficient hardware implementation. By designing adder graphs, our first goal is to produce dedicated hardware instead of simply mimic generic processors, as in FloPoCo, whose simplified interface is represented in Figure 3.1. Many hardware problems, like MCM, have a combinatorial component and the pen and paper approach to optimize them is usually not well-suited, thus we prefer automatic methods.

Finally, we will be interested in computing just right, as in FloPoCo. This means that internal operator computations should drop unnecessary bits to save hardware and computational resources, as long as the output fits a given accuracy. As much as possible, we will also avoid wasting resources.

Overall, our idea is to (i) optimize automatically, based on a formal model of the hardware and of the operator; (ii) guarantee an output error bound to ensure quality but also performance as we do not compute unnecessary bits. Our approach relies on the mathematical modeling of the operator architecture. Hardware design usually involves combinatorial problems which fit well for MILP-based approaches.

To alleviate hardware designers from a tedious work, we will provide automatic HDL code-generation for arithmetic operators we study. We will compare to approaches embedded in FloPoCo and we include some solutions into this arithmetic core generator.

MCM operator ecosystem

FloPoCo, a the time of writing this thesis, embeds three operators which generate shift-and-add VHDL code for the MCM arithmetic operator. First, `IntConstMultShiftAdd` permits to convert an adder graph, formatted as a string, into VHDL code. The input parameters are simply the word length of the input and the adder graph. Suppose we want to produce the VHDL code for the adder graph represented in Figure 2.1a, which outputs

7 and 23. We choose to have 8-bit input, and using FloPoCo, we generate the VHDL code with the following command:

```
flopoco IntConstMultShiftAdd graph="{{'O',[7],2,[7],1,0},{'O',[23],2,[23],2,0},
    {'A',[7],1,[1],0,3,[-1],0,0},{'A',[23],2,[1],0,4,[7],1,0}}" wIn=8
```

Using the `IntConstMultShiftAdd` operator already requires knowing the adder graph we have decided to implement. FloPoCo provides two others operators, for the shift-and-add MCM implementation, which produce the adder graph on-the-fly given the target constants. The first operator, `IntConstMultShiftAddRPAG`, uses the RPAG heuristic [KZFC12] and the second operator, `IntConstMultShiftAddOpt`, relies on the existing MILP models [Kum16, Kum18]. In both cases, the optimization is done w. r. t. the number of adders. For the hardware implementation, it is necessary to provide the input data path. This will determine the cost of each intermediate operator and, despite being an important metric, the adder count does not benefit from this information.

So far, only exact MCM has been done "optimally" w. r. t. the number of adders. *A posteriori* truncation approaches have been recently proposed [GDJ10, dDFKF19]. Moreover, the heuristic approach [GDJ10] does not work with a user-given error bound and the output error need to be verified *a posteriori*. This is not completely fixed by the MILP approach [dDFKF19] which underestimates the error propagation. In any case, both methods consist in applying truncations over a fixed adder graph, which might not be "truncation-friendly" and the results could be improved.

In comparison to previous approaches, the MILP one has often proven its efficiency. In our work, we chose to optimize hardware operators using mathematical modeling as it as been done several times for the MCM problem [Gus08, Kum16, Kum18, LM20].

The MCM operator could also be used for the implementation of digital filters. For FIR filters, Kumm *et al.* [KVF23] demonstrated that it is possible and efficient to do the co-design of the filter coefficients and their implementation with MCM. In the case of IIR filter design, the state-of-the-art either relies on KCM [Cha94], or on coefficient design and quantization steps, followed by MCM, which leaves room for improvement. With our work, we aim at unifying this three-step filter implementation process into a single one, which relies on mathematical modeling, as for FIR filters [KVF23].

Operations research for operator generation

Overall, with this thesis, we aim at paving the way towards automatic optimization of hardware architectures for arithmetic operations from multiplication to digital filters. With this work, we are going to address the following problems:

- **MCM-Adders.** Optimizing the MCM problem can be done according to different metrics. First, we will consider shift-and-add implementations minimizing the number of adders, #A, solving the MCM-Adders problem. This metric has been the most widely used and we found an error in one of the MILP MCM-Adders approaches [Kum18], also present in the SMT one [LM20]. Thus, in Chapter 4, we will present the problem and our contribution to correcting it. We also note that these models

usually only embed the concept of adders and other metrics such as the adder depth, are evaluated outside. With our work, we want to provide less abstract models, closer to real-life. Hence, our goal is to have more metrics computed within the model so we can update the objective or the constraints on these metrics and not rely anymore on outside computation. This is a greater burden for the solver that we will try to alleviate using operations research techniques such as valid inequalities or parameter fine-tuning.

- **MCM-Bits.** For hardware implementation, the #A metric can be refined and it is possible to count the number of one-bit adders, $\#A_b$, instead. In Chapter 5, we will focus on this lower-level metric by minimizing $\#A_b$, solving the MCM-Bits problem. It requires to decompose adders into one-bit adders and to thoroughly look for the adder that are best-suited to minimize the total number of one-bit adders.

- **tMCM.** Following the "computing just right" principle, it is desirable to truncate unnecessary bits to save computing resources. In [dDFKF19, GVK22a], truncations have been included to the MCM problem to compute just right. However, the error propagation is incorrect and the user-given error could be exceeded. In Chapter 6, we will provide a correct, and tighter, error propagation rule. We propose an alternative MILP-based model, which permits up to have much tighter, and actually correct, error bounds. We model various cases that can occur and, for the first time, incorporate truncations into the adder graph search. With our solution, the solver automatically searches for the topologies well-suited for truncations. We refer to the problem of finding the adder graph with the minimal cost, including truncations, in a single global step, as the tMCM problem.

- **PMCM.** Pipelining adder graphs is a standard way to increase the throughput. Given a fixed adder graph, it is possible to to pipeline it efficiently [KZ11]. There also exist methods to heuristically find the best pipelined adder graph from the set of target constants [KZFC12], solving the PMCM problem. In our work, we aim at solving this problem within an MILP-based approach, in contrast to the state-of-the-art [Kum16] which relies on complete enumeration. We present our work on this topic in Chapter 7.

- **IIR filter design.** In Chapter 8, we demonstrate the usage of MCM as part of a larger model which can be used to design second-order IIR filters. Overall, the benchmarks we will use throughout this thesis come precisely from digital signal processing (DSP) applications. Although with this application we put an additional focus on DSP, multiple other domains, such as cryptography or neural networks, involve the MCM problem or a close variation.

Overall, with this thesis, we will demonstrate that MILP-based models are extensible and can include the computation of many hardware metrics internally. Each chapter features a description of a new MILP model. These models are available in Appendix A, as well as detailed benchmarks in Appendix B and complete tables of results in Appendix C.

Part II

Multiple Constant Multiplication

CHAPTER 4

Improving high-level MCM model

"It's still magic even if you know how it's done."

Terry Pratchett (1948-2015),
A Hat Full of Sky (2004)

4.1 Introduction

The objective of this chapter is to tackle the Multiple Constant Multiplication (MCM) problem using the shift-and-add approach. As detailed in Section 1.3.1, this is a common approach to efficiently solve MCM problems. In this chapter, our goal is to find optimal solutions with respect to high-level metrics, such as the number of adders, solving the MCM-Adders problem.

The MCM problem has been studied for decades and is conjectured to be NP-hard, even if the instance involves only a single constant. We want to stress out that the difficulty of the problem of finding a minimal cost algorithm depends, in particular, on the target constants. For these reasons, instead of trying to find a specific algorithm for this problem, we prefer to take advantage of efficient Mixed-Integer Linear Programming (MILP) generic solvers and we present an ILP model to solve MCM instances. From a mathematical description of the problem, we will show that we are able to obtain interesting results in a reasonable time using MILP solvers.

Previous works [Gus08, Kum16, Kum18] already used MILP-based models to solve the MCM-Adders problem. In [Gus08] and [Kum16], models were used to find the optimal solution within a search space which is precomputed. Our goal is to avoid this costly precomputation which is impossible when the word length of the target constants is high or when the adder depth is unbounded. In [Kum18], Kumm proposed an MILP-based model which directly solves MCM-Adders, *i. e.*, without any precomputation. It is a satisfaction problem, which requires an outside loop to minimize the number of adders, answering the

following question: "Given a set of target constants $C \in \mathbb{N}^{N_c}$, is there a way to build a shift-and-add tree, with N adders, that outputs all the constants in C?" Many internal subroutines of MILP solvers are specifically designed for minimization/maximization and using an outside loop does not benefit from this. Moreover, outside loops on MILP models are realistic if the number of iterations is limited, which is not the case for fine-grained metrics. Finally, we demonstrate, using a counterexample, that this model misses optimal solutions for some instances.

Overall, we will improve the high-level model for MCM by including secondary metrics to minimize within the model and by using solvers' parameters in order to improve their performance. This work has been published in the IEEE Transactions on Circuits and Systems I [GV23b].

4.2 Modeling the adder graph topology

4.2.1 A counter example to the state-of-the-art MILP model

To be able to express the adder graph topology as constraints, we need to declare the topology as a mathematical equation. To do so, Equation (2.12) links fundamentals together, but for convenience we recall it here:

$$c_a = (-1)^{\sigma_{a,l}} 2^{s_{a,l}} \times c_{a,l} + (-1)^{\sigma_{a,r}} 2^{s_{a,r}} \times c_{a,r}. \tag{4.1}$$

The adder graph topology is in direct relationship with these fundamentals, see Figure 4.2a. Kumm [Kum18] linearized this equation to provide an MILP model which encodes the topology. His model includes inconspicuous additional constraints which remove *valid* solutions. This could have been harmless if no optimal solutions were removed too.

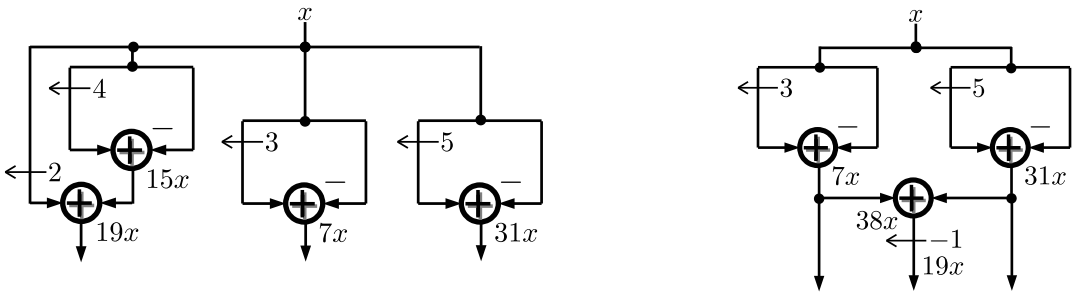
The issue comes from the fact that, first, Kumm based his model over (4.1), and, second, applied theoretical results from Dempster and Macleod [DM94] to reduce the model size. In particular, [DM94] demonstrated that it is possible to consider only odd fundamentals in the adder graph, and still obtain optimal solutions. Due to the linearization, the initial conditions of [DM94, Theorem 2] are not met anymore.

As we presented in Chapter 2, for its linearization Kumm introduced

$$c^{\text{sh}}_{a,l} = 2^{s_{a,l}} c_{a,l}, \tag{4.2}$$

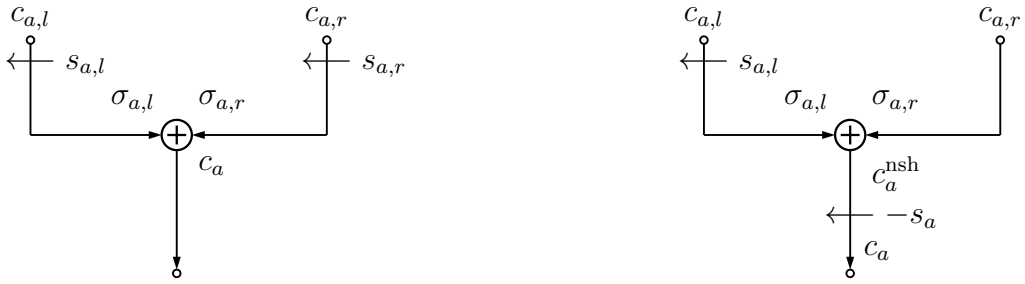
where $c^{\text{sh}}_{a,l} \in \mathbb{N}$. Thus, if inputs, $c_{a,l}$, are odd, negative shifts are out of question and, using the result from [DM94, Theorem 2], Kumm [Kum18] forbade even-valued fundamentals which removed completely the possibility to use negative shifts. However, these negative shifts are necessary and could still be applied to even-valued fundamentals.

For example, 3 adders are enough to produce an adder graph for the set of target constants $C = \{7, 19, 31\}$, as illustrated in Figure 4.1b while Kumm's model [Kum18] leads to 4 adders, in Figure 4.1a, where no negative shifts are used. In other words, the additional constraints, which were supposed to only remove equivalent solutions, also



(a) With the original model [Kum18]. (b) With our model, Chapter 4.

Figure 4.1: Adder graphs obtained for target constants $\{7, 19, 31\}$.



(a) Adder model in adder graph [Kum18]. (b) Adder in our MCM modeling.

Figure 4.2: Classic (left) and proposed (right) adder models.

removed important, and actually optimal, solutions. Due to using a two step process, linearization first and using theoretical results second, a conflict between the integrality constraints and the restriction to odd fundamentals only [DM94, Theorem 2] arose.

4.2.2 Our model

To avoid the issue presented above, we propose to rewrite Equation (4.1), which links adders together, directly taking into account the different theorems proposed in [DM94]. Basically, these theorems have the following consequence: for each adder, either one (does not matter which) of the inputs is left-shifted or the sum of the inputs is right-shifted. Arbitrarily, we decide that the left input can be shifted and not the right one. This leads to a new equation to link fundamentals:

$$c_a = 2^{-s_a} \left((-1)^{\sigma_{a,l}} \, 2^{s_{a,l}} c_{a,l} + (-1)^{\sigma_{a,r}} \, c_{a,r} \right), \tag{4.3}$$

for which we give a representation in Figure 4.2b.

The shifts, s_a and $s_{a,l}$, are not specifically constrained with this equation and even-valued fundamentals can still be computed. As these are not necessary, we can avoid this by adding a constraint on c_a,

$$c_a = 2c_a^{\text{odd}} + 1, \tag{4.4}$$

where c_a^{odd} is an integer variable. With this additional constraint, we removed *equivalent* valid solutions.

Now, Equation (4.3) must be adapted to MILP. First, we linearize it using intermediate variables similarly to (2.15):

$$\underbrace{2^{s_a} c_a}_{c_a^{\text{nsh}}} = \underbrace{(-1)^{\sigma_{a,l}} \underbrace{2^{s_{a,l}} c_{a,l}}_{c_{a,l}^{\text{sh}}}}_{c_{a,l}^{\text{sh,sg}}} + \underbrace{(-1)^{\sigma_{a,r}} \underbrace{c_{a,r}}_{c_{a,r}^{\text{sh}}}}_{c_{a,r}^{\text{sh,sg}}}, \tag{4.5}$$

to store intermediate results

$$c_a^{\text{nsh}} = 2^{s_a} c_a, \tag{4.6}$$
$$c_{a,l}^{\text{sh}} = 2^{s_{a,l}} c_{a,l}, \tag{4.7}$$
$$c_{a,l}^{\text{sh,sg}} = (-1)^{\sigma_{a,l}}\, c_{a,l}^{\text{sh}}, \tag{4.8}$$
$$c_{a,r}^{\text{sh}} = c_{a,r}, \tag{4.9}$$
$$c_{a,r}^{\text{sh,sg}} = (-1)^{\sigma_{a,r}}\, c_{a,r}^{\text{sh}}. \tag{4.10}$$

It can be noticed that, for uniformity, we add a symbolic link between $c_{a,r}^{\text{sh}}$ and $c_{a,r}$, though right inputs to adders are never shifted. We linearize the product (4.6) using binary variables $\Psi_{a,s} \in \{0,1\}$, that basically select the shift s for the adder a, and the following indicator constraints

$$c_a^{\text{nsh}} = 2^s c_a \quad \text{if } \Psi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!], s \in [\![0; w]\!], \tag{4.11}$$

where terms $2^s, s \in [\![0; w]\!]$, become precomputed parameters, selected by $\Psi_{a,s}$. This can also be written as big M constraints.[1] Similarly, we use binary variables $\Phi_{a,s} \in \{0,1\}$, to linearize (4.7):

$$c_{a,l}^{\text{sh}} = 2^s c_{a,l} \quad \text{if } \Phi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!], s \in [\![0; w]\!], \tag{4.12}$$

where $\Phi_{a,s}$ encodes that the left input of the adder a is shifted by s bits. In both cases, 4.11 and 4.12, we also need to enforce that one and only one shift per adder, even if zero-valued, is chosen,

$$\sum_{s=0}^{w} \Phi_{a,s} = 1, \qquad \forall a \in [\![1; N]\!], \tag{4.13}$$
$$\sum_{s=0}^{w} \Psi_{a,s} = 1, \qquad \forall a \in [\![1; N]\!]. \tag{4.14}$$

[1] We recall that, for conciseness, in this thesis, we will present all the constraints that could also be written with big M as indicator constraints instead.

Without these constraints, none of the constraints on c_a^{nsh} and $c_{a,l}^{\text{sh}}$, (4.11) and (4.12), would actually apply.

Using binary variables to encode the sign of each input, $\sigma_{a,i} \in \{0,1\}$, we linearize (4.8) and (4.10) as follows:

$$c_{a,i}^{\text{sh,sg}} = -c_{a,i}^{\text{sh}} \quad \text{if } \sigma_{a,i} = 1, \qquad \forall a \in [\![1;N]\!], i \in \{l,r\}, \tag{4.15}$$

$$c_{a,i}^{\text{sh,sg}} = c_{a,i}^{\text{sh}} \quad \text{if } \sigma_{a,i} = 0, \qquad \forall a \in [\![1;N]\!], i \in \{l,r\}. \tag{4.16}$$

Finally, we have

$$c_a^{\text{nsh}} = c_{a,l}^{\text{sh,sg}} + c_{a,r}^{\text{sh,sg}}, \qquad \forall a \in [\![1;N]\!], \tag{4.17}$$

which achieves the linearization of (4.3).

Now that we encoded, in the MILP-based model, the fundamental computation from its inputs, we need to link adders together. To do so, we use binary variables $c_{a,i,k} \in \{0,1\}$ which encode that input i of adder a comes from the adder k. This way, we link the inputs, $c_{a,i}$, with preceding adders,

$$c_{a,i} = c_k \quad \text{if } c_{a,i,k} = 1, \qquad \forall a \in [\![1;N]\!], i \in \{l,r\}, k \in [\![0;a-1]\!], \tag{4.18}$$

and we make sure that each input of each adder is linked with one preceding adder:

$$\sum_{k=0}^{a-1} c_{a,i,k} = 1, \qquad \forall a \in [\![1;N]\!], i \in \{l,r\}. \tag{4.19}$$

Similarly, we link adders with target constants using binary variables $o_{a,j} \in \{0,1\}$ which encode that the adder a is the output for the j-th output. Given the set of target constants C, we denote $|C|$ its cardinality and C_j its j-th element. This way, we add the constraints

$$c_a = C_j \quad \text{if } o_{a,j} = 1, \qquad \forall a \in [\![0;N]\!], j \in [\![1;|C|]\!], \tag{4.20}$$

$$\sum_{a=0}^{N} o_{a,j} = 1, \qquad \forall j \in [\![1;|C|]\!], \tag{4.21}$$

to ensure that each target constant is computed by one adder. We summed up the constants and variables used, up to this point, in Table 4.1.

Now that the model is complete, we propose two valid inequalities to tighten the linear relaxation. First, we can realize that a shift to the left input is possible if and only if no right-shift occurs at the output of the adder. Formally, the left input is shifted if and only if $\Phi_{a,0} = 0$. This literally means that the left shift is not equal to 0 bits but some other value. Similarly, the absence of right-shift is denoted by $\Psi_{a,0} = 1$. This leads to the following set of valid inequalities:

$$\Phi_{a,0} + \Psi_{a,0} = 1, \qquad \forall a \in [\![1;N]\!], \tag{4.22}$$

Table 4.1: Constants (top) and variables (bottom) used in the MILP formulation

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

which denotes that either the left-shift or the right-shift has to be zero-valued, but not both. Second, we can use the constraints already proposed by Kumm [Kum18],

$$\sigma_{a,l} + \sigma_{a,r} \leq 1 \qquad \forall a \in [\![1; N]\!], \tag{4.23}$$

which state that at most one of the inputs can be negative. This is already verified due to the lower bound on fundamentals, $c_a \geq 0$ which would not hold with two negative inputs. However, these additional constraints on σ's will efficiently tighten the linear relaxation of our model. Finally, we have a complete model to solve the MCM-Adders satisfiability problem for a given number of available adders N. All the constraints are gathered together in a model available in Appendix A.1.1.

Minimization problem

In practice we want to minimize the number of adders to produce an adder graph which computes a given set of target constants. As Kumm [Kum18] proposed, to find the minimal number of adders required we use an outside loop to the model definition and optimization which increments the number of available adders N. We start from a known lower bound [Gus07]. In the previous section, we presented an MILP model that is either infeasible or provides an adder graph that permits to compute a set of target constants. Proving infeasibility can be overly difficult and hours could be spent without obtaining a proof.

Hence, instead of starting from a known lower bound [Gus07], which could be infeasible, we could be tempted to tackle the problem from the other end, *i. e.*, starting from a number of adders N big enough and decrementing this value until the model is infeasible. However, in both cases, at each new optimization step, information from the previous ones is lost. Hence, it would be preferable to start with a number of available adders N big enough to obtain a solution, and then to "ask" the solver to minimize their usage.

We start by fixing N to a known upper bound, obtained using a heuristic solution [VP07, KZFC12] or a greedy algorithm [Ber86]. Then we add to the original model one binary variable per adder, $u_a \in \{0, 1\}$, $\forall a \in [\![1; N]\!]$ which is true if adder a is used. This permits to deactivate unnecessary adders:

$$c_a = 0 \text{ if } u_a = 0, \qquad \forall a \in [\![0; N]\!]. \tag{4.24}$$

Finally, adding to the model the objective function

$$\min \sum u_a, \tag{4.25}$$

permits to solve the MCM-Adders problem as one minimization problem.

Thanks to the use of a heuristic, besides having a valid value for N to start with, we obtain a complete solution which we use as a warm start. This speeds up solving. Furthermore, we can use knowledge on the minimal number of adders [Gus07], $\underline{N}$, to fix $u_a = 1$ for the $\underline{N}$ adders. This avoids that the solver searches for solutions which do not involve enough adders anyway.

It is straightforward to see that it does not matter if the used adders are the first ones, the last ones, or randomly spread among the available ones: these are symmetries in the search space. Adding symmetry breaking constraints in order to avoid wasting solving time finding symmetrical solutions over and over can be done with

$$u_{a-1} \geq u_a, \qquad \forall a \in [\![1; N]\!], \tag{4.26}$$

which ensures that the first adders are used in priority.

We proposed a new MILP model which permits to solve to optimality the MCM-Adders problem. This model, seemingly simple, is the first formalization of the *minimization* problem for MCM-Adders. Building on top of it, we will be able to extend our model to different metrics and to incorporate constraints on the problem, such as a bound on the adder depth. Moreover, thanks to our extension to avoid the costly outside loop on the number of adders, we claim that we can improve the quality of the solutions even when solvers do not provide optimality proof.

4.2.3 Optimization results

We compare our models with the publicly available implementation[2] of the RPAG heuristic [KZFC12] and our implementation of a greedy algorithm based on CSD [Ber86]. For

Table 4.2: Number of adders obtained with each method within the available solving time. Results followed by a star (*) are not proven optimal w. r. t. the number of adders. MILP_I and MILP_M correspond to our model with indicator constraints and big-M constraints, respectively.

Benchmark			Loop		Min	
	CSD	RPAG	MILP_I	MILP_M	MILP_I	MILP_M
`Gaussian 3`	5*	4*	4	4	4	4
`Gaussian 5`	10*	6*	5	5	5	5
`Highpass 5`	5*	4*	4	4	4	4
`Highpass 9`	7*	5*	5	5	5	5
`Highpass 15`	14*	12*	12	12	12	12
`Laplacian 3`	6*	4*	3	3	3	3
`Lowpass 5`	11*	7*	6	6	6	6
`Lowpass 9`	17*	13*	12	12	12	12
`Lowpass 15`	51*	27*	-	-	27*	26*
`Unsharp 3-1`	5*	4*	4	4	4	4
`Unsharp 3-2`	11*	6*	5	5	5	5

the sake of completeness, we present solving-time results for our models with an outside loop and as a minimization problem using big M constraints and indicator constraints.

First, we need to report that within the 8-hour time limit, the model which directly minimizes the number of adders permited to obtain the best known solution in most cases: for 81 and 83 instances out of 86 instances using indicator and big M constraints, respectively. Using the outside loop, the best known solution was obtained for 56 and 60 instances, using indicator and big-M constraints, respectively. This clearly demonstrates advantage of minimization over satisfiability. In Table 4.2, we report the number of adders obtained with each approach on a part of the benchmarks, we provide the rest of the results in Appendix C.1.

Both MILP-based approaches show significantly better results than the greedy algorithm (CSD) and the heuristic (RPAG) which found the best solution only for only 10 and 48 instances, respectively. It was expected that the minimization model would outperform RPAG as the solution of RPAG is used as a warm start. However, the solution obtained with RPAG has been improved for more than 30 instances and our results highlight that there was room for improvement which we exploited. For instance, we note in Table 4.2 for the benchmark `Gaussian 5` that RPAG found a solution with 6 adders when our models led to a solution with only 5 adders. It is also interesting to see that the MILP-based model which relies on the outside loop is also more efficient than the heuristic.

Nevertheless, it should be noted that heuristics and greedy algorithms permit to obtain solutions almost instantaneously while MILP-based methods reach the time limit for one-

[2] `https://gitlab.com/kumm/pagsuite`

third of the benchmarks, 30 instances for the outside loop model with indicator constraints.

Overall, it seems that the solver performs best with big-M constraints over indicator ones. Hence, in the following, we will implement the models with big-M constraints, even though we write indicator constraints for conciseness. Moreover, with the minimization model, we obtain the best adder graph, w. r. t. the number of adders, more often and we will carry on with this model.

4.3 Bounding and minimizing the adder depth

4.3.1 Adder depth

The adder depth of the adder graph corresponds to the total number of cascaded adders, see Definition 1. We also refer to the depth of an adder as its adder depth. Bounding the adder depth is often considered to reduce the adder graph delay [DDK00, SY11]. We also recall that a bound on the adder graph is a necessary condition to perform a complete enumeration of the search space. Kumm proposed an optimal approach [Kum16, Section 5.2] relying on this complete enumeration which can, *de facto*, solve MCM-Adders with the additional constraint of a bounded adder depth. However, as shown in [Kum16, Section 5.5], it is not reasonable to enumerate the whole search space for depths starting from 4. Hence, it is desirable to have the possibility to bound the adder depth within an MILP model to avoid precomputing the search space.

In order to track the adder depth, we introduce the variable $ad_{\max} \in \mathbb{N}$. We also add two sets of integer variables to encode the adder depth of each adder:

- $ad_a \in \mathbb{N}$, $\forall a \in [\![0; \mathrm{N}]\!]$, corresponds to the adder depth of the adder a;
- $ad_{a,i} \in \mathbb{N}$, $\forall a \in [\![1; \mathrm{N}]\!], i \in \{l, r\}$ is the adder depth of the left and right inputs of adder a.

Naturally, the adder depth of the input to the adder graph is set at zero, $ad_0 = 0$. Obviously, the adder depth of an adder or of the adder graph cannot exceed the number of adders, thus we actually have $ad_{\max} \leq N$, $ad_a \leq N$ and $ad_{a,i} \leq N$ for every adder a. These constraints may seem repetitive but we should explicit as much knowledge we have on the variables as we can. This drastically helps the solver.

The adder depth propagation can then be handled with the equation provided in Definition 1:

$$ad_a = \max\left(ad_{a,l} + 1, ad_{a,r} + 1\right), \qquad \forall a \in [\![1; \mathrm{N}]\!]. \tag{4.27}$$

To fit in an MILP-based model, we need to linearize the "max" operator. First, we simply ensure that the adder depth is at least as large as it should be:

$$ad_a \geq ad_{a,l} + 1, \qquad \forall a \in [\![1; \mathrm{N}]\!], \tag{4.28}$$

$$ad_a \geq ad_{a,r} + 1, \qquad \forall a \in [\![1; \mathrm{N}]\!]. \tag{4.29}$$

Then, as it is commonly done, we introduce a binary variable $ad_a^b \in \{0,1\}$, for each adder, to enforce that ad_a is smaller or equal than $ad_{a,l}+1$ or $ad_{a,r}+1$:

$$ad_a \leq ad_{a,l} + 1 + \mathrm{N}ad_a^b, \qquad \forall a \in [\![1;\mathrm{N}]\!], \tag{4.30}$$

$$ad_a \leq ad_{a,r} + 1 + \mathrm{N} \times \left(1 - ad_a^b\right), \qquad \forall a \in [\![1;\mathrm{N}]\!]. \tag{4.31}$$

Hence, combined with (4.28)-(4.29), ad_a is exactly equal to one parameter of the max operator. In the above constraints, we have used the fact that the adder depth is always bounded by N.

Similarly to what has been done for the propagation of fundamentals, we reuse the binary variables encoding the topology, $c_{a,i,k}$. This way, we impose that the adder depth of the inputs actually comes from the topology and this is done as follows:

$$ad_{a,i} = ad_k \quad \text{if } c_{a,i,k} = 1, \qquad \forall a \in [\![1;\mathrm{N}]\!], i \in \{l,r\}, k \in [\![0;a-1]\!]. \tag{4.32}$$

Finally, we enforce that the adder depth of the adder graph is at least as large as the adder depth of any adder:

$$ad_{\max} \geq ad_a, \qquad \forall a \in [\![1;\mathrm{N}]\!]. \tag{4.33}$$

As our goal is to bound or minimize the adder depth, the solver will reduce the value of $ad_{\max}$ as much as possible until it is exactly equal to one of the ad_a's, thus to the adder depth.

In case we have a user-given bound, $\overline{ad}$, we simply impose

$$ad_{\max} \leq \overline{ad}, \tag{4.34}$$

to make sure that the solver will only produce adder graphs which conform to this bound.

Instead of a bound on the adder depth, it could be preferable to have the minimal adder depth possible as a second objective, while simultaneously minimizing the number of adders as the main objective. Hence, we do not use (4.34) and, instead, the variable $ad_{\max}$ is used to modify the objective function to

$$\min \sum_{a=1}^{\mathrm{N}} (\mathrm{N}u_a) + ad_{\max}. \tag{4.35}$$

This new objective function is a weighted sum between two objectives, minimizing the number of adders with weight N and the adder depth, with weight 1. Because $\mathrm{N} \geq ad_{\max}$, the weights ensure that it is unconditionally stronger to reduce the number of adders over increasing the adder depth. Thus, it enforces a lexicographic optimization with the number of adders as first objective and the adder depth as second.

4.3.2 Tightening the search-space

With this section, our goal is to provide redundant constraints that should help to reduce the search space or tighten the linear relaxation. We are able to *a priori* know a lower bound on the adder depth which is necessary to compute a given product by a constant or set of constants [Gus07]. We denote $D_{\mathrm{LB}}(C_j)$ the function which defines a lower bound on the adder depth of a constant C_j:

$$D_{\mathrm{LB}}(C_j) = \lceil \log_2 S(C_j) \rceil, \tag{4.36}$$

where $S(C_j)$ corresponds to the number of nonzero digits for C_j in its CSD representation. For every constant C_j of the set of target constants C, there is no point in considering for output an adder whose adder depth is not at least equal to $D_{\mathrm{LB}}(C_j)$. This way, we can add the set of constraints,

$$ad_a \geq D_{\mathrm{LB}}(C_j) \times o_{a,j}, \qquad \forall a \in [\![1; N]\!], j \in [\![1; |C|]\!], \tag{4.37}$$

which fixes $o_{a,j} = 0$ if the adder depth of a could not produce the output C_j at its depth ad_a. This constraint is obviously verified in any case but we add this set of valid inequalities to tighten the linear relaxation of the model.

Example 6. The constant 49 has a minimal adder depth $D_{\mathrm{LB}}(49) = 2$, hence Constraint 4.37 becomes,

$$ad_a \geq 2o_{a,1}, \qquad \forall a \in [\![1; \mathrm{N}]\!],$$

enforcing adders to have an adder depth of at least 2 before they can be even considered as potential output adders for the target constant 49.

Besides using valid inequalities, we can break symmetries by enforcing an order for the used adders. In that respect, we ensure that adders are ordered by depth, which is done with the following set of constraints:

$$ad_{a-1} \leq ad_a + N(1 - u_a), \qquad \forall a \in [\![1; N]\!], \tag{4.38}$$

where $N(1 - u_a)$ is necessary to avoid a collision between the adder depth of unused adders and the symmetry breaking constraint.

4.3.3 Optimization results

Choosing the right bound on the adder depth requires an expertise which is not compatible with generic benchmarks. For this reason, we will only consider the minimization of the adder depth as a second objective, with the objective function (4.35), and we will not use a strict bound. With the following experiments, we aim to evaluate the cost-benefit of

Table 4.3: Number of adders and adder depth obtained with each method within the available solving time. MCM-Adders, BiObj. and Tighten correspond, respectively, to the basis model with big-M constraints, the lexicographic bi-objective one and the model with tightening constraints. Solving times are given in seconds and TO stands for timed out.

Benchmark	MCM-Adders			BiObj.			Tighten		
	#A	AD	time	#A	AD	time	#A	AD	time
`Gaussian 3`	4	2	1	4	2	1	4	2	1
`Gaussian 5`	5	4	5	5	4	210	5	4	154
`Highpass 5`	4	2	1	4	2	1	4	2	1
`Highpass 9`	5	2	1	5	2	1	5	2	1
`Highpass 15`	12	2	1	12	2	4	12	2	1
`Laplacian 3`	3	3	1	3	3	1	3	3	1
`Lowpass 5`	6	3	1	6	3	52	6	3	51
`Lowpass 9`	12	3	6	12	3	TO	13	3	TO
`Lowpass 15`	26	4	TO	-	-	TO	27	3	TO
`Unsharp 3-1`	4	2	1	4	2	1	4	2	1
`Unsharp 3-2`	5	4	2	5	3	23	5	3	57

considering the adder depth within the model in terms of the solving time and the quality of the solutions.

In Table 4.3, we compare results solving the MCM-Adders problem with/without the adder depth as a second objective, and with/without tightening constraints (4.37) and (4.38). Overall, including the adder depth as an objective induced larger solving times. In particular, the solver was not able to obtain the best known solutions, in terms of number of adders, for 27 instances, within the available time. However, the adder depth successfully decreased for 6 instances without impacting the number of adders, see instance `Unsharp 3-2` for example, and for 12 instances at the cost of a few adders.

Note that we did not provide a warm start for all the additional variables and expected that the solver would infer the adder depth from the starting value of shifts, signs and fundamentals. Yet, for the largest instances the solver stopped trying to repair the warm start before obtaining a valid solution. For instance, for `Lowpass 15`, no solution was obtained after the 8-hour time limit while we could find one in just a few milliseconds using a heuristic. To overcome this issue we could, for example, put more work on the warm start to provide all the initial values or use some solver parameters, such as `SubMIPNodes` or `StartNodeLimit` with Gurobi, which permit to increase the solving efforts using the warm start.

With tightening constraints, for a consequent number of instances, we obtained solutions with a larger number of adders than the best known solutions. For 20 instances out of 86 it permitted to decrease the adder depth, sometimes at the cost of an extra adder, as for instance `Lowpass 15`. Finally, we remark that the solver proved that, for benchmark

`Gaussian 5`, the optimal solution, in terms of adder count first and adder depth second, requires 5 adders and an adder depth equal to 4. Hence, bounding the adder depth to 3 would either lead to an infeasible problem or increase the number of adders in the solution: there usually is a trade-off between adder depth and number of adders.

Overall, modeling the adder depth permits to obtain solutions with lower adder depth in comparison with the original model. It comes with a cost in terms of solving time, which is partly mitigated thanks to tightening constraints.

In this section we illustrated that it is possible to consider the adder depth within the model at a reasonable cost. This is not limited to this metric and other metrics such as the glitch path count [DDK02, Kum18] can be incorporated as well. In the following, we will not use any specific second metric, as our main focus is on the hardware cost in terms of resources utilization.

4.4 Fine-tuning the MILP solver

In Section 2.2.3, we presented basic information on the solving process and a few parameters we can fine-tune. In this section, we will discuss specificities of the MCM problem in order to find values for these parameters which should help the solving process.

First, it should be noted that the MCM problem is not commonly known in the optimization and operations research community. Thus, default parameter values might not be particularly well suited for this problem. Moreover, heuristics are not designed specifically to work well on MCM. Yet, we believe that primal heuristics, such as feasibility pump, are generic enough to be suitable. For that reason, we will not specifically tweak parameters which influence heuristics but focus our efforts on the branch-and-bound parameters.

The branch-and-bound will build solutions step by step, one variable at a time. An inadequate search space exploration of the MCM problem is unlikely to produce valid solutions and let alone an interesting solution. Fixing the input shift of the first adder and then the fundamental of the last adder before building anything in the middle is unlikely to produce a valid solution. Guiding the order of the branching, using the `BranchPriority` parameter on Gurobi, could permit to build solution in a way which is similar to building an adder graph, adder by adder.

Hence, we will prioritize the first adder first, the second adder second, etc. This way we ensure that the fundamental of every adder is coherent with the adders which have already been fixed. Each adder is defined by multiple variables, the fundamental but also the inputs, the shifts, the sign, and so on. We decide to branch on the fundamental first, and then to branch on the variables which fix the topology. Finally, we branch on other variables such as $o_{a,j}$, since our goal is to fix all the variables which characterize an adder before branching on the next adder. We sum up the order of the branching in Figure 4.3.

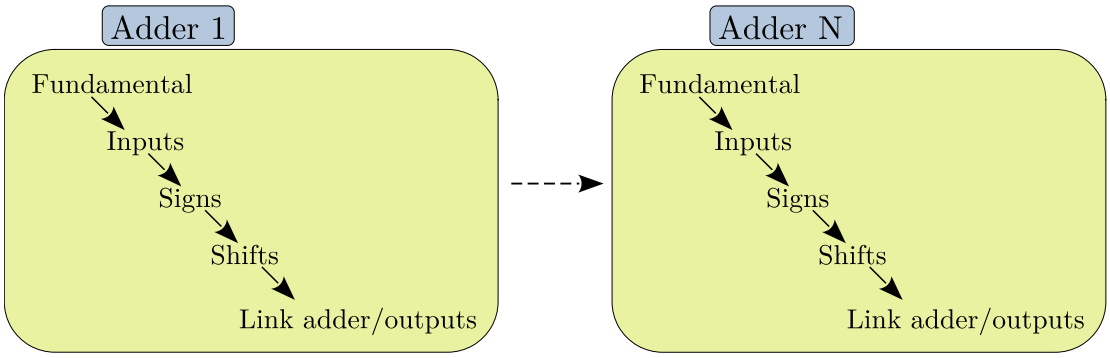


Figure 4.3: Branching order.

Table 4.4: Solving time (in seconds) to prove optimality using the model in Appendix A.1.2 without and with branching priority (BP) parameter. TO means that the time limit of 28800 seconds has been reached without proving optimality.

Benchmark	G.3	G.5	HP5	HP9	HP15	L.3	LP5	LP9	LP15	U.3-1	U.3-2
Without branch. priority	1	5	1	1	1	1	1	6	TO	1	2
With branch. priority	1	3	1	1	1	1	1	3	TO	1	2

4.4.1 Optimization results

In this section we compare solving times between the base model in Appendix A.1.2 and the same model including branching priority as illustrated in Figure 4.3. In Table 4.4, we present solving times for a few instances, the rest of the results is available in Appendix C.1.

Overall, out of 86 benchmarks, the branching priority parameter permitted to solve the model faster for 25 instances while it worsens the solving time for only 2 instances, but this could simply be due to performance variability. On average, in the whole set of benchmarks, solving times were reduced by 16.9% using branching priority. Although the number of instances which timed out is the same, 23 for both the basis model and when setting solver parameters, the trend is that using our branching priority is efficient.

These results are encouraging and it seems promising that a deeper analysis would permit to find the best branching priority parameter setting. In particular, the cost we paid in terms of solving times to include the adder depth within the model could probably be reduced by fine-tuning the branching priority. For the sake of simplicity and conciseness, in the following the model parameters will not be tweaked in the experiments.

4.5 Conclusion

In this chapter, we have improved the state-of-the-art with an MILP-based approach for MCM-Adders. We have demonstrated the limitations of existing models, and in partic-

ular we have corrected the state-of-the-art model [Kum18]. We have proposed different implementations, an outside loop for a satisfaction problem versus a minimization problem and using big-M or indicator constraints. We have compared these implementations with a heuristic and a greedy algorithm. As expected, all the proposed implementations outperformed the state-of-the-art in terms of the targeted metric, the number of adders. Actually, in the minimization case, the RPAG heuristic was used as a warm start and our approach built upon this initial solution.

Our experiments illustrated the superiority of solving a minimization problem instead of relying on an outside loop around satisfaction problems solved with MILP, like in literature [Kum18]. For the MCM-Adders problem, we also demonstrate that big-M constraints were better-suited than indicator constraints. For the rest of this thesis, we will use the minimization model with big-M constraints as a base model.

Then, we included the adder depth computation within the model. Moreover, we added the simultaneous minimization of the number of adders and the adder depth, for the first time. However, considering the adder depth within an MILP-based model comes with an extra cost in terms of solving times for a limited gain. Still, the new feature opens horizons for bounding the adder depth instead of minimizing it, or for including other metrics such as the glitch path count which could provide a significant gain despite the extra solving time cost it would induce.

Finally, by applying informed choices over solver parameters, we demonstrated that tweaking the model is a very promising path in order to improve solver performance for MCM and obtain better solutions faster. In the following models, we will not explore this option anymore due to the engineering cost it comes with. However, once a model seems here to stay, this step is essential to improve it again.

CHAPTER 5

MCM with a low-level cost metric

"Insanity is catching."

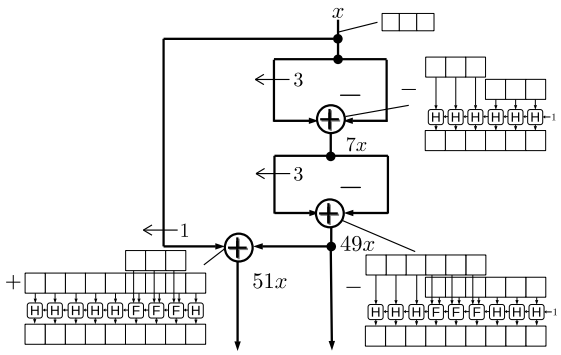
Terry Pratchett (1948-2015),
Making Money (2007)

5.1 Introduction

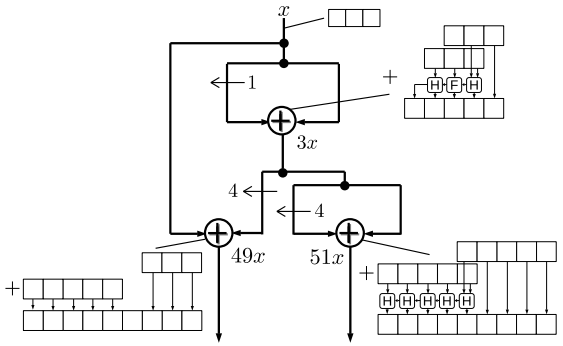
In Chapter 4, we presented our high-level MILP-based model to solve the MCM-Adders problem and we are now able to model adder graph topology as sets of variables and constraints. This permits to find optimal solutions, in terms of the number of adders. However, there are many solutions which have the same optimal number of adders and we want to find the best one among them. For that, we need a metric which would permit to differentiate equivalent MCM-Adders solutions. To do so, in this chapter, we will have a deeper look into adder graph implementation and we dissect adders into smaller components: half-adders and full-adders that we simply call one-bit adders. Taking into account this low-level metric, we will see significant differences between adder graphs which were equivalent w. r. t. the number of adders. For example, in Figure 5.1 we represent two adder graphs, which both compute the same outputs and both require 3 adders, but which require a significantly different number of one-bit adders for 3-bit inputs.

In this chapter, we will solve the MCM problem taking into account this low-level metric. In the following, we will refer to this specific problem as MCM-Bits. Naturally, we may consider using our MILP-based model for MCM-Adders to produce all the optimal solutions, in terms of adders, and search within them to solve the MCM-Bits problem. However, this is unrealistic as the number of optimal solutions for MCM-Adders significantly increases with the word length and the number of target constants.

It has also been demonstrated that the MCM-Bits problem can be solved heuristically [DM94, KZFC12]. However, our goal is to find *optimal* solutions. To that end, we build upon our MILP-based model to establish the adder graph topology and we incorporate



(a) Adder graph with 24 one-bit adders for 3-bit input.



(b) Adder graph with 8 one-bit adders for 3-bit input.

Figure 5.1: Different adder graph topologies computing the same outputs: $49x$ and $51x$

variables and constraints to count the number of one-bit adders of the adder graph. Unlike MCM-Adders, solving MCM-Bits requires to *a priori* know the word length of the input as it directly impacts the one-bit adder count.

Note that from a fixed adder graph, it is possible to precisely extract information on the number of one-bit adders required for its implementation [JGW07]. We derive from this information new sets of constraints in order to keep track of the number of one-bit adders *on-the-fly* within the MILP-based model. It comes with many challenges such as counting the number of one-bit adders for each adder within the model. This adds more constraints to the MILP problem, however, it has the potential to pay off as illustrated in Figure 5.1.

We compare hardware implementation results between adder graphs obtained using high-level and low-level approaches and demonstrate this way the benefit of fined-grain MILP models. The preliminary results on this work have been presented at the IEEE International Symposium on Circuits and Systems (ISCAS) [GVK22a] in 2022 and the full version is part of the publication [GV23b] in IEEE TCAS-I.

5.2 Modeling the one-bit adder count

We build upon the MILP-based model we presented in Chapter 4 to solve the MCM-Bits problem. In addition to the constraints fixing the adder graph topology, we need to keep track of the number of one-bit adders for each node of the adder graph. The precise number of one-bit adders depends on the word length of the inputs, as well as on the shifts or subtractions. There are many cases to be considered, as presented in [JGW07], if we want to tighten the upper bound as necessary. We first study how we can propagate the word length.

To do so, we decompose this into tracking the Most Significant Bit (MSB) and Least Significant Bit (LSB) positions, which, for integers, is roughly the same as simply tracking the word length. Indeed, the input x is a FxP number which is represented as an integer

and its LSB position is implicit. Hence, without loss of generality we can consider that x is an integer, thus its LSB is equal to zero. Then, each adder computes the product between an integer fundamental and the integer x, thus, it produces an integer whose LSB is still equal to zero. This permits to consider that the LSB is always equal to 0 and simply track the MSB.

The MSB after each adder can be computed without specifically considering the topology of the adder graph but simply from an upper bound on the input, $\overline{x}$, and the fundamental associated with the adder:

$$\mathrm{msb}_a = \lceil \log_2 (\overline{x} c_a + 1) \rceil . \tag{5.1}$$

This constraint is nonlinear but we can already remove the rounding operator by relaxing the equality:

$$\mathrm{msb}_a \geq \log_2 (\overline{x} c_a + 1) \tag{5.2}$$

We need the MSB of the adder a to be greater than $\log_2 (\overline{x} c_a)$ and the integrality constraint ensures that msb_a rounds up. We dropped the equality, hence msb_a could become arbitrarily large. However, it is not important if an MSB position larger than necessary is stored in msb_a provided this does not lead to more one-bit adders in the cost function we will propose. When needed, the solver could reduce the msb_a value exactly down to $\lceil \log_2 (\overline{x} c_a) \rceil$.

Still, the nonlinearity due to the presence of a $\log_2$ operator keeps us from using this constraint as it is. Exponentiation of both sides leads to

$$2^{\mathrm{msb}_a} - 1 \geq \overline{x} c_a, \tag{5.3}$$

which can be linearized similarly to shifts with the challenge of the wider range of possible values for msb_a, hence more variables and constraints. Indeed, possible shifts only range from 0 to the word length of the maximum input [DM94].

We are now able to compute the data word length, which is the MSB value plus one but we also need to link it to topology and propagate it. We introduce integer variables, $\mathrm{msb}_{a,i} \in \mathbb{N}$, corresponding to the MSB of the input i of adder a. Then we propagate the MSB position using variables $c_{a,i,k}$ that handle topology:

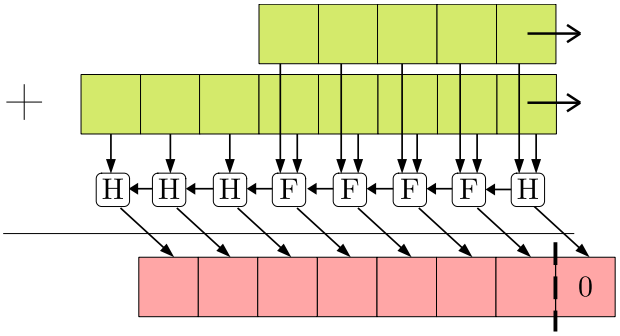
$$\mathrm{msb}_{a,i} = \mathrm{msb}_k \quad \text{if } c_{a,i,k} = 1, \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, k \in [\![0; a-1]\!]. \tag{5.4}$$

This will permit to correctly compute the number of one-bit adders within every adder.

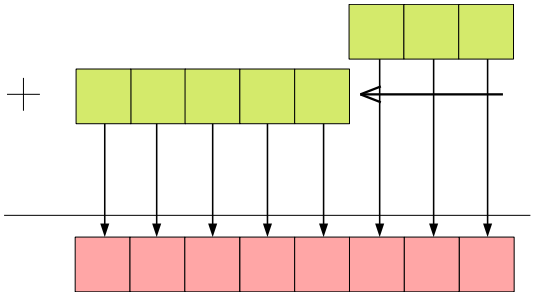
Note that, in the worst case, each bit of the adder output could need a one-bit adder to compute its value and, the LSB value might have been impacted by the carry of one-bit adders which output 0's, as in Figure 5.2a, in case of a right-shift after the adder. Hence we have this first upper bound on the number of one-bit adders B_a corresponding to an adder a:

$$\mathrm{B}_a \leq \mathrm{msb}_a + 1 - s_a. \tag{5.5}$$

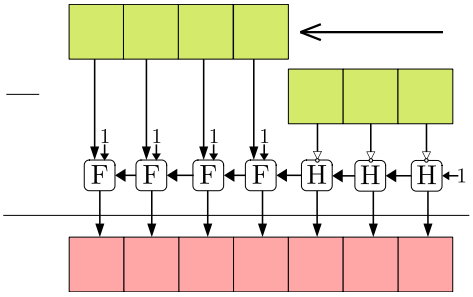
However, this bound is not reached in the general case, since one-bit adders can be saved due to data alignment as illustrated in 5.2.



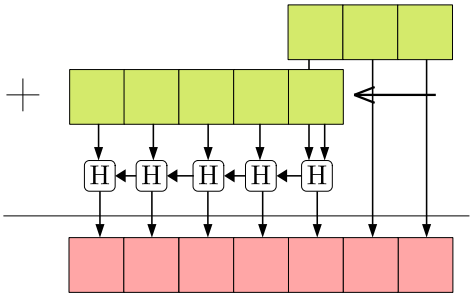
(a) In case of right-shift, the MSB of the output decreases but not the one-bit adder cost.



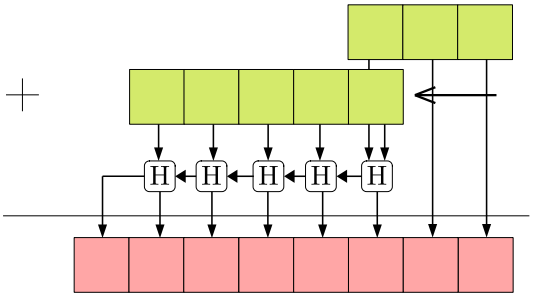
(b) When shifts are large enough, no one bit adders are required.



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



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



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

Figure 5.2: Counting one-bit adders.

Counting the number of one-bit adders, B_a, can be seen as computing the word length of the adder, $\mathrm{msb}_a + 1 - s_a$, and deducting the gains $g_a \in \mathbb{N}$ and $\psi_a \in \{0, 1\}$ we can obtain with shifts and carry, respectively:

$$\mathrm{B}_a = \mathrm{msb}_a + 1 - s_a - g_a - \psi_a. \tag{5.6}$$

In particular, the MSB computation is either done by a dedicated one-bit adder, as represented in Figure 5.2d, or obtained from the carry of the last one-bit adder, as in Figure 5.2e. We encode this information in the binary variable ψ_a, for each adder, so $\psi_a = 1$ if and only if the MSB value is computed with a carry. We enforce this by adding indicator constraints which ensure that, if the MSB value is computed with a carry, then the input MSB, which is shifted, is strictly smaller than the MSB of the adder:

$$\mathrm{msb}_{a,l} + s_{a,l} + s_a \leq \mathrm{msb}_a - 1 \quad \text{if } \psi_a = 1, \qquad \forall a \in [\![1; N]\!], \tag{5.7}$$

$$\mathrm{msb}_{a,r} + s_a \leq \mathrm{msb}_a - 1 \quad \text{if } \psi_a = 1, \qquad \forall a \in [\![1; N]\!]. \tag{5.8}$$

These constraints do not ensure that ψ_a will be equal to 1 if a carry is used to compute the MSB but that a carry needs to be used if $\psi_a = 1$. As the objective is to minimize the number of one-bit adders, the solver will maximize carry usage, thus it will fix $\psi_a = 1$ if possible.

Example 7. In Figure 5.1b, the MSB of $3x$ is done by the carry while the MSB of $51x$ requires a dedicated one-bit adder.

Together with the MSB carry, one-bit adders can be saved thanks to shifts. In the best case, shifts can lead to output bits computed without any one-bit adder, we illustrate this in Figure 5.2b where the shift of one side of the addition exceeds the MSB of the other side. However, in case of subtractions, these large shifts are not beneficial, as illustrated in Figure 5.2c. Formally, the following constraint sums up both figures:

$$g_a = \mathrm{msb}_a \quad \text{if } s_{a,l} > \mathrm{msb}_{a,r} \wedge \sigma_{a,r} = 0, \qquad \forall a \in [\![1; N]\!]. \tag{5.9}$$

In any case, shifts are usually smaller than the data word length and permit to save trailing one-bit adders for additions or subtractions, see Figure 5.2d. In our case, where the left input can be shifted but not the right one, this leads to the constraint

$$g_a = s_{a,l} \quad \text{if } s_{a,l} \leq \mathrm{msb}_{a,r} \wedge \sigma_{a,r} = 0, \qquad \forall a \in [\![1; N]\!]. \tag{5.10}$$

The same constraint $g_a = s_{a,l}$ holds if the left input is subtracted to the right one. Lastly, if the right input is subtracted to the left one, all the one-bit adders are necessary and we have

$$g_a = 0 \quad \text{if } \sigma_{a,r} = 1, \qquad \forall a \in [\![1; N]\!]. \tag{5.11}$$

Finally, we simplify the conditions in (5.9) and (5.10) in order to compute g_a within the model. The solver will maximize the value of g_a, hence we only need to prevent g_a from exceeding certain values. First,

$$g_a \leq \mathrm{msb}_a, \qquad \forall a \in [\![1; N]\!], \tag{5.12}$$

$$g_a \leq s_{a,l} \quad \text{if } s_{a,l} \leq \mathrm{msb}_{a,r}, \qquad \forall a \in [\![1; N]\!], \tag{5.13}$$

ensures that (5.9) is always verified and (5.12) will prevail if $s_{a,l} > \mathrm{msb}_a$ and the left input is not subtracted.

Naturally, the objective function changes from minimizing the number of adders to minimizing the number of one-bit adders,

$$\min \sum_{a=1}^{\mathrm{N}} \mathrm{B}_a, \tag{5.14}$$

and can also be adjusted to solve the lexicographic bi-objective problem minimizing the number of one-bit adders first and the adder depth second:

$$\min \sum_{a=1}^{\mathrm{N}} \mathrm{NB}_a + ad_{\max}. \tag{5.15}$$

Conjecture 1. *Let C be a set of target constants, $\forall N \in \mathbb{N}$ such that there exists an adder graph with N adders that outputs C, the optimal adder graph w. r. t. the number of one-bit adders which outputs C requires at most K adders, where $K \leq N$.*

We make the conjecture that an optimal solution for MCM-Adders, whilst not necessarily optimal for MCM-Bits, provides a valid upper bound on the number of adders required for the optimal solution of MCM-Bits. As a consequence, we will use a bound on the number of adders and assume we do not exclude optimal solutions w. r. t. one-bit adders.

5.3 Experiments

We have implemented the MILP model-generation as an open-source tool called `jMCM`[1]. As we did in the previous chapter, we used the RPAG heuristic [KZFC12] to obtain a warm start. We solved our model using Gurobi 10.0.1 [Gur20] executed with 4 threads on an Intel® Xeon™ E7-8870 CPU at 2.10GHz. For each instance, the time limit is of 8 hours. Then, we generated VHDL from adder graphs, with our open-source Julia package `AdderGraphs`[2], which we then synthesized for FPGA using Vivado v2022.2 for the xc7k70tfbv484-3 Kintex 7 device.

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

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

[2] `https://github.com/remi-garcia/AdderGraphs`

Table 5.1: Number of adders, #A, and one-bit adders, #A_b, obtained with MCM-Adders vs MCM-Bits within the available solving time. Best results are in bold.

Benchmark	8-bit input				16-bit input				32-bit input			
	MCM-A.		MCM-B.		MCM-A.		MCM-B.		MCM-A.		MCM-B.	
	#A	#A_b	#A	#A_b	#A	#A_b	#A	#A_b	#A	#A_b	#A	#A_b
`Gaussian 3`	**4**	43	**4**	**40**	**4**	75	**4**	**72**	**4**	139	**4**	**136**
`Gaussian 5`	**5**	**57**	**5**	**57**	**5**	**97**	**5**	**97**	**5**	**177**	**5**	**177**
`Highpass 5`	**4**	42	**4**	**39**	**4**	74	**4**	**71**	**4**	138	**4**	**135**
`Highpass 9`	**5**	50	**5**	**47**	**5**	90	**5**	**87**	**5**	170	**5**	**167**
`Highpass 15`	**12**	122	**12**	**105**	**12**	218	**12**	**201**	**12**	410	**12**	**393**
`Laplacian 3`	**3**	34	**3**	**31**	**3**	58	**3**	**55**	**3**	106	**3**	**103**
`Lowpass 5`	**6**	67	**6**	**60**	**6**	115	**6**	**108**	**6**	211	**6**	**204**
`Lowpass 9`	**12**	130	13	**128**	**12**	**226**	13	232	**12**	**418**	13	440
`Lowpass 15`	**26**	316	27	**250**	**26**	524	27	**466**	**26**	940	27	**898**
`Unsharp 3-1`	**4**	**32**	**4**	**32**	**4**	**64**	**4**	**64**	**4**	**128**	**4**	**128**
`Unsharp 3-2`	**5**	57	**5**	**49**	**5**	97	**5**	**89**	**5**	177	**5**	**171**

that minimizing the one-bit adder count directly permits to obtain better adder graphs, w. r. t. to the number of one-bit adders, in comparison to simply solving MCM-Adders. The additional variables and constraints we included in the model could have made it too difficult for the solver to obtain interesting solutions. We also want to confirm that minimizing the number of one-bit adders effectively translates into a reduced hardware cost. The research questions we answer in this section are summarized into two main questions:

(RQ1) Does minimizing the one-bit adder count within an MILP-based model permit to obtain better adder graphs?

(RQ2) Do adder graphs with less one-bit adders permit cost reduction in the final hardware?

For conciseness, detailed results in tables are shown only for a subset of the benchmarks, while the statistics are discussed over the full set of 86 benchmarks.

5.3.1 Optimization results

In the following we will compare the number of one-bit adders across adder graphs which have been obtained by solving MCM-Adders and MCM-Bits, respectively. For the former adder graphs, we will compute the number of one-bit adders for 3 different input word lengths: 8-bit, 16-bit and 32-bit. Then, we will solve our model for MCM-Bits for these 3 word lengths. Note these usually are common choices but we could have work with any word length.

RQ1: Solving MCM-Bits leads to better adder graphs

In Table 5.1 we present our results in terms of adder count, #A, and one-bit adder count, #A_b. In grand majority of overall 86 instances, solving MCM-Bits led to better adder graphs, w. r. t. the number of one-bit adders, than solving MCM-Adders. For example, for 8-bit inputs with the instance `Gaussian 3`, optimizing directly for MCM-Bits led to a different topology than solving the MCM-Adders problem, which reduced the number of one-bit adders from 43 to 40. Over the 86 instances, better results have been obtained for 69 instances for 8-bit inputs, 66 instances for 16-bit inputs and 62 instances for 32-bit inputs. On average, we reduced the number of one-bit adders by 11.3% for 8-bit inputs. This average reduction is the most significant for smaller word length but, for the 32-bit inputs, we still have a gain of 3.4% one-bit adders, on average.

In a few cases, the adder graph obtained solving MCM-Bits within the available time was worse than the one obtained solving MCM-Adders, in terms of one-bit adder count. This is the case for `Lowpass 9`, and interestingly only for 16-bit and 32-bit inputs. In this case, we note that the number of adders is also larger. Although the number of possible adder graphs is the same for both MCM-Adders and MCM-Bits, the search space is actually more difficult to explore, as equivalent adder graphs for MCM-Adders may not be equivalent for MCM-Bits. Moreover, the cost of adder graphs is harder to compute in the MCM-Bits case, thus a deeper exploration of the branch-and-bound is necessary to find that a solution is in fact worse than the current best known.

Due to this additional difficulty, the solver timed out for 47 instances, for MCM-Bits, in the 8-bit input case, while it only timed out for 23 instance when solving the MCM-Adders problem. Optimality is harder to prove but, despite a more complex model, we still obtain good quality solutions as stated above. In most cases, it is a matter of minutes, or even seconds, until the best known solution is obtained and the main difficulty lies in the proof of optimality.

5.3.2 Hardware results

Optimization approaches for MCM-Adders have been studied for decades and we believe that it is a good marker of the robustness of this metric for the hardware implementation. Solving MCM-Bits is still novel and we believe it is needed to assert its interest to produce efficient hardware implementations. We present hardware results on the adder graphs obtained for 8-bit inputs, after *out-of-context* place and route.

RQ2: The one-bit adder metric is correlated with the number of LUTs

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 7.8% on average, and by 25.9% in the best case, compared to our model for MCM-Adders. However, some outliers exist and, for 7 instances, the number of LUTs actually increased. Our method significantly increases the delay, by 28.3% on average.

Table 5.2: Hardware results for 8-bit inputs. Best results are in bold.

Benchmark	MCM-Adders			MCM-Bits		
	LUTs	Delay (ns)	Power (mW)	LUTs	Delay (ns)	Power (mW)
`Gaussian 3`	44	**1.833**	5	**41**	1.936	**4**
`Gaussian 5`	**58**	3.166	**6**	**58**	**3.039**	**6**
`Highpass 5`	42	**1.922**	**5**	**41**	2.501	**5**
`Highpass 9`	51	1.93	**6**	**48**	**1.848**	**6**
`Highpass 15`	122	**1.933**	**11**	**108**	2.4	**11**
`Laplacian 3`	35	2.429	**4**	**33**	**2.378**	**4**
`Lowpass 5`	68	**2.449**	8	**64**	2.468	**7**
`Lowpass 9`	135	**2.6**	**14**	**134**	4.57	16
`Lowpass 15`	310	**3.263**	33	**259**	3.871	**29**
`Unsharp 3-1`	**33**	**2.066**	**4**	**33**	**2.066**	**4**
`Unsharp 3-2`	58	3.205	**6**	**51**	**3.162**	**6**

Table 5.3: Correlation between different metrics, #A the number of adders and $\#A_b$ the number of one-bit adders, and actual hardware cost

	LUTs	Delay	Power
#A	0.9586	0.443	0.9617
$\#A_b$	0.973	0.4156	0.9606

Table 5.2 demonstrates detailed hardware results for the 11 image-processing instances. In here, MCM-Bits always reduces the number of LUTs, though in rare cases the results coincide with the MCM-Adders solution, *e. g.*, for `Unsharp 3-1`. We observed in Table 5.1 that minimizing the one-bit adders can sometimes lead to a larger adder count, as for instance `Lowpass 15`, however, it significantly reduced the number of LUTs cost.

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 5.3 clearly demonstrates that the one-bit adders are most certainly in a linear relationship with the LUT count, with a correlation factor $r = 0.973$, which is stronger than $r = 0.9586$ 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 0.4156 and 0.9606, respectively. This indicates a less strong relationship with delay, which is also worse than for the number of adders. This analysis also confirms the detailed results in Table 5.2, where sporadic increase in delay and power for seemingly better MCM-Bits solutions can be observed. In general, the increase in power consumption 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 [Kum18].

5.4 Conclusion

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 number of adders in comparison to MCM-Adders solutions. The warm start comes from RPAG whose solution has been improved by MCM-Adders and, sometimes we did not reach the same number of adders with MCM-Bits within the available solving time. This can be dealt with by solving MCM-Adders first and using the optimal solution w. r. t. the number of adders as a bound for MCM-Bits. In this chapter, we have proposed for the first time an MILP-based approach for MCM-Bits. As expected, our model outperforms the higher-level model which solves MCM-Adders w. r. t. the one-bit adder count.

The proposed model is larger and harder to solve than our original model for MCM-Adders. We believe that our approach can be fine-tuned using techniques we presented in the previous chapter, in order to speed up the solving process and eventually to improve further the solutions we obtained. With extensive experiments we demonstrated significant gains in terms of cost estimation through the number of one-bit adders and in terms of hardware cost. Moreover, including other metrics as a second objective, we are confident that the increase of the delay can be mitigated. In any case, as we will see in Chapter 7, it is possible to reduce the delay by pipelining the adder graph.

Finally, we confirmed the interest of the one-bit adders low-level metric, which we used to define the MCM-Bits problem. This metric is well correlated with the LUT count and the power consumption. In future work it would be interesting to find a metric which is better correlated with the delay. However, these hardware results have been obtained out-of-context and delay and power consumption data needs to be confirmed with actual hardware experiments.

CHAPTER 6

Truncations

"If failure had no penalty success would not be a prize."

Terry Pratchett (1948-2015),
Sourcery (1988)

6.1 Introduction

Arithmetic operations usually increase the size of the data paths and this can rapidly overgrow the available resources. Giving up on the full-precision output is considered acceptable and, the output of different arithmetic operators is often rounded before being used as the input of the next operator. But this means that computing resources were used to compute information that would be discarded anyway. To avoid computing unnecessary bits, we can introduce intermediate truncations to save resources. Yet, because of truncations, errors crop up and need to be tamed below a user-given admissible bound.

An important aspect here is automation of the design choices, such as these intermediate truncations, since they are nontrivial and should be discharged to tools. This is the philosophy that we share with FloPoCo code generator [dDP11], which generates arithmetic operators computing just right [dDIM14, VIDDH19].

Incorporating truncations in adder graphs has been considered with heuristics and on a fixed topology [GDJ10, dDFKF19]. The first method [GDJ10] heuristically propagates truncations in the adder graph and experimentally verifies that the induced errors are acceptable. This does not permit to *a priori* ensure that a user-given error bound is met. To overcome this issue, an MILP-based model [dDFKF19], has been designed to truncate as much as possible and to verify an error bound on the outputs. The first obvious drawback of this approach is fixed adder graph. Moreover, we found a mistake in the error propagation model which can underestimate the effect of truncation-induced errors.

In contrast to previous works [GDJ10, dDFKF19], which focus on truncations of a *fixed*

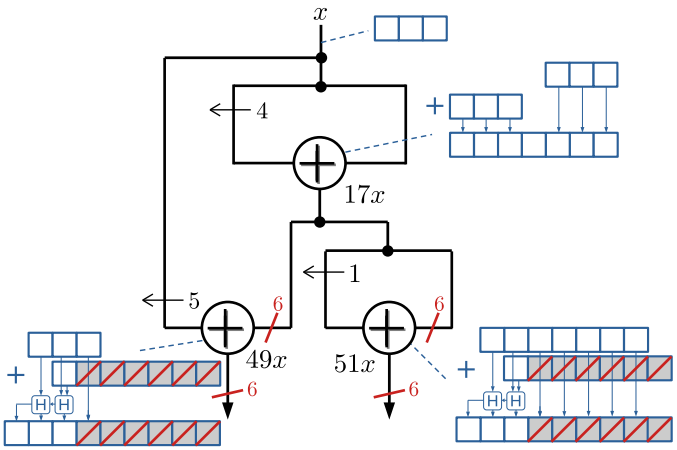


Figure 6.1: Computing $49x$ and $51x$ via the fundamental 17 leads to only 4 one-bit adders for a 3-bit input/output.

adder graph, we present an MILP which permits to solve the Truncated MCM (tMCM) problem in one step. This should permit to guide the topology, potentially leading to *different* adder graphs than the ones obtained solving the MCM-Adders or MCM-Bits problems as illustrated in the following example.

Example 8. Multiplication by 49 and 51 can be done as illustrated by the adder graph in Figure 5.1a with #A = 3, and then the adder graph can be truncated to ensure 3 output bits leading to a total of 6 one-bit adders. With the same requirements, a different topology requiring only 4 one-bit adders exists, see in Figure 6.1.

In Figure 6.1, we represent truncations with red bars with a number which indicates the number of truncated bits and *not* the remaining word length.

In this chapter, we complete our model which solves the MCM-Bits problem adding truncations to further reduce the hardware cost. From user-given error bounds on the outputs, which must be guaranteed by construction, our goal is to find an adder graph with internal truncations at the minimal cost in terms of the number of one-bit adders. For example, we want to find the adder graph presented in Figure 6.1 instead of relying on a two-step process which might miss optimal truncated adder graphs. We presented our first contribution to truncated adder graphs at the IEEE International Symposium on Circuits and Systems (ISCAS) [GVK22a] in 2022 and we published our completed work on tMCM in the IEEE Transactions on Circuits and Systems journal in 2023 [GV23b].

6.2 Modeling truncations and error propagation

The classic approach consists in finding a formal relation between output error and intermediate truncations, then, given an error budget, error requirements are back-propagated

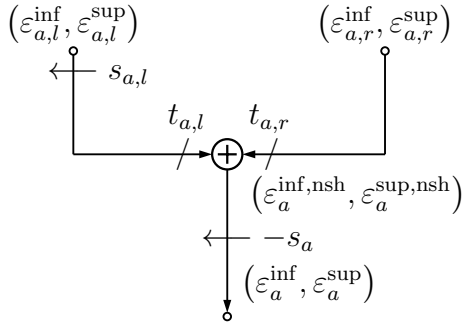


Figure 6.2: Model of an adder in presence of errors ε and truncations t.

to find sufficient parameters. In this thesis, we discharge this job to MILP. So, first we define error propagation rules, then we precisely define the effect of truncations over the errors. Finally, we model the gain of each truncation and maximize it by minimizing the hardware cost. This is done while respecting the output error constraints.

Error propagation in the state-of-the-art

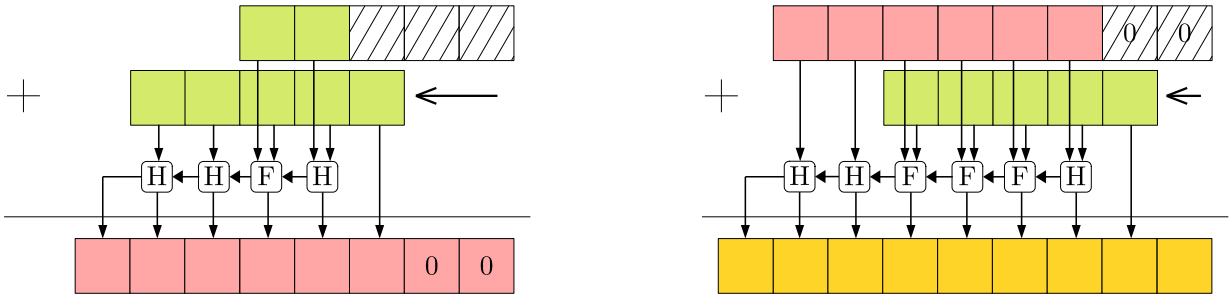
In the state-of-the-art model [dDFKF19], the error at each input of every adder is computed as the maximum between: (i) the truncation induced error and (ii) the error propagation. In our conference paper [GVK22a], we used this same error propagation rule. However, if we truncate a signal which is already tainted by an error, the errors *accumulate* and taking the maximum between both errors underestimates the actual error. In [dDFKF19, GVK22a], this can result in an output with an error that exceeds the user-given error bound. This error was difficult to detect since the error model that was used in both papers was based on absolute error which was a slight overestimation with respect to the nonsymmetric truncation error. This mitigated the underestimation presented above, partly hiding the problem.

In the following, we will not only correct the mistake but also consider tighter yet reliable bounds on the error. To do so, we will not consider absolute error bounds but asymmetric ones.

Error modeling

In Section 1.1.2, we presented the error propagation rule of different basic operators such as addition, negation or truncation. Our goal is to bring them all and to bind them in a single error propagation rule. Hence, for each adder, a, we start by defining integer variables for error bounds, $\varepsilon_a^{\text{inf}} \in \mathbb{N}$ and $\varepsilon_a^{\text{sup}} \in \mathbb{N}$. As illustrated in Figure 6.2, these errors come from the inputs of the adder and from the truncations. We denote $\varepsilon_{a,i}^{\text{inf}} \in \mathbb{N}$ and $\varepsilon_{a,i}^{\text{sup}} \in \mathbb{N}$, for each input $i \in \{l, r\}$, the errors of the inputs, and $\varepsilon_a^{\text{inf,nsh}} \in \mathbb{N}$ and $\varepsilon_a^{\text{sup,nsh}} \in \mathbb{N}$ the errors before the adder right-shift. The truncation-induced errors are encoded by the integer variables $\varepsilon_{t_{a,i}} \in \mathbb{N}$.

We combine the error propagation of each operator, as presented in Section 1.1.2, to propagate adder errors. In particular we take the sum of (i) the error induced by truncations



(a) Truncations might induce zeros in the output.

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

Figure 6.3: Counting one-bit adders and propagating errors in presence of truncations.

and (ii) the propagated error instead of the maximum as done in the state-of-the-art. This leads to the following propagation rule:

$$2^{s_a}\varepsilon_a^{\text{inf}} = \varepsilon_a^{\text{inf,nsh}} = \begin{cases} \left(2^{s_{a,l}}\varepsilon_{a,l}^{\text{inf}} + \varepsilon_{t_{a,l}}\right) + \left(\varepsilon_{a,r}^{\text{inf}} + \varepsilon_{t_{a,r}}\right), & \text{if } \sigma_{a,l} = \sigma_{a,r}, \\ \left(2^{s_{a,l}}\varepsilon_{a,l}^{\text{inf}} + \varepsilon_{t_{a,l}}\right) + \left(\varepsilon_{a,r}^{\text{sup}} + \varepsilon_{t_{a,r}}\right), & \text{if } \sigma_{a,r} = 1, \\ \left(2^{s_{a,l}}\varepsilon_{a,l}^{\text{sup}} + \varepsilon_{t_{a,l}}\right) + \left(\varepsilon_{a,r}^{\text{inf}} + \varepsilon_{t_{a,r}}\right), & \text{if } \sigma_{a,l} = 1, \end{cases} \tag{6.1}$$

$$2^{s_a}\varepsilon_a^{\text{sup}} = \varepsilon_a^{\text{sup,nsh}} = \begin{cases} 2^{s_{a,l}}\varepsilon_{a,l}^{\text{sup}} + \varepsilon_{a,r}^{\text{sup}}, & \text{if } \sigma_{a,l} = \sigma_{a,r}, \\ 2^{s_{a,l}}\varepsilon_{a,l}^{\text{sup}} + \varepsilon_{a,r}^{\text{inf}}, & \text{if } \sigma_{a,r} = 1, \\ 2^{s_{a,l}}\varepsilon_{a,l}^{\text{inf}} + \varepsilon_{a,r}^{\text{sup}}, & \text{if } \sigma_{a,l} = 1. \end{cases} \tag{6.2}$$

To include the above error propagation rule into an MILP-based model, we have to linearize the shifted input error as we linearized shifts over input fundamentals: using intermediate variables, $\varepsilon_{a,l}^{\text{inf,sh}}, \varepsilon_{a,l}^{\text{sup,sh}} \in \mathbb{N}$, and the binary variables $\Phi_{a,s}$ to obtain the constraints

$$\varepsilon_{a,l}^{\text{inf,sh}} = 2^s\varepsilon_{a,l}^{\text{inf}} \quad \text{if } \Phi_{a,s} = 1, \qquad \forall a \in [\![1; N]\!], s \in [\![0; w]\!], \tag{6.3}$$

$$\varepsilon_{a,l}^{\text{sup,sh}} = 2^s\varepsilon_{a,l}^{\text{sup}} \quad \text{if } \Phi_{a,s} = 1, \qquad \forall a \in [\![1; N]\!], s \in [\![0; w]\!]. \tag{6.4}$$

For uniform notations we also add the variables for the right input, $\varepsilon_{a,r}^{\text{inf,sh}} \in \mathbb{N}$ and $\varepsilon_{a,r}^{\text{sup,sh}} \in \mathbb{N}$, which are simply equal to $\varepsilon_{a,r}^{\text{inf}}$ and $\varepsilon_{a,r}^{\text{sup}}$, respectively

We also need to linearize the adder shift s_a with the variables $\Psi_{a,s}$ and to link the input errors with previous adders, using the binary variables $c_{a,i,k}$:

$$\varepsilon_a^{\text{inf,nsh}} = 2^s\varepsilon_a^{\text{inf}} \quad \text{if } \Psi_{a,s} = 1, \qquad \forall a \in [\![1; N]\!], s \in [\![0; w]\!], \tag{6.5}$$

$$\varepsilon_a^{\text{sup,nsh}} = 2^s\varepsilon_a^{\text{sup}} \quad \text{if } \Psi_{a,s} = 1, \qquad \forall a \in [\![1; N]\!], s \in [\![0; w]\!], \tag{6.6}$$

$$\varepsilon_{a,i}^{\text{inf}} = \varepsilon_k^{\text{inf}} \quad \text{if } c_{a,i,k} = 1, \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, k \in [\![0; a-1]\!], \tag{6.7}$$

$$\varepsilon_{a,i}^{\text{sup}} = \varepsilon_k^{\text{sup}} \quad \text{if } c_{a,i,k} = 1, \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, k \in [\![0; a-1]\!]. \tag{6.8}$$

This is similar as we did in previous chapters.

Finally, we need to compute the truncation errors $\varepsilon_{t_{a,i}}$. We recall that truncations to the t-th bit, $\diamond_t(\cdot)$, induce error, $\varepsilon_t \geq 0$, which is bounded by the quantity it removes:

$$\varepsilon_t \leq 2^t - 1. \tag{6.9}$$

This 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 Figure 6.3a, and keeping track of these trailing 0's, denoted z, allows for a tighter bound on the truncation errors:

$$\varepsilon_t \leq \max\left(2^t - 2^z, 0\right). \tag{6.10}$$

To be able to use this bound, we first need to introduce variables to encode the number of trailing zeros:

- $z_a \in \mathbb{N}$, for each adder, is the number of trailing zeros after the adder.
- $z_{a,i} \in \mathbb{N}$, for each adder, corresponds to the number of trailing zeros in each input and is linked with z_a's with the topology variables:

$$z_{a,i} = z_k \quad \text{if } c_{a,i,k} = 1, \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, k \in [\![0; a-1]\!]. \tag{6.11}$$

- $z^{\text{sh}}_{a,i} \in \mathbb{N}$, for each adder, encodes to the number of trailing zeros in each input after the shift:

$$z^{\text{sh}}_{a,l} = z_{a,l} + s_{a,l}, \qquad \forall a \in [\![1; N]\!], \tag{6.12}$$
$$z^{\text{sh}}_{a,r} = z_{a,r}, \qquad \forall a \in [\![1; N]\!]. \tag{6.13}$$

- $z^{\text{nsh}}_a \in \mathbb{N}$, for each adder, is the number of trailing zeros before the right-shift. This variable is equal to the minimum number of trailing zeros of both addition inputs:

$$z^{\text{nsh}}_a = \min\left(\max\left(z^{\text{sh}}_{a,l}, t_{a,l}\right), \max\left(z^{\text{sh}}_{a,r}, t_{a,r}\right)\right), \qquad \forall a \in [\![1; N]\!]. \tag{6.14}$$

  This constraints involve max operators which we know how to linearize using an additional binary variable. Then, z^{nsh}_a is linked with z_a using right-shifts:

$$z^{\text{nsh}}_a = z_a + s_a, \qquad \forall a \in [\![1; N]\!], s \in [\![0; w]\!]. \tag{6.15}$$

Now that we are able to propagate trailing 0's, we linearize both powers of two of (6.10). To do so, we add binary variables $t_{a,i,b} \in \{0, 1\}$ and $z_{a,i,b} \in \{0, 1\}$, in a similar manner as we added $\Phi_{a,s}$ in Chapter 4. These variables correspond, for each adder a and each bit b, to the position of the truncations and of trailing 0's and we enforce that one position is chosen, even if zero-valued:

$$\sum_{b=0}^{w+w_{in}} t_{a,i,b} = 1, \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, \tag{6.16}$$
$$\sum_{b=0}^{w+w_{in}} z_{a,i,b} = 1, \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}. \tag{6.17}$$

We link these variables with the truncations and trailing zeros:

$$\sum_{b=0}^{w+w_{in}} b \times t_{a,i,b} = t_{a,i}, \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, \tag{6.18}$$

$$\sum_{b=0}^{w+w_{in}} b \times z_{a,i,b} = z^{\text{sh}}_{a,i}, \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}. \tag{6.19}$$

Then, use intermediate variables, $\varepsilon^T_{a,i} \in \mathbb{N}$ and $\varepsilon^Z_{a,i} \in \mathbb{N}$, for each adder and input, to encode the truncation error and the part which is error-free due to trailing zeros:

$$\varepsilon^T_{a,i} = 2^{b-1}, \quad \text{if } t_{a,i,b} = 1, \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, b \in [\![1; w + w_{in}]\!], \tag{6.20}$$

$$\varepsilon^T_{a,i} = 0, \quad \text{if } t_{a,i,0} = 1, \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, \tag{6.21}$$

$$\varepsilon^Z_{a,i} = 2^{b-1}, \quad \text{if } z_{a,i,b} = 1, \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, b \in [\![1; w + w_{in}]\!], \tag{6.22}$$

$$\varepsilon^Z_{a,i} = 0, \quad \text{if } z_{a,i,0} = 1, \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}. \tag{6.23}$$

And we finally have

$$\varepsilon_{t_{a,i}} \geq \varepsilon^T_{a,i} - \varepsilon^Z_{a,i}, \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}. \tag{6.24}$$

In some cases where truncated bits are all zeros, as in Figure 6.3b, we can even have error-free truncations. The last step is to bound errors with the user-given acceptable output error $\overline{\varepsilon}$:

$$\varepsilon^{\text{inf}}_a \leq \overline{\varepsilon} \quad \text{and} \quad \varepsilon^{\text{sup}}_a \leq \overline{\varepsilon}. \tag{6.25}$$

Finally, an overflow could occur because of the propagated errors which can force the result of the adder to the next binade w. r. t. the one deduced by (5.3). To prevent this risk, we adjust the MSB taking the error into account:

$$2^{\text{msb}_a} - 1 \geq \overline{x} c_a + \varepsilon^{\text{sup}}_a, \tag{6.26}$$

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. This ends the error propagation analysis. However, we did not model the gain which can be obtained with truncations yet. Thus, there is no incentive for the solver to use truncations at all.

Reducing adder graph cost with truncations

When computing the one-bit adders, we used to motivate certain topologies by the gains due to shifts, see the constraints on g_a which intervenes in the one-bit adder count, (5.6), which we recall here:

$$\text{B}_a = \text{msb}_a + 1 - s_a - g_a - \psi_a. \tag{6.27}$$

However, we can also interpret the gain of certain number of one-bit adders induced by shifts, as truncations of s bits with a zero error. Indeed, shifting the input, induce implicit

trailing zeros which can be truncated at no cost. Hence, we can simply add a constraint enforcing the truncations to be at least as large as shifts:

$$t_{a,l} \geq s_{a,l}, \qquad \forall a \in [\![1; N]\!]. \tag{6.28}$$

This way, it is clearly not necessary to consider shifts anymore for the one-bit adder savings. Actually, we can even constrain truncations, on both inputs, to be larger than the number of shifted trailing 0's without any effect on the actual error propagation:

$$t_{a,i} \geq z^{\text{sh}}_{a,i}, \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}\,. \tag{6.29}$$

Then, while shifts were asymmetric, we need to take into account truncations of both inputs as truncation of the right input could also reduce the computation cost. For example, in addition to large shifts (or truncations) on the left input, which we represented in Figure 5.2b, we can also have large truncations on the right input such that output bits are computed without any one-bit adder. We illustrate this in Figure 6.4a and formalize it as follows:

$$g_a = \text{msb}_a \quad \text{if } t_{a,r} > \text{msb}_{a,l} \wedge \sigma_{a,l} = 0, \qquad \forall a \in [\![1; N]\!]. \tag{6.30}$$

In case of addition, the possible gain, which initially came only from the left shift, is induced by the maximum truncation between the left and right inputs:

$$g_a \leq \max\left(t_{a,l}, t_{a,r}\right) \quad \text{if } t_{a,r} \leq \text{msb}_{a,l} \wedge t_{a,l} \leq \text{msb}_{a,r}, \qquad \forall a \in [\![1; N]\!]. \tag{6.31}$$

This max operator can be linearized, and we do so in order to include (6.31) within our MILP-based model. For this purpose, we introduce binary variables $t^B_a \in \{0, 1\}$, for each adder, and integer variables, $t^{\max}_a \in \mathbb{N}$, to store the maximum value between $t_{a,l}$ and $t_{a,r}$. This is formalized as follows:

$$t^{\max}_a = t_{a,l} \quad \text{if } t^B_a = 1, \qquad \forall a \in [\![1; N]\!], \tag{6.32}$$

$$t^{\max}_a = t_{a,r} \quad \text{if } t^B_a = 0, \qquad \forall a \in [\![1; N]\!], \tag{6.33}$$

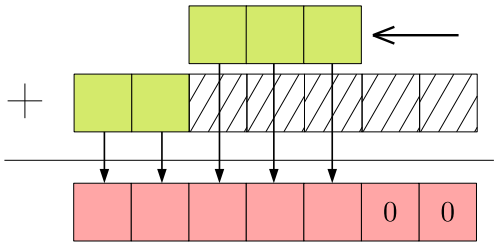
$$g_a \leq t^{\max}_a \quad \text{if } t_{a,r} \leq \text{msb}_{a,l} \wedge t_{a,l} \leq \text{msb}_{a,r}, \qquad \forall a \in [\![1; N]\!]. \tag{6.34}$$

Finally, in the case of a subtraction, the gain can only come from the input which is subtracted to the other, the so-called *subtrahend*. We illustrate in Figure 6.4b, a truncation on the positive input which has no impact over the computation cost. However, as illustrated in Figure 6.4c, truncation of the subtrahend permits to save one-bit adders. To benefit from this possible gain, we introduce additional constraints which bound the number of one-bit adders that can be saved in case of subtractions:

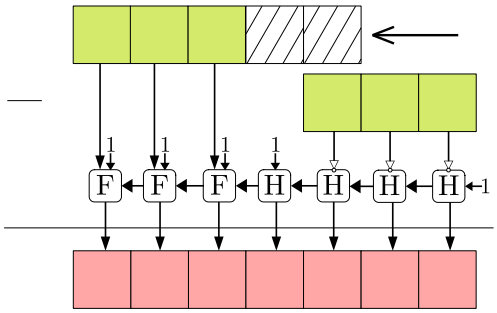
$$g_a \leq t_{a,l} \quad \text{if } \sigma_{a,l} = 1 \wedge \sigma_{a,r} = 0, \qquad \forall a \in [\![1; N]\!], \tag{6.35}$$

$$g_a \leq t_{a,r} \quad \text{if } \sigma_{a,l} = 0 \wedge \sigma_{a,r} = 1, \qquad \forall a \in [\![1; N]\!]. \tag{6.36}$$

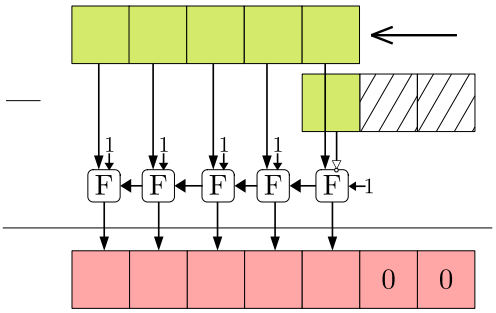
If we consider truncations of both inputs simultaneously, it is direct to see that the gain will only come from truncating the subtrahend and it was not necessary to truncate



(a) Truncation of the nonshifted input exceed the MSB of the shifted input.



(b) Truncation on the positive input in case of subtraction.



(c) Truncation on the negative input in case of subtraction.

Figure 6.4: Counting one-bit adders in case of truncations.

the other input. Hence, we can speed up the solving process by removing solutions with an equivalent cost. To do so, we add two sets of constraints which prevent unnecessary truncations in subtractions:

$$t_{a,l} = z^{\text{sh}}_{a,l} \quad \text{if } \sigma_{a,r} = 1, \qquad \forall a \in [\![1; N]\!], \tag{6.37}$$

$$t_{a,r} = z^{\text{sh}}_{a,r} \quad \text{if } \sigma_{a,l} = 1, \qquad \forall a \in [\![1; N]\!]. \tag{6.38}$$

In any case, the solver would not have increased the error for 0 gain but adding these constraints reduces the search space.

Remark. Our analysis on subtractions holds for the FPGA implementation. However, in Figure 6.4b, we see that multiple one-bit adders are actually half-adders. Hence, on ASICs, truncation on both side will modify the hardware cost.

Our analysis permits to precisely track the truncations-induced gain in terms of one-bit adders count. The complete linearization is available in Appendix A.3. Overall, this adds a significant number of variables and constraints to the model. They are very closely linked with the variables handling the topology and we believe that it should help the solver to quickly find interesting adder graph topologies for truncations.

6.3 Optimization and hardware results

The proposed MILP-based model including truncations is available within our open-source tool `jMCM`[1]. We solved our model using Gurobi 10.0.1 [Gur20] executed with 4 threads on an Intel® Xeon™ E7-8870 CPU at 2.10GHz. For each instance, the time limit is of 8 hours. Then, we generated VHDL from truncated adder graphs, with our open-source Julia package `AdderGraphs`[2], which we synthesized for FPGA using Vivado v2022.2 for the xc7k70tfbv484-3 Kintex 7 device.

With the following experiments we aim to study the impact of truncations on adder graphs. We will compare results obtained from the resolution of the MCM-Bits problem and the tMCM problem. For conciseness, we limit our experiments to 8-bit inputs but discuss truncations for multiple error bounds. We remind that full-precision is strictly the same as simply solving MCM-Bits. For each instance, we will perform two experiments: first setting an appropriate error-bound ensuring that we keep at least 50% of the full-precision, and a second ensuring at least 75% of the full-precision. Then, we can compare results with decreasing precision and we expect to observe a decrease of the necessary hardware usage with the reduction of the precision. The research questions we answer in this section are summarized as:

(RQ3) What is the impact of intermediate truncations on optimization results in terms of $\#A_b$?

(RQ4) How does this impact passes through to synthesized hardware?

We only provide detailed results for subset of the benchmarks, however we discuss statistics over all the 86 instances and complete results will be provided in appendices for the final version.

RQ3: Truncations permit to drastically reduce the one-bit adder count

In Figure 6.5, we illustrate the adder count reduction between full-precision, three quarter of the precision and half-precision for multiple instances. As expected, the one-bit adder count drops with the lower precision w. r. t. full-precision. On average, truncations to half-precision induce a one-bit adder count reduction of 36.7%. For example, we observe a 43% reduction of the number of one-bit adders for the instance `Unsharp 3-1`. However, we note a few cases where smaller precision did not lead to less one-bit adders. The model including truncations is larger than the full-precision one and we believe that with more available time the solver would have at least obtained the same solution.

Similarly to the case of MCM-Bits, truncated adder graphs, in some cases, require more adders. This 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. In Table 6.1, we provide detailed values and we remark that the different precision do not induce significant

[1] `https://github.com/remi-garcia/jMCM`
[2] `https://github.com/remi-garcia/AdderGraphs`

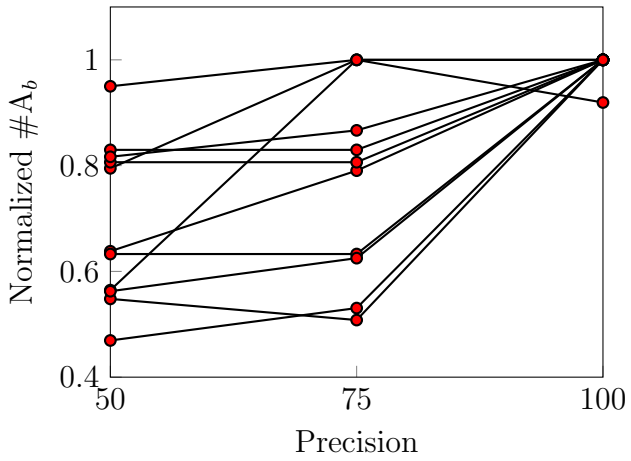


Figure 6.5: Comparing one-bit adder count w. r. t. precision. The lines correspond to the normalized number of one-bit adders for three precision: full-precision, 75% of the full-precision and half-precision.

Table 6.1: Number of adders, #A, and one-bit adders, #A$_b$, obtained with MCM-Bits (exact outputs) vs tMCM keeping 50% and 75% of output precision.

Benchmark	MCM-Bits		tMCM$_{75\%}$		tMCM$_{50\%}$	
	#A	#A$_b$	#A	#A$_b$	#A	#A$_b$
`Gaussian 3`	4	40	4	40	4	38
`Gaussian 5`	5	57	6	62	6	35
`Highpass 5`	4	39	4	39	4	31
`Highpass 9`	5	47	5	39	5	39
`Highpass 15`	12	105	12	83	12	67
`Laplacian 3`	3	31	3	25	3	25
`Lowpass 5`	6	60	6	52	7	49
`Lowpass 9`	13	128	13	81	13	81
`Lowpass 15`	27	250	27	127	27	137
`Unsharp 3-1`	4	32	4	20	4	18
`Unsharp 3-2`	5	49	5	26	6	23

Table 6.2: Hardware results for 8-bit inputs. The delay is given in ns and the power in mW.

Benchmark	MCM-Bits			$tMCM_{75\%}$			$tMCM_{50\%}$		
	LUTs	Delay	Power	LUTs	Delay	Power	LUTs	Delay	Power
`Gaussian 3`	41	7.305	105	33	7.142	94	26	7.578	76
`Gaussian 5`	58	8.937	169	52	9.024	157	37	8.136	113
`Highpass 5`	41	8.574	130	34	7.65	104	21	7.305	71
`Highpass 9`	48	7.675	161	44	7.606	150	33	8.036	132
`Highpass 15`	108	10.377	408	100	10.477	360	62	9.842	263
`Laplacian 3`	33	8.377	109	30	8.533	99	29	7.925	98
`Lowpass 5`	65	9.142	202	58	8.895	187	45	8.315	148
`Lowpass 9`	138	12.439	493	118	11.367	441	95	11.379	345
`Lowpass 15`	258	15.617	1153	253	14.776	1091	218	14.291	930
`Unsharp 3-1`	33	7.372	100	27	7.1	82	13	6.456	55
`Unsharp 3-2`	49	9.625	162	46	9.239	144	35	9.474	118

differences in adder graphs in terms of adder count: compared with MCM-Bits, keeping only half of the output bits led to solutions with more adders in 13 cases.

Although the additional variables and constraints we incorporated to the model have an impact on solving times, the solver could not prove optimality for only 56 instances. As a reminder, when solving the simpler model for MCM-Bits, the solver already timed out for 47 instances.

RQ4: Reducing synthesized hardware cost with truncations

In Chapter 5, we observed a strong correlation between the number of one-bit adders and both the number of LUTs and the power consumption. Table 6.2 confirms that tMCM provides major hardware cost reductions w. r. t. full precision: for 8-bit inputs and half-precision outputs, the number of LUTs is decreased by 25.5%, the delay by 5.0% and the power by 22.0%. Compared to base model for the MCM-Adders problem, the gains are even greater: in [GV23b], we reported up to 61% reduction in LUTs. However, in one case the truncations led to a delay increased by 76%. 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.

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. Furthermore, generic static analysis tools can be used to compute such bounds for numerical programs, such as [GP11, DIN+18], or they can be tightened using domain specific properties, such as has been done for digital filters [VIDDH19].

6.4 Conclusion

We solve, for the first time, the MCM problem taking truncations and output error bound into account. In this chapter, we formalized the problem of searching for truncated adder graphs as the tMCM problem, and we proposed an analysis of the error propagation within adder graphs. This way, by construction, we ensure that the error budget is automatically shared among the adders to maximize the gain.

We propagate rounding errors in a tight manner, using a nonsymmetric error model for truncations. Then, by presenting the similarities between truncations and shifts when counting one-bit adders, we simplified the model. With our experiments, including synthesis results, we demonstrate that it is efficient to automate the tMCM design-space exploration liberating the designers to study high-level questions. As expected, we confirm that, the less output bits are necessary, the more internal truncations can be used to obtain a significant hardware cost reduction.

This chapter is the result of the balanced combination of computer arithmetic with hardware design and operations research, where we demonstrated the versatility of the MILP approach for diving into *arithmetic* low-level metric.

CHAPTER 7

Pipelined adder graphs

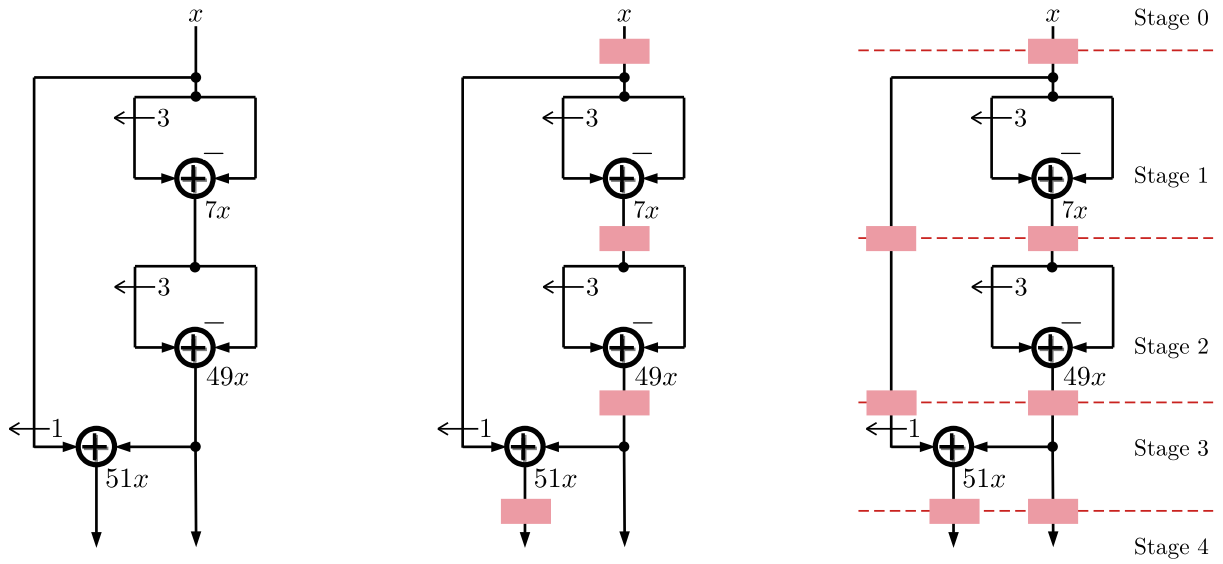
> "Everything makes sense a bit at a time. But when you try to think of it all at once, it comes out wrong."
>
> Terry Pratchett (1948-2015),
> *Only You Can Save Mankind (1992)*

7.1 Introduction

In the previous chapters we have focused on resource consumption. However the hardware cost is actually manifold and it includes delay, throughput or power consumption. Using registers is a common practice for increasing the throughput and even for decreasing the power consumption, as it splits the glitch path. In this chapter, we will consider pipelined adder graphs in order to prioritize the throughput [MBCCD06, KZ11, KZFC12, Kum16] and, with the constraint of pipelining, we will minimize resource usage.

We assume that all outputs should be available at the same clock cycle and this property is guaranteed in adder graphs with registers before each stage. Pipelining of a fixed adder graph can be obtained by performing cut-set retiming (CSR) [Par99, KZ11]. For example, to pipeline the stages of the adder graph represented in Figure 7.1a, one must put a register at the input, one register after each adder, see Figure 7.1b, and finally a few additional registers to ensure the synchronization of the signals, as shown in Figure 7.1c.

The fixed adder graph might not be "pipeline friendly" and it has been shown [KZFC12] that a better solution consists in searching directly for pipelined adder graphs. The idea is to search for the best multiplierless topologies minimizing the resources as a whole. To address this issue, there exists a heuristic [KZFC12]. An optimal method, in which the whole design space is enumerated, was proposed by Kumm [Kum16]. The main disadvantage of this approach is its lack of scalability since for larger and interesting adder graphs the method cannot be applied.



(a) Adder graph to compute $49x$ and $51x$.

(b) Adding registers.

(c) Synchronizing the signals.

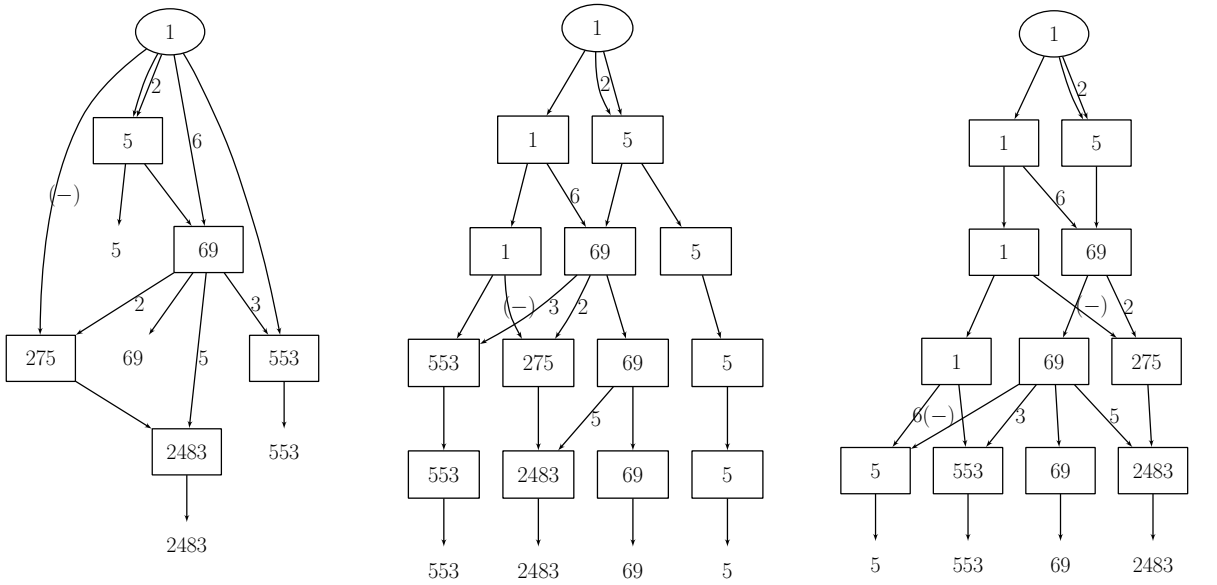
Figure 7.1: Pipelining a fixed adder graph.

Our goal is to avoid a heavy precomputation and similarly to other models the register cost directly within the model and let the solver do its job. Our second objective is to also optimize w. r. t. lower-level metrics which are closer to LUTs and flip-flops, namely the one-bit adders and one-bit registers, respectively.

In the following, we will refer to the problem of searching for an optimal pipelined adder graph w. r. t. the number of high-level elements (adders and registers) as PMCM-Adders. Despite the presence of registers, which we want to minimize, we keep "-Adders" for simplicity and uniformity with MCM. Naturally, when we will discuss the optimization of pipelined adder graph with respect to one-bit adders and registers, we will refer to this problem as PMCM-Bits.

In this chapter, we will illustrate our examples with large adder graphs and our original representation becomes confusing. Hence, we adopt the graphical representation of Figure 7.2. The rectangle nodes correspond to adders if there are two incoming arrows, or to registers if there is only one input. Labels within the nodes are the fundamentals. Numbers alongside the edges correspond to shifts and a minus indicates the negation of the input. For example, in Figure 7.2b, the node with the fundamental 275 corresponds to an addition of the input $69x$ shifted by 2 bits and from which the register, containing the value $1x$, is subtracted.

This work is published at the International Conference on Field-Programmable Logic and Applications (FPL) in September 2023 [GV23a].



(a) Adder graph.

(b) Pipelined adder graph.

(c) Pipelined adder graph with a same fundamental twice.

Figure 7.2: (Pipelined) adder graphs which compute $5x$, $69x$, $553x$ and $2483x$.

7.2 Pipelining in a high-level model

The solution of the MCM-Adders problem is guided by the number of adders, and naturally, each fundamental should be unique in the adder graph. However, when incorporating the pipelining, it is the length of the connection between an adder and the maximum stage it interacts with, which is guiding the topology. Hence, it might sometimes be beneficial to compute the same fundamental twice, as it could save multiple stages of registers. For example, consider the adder graph in Figure 7.2a, computing $5x$, $69x$, $553x$ and $2483x$. To pipeline this adder graph without refactoring the topology, we need 8 registers in addition to the 5 adders, as illustrated in Figure 7.2b, thus, the total cost is 13. However, there exists an adder graph, represented in Figure 7.2c, whose total cost is only 12, with 6 registers and 6 adders. Its particularity is that it computes the same fundamental (5) twice: on Stage 1, and then on Stage 4. This permits to avoid propagating the fundamental 5 through all the stages and to recompute it when needed, which reduced the overall cost.

Now that we have presented a few specifics of pipelined adder graphs, we can dive in the modeling of registers. Actually, our idea is to avoid an explicit model for registers, as entities similar to adders, in contrast to Kumm [Kum16]. Instead, we assume that registers are always associated with adders, which is often the case, and we build upon the existing topology part of our models. Hence, we model the pipelining stages and we will deduce the number of registers afterwards. To do so, we consider integer variables for each adder, which store the stage when the adder is computed, $p_a \in \mathbb{N}$, and the maximum stage the adder interacts with as a direct input, $\overline{p}_a \in \mathbb{N}$.

Example 9. In Figure 7.2c, the initial input stage is $p_0 = 0$, by definition, and the maximum stage it interacts with is $\overline{s}_0 = 4$ (to compute the fundamental 553). Then, its register cost is $r_0 = 4 - 0$, including the register at the adder stage.

We have defined variables p_a and $\overline{p}_a$ but we still need to add constraints in order to assign them the right value. To do so, we remark that the adder depth and pipeline stage are actually synonyms. Hence, we first compute p_a by introducing variables $p_{a,i} \in \mathbb{N}$ and $p_a^b \in \{0, 1\}$, exactly as we did for the adder depth in Chapter 4:

$$p_a \geq p_{a,l} + 1, \qquad \forall a \in [\![1; \mathrm{N}]\!], \tag{7.1}$$
$$p_a \geq p_{a,r} + 1, \qquad \forall a \in [\![1; \mathrm{N}]\!], \tag{7.2}$$
$$p_a \leq p_{a,l} + 1 + \mathrm{N}p_a^b, \qquad \forall a \in [\![1; \mathrm{N}]\!], \tag{7.3}$$
$$p_a \leq p_{a,r} + 1 + \mathrm{N} \times \left(1 - p_a^b\right), \qquad \forall a \in [\![1; \mathrm{N}]\!]. \tag{7.4}$$

Then, we simply link the pipeline stage of each input with the pipeline stage of the original adder:

$$p_{a,i} = p_k \quad \text{if } c_{a,i,k} = 1, \qquad \forall a \in [\![1; \mathrm{N}]\!], i \in \{l, r\}, k \in [\![0; a-1]\!]. \tag{7.5}$$

Finally, we also encode the output pipeline stage, $p_{\max} \in \mathbb{N}$, as the maximum of the pipeline stages plus one:

$$p_{\max} \geq p_a + 1, \qquad \forall a \in [\![1; \mathrm{N}]\!]. \tag{7.6}$$

We are now able to deduce the pipeline stage of each adder as we did for the adder depth. Then, we also need to extract the information of the maximum stage the adder interact with as a direct input, which we encode in $\overline{p}_a$. To do so, we add binary variables $\overline{p}_{a,s} \{0, 1\}$ which, for each adder a and stage p, encodes that the adder a intervenes as an input for the stage p. Then, we enforce that $\overline{p}_{a,p} = 1$ if adder a is used as an input for an adder at stage p:

$$p_k \leq N \times \overline{p}_{a,p} + p - 1 \quad \text{if } c_{k,i,a} = 1, \qquad \forall p \in [\![1; N]\!], a \in [\![0; N-1]\!], \forall k \in [\![a+1; N]\!], i \in \{l, r\}. \tag{7.7}$$

We can notice that we have voluntarily ordered indices of $c_{a,i,k}$ as $c_{k,i,a}$ instead. Originally, we use the variable $c_{a,i,k}$ to link adders with their inputs, here we alternatively use it to link adders with their outputs.

Actually, we also need to consider adders which are used for outputs of the pipelined adder graph:

$$p_{\max} \leq (N+1) \times \overline{p}_{a,p} + p - 1 \quad \text{if } o_{a,j} = 1, \quad \forall s \in [\![1; N+1]\!], a \in [\![0; N]\!], j \in [\![1; |C|]\!], \tag{7.8}$$

as they intervene at the stage $p_{\max}$. Finally, we add the constraints

$$\overline{p}_a \geq p \times \overline{p}_{a,p}, \qquad \forall p \in [\![1; N+1]\!], a \in [\![0; N-1]\!], \tag{7.9}$$

in order to compute the maximum stage with which the adder a interacts with. Using the constraints above, we are able to integrate into the MILP model additional information which we use to infer the register cost, $r_a \in \mathbb{N}$, for each adder:

$$r_a = \overline{p}_a - p_a, \qquad \forall a \in [\![0; N]\!]. \tag{7.10}$$

When implementing on FPGAs, we will consider that registers that follow adders basically come for free as they actually are on the same basic logic element (BLE). To look at it in another way, adding registers is necessary to have a pipelined adder graph, hence their cost is mandatory and it is the adders that basically come for free. This naturally leads to the following objective function which minimizes the number of registers of the pipelined adder graph:

$$\min \sum_{a=0}^{\mathrm{N}} r_a. \tag{7.11}$$

With this objective function we directly search for the best pipelined adder graphs w. r. t. the number of registers. Ultimately, we expect that this high-level cost will positively impact to hardware resource and power consumption. Naturally, it would be desirable to directly optimize w. r. t. lower-level metrics.

7.2.1 Pipelining in a low-level model

In Chapter 5, we were able to model one-bit adders in adder graphs and we obtained substantial gain optimizing w. r. t. this metric in comparison to the number of adders. Intuitively, it seems to be reasonable to split registers into one-bit registers in pipelined adder graphs.

If we zoom into registers, we find that they simply consist of a number of one-bit registers equal to the data path length, storing each bit into a single one-bit registers. Hence, the number of one-bit registers, B_a^r, following an adder a is exactly

$$B_a^r = \mathrm{msb}_a + 1. \tag{7.12}$$

Thus, the cost in terms of one-bit registers associated to each adder a, which we denote C_a^r, is

$$C_a^r = B_a^r \times r_a. \tag{7.13}$$

We note that the number of one-bit registers per adder, B_a^r, is at least as large as the number of one-bit adders, B_a, defined in (5.6). Indeed, the number of one-bit adders is bounded by the data path, which corresponds to the number of one-bit registers.

Similarly to the pair adder/register, we will consider that one-bit adders come for free with the one-bit registers. This greatly simplifies the problem we solve, as the parts of the

model used to compute g_a or ψ_a are not necessary anymore. We only need to keep the set of variables and constraints used to determine the MSB of each adder. However, due to the product in (7.13) which we have to linearize, the model still remains somewhat complex.

The number of registers per adder, r_a, is bounded by the maximum number of adders N. This bound on r_a is small enough to include in the model N^2 binary variables $r^B_{a,b} \in \{0,1\}$ such that $r^B_{a,b} = 1$ if $r_a = b$. This can be encoded with the following constraints:

$$\sum_{b=1}^{N} b \times r^B_{a,b} = r_a, \qquad \forall a \in [\![0; N]\!], \tag{7.14}$$

$$\sum_{b=1}^{N} r^B_{a,b} = 1, \qquad \forall a \in [\![0; N]\!]. \tag{7.15}$$

Then, the product can be linearized with

$$C^r_a = b \times B^r_a \quad \text{ if } r^B_{a,b} = 1, \qquad \forall a \in [\![0; N]\!], b \in [\![1; N]\!], \tag{7.16}$$

and we can replace the objective function by

$$\min \sum_{a=0}^{\mathrm{N}} C^r_a, \tag{7.17}$$

ensuring the minimization of the number of one-bit registers.

Overall, it is interesting to remark that this new model combines concepts from the models for MCM-Bits and PMCM-Adders, but parts of the MCM-Bits model are not necessary anymore, resulting in an alleviated model.

7.2.2 Critical path

In Chapter 4, we presented a way to incorporate the adder depth in the MCM model. The goal was to minimize it in order to reduce the adder graph delay. In the case of pipelined adder graphs, the delay is not the main priority and we are more interested in the throughput. To maximize it, we need to minimize the critical path, which corresponds to the longest path between pairs of registers.

Apart from the routing, we suppose at most one adder between each pair of registers. Hence, the critical path is not relevant if we do not explore lower-level metrics. Actually, the critical path of the pipelined adder graph is determined by the adder with the largest number of one-bit adders. Using MILP, it is straightforward to store this value into an integer variable, $B_{\max} \in \mathbb{N}$, as follows:

$$B_{\max} \geq \mathrm{B}_a, \qquad \forall a \in [\![1; N_A]\!]. \tag{7.18}$$

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

$$\min M \times \sum_{a=1}^{N_A} C^r_a + B_{\max}, \tag{7.19}$$

where the integer parameter M is fixed to a known upper bound of $B_{\max}$ such as the maximum data word length in the adder graph, in order to ensure that the number of one-bit registers is minimized first, and $B_{\max}$ second.

The incorporation of the critical path within the MILP model requires computing the number of one-bit adders again. Because of this extra cost, minimizing the critical path as we did might not pay off. We will investigate a cost/benefit analysis with our experiments.

7.3 Experiment results

The proposed MILP-based models are available within our open-source tool `jMCM`[1]. In our experiments, we fix the input word length at 8 bits as it is a logical choice for the image-processing benchmarks. We solved our model using Gurobi 10.0.1 [Gur20] executed with 4 threads on an Intel® Xeon™ E7-8870 CPU at 2.10GHz. For each of the 86 instances, the time limit is of 30 minutes. Then, we generated VHDL from pipelined adder graphs, with our open-source Julia package `AdderGraphs`[2], which we then synthesized for FPGA using Vivado v2018.2 for the xc7v585tffg1761-3 Kintex 7 device. We provide detailed results only for a subset of the benchmarks while statistics integrate all 86 benchmarks.

7.3.1 Optimization results

PMCM-Adders

There is a trade-off between the number of adders and the number of registers. Consequently, for PMCM we discourage using tight bounds on the initial bound on the maximum number of adders, which is otherwise advised for MCM-Adders. For instance, in our experiments we arbitrarily increased by two every (initially tight) bound on the number of adders for each benchmark. This looser upper bound permitted to obtain lower cost adder graphs for 11 instances out of 86.

In Table 7.1, we compare the state-of-the-art RPAG results with our optimal approach for the PMCM-Adders. With our tool, we can observe a 17.57% reduction of the number of adders plus registers, on average and across all 86 benchmarks, compared to RPAG. In details, for the 11 instances that require more adders than RPAG, it permitted to reduce the number of registers by 49.26%. For example, we see that, for `Lowpass 9`, RPAG found a solution with 12 adders and 7 registers leading to a total cost of 19, while our model for PMCM-Adders led to a solution with an additional adder but two less registers for a total cost of 18.

[1] `https://github.com/remi-garcia/jMCM`
[2] `https://github.com/remi-garcia/AdderGraphs`

Table 7.1: Comparison of our results to RPAG [KZFC12]. Here #A, #A_b, #R and #R_b denote the number of adders, one-bit adders, registers, and one-bit registers, respectively. The cost is $C = \#A + \#R$ and $C_b = \#A_b + \#R_b$. Numbers in bold indicate best results.

Benchmark	RPAG						PMCM-Adders						PMCM-Bits					
	#A	#R	C	#A_b	#R_b	C_b	#A	#R	C	#A_b	#R_b	C_b	#A	#R	C	#A_b	#R_b	C_b
`Gaussian 3`	4	2	6	43	25	68	4	1	**5**	43	15	**58**	4	1	**5**	43	15	**58**
`Gaussian 5`	6	2	**8**	69	34	**103**	6	2	**8**	69	34	**103**	6	2	**8**	69	34	**103**
`Highpass 5`	4	4	8	42	45	87	4	2	**6**	42	24	**66**	4	2	**6**	42	24	**66**
`Highpass 9`	5	4	9	50	49	99	5	2	**7**	50	27	**77**	5	2	**7**	50	27	**77**
`Highpass 15`	12	7	19	122	104	226	12	3	**15**	122	56	**178**	12	3	**15**	122	56	**178**
`Laplacian 3`	4	2	6	45	24	69	4	1	**5**	45	16	**61**	5	0	**5**	58	3	**61**
`Lowpass 5`	7	2	9	75	40	115	7	1	**8**	68	30	**98**	8	1	9	85	25	110
`Lowpass 9`	13	6	19	152	102	254	15	3	18	167	68	235	13	4	**17**	145	81	**226**
`Lowpass 15`	27	26	53	306	434	740	27	9	**36**	304	191	**495**	27	10	37	306	201	507
`Unsharp 3-1`	4	2	6	32	34	66	4	1	**5**	38	18	**56**	4	1	**5**	35	21	**56**
`Unsharp 3-2`	6	3	9	62	50	112	5	2	**7**	60	33	93	5	2	**7**	59	32	**91**
Total:	92	60	152	998	941	1939	93	27	**121**	1025	508	**1533**	93	29	123	1029	517	1546

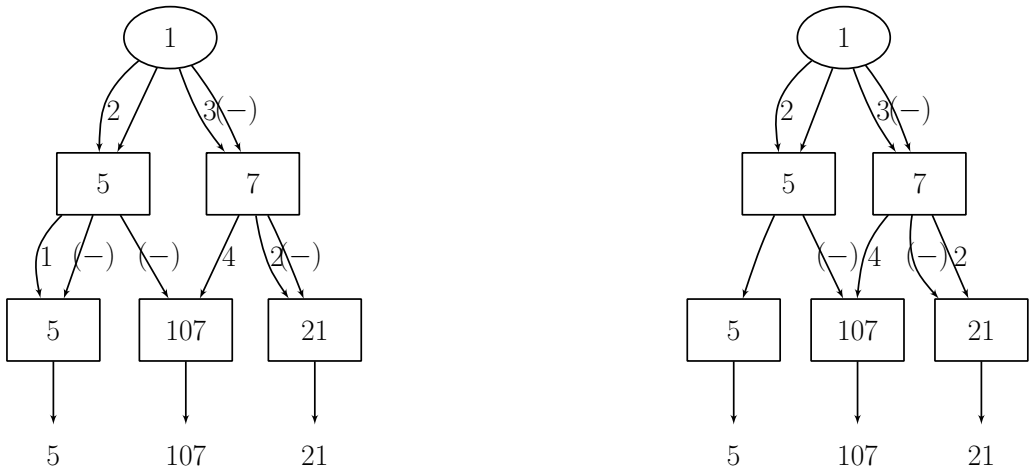
Table 7.2: A posteriori pipelining of MCM-Bits solution vs one-step optimization with PMCM-Bits. $C_b = \#A_b + \#R_b$. Best results are in bold.

Benchmark	MCM-B.+Pipelining			PMCM-Bits		
	$\#A_b$	$\#R_b$	C_b	$\#A_b$	$\#R_b$	C_b
`Gaussian 3`	40	28	68	43	15	**58**
`Gaussian 5`	57	55	**112**	84	32	116
`Highpass 5`	39	77	116	42	24	**66**
`Highpass 9`	47	41	88	50	27	**77**
`Highpass 15`	105	230	335	122	56	**178**
`Laplacian 3`	31	59	90	58	3	**61**
`Lowpass 5`	60	89	149	85	25	**110**
`Lowpass 9`	137	218	355	145	81	**226**
`Lowpass 15`	250	688	938	306	201	**507**
`Unsharp 3-1`	32	34	66	35	21	**56**
`Unsharp 3-2`	49	85	134	59	32	**91**
Total:	847	1604	2451	1029	517	**1546**

PMCM-Bits

The state-of-the-art RPAG heuristic demonstrated [KZFC12] that solving the PMCM-Adders as one step is better than *a posteriori* pipelining an optimal adder graph. With our work, we confirm that this holds for the finer-grained metric of counting one-bit adders and one-bit registers, as illustrated in Table 7.2. By solving the PMCM-Bits problem, instead of pipelining *a posteriori* the MCM-Bits solution, we achieve a 33.15% cost reduction in terms of number of one-bit registers, on average. For example, the solution with *a posteriori* pipelining of `Lowpass 15` required more than 600 one-bit registers while the pipelined adder graph obtained in one step only requires 201 one-bit registers. The original adder graph was not pipelining-friendly and the total cost was drastically reduced.

However, the PMCM-Bits solutions demonstrate no improvement over the PMCM-Adders ones. Actually, on average, the cost of the adder graphs we obtained increased by 1.35%. We observed that solving PMCM-Bits is more challenging for the solver, which translates into obtaining, in rare cases, worse solutions than PMCM-Adders as illustrated in Table 7.1. Our hypothesis is that, for hard instances, solving the PMCM-Adders problem leads to an adder and register reduction that is not reached by solving the PMCM-Bits within the available time. This might be explained in the following way: the interest of MCM-Bits is to count the number of one-bit adders and look for corner cases that permit to avoid using them, but it is somewhat leveled by the register bit count, which always depends on the word length of the adder output. Though, in Table 7.1, we did observe one case, the benchmark `Unsharp 3-2`, where PMCM-Bits found a better, w. r. t. to high-level metric, adder graph than PMCM-Adders.



(a) Adder is used instead of a register

(b) Register can be used

Figure 7.3: Postprocessing replaces unnecessary adders with same-cost registers

Post-processing

We observed that sometimes the solver provides solutions in which adders could have been replaced by registers such as in Figure 7.3a. For our model this has the same cost as the register used in Figure 7.3b, since we considered that the adders come for free. Hence, the solver will not differentiate between these solutions. Hardware costs, such as power consumption, would however differ. These unnecessary adders can be replaced by registers with a trivial post-processing.

7.3.2 Hardware results

In Table 7.3, we compare the critical path delay of five different approaches: RPAG, MCM-Adders (no pipelining), PMCM-Adders, PMCM-Bits and PMCM-Bits$_{\text{CP}}$. As expected, pipelining leads to smaller critical path which permits to increase the frequency and the throughput of the final hardware. Interestingly, minimizing the critical path as a second objective was not as efficient as we expected: on average, the critical path delay was reduced by just 0.37% and simply solving PMCM-Bits was even better on multiple instances, in the allocated solving time. For example, the critical path obtained with PMCM-Bits for the instance `Lowpass5` was smaller than the one obtained with PMCM-B$_{\text{CP}}$. The main objective of PMCM-B$_{\text{CP}}$ was to minimize resources, and the critical path was secondary. If throughput needs to be pushed further, we think that modifying the cost function to allow for a small increase in resource usage, in favor of a smaller critical path, could improve the delay. This change would be straightforward to apply as we provided models and not a dedicated algorithm.

The Table 7.1 and 7.2 demonstrated the advantages of the proposed approach over the state-of-the-art heuristic RPAG on the proxy variables (one-bit) adders and registers. In previous chapters, we demonstrated the interest of counting adders and one-bit adders.

Table 7.3: Critical path delay (ns) for each method. Best results are in bold.

Benchmark	MCM-A.	RPAG	PMCM-A.	PMCM-B.	PMCM-B_{CP}
`Gaussian 3`	2.643	**1.452**	1.494	1.494	1.494
`Gaussian 5`	4.608	1.792	**1.772**	**1.772**	**1.772**
`Highpass 5`	2.629	1.887	**1.566**	**1.566**	**1.566**
`Highpass 9`	2.731	**1.698**	1.712	1.712	1.712
`Highpass 15`	5.142	**2.791**	3.423	3.423	3.423
`Laplacian 3`	3.926	**1.497**	1.502	1.502	1.502
`Lowpass 5`	3.687	1.931	1.705	**1.662**	1.824
`Lowpass 9`	7.458	**3.079**	2.875	3.304	3.099
`Lowpass 15`	13.147	5.897	7.586	**5.178**	5.415
`Unsharp 3-1`	2.535	1.641	1.630	1.630	**1.547**
`Unsharp 3-2`	4.020	1.547	1.623	**1.409**	**1.409**

We are optimistic that registers and one-bit registers are useful proxy variables too.

In Table 7.4, we recap the synthesis results for the RPAG, the two-step process consisting in solving MCM-Adders (respectively, MCM-Bits) first and pipelining second, and our optimal approach. Interestingly, the RPAG heuristic is not better than the two-step approaches, but this can be explained by significant improvements of the first step (MCM-Adders and MCM-Bits) in comparison to previous solutions. We see the clear advantage of our optimal solutions with respect to the number of FFs, which are almost everywhere significantly reduced by both PMCM-Adders and PMCM-Bits. For example, for the instance `Highpass 5`, there are only 73 FFs in the solutions obtained with PMCM-Adders and PMCM-Bits while other methods led to more than 92 FFs and up to 118. However, synthesis results confirm our observation that PMCM-Bits in general has shown no advantage to PMCM-Adders in the allocated time. Finally, the gains in results of number of LUTs and power are more erratic and the one-step process does not lead to a clear cost reduction compared to *a posteriori* pipelining. This inconclusiveness is nevertheless not discouraging since all these values are close.

7.4 Conclusion

With this work, we solve the PMCM problem optimally using the MILP approach. Our results, evaluated on a large set of benchmarks, demonstrate the superiority of *optimal* approach compared to the state-of-the-art heuristic RPAG. We also compared our new model with *a posteriori* pipelined solutions obtained solving our model for MCM-Adders. This way, we demonstrated that our one-step process permits to find better solutions than the ones obtained with the two-step process.

In contrast to previous approaches, we also incorporate the pipelining into a model that targets a *low-level metric*, based on counting one-bit adders and registers. Our experiments confirm that optimizing in one step, *i. e.* PMCM-Bits, is better than *a posteriori* pipelining

Table 7.4: Comparison of adder graphs obtained with different high- and low-level models, including pipelining or applied on a fixed adder graph. Registers are also added on the I/Os. We report the number of LUTs and FF and the power consumption (mW) after place and route. Numbers in bold indicate best results. GM stands for geometric mean.

Bench				Two-step vs one-step counting adders						Two-step vs one-step counting one-bit adders					
	RPAG			MCM-Adders + P.			PMCM-Adders			MCM-Bits + P.			PMCM-Bits		
	#LUTs	#FF	power	#LUTs	#FF	power	#LUTs	#FF	power	#LUTs	#FF	power	#LUTs	#FF	power
G.3	43	75	116	42	76	114	42	**65**	114	**39**	74	**112**	42	**65**	114
G.5	69	109	**170**	58	148	174	68	**108**	**170**	**57**	138	180	68	**108**	**170**
HP5	42	93	127	41	92	126	41	**73**	**125**	**37**	118	134	41	**73**	**125**
HP9	50	105	163	48	104	**156**	48	**84**	**156**	**44**	94	161	48	**84**	**156**
HP15	121	226	408	119	225	400	118	**184**	**398**	**100**	315	413	118	**184**	**398**
L.3	45	75	121	34	95	122	44	**68**	126	**31**	93	**118**	44	**68**	126
LP5	76	115	213	66	157	218	77	111	221	**59**	142	212	70	**104**	**207**
LP9	149	250	**483**	143	354	510	179	**229**	510	**139**	396	545	153	**229**	507
LP15	303	691	1181	302	690	**1175**	311	**487**	1181	330	791	1219	**296**	498	1204
U.3-1	32	73	109	**30**	72	**103**	35	**63**	106	**30**	72	**103**	35	**63**	106
U.3-2	62	117	179	59	**97**	**174**	116	126	197	**51**	139	178	61	116	178
GM:	71	133	217	66	146	**215**	77	117	218	**62**	158	219	70	**116**	**215**

of MCM-Bits solutions, following the same trend as with high-level metrics. However, in the allocated time, PMCM-Bits design exploration is rarely revealing better results than the high-level metric-based PMCM-Adders. Nevertheless, this low-level metric permitted to include the critical path minimization within the MILP-based approach.

With this chapter, we demonstrate the versatility and power of the MILP-based approach: it is possible to directly optimize for pipelined adder graph using similar methods as for adder graph and truncated adder graphs. This new design automation method, dedicated to pipelined circuits, permits to alleviate time and efforts embedded system designers put into pipelining.

Part III

Digital Filters

CHAPTER 8

IIR Filters

> "Vimes found it better to look to Authority for orders and then filter those orders through a fine mesh of common sense, adding a generous scoop of creative misunderstanding."
>
> Terry Pratchett (1948-2015),
> *Night Watch (2002)*

8.1 Introduction

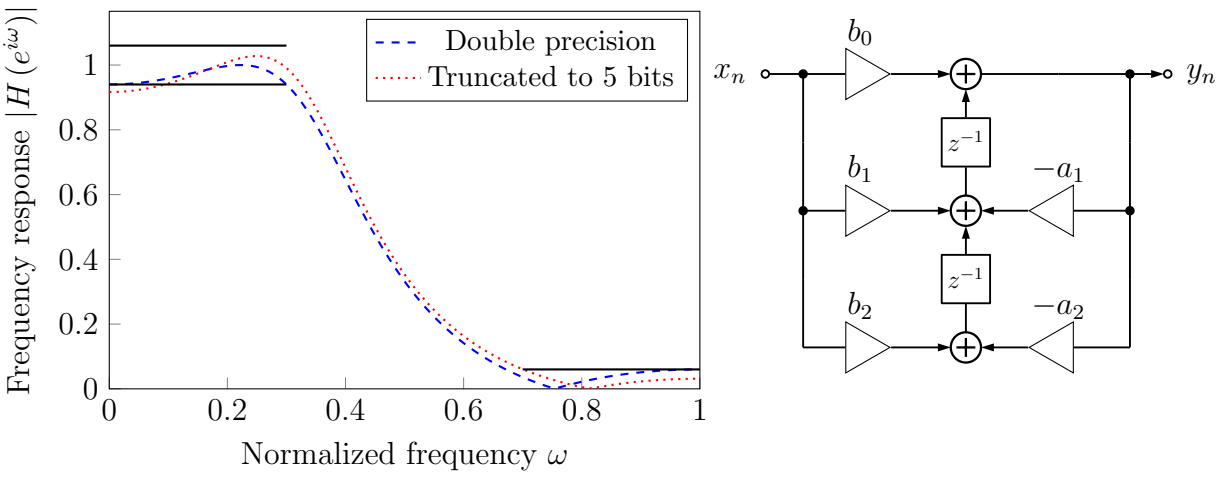
Digital filters are essential components of modern technology, from medical equipment to scientific instruments. Finite impulse response (FIR) and infinite impulse response (IIR) filters can be relatively easily designed in software, but hardware implementation presents a new layer of difficulties to overcome. The filter implementation, from the specifications, follows three separate steps:

(FD) the *Filter Design* consists in finding real filter coefficients, usually in double precision, such that the resulting filter fits the given frequency specification.

(Q) the *Quantization* converts the coefficients to the FxP format. This does not simply consist in rounding the real coefficients as resulting filter could not fit the specification anymore.

(I) the *Implementation* consists in generating a valid hardware description which depends on the hardware components available.

In the previous chapters, we have presented our contribution to the MCM problem which can be used to optimize the implementation step. In the current chapter, our goal will



(a) Specification $\texttt{lp1}_4$ and its reference Matlab, quantized to 5-bit.

(b) Transposed direct form II of second-order IIR filter

Figure 8.1: Second-order IIR specification, transfer function and hardware implementation.

be to demonstrate that mathematical modeling is particularly interesting for multi-layer problems, since models can be merged together to tackle the complete problem.

In Chapter 1, we presented each step and different approaches to merge multiple steps together, such as KCM [Cha94] for Q+I. In case of FIR filters, [KVF23] successfully combined all the steps into one global MILP-based model. The idea was to linearize the frequency response so the filter coefficients search, as scaled integers thus FxP, can be merged with the MCM minimization. Despite the additional difficulties inherent to IIR filters, such as the recursive part and the stability constraints, our ambition is to solve the combined FD+Q+I steps for second-order IIR sections within one global optimization.

In this chapter we present our approach which permits the hardware-aware design of second-order IIR filters. Our work on this topic has been published in IEEE Transactions on Signal Processing [GVK+22b] before we explored the MCM problem, thus we do not use low-level metrics in the following but optimize w. r. t. the number of adders.

8.2 The filter design and quantization steps

We recall the transposed direct form II structure of second-order digital filters in Figure 8.1b and its output is computed as

$$y_n = \sum_{i=0}^{2} b_i x_{n-i} - \sum_{i=1}^{2} a_i y_{n-i}, \tag{8.1}$$

where $a_i, b_i \in \mathbb{R}$ are the filter coefficients. The corresponding transfer function is

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}, \tag{8.2}$$

with $z \in \mathbb{C}$.

Given some frequency specifications, there exists a large number of real-valued coefficient sets which fit, but only a subset of those is representable in FxP arithmetic with a given word length. We propose to directly search for a_i and b_i in a FxP representation and introduce their integer counterparts, $a'_i, b'_i \in \mathbb{Z}$, that are linked with the real-valued coefficients via

$$a_i = 2^{-w+1}\, 2^{g_a}\, a'_i, \qquad a'_i \in [\![-2^{w-1}; 2^{w-1} - 1]\!], \tag{8.3}$$

$$b_i = 2^{-w+1}\, 2^{g_b}\, b'_i, \qquad b'_i \in [\![-2^{w-1}; 2^{w-1} - 1]\!], \tag{8.4}$$

where w is the word length, g_a and g_b are the largest MSB positions of the FxP representations of a_i and b_i, respectively. It is typical to use the same word length for a_i and b_i but we allow different FxP formats in the numerator and denominator of the transfer function. While the word length w is a user-given parameter in our setting and is typically desired to be as small as possible in order to save resources, the values of the MSB positions g_b and g_a are not known. Hence, our first goal is to predict the MSB positions that, on the one hand, encompass all representable coefficients, *i. e.*, do not lose solutions, and that do not overestimate the MSB on the other hand. To do so, we first need to consider the specifications.

8.2.1 Linearizing specifications and stability constraints

One of the key elements of merging all the steps is to be able to model the specifications, required for FD, into an MILP model. Yet, we recall that the frequency constraints

$$\underline{\beta}(\omega) \leq \left|H\left(e^{j\omega}\right)\right| \leq \overline{\beta}(\omega)\,, \qquad \forall \omega \in [0; \pi]\,, \tag{8.5}$$

are highly nonlinear. First of all, (8.5) actually enforces one constraint for each $\omega \in [0; \pi]$ and there are infinitely many such constraints to be satisfied. Actually, this constraint is called semi-infinite [HK93]. The standard discretization approach to handling semi-infinite constraints consists in discretizing $[0; \pi]$ into a finite set of frequencies Ω_d, leading to a finite number of constraints. This discretization can be dynamically updated, but we consider here a fixed discretization and an *a posteriori* verification of the semi-infinite frequency constraint [VLH17].

Second, for a fixed $\omega \in \Omega_d$, the constraint (8.5) includes a complex absolute value, which involves a square-root of the sum of squared terms, and a fraction. By incorporating (8.2) into (8.5), then multiplying with the denominator $|A(e^{j\omega})|$, and finally squaring the result we obtain a constraint equivalent to (8.5)

$$\left|A\left(e^{j\omega}\right)\right|^2 \underline{\beta}(\omega)^2 \leq \left|B\left(e^{j\omega}\right)\right|^2 \leq \left|A\left(e^{j\omega}\right)\right|^2 \overline{\beta}(\omega)^2\,, \tag{8.6}$$

where,

$$\left|B\left(e^{j\omega}\right)\right|^2 = \sum_{k=0}^{2}\sum_{l=0}^{2} b_k b_l \cos((k-l)\,\omega), \tag{8.7}$$

$$\left|A\left(e^{j\omega}\right)\right|^2 = \sum_{k=0}^{2}\sum_{l=0}^{2} a_k a_l \cos((k-l)\,\omega), \text{ with } a_0 = 1. \tag{8.8}$$

Yet, in (8.7) and (8.8), the filter coefficients are still involved in bilinear terms $b_k b_l$ and $a_k a_l$. In Section 2.1.2, we presented of products of positive integers linearization proposed by Billionnet *et al.* [BEL08]. Here however, b_k, b_l and a_k, a_l correspond to filter coefficients that have no sign restriction. We extend the linearization exposed before to signed integers by adding auxiliary variables $x^+, y^+ \in \mathbb{N}$ and $x^{\text{sg}}, y^{\text{sg}} \in \{0, 1\}$, and link them by following constraints

$$x^+ = |x|\,, \qquad y^+ = |y|\,, \tag{8.9}$$

$$x^{\text{sg}} = \text{sign}(x)\,, \qquad y^{\text{sg}} = \text{sign}(y)\,, \tag{8.10}$$

$$z^+ = x^+ y^+, \tag{8.11}$$

where the linearization of the absolute values (8.9) and the sign constraints (8.10) are well-known and involve indicator or big-M constraints [BT97, Man06], and where $z^+ = x^+ y^+$ is the positive case we already presented. Finally, $z = \pm z^+$ and the sign is determined by the values of x^{sg} and y^{sg} directly in the model. This whole linearization relies on the fact that bounds on x and y are known which will be verified in our case once we will have upper bounds on g_a and g_b.

Finally, we recall necessary and sufficient stability conditions for second-order filters,

$$-2 < a_1 < 2, \tag{8.12}$$

$$|a_1| - 1 < a_2 < 1, \tag{8.13}$$

which we can use in an MILP model, almost as is. As explained before, the absolute value is standardly linearized using indicator or big-M constraints.

8.2.2 Bounds on filter coefficients

Now that we have included in an MILP-based model the specifications and the stability, we can deduce the values of g_a and g_b. From stability constraints (8.12) and (8.13) it is straightforward to derive bounds on a_i: $a_1 \in]-2; 2[$ and $a_2 \in]-1; 1[$. These bounds are independent of the frequency specification of the filter and yield an upper bound, $g_a = 1$, for the MSB of the coefficients a_i.

Bounds on b_i, however, cannot be obtained independently of the filter specifications. Using the bounds on coefficients a_i and the fact that the cosine in (8.8) belongs to $[-1; 1]$,

we deduce that

$$a_0 a_1 \cos((0-1)\,\omega) \leq 2, \tag{8.14}$$
$$a_0 a_2 \cos((0-2)\,\omega) \leq 1, \tag{8.15}$$
$$a_1 a_2 \cos((1-2)\,\omega) \leq 2, \tag{8.16}$$
$$a_0 a_0 \cos((0-0)\,\omega) \leq 1, \tag{8.17}$$
$$a_1 a_1 \cos((1-1)\,\omega) \leq 4, \tag{8.18}$$
$$a_2 a_2 \cos((2-2)\,\omega) \leq 1. \tag{8.19}$$

Hence,

$$\left|A\left(e^{j\omega}\right)\right|^2 = \sum_{k=0}^{2}\sum_{l=0}^{2} a_k a_l \cos((k-l)\,\omega) \leq 1+2+1+2+4+2+1+2+1 \tag{8.20}$$
$$\leq 16. \tag{8.21}$$

This bound, together with the frequency specification constraints in Equation (8.6), leads to the constraint

$$\left|B\left(e^{j\omega}\right)\right|^2 \leq 16\overline{\beta}(\omega)^2, \qquad \forall \omega \in \Omega, \tag{8.22}$$

that needs to be satisfied by the coefficients b_i.

As can be seen from (8.7), $\left|B(e^{j\omega})\right|^2$ is a quadratic form $b^T Q b$ with respect to the variables b_i. Its characteristic matrix Q, whose entries are

$$Q_{kl} = \cos((k-l)\omega), \qquad \forall k, l \in [\![1;3]\!], \tag{8.23}$$

is symmetric and its spectrum is $\{0, 1-\cos(2\omega), 2+\cos(2\omega)\}$. Its eigenvalues being non-negative, the inequality constraint (8.22) is convex. As a consequence, solving the convex quadratic problems consisting in minimizing or maximizing b_i subject to the convex quadratic constraints (8.22) can be easily done by local solvers at a linear cost. Actually, common mixed MILP solvers can solve this kind of nonlinear problem. Solving these quadratic models permits to deduce an upper bound for g_b.

Note that the bounds we obtain on b_i do not fully take into account the specificity of the filter we are designing and use the worst case bounds in (8.22). It is possible to compute tighter bounds on the coefficients in order to reduce the search space. Currently, we have modeled the frequency and stability constraints using integer variables to represent FxP coefficients. Hence, we have a model which can be used to solve steps FD+Q. We are not interesting in the solutions of this model as their actual cost in terms of the I step is not defined. However, by minimizing each a_i and b_i subject to the constraints we have already defined, we quickly obtain lower bounds for each coefficient which can drastically reduce the search space when solving the complete model including the I step. Similarly, we obtain upper bounds by maximizing each a_i and b_i. As it has been shown in [KVF23] in the context of the design of FIR filters, solving these simpler MILPs to obtain tighter bounds for the solving of the final MILP is worthwhile.

8.3 Incorporating the implementation step

The FxP coefficients satisfying the above frequency and stability constraints are also subject to hardware constraints. To optimize according to these constraints, we use the minimization ILP-based hardware model for MCM-Adders we presented in Chapter 4. With this model, given a set of constant coefficients, we are able to find an adder graph with a minimal number of adders. For the global IIR filter design problem, the coefficients are, however, the unknowns. Similarly to [KVF23], we first design a number of linking constraints that bind the coefficient variables for filter design with the inputs of an MCM problem. We refer to them as *glue constraints* in Figure 8.2.

The challenge is that for the IIR filters two multiplier blocks should be optimized simultaneously, one for coefficients a_i and one for b_i, with the goal of minimization of the *total* number of adders, *i. e.*, in both multiplier blocks and in the filter structure. This can be achieved by incorporating twice the MCM constraints, one for each multiplier block, and combining the objective functions. However, the original model is already costly so we propose a more efficient solution, where the idea is to produce one adder graph, which can be *a posteriori* separated into two independent multiplier blocks, one for a_i and one for b_i. In other words, we propose an MCM model that, given a certain number of input coefficient variables, adds the constraint that one part of those is separable from the others.

This is achieved in the following way. In the original model, a variable c_a is associated to every adder a and these variables are linked together with binary variables $c_{a,i,k}$ where $c_{a,i,k} = 1$ if, and only if, the adder k is the i-th input of adder a. In the new model, we need to ensure that the adders related to the coefficients a_i are not computed using the adders related to the coefficients b_i, and *vice versa*.

We propose to add binary variables mcm_a^a and mcm_a^b, for each adder a, in order to ensure that the adder can be used for one adder graph or the other, but not both. We add a set of binary variables, $s_{a,k}$, which encodes the information about two adders a and k being used in the same adder graph, and a set of the following constraints

$$c_{a,i,k} \leq s_{a,k} \qquad \forall a, k, i. \tag{8.24}$$

They ensure that an adder cannot be used as the input of another if they are not in the same adder graph. In a similar way, using mcm_a^a and mcm_a^b, we ensure that the glue between the filter coefficients and the adders is only possible for adders in the corresponding adder graph. Note that the solver might even find FIR filters, as a_i can be equal to 0.

8.4 Enhancing the model and wrapping up the solving process

The explicit expression of (8.6) involved in the constraint (8.22) is as follows

$$b_0^2 + b_1^2 + b_2^2 + 2b_0b_1\cos(\omega) + 2b_1b_2\cos(\omega) + 2b_0b_2\cos(2\omega). \tag{8.25}$$

We remark that the above equation is insensitive to changing all coefficients sign simultaneously and to exchanging variables b_0 and b_2. This corresponds to symmetries in our model which we will break in order to reduce the size of the search space. In details, the first symmetry consists in simultaneously changing the sign of the values taken by b_0, b_1 and b_2 and the second in exchanging the values taken by b_0 and b_2. Note that both symmetries have no incidence on the MCM problem defined in Chapter 4, the existence of symmetric adder graphs being obvious for opposed or exchanged coefficients.

As a consequence, from an arbitrary solution with

$$b_0 = b_0^*,\ b_1 = b_1^* \text{ and } b_2 = b_2^*, \tag{8.26}$$

we can build three new solutions by simply applying these symmetries to obtain

$$b_0 = -b_0^*,\ b_1 = -b_1^* \text{ and } b_2 = -b_2^*, \tag{8.27}$$
$$b_0 = b_2^*,\quad b_1 = b_1^* \quad \text{and } b_2 = b_0^*, \tag{8.28}$$
$$b_0 = -b_2^*,\ b_1 = -b_1^* \text{ and } b_2 = -b_0^*. \tag{8.29}$$

The fourth solution (8.29) is obtained by applying the two symmetries consecutively, in any order. Note that some of these four symmetric solutions (8.26)-(8.29) may be equal in some special cases, *e. g.*, when $b_0 = b_1 = b_2 = 0$.

In order to derive symmetry breaking constraints, we divide the search space for b_0, b_1 and b_2, which is $\mathbb{R}^3$, into four areas:

$$\Sigma_1 = \left\{(b_0, b_1, b_2) \in \mathbb{R}^3 \mid b_0 \geq |b_2|\right\}, \tag{8.30}$$
$$\Sigma_2 = \left\{(b_0, b_1, b_2) \in \mathbb{R}^3 \mid -b_0 \geq |b_2|\right\}, \tag{8.31}$$
$$\Sigma_3 = \left\{(b_0, b_1, b_2) \in \mathbb{R}^3 \mid b_2 \geq |b_0|\right\}, \tag{8.32}$$
$$\Sigma_4 = \left\{(b_0, b_1, b_2) \in \mathbb{R}^3 \mid -b_2 \geq |b_0|\right\}. \tag{8.33}$$

Then, one can verify that: a solution lying inside Σ_2 moves to Σ_1 applying the sign symmetry; a solution lying inside Σ_3 moves to Σ_1 applying the exchange symmetry; a solution lying inside Σ_4 moves to Σ_1 applying both symmetries consecutively. As a consequence, one can restrict the search to Σ_1 and reconstruct all solutions using symmetries. This restriction to Σ_1 is achieved by adding the symmetry breaking constraint

$$b_0 \geq |b_2|, \tag{8.34}$$

to the model. The absolute value can be linearize as presented previously. Restricting to another Σ_i would be lead to another symmetry breaking constraint, with an equivalent improvement of the resolution process.

For completeness, it should be noted that two equivalent solutions might still be kept on the frontier $\Sigma_k \cap \Sigma_l$ between areas despite our additional constraint. However, resolving this issue is counterproductive, because the great majority of the search space lies in the interior of the sets Σ_i, and removing the symmetries on the boundaries would introduce too many additional constraints.

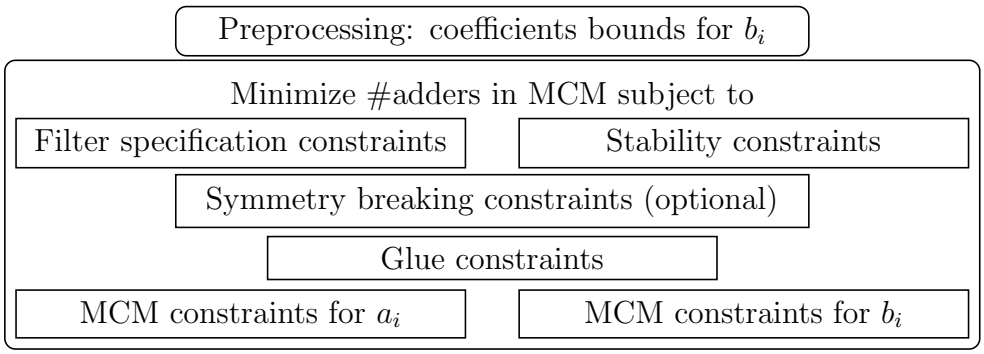


Figure 8.2: High-level structure of the global ILP model IIRoptim.

Bird-view of the model

We provide a complete model, which is preceded by a simpler quadratic convex problem we solve with the same tools. As Figure 8.2 shows, the high-level IIR model consists of the following constraints:

- linearized frequency-specification constraints (8.6)-(8.8);
- stability constraints (1.27)-(1.28);
- (optional) symmetry breaking constraints (8.34) enabling a significant reduction of the design space;
- constraints responsible for the design of optimal constant-multiplication blocks for a_i and b_i;
- so-called glue constraints, connecting the unknown filter coefficients with multiplier blocks.

Thanks to the great modularity of mathematical modeling, we should be able to merge our MCM-Bits or tMCM models with the FD+Q as easily as it was for MCM-Adders. With MCM-Bits or tMCM, the model might become too large for current solvers but they will evolve and we believe they will soon be able to tackle problems which our currently out of reach.

8.5 Experimental results and discussion

8.5.1 Implemented toolflow

We implemented our approach into a tool, whose minimalist interface is shown in Figure 8.3. The input specification includes the frequency-domain specification of the filter, and also the information for hardware implementation, *i. e.*, the coefficient word length,

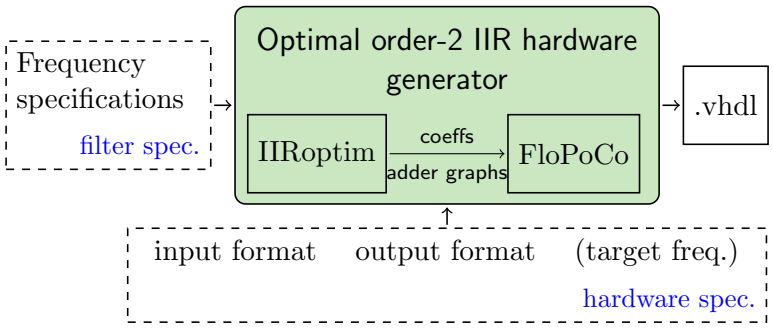


Figure 8.3: Interface of the proposed tool.

Table 8.1: Sets of lowpass filters used for the IIR experiments. First the set with decreasing δ, next the sets with increasing passband and stopband. Finally, a lowpass filter.

Benchmark	$\texttt{lp1}_k$	$\texttt{lp2}_k$	$\texttt{lp3}_k$	$\texttt{lp4}$
k	$\{0, 1, \ldots, 6\}$	$\{0, 1, \ldots, 4\}$	$\{0, 1, \ldots, 4\}$	–
passband/π	$[0; 0.3]$	$[0; 0.3 + 0.05k]$	$[0; 0.3]$	$[0; 0.5]$
stopband/π	$[0.7; 1]$	$[0.7; 1]$	$[0.7 - 0.05k; 1]$	0.91
δ	$0.1 - 0.01k$	0.1	0.1	0.1

the input/output formats and performance parameters, such as the required target frequency. All the other parameters (filter coefficients and their FxP format, the multiplierless operator architecture, the representation of intermediate data, etc.) are determined automatically as a part of a global optimization process.

Given the user input, we first construct an MILP model and solve it, with a few possible intermediate steps we presented above. The result of the global optimization problem, *i. e.*, the list of filter coefficients and the adder graphs defining the optimal shift-and-add architectures, is then passed on to FloPoCo [dDP11], which was extended for that. In collaboration with Jonas Kühle, Fulda University, we implemented a new operator `FixIIRShiftAdd` within FloPoCo, generating faithfully-rounded multiplierless IIR filters, *i. e.*, only the last output bit might be erroneous and all other bits are guaranteed to be correct. The new operator alleviates the filter designer from all internal architectural decisions and presents a final VHDL code.

8.5.2 Set of benchmarks and comparison approaches

Benchmarks

Although the design of second-order IIR filters is an important part of the design of larger order filters, benchmarks are rarely targeting frequency specifications of individual second-order sections. Hence, we propose three sets of filter specifications with increasing filter design difficulty that could without doubt be used in real-life applications. In addition to that, we add another artificial low-pass filter, and a real-life example from [VIDDH19].

The normalized low-pass filter specifications are here defined as

$$1-\delta \leq \left|H(e^{i\omega})\right| \leq 1+\delta, \qquad \forall \omega \in [0;\omega_p], \qquad \text{(passband)}$$
$$0 \leq \left|H(e^{i\omega})\right| \leq \delta, \qquad \forall \omega \in [\omega_s;1]. \qquad \text{(stopband)}$$

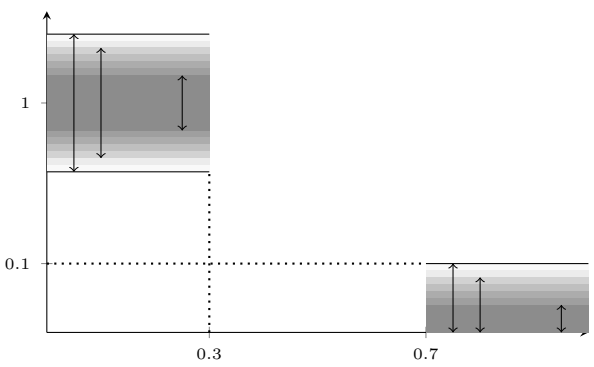
We fix the initial passband to $[0;0.3]$, stopband to $[0.7;1]$ and $\delta = 0.1$. Then, for each of the families of filter specifications, we vary one of the parameters in dependence of a variable k to increase the filter design difficulty. The detailed frequency specifications are given in Table 8.1, and their graphical representation is sketched in Figure 8.4. For example, in family `lp1`, the δ varies from 0.1 to 0.04 with step $0.01k$ where $k = [\![0;6]\!]$. However, the designs were possible only up to $k = 5$, reaching the maximum design possibilities for second-order IIR filters. Analogously, the families `lp2` and `lp3` increase/reduce the passband/stopband, respectively. The filter specification `lp4` is a lowpass with a short stopband.

Finally, our last benchmark `hp0` is a real-life application, it is a high-pass filter, represented in Figure 8.5, which is a compensator used in a magnetic-bearing control system. It was initially derived by discretizing the analog controller [KFCV03] and further used as an example in word length optimization literature [SRZ12, VIDDH19]. It is most recently used in [VIDDH19] to demonstrate a KCM-based faithfully-rounded implementation of IIR filters. This filter is not defined in terms of frequency specifications but by its frequency response, hence to implement the three-step approaches we approximate the frequency specification in Matlab using the coefficients given in [SRZ12, VIDDH19]. For our implementation, the versatility of an MILP modeling permits to easily integrate frequency response bounds as functions of ω and similarly approximate the filter `hp0`.

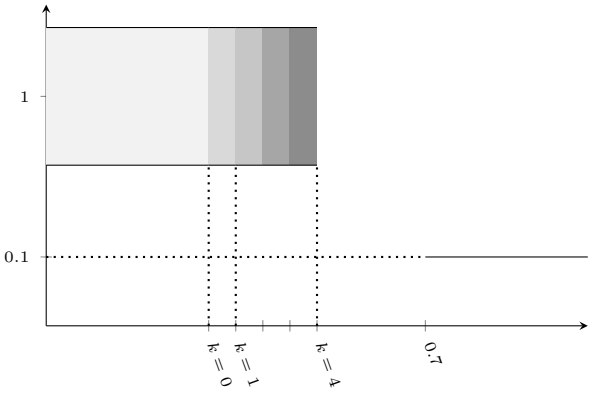
Comparison approaches

We compare our approach with the classical and state-of-the-art three step approach for the IIR design. We will compare optimization results in terms of high level metrics such as the number of adders, and implementation results. For the three step approach, we will first obtain filter coefficients using Matlab's elliptic method. Then, as in Matlab's Fixed-Point design Toolbox, we convert the double-precision coefficients to FxP. However, quantization often leads to errors in frequency response or instabilities, hence we start with a relatively small word length and increase it until the frequency-domain error is above a threshold ($> 10^{-7}$) or the filter is unstable. Finally, we perform different implementation:

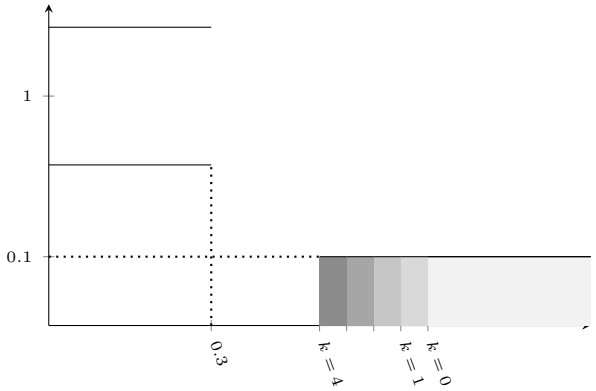
- 3-step Generic: using generic multipliers (using the VHDL '`*`' operator, potentially using DSP blocks on FPGA);
- 3-step Generic NoDSP: same as above with disabled DSP blocks for synthesis and implementation;
- 3-step MCM: we apply the optimal shift-and-add implementation on the quantized coefficients;
- 2-step KCM: we skip the quantization step and use the approach from [VIDDH19], which directly obtains a KCM-based implementation for real coefficients.



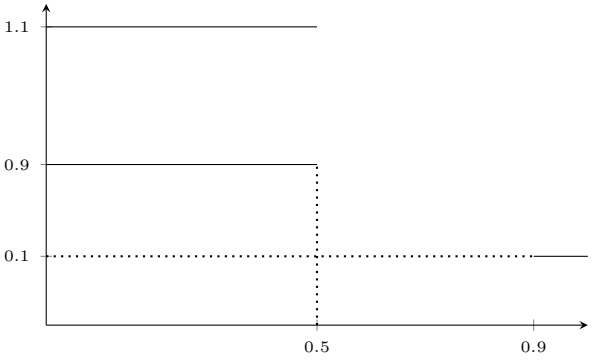
(a) $\texttt{lp1}_k$ – decreasing amplitude



(b) $\texttt{lp2}_k$ – increasing passband



(c) $\texttt{lp3}_k$ – increasing stopband



(d) `lp4`

Figure 8.4: Proposed families of benchmarks.

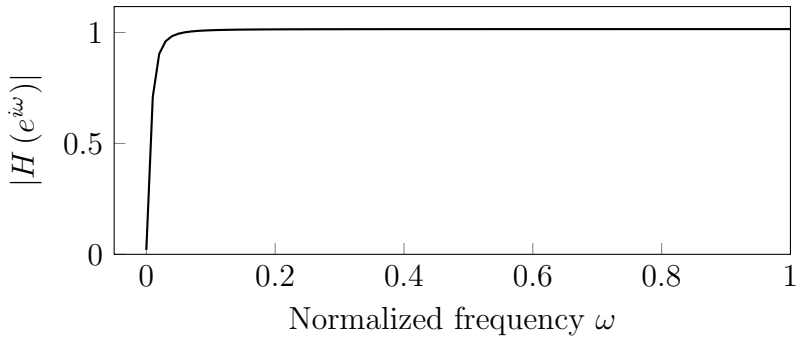


Figure 8.5: Frequency response of the compensator `hp0` [CVKF03, SRZ12].

Table 8.2: Results for our global optimization method vs applying optimal MCM upon quantized coefficients. The total number of adders #A=#A_M+#A_S consists of the multiplier block #A_M, and structural #A_S adders. Results are reported for the smallest coefficient word length W yielding stable filters for 3-step MCM. Best results are in bold.

Benchmark	Our method				3-step MCM			
	W	#A_M	#A_S	#A	W	#A_M	#A_S	#A
$lp1_0$	4	1	4	**5**	10	8	4	12
$lp1_1$	4	1	4	**5**	16	11	4	15*
$lp1_2$	4	2	4	**6**	6	5	4	9
$lp1_3$	4	3	4	**7**	10	7	4	11
$lp1_4$	5	4	4	**8**	9	7	4	11
$lp1_5$	5	4	4	**8**	–	–	–	–
$lp2_0$	4	1	4	**5**	10	8	4	12
$lp2_1$	4	1	4	**5**	10	8	4	12
$lp2_2$	5	3	4	**7**	10	8	4	12
$lp2_3$	6	4	4	**8**	–	–	–	–
$lp3_0$	4	1	4	**5**	10	8	4	12
$lp3_1$	4	2	4	**6**	4	2	4	**6**
$lp3_2$	4	3	4	**7**	23	18	4	22*
$lp3_3$	5	3	4	**7**	–	–	–	–
lp4	4	1	4	**5**	4	3	4	7
hp0	6	1	2	**3**	11	6	4	10

* heuristic solution using [KZFC12]

Basically, 3-step MCM and 2-step KCM is the state-of-the-art but we also compare against 3-step Generic (NoDSP) which is typically used by engineers in Matlab.

The coefficient word length is an input parameter for our tool, hence, for each specification we will explore a range of different coefficient word lengths. For the input/output data in hardware implementation, we used 16-bit.

8.5.3 Evaluation of the MILP model and design results

In this section, we evaluate the performance of our MILP model and compare with the 3-step MCM-based approach to see the benefits w. r. t. number of adders in a shift-and-add implementation.

First, note that the solving times are reasonable, ranging from a few seconds for small word lengths (4-5 bits) and going up to a few minutes in the worst case if we push the word length to unnecessarily large values. We observe similar solving times for the 3-step MCM approach. Symmetry breaking constraints permitted to drastically reduce solving times. Depending on the problem complexity, solving times were at least divided by two.

Although increasing the word length doubles the ranges of integer variables with each new coefficient bit, the main bottleneck in pushing the coefficient word lengths further than 10-11 bits is due to numerical instabilities in the MILP solvers.

In Table 8.2, we present the optimization results for our method and for the 3-step MCM method. We report results for the smallest word lengths possible, in which our method has a feasible result (and, by construction, no frequency domain error), and where the Q step in 3-step MCM yields stable filters with frequency-domain error smaller than 10^{-7}. It can be noticed that our method finds solutions for smaller coefficient word lengths than the 3-step MCM, and with significantly smaller total number of adders for most of the cases (46% on average in Table 8.2). In some cases, marked with asterisk, target word lengths for 3-step MCM were too big s. t. the optimal MCM timed-out and the RPAG heuristic [KZFC12] was used to obtain adder graphs instead. For the specification `lp4` both approaches find designs with 4-bit coefficients, but thanks to efficiently covering the whole design space of all possible FxP coefficients, our method determines coefficients that require *fewer* adders in the multiplier block and the total of 5 adders, instead of 7. Finally, the benchmark $\texttt{lp3}_1$ is the only case when the coefficients in 3-step MCM coincide with the ones found by our tool.

Another advantage of the proposed MILP formulation is privileging sparse implementations, which even for second-order filters largely improves performance. For instance, while the benchmark `hp0` was traditionally implemented with all nonzero coefficients [VIDDH19, SRZ12], we found a sparse implementation with 6-bit coefficients, having as coefficients $b_0 = 1$, $b_1 = -1$, $b_2 = 0$, $a_1 = -31/32$, $a_2 = 0$ and leading to the total of 3 adders for the whole filter. To compare, the 3-step MCM can provide at minimum 11-bit coefficients implemented with 10 adders.

Note that for benchmarks $\texttt{lp1}_5$, $\texttt{lp2}_3$ and $\texttt{lp3}_3$ Matlab failed to find any IIR filter with double-precision coefficients which fits the specifications. Our method, however, successfully completes the task with a small coefficient word length. It is important to note that our tool provides optimal implementations w. r. t. the total number of adders for a given coefficient word length. While we stop at the smallest possible coefficient word length, as in Table 8.2, it does not necessarily lead to the optimal implementation w. r. t. all possible word lengths, and increasing the coefficient word length could actually lead to fewer adders.

To illustrate this, Figure 8.6 shows synthesis results for the $\texttt{lp1}_4$ benchmark with coefficient word lengths varying from 4 to 10 bits. First, one can note that our results start at 5-bit coefficients. Indeed, our tool proved that no stable solution satisfying the given frequency specifications is possible for coefficients with word length 4. It can be seen that increasing the word length from 6 bits to 7 bits actually reduces the total number of adders from 8 to 7 and improves the number of used LUTs. Then, increasing coefficient word length does not worsen the solution as we can simply multiply values by 2 for each additional bit to obtain an equivalent solution. In other words, even if the user specifies a larger word length than required, our tool finds the best coefficients that might fit in a smaller format and guarantees that trying smaller word length will not give smaller number of adders. However, the larger the word length the harder is the problem for the solver which, at some point, might not find the optimal solution within its available time.

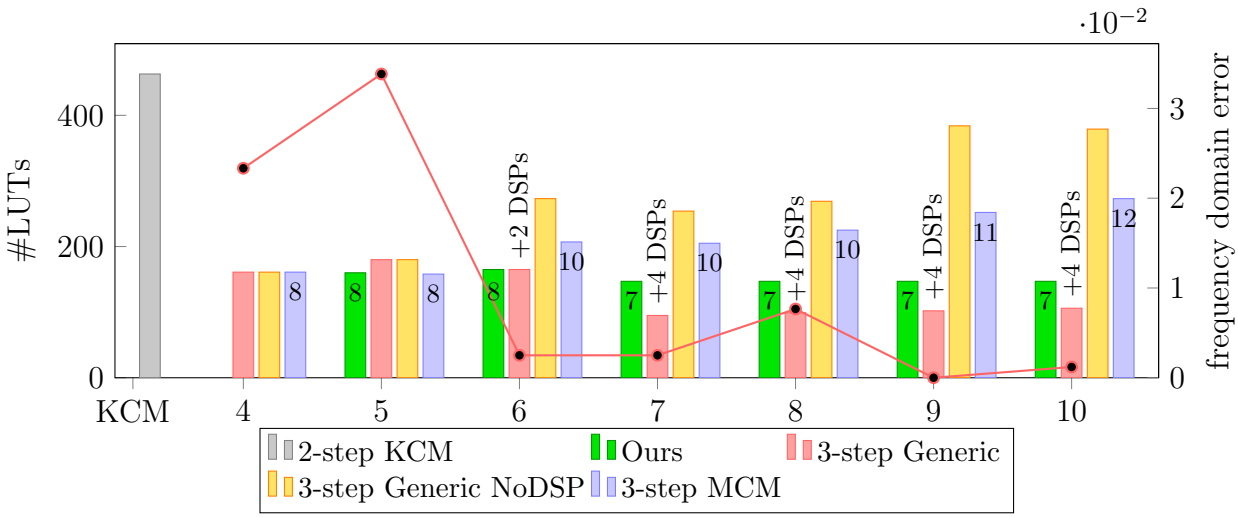


Figure 8.6: Implementations for $\texttt{lp1}_4$ benchmark with coefficient word lengths varying from 4 to 10 bits. The bars correspond to the number of LUTs (left axis) and labels on bars denote number of adders. The frequency-domain error of 3-step methods is the red line (right axis).

For the classical 3-step approaches, design-space exploration is more difficult and irregular. For these methods, coefficient quantization introduces a frequency-domain error meaning that the quantized filter does not satisfy the frequency specifications any more (see the red line in Figure 8.6). This error is highly nonlinear, and a typical intuition that increasing coefficient word length improves the quality of filter is simply not true (see the frequency-domain error for 7- and 8-bit coefficients in Figure 8.6). Hence, the search for the best coefficient size must be exhaustive for 3-step methods.

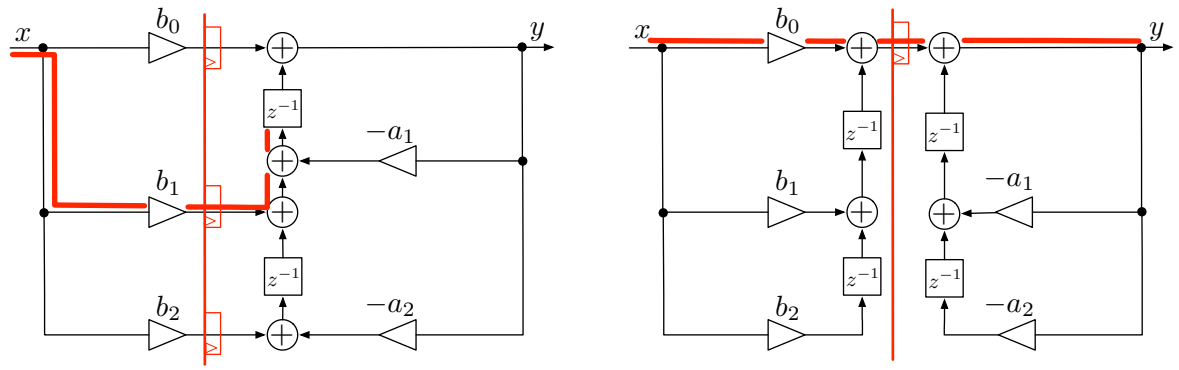
8.5.4 Hardware implementation and discussion

In the following, we first describe in details the faithfully-rounded architectures that we implement in FloPoCo and then discuss the synthesis results obtained for our benchmarks and each approach.

Implemented architecture

The addition in the center of the high-level description of the transposed direct form II filter, in Figure 8.1b, requires the addition of three inputs, hence it must be first separated into two consecutive two-input add operators. The critical path then contains one multiplication and two additions as illustrated in Figure 8.7a. To cut the critical path, pipeline registers are usually added, leading to three additional registers as indicated by the red registers in Figure 8.7a.

We propose to slightly modify the classic structure and separate the recursion into its own branch as in Figure 8.7b. The number of two-input adders stays the same, and the two



(a) Transposed Direct Form 2 (b) Proposed structure

Figure 8.7: Comparison of the classic and proposed structures. The red line corresponds to the critical path.

additional registers are compensated by the fact that only one pipeline register is required to cut the critical path, instead of three for the original structure. Hence, at the same hardware cost we get a better modularity of the feedforward/feedback loops.

Faithfully-rounded output

We aim at providing faithfully-rounded implementations, *i. e.*, the precision of the output y serves as the accuracy constraint. The goal is to compute just right to avoid assigning larger data sizes than strictly required as this would waste resources to compute unnecessary bits. Hence, we provide a code generator that, given the input/output format and filter coefficients, automatically computes the word sizes of all internal data paths to guarantee the time-domain error smaller than $2^{\ell_{\text{out}}}$ but not more.

Figure 8.8 presents our approach for multiplierless hardware IIR on the example of $\texttt{lp1}_4$ benchmark, fulfilling frequency specifications in Figure 8.1a with 7 adders. Its transfer function was obtained as

$$H_{\texttt{lp1}_4}(z) = \frac{25 \times 2^{-7} + 40 \times 2^{-7}\, z^{-1} + 25 \times 2^{-7}\, z^{-2}}{1 - 40 \times 2^{-6}\, z^{-1} + 20 \times 2^{-6}\, z^{-2}}. \tag{8.35}$$

The inputs to the architecture generator are the MSB and LSB positions of the input x and output y, the adder graphs for multiplier blocks a_k and b_k, and their corresponding LSBs. For instance, here $\ell_b = -7$, $\ell_a = -6$.

Obviously, one cannot compute exactly (or with some fixed precision) on each iteration, truncate to ℓ_{out} and simply feed truncated values back into the loop, as this will degrade tremendously numerical quality and accumulated errors will explode. The Worst-Case Peak Gain (WCPG) measure for IIR filters [VHL15], which has been applied for hardware IIR filters implemented with KCM multipliers [VIDDH19], permits to determine the necessary extended internal precision ℓ_{ext} s. t. the propagated error never reaches the LSB of the

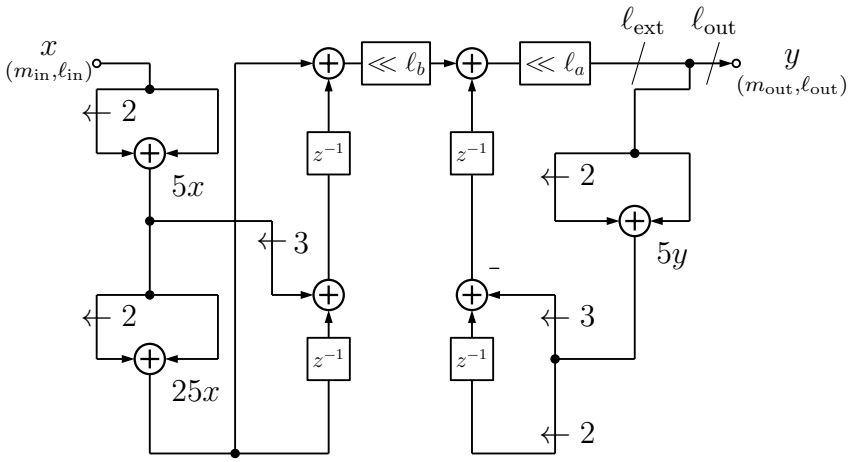


Figure 8.8: The $\texttt{lp1}_4$ benchmark can be implemented with mere 7 adders for 7-bit coefficients. All additions are exact, the truncation to internal extended format ℓ_{ext} is determined s.t. the output is faithfully rounded to ℓ_{out}.

output. For example, in Figure 8.8 $\ell_{\text{ext}} = \ell_{\text{out}} + G$, where the number of guard bits G for the filter $\texttt{lp1}_4$ determined with its WCPG is $G = 3$. Then, the output of the multiplier blocks needs to guarantee its result with accuracy ℓ_{ext}. We perform all additions and shifts exactly, increasing the size of data paths until their truncation to ℓ_{ext}.

For the generic approach, based on plain VHDL multipliers (using the $*$ operator), we adopt a similar approach.

Synthesis results

For the hardware experiments, the FPGA synthesis was performed using Vivado v2020.2 for a Kintex 7 device (xc7k70tfbv484-3). The delay and power results are obtained after place and route. The power was evaluated for a 100 MHz clock for all dynamic parts including clocks, logic, signals and DSPs but no static or I/O power to ease the comparison of the cores. We compare the designs with the smallest coefficient word lengths, *i. e.*, for which our MILP has a feasible solution and the 3-step approach has a frequency-domain error smaller than 10^{-7}.

Figure 8.9 summarizes the number of LUTs, the critical path delay and the power. For each benchmark, below the x-axis, we see the coefficient word length for our result (left) and for the 3-step quantization (right). For instance, we recognize the values 5 and 9 for the $\texttt{lp1}_4$. First, we observe that our method is always superior to any of the 3-step or KCM-based methods w. r. t. all metrics. We also observe an average LUT improvement of 48% compared to the best results of other methods, excluding the one using DSPs. In addition to that, the proposed approach offers the lowest delay with an improvement of 27%, which is not as drastic as LUT improvement. Finally, the resource reduction translates to the significant power reduction of 57%, on average, and up to 95% in the best case.

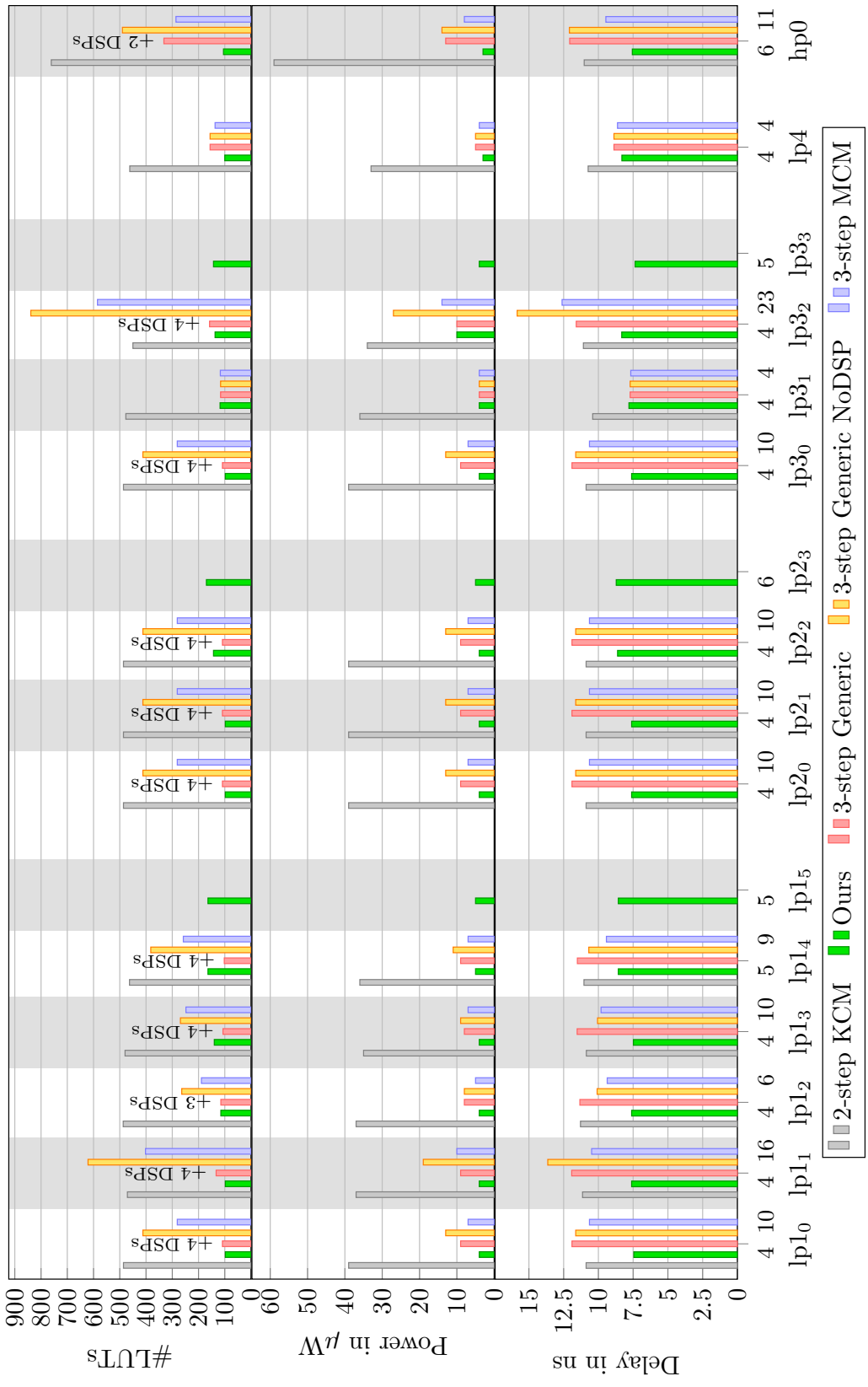


Figure 8.9: Benchmark results of resources, power and critical path delay, using I/O word length 16 bits, on FPGAs. The x-axis features the benchmark name and the smallest possible coefficient word length for our designs (left) and 3-step approaches (right).

Of course, our method is more efficient partly due to the smallest possible coefficient word lengths, but even when using the same word sizes (see benchmark `lp4`) we obtain solutions with less adders which leads less LUTs, a smaller delay and power consumption. The KCM-based approach, representing the state-of-the-art for faithfully-rounded IIR filters, had worse general performance than our method in all benchmarks and requires, on average, more LUTs than other approaches. However, KCM-based multipliers have similar, or smaller, delays than 3-step methods. Moreover, KCM would have no issue dealing with large word length coefficients while optimal shift-and-add based method, including ours, are currently limited in this regard. Yet, we proved that for second-order IIR filters, it is unnecessary to deal with large word lengths.

One of the best performance improvements was achieved for the `hp0` compensator. This is due to the much smaller coefficient word length than in all the previous literature, the sparsity and the fact that we succeeded in finding a filter which is not on the edge of stability. This results in only 105 LUTs compared to 286 in the 3-step MCM and 760 in the KCM-based implementation.

Thanks to the help of Efstratios Zacharelos from the University of Naples Federico II, we have been able to also run the hardware experiments on ASIC. The synthesis was performed using Cadence Genus v18.13 targeting a commercial standard-cell library in 14 nm FinFET technology from Global Foundry with 0.8 V supply voltage and regular VTH. Power dissipation is estimated by simulating the final netlist with 1000 random input vectors using Cadence NCSIM to obtain the switching activity. The timing constraints were set to 10 ns to obtain rather compact circuits. ASICs comparison with implementations using DSPs or the KCM table-based method is not representative due to technology differences, thus we just report implementation results for the our approach and the 3-step Generic NoDSP and 3-step MCM approaches.

We reported our hardware results, in terms of area, power and delay, in Figure 8.10. On ASICs benchmarks, our designs yield similar results, reducing the area by 48% and the delay by 27% on average, with the peak reduction by 76% and 51%, respectively. Our new designs also demonstrate an impressive average power improvement of 65%, and in the best cases of 90%. This means that our designs divide the power consumption by 4 on average, and by almost 10 in the best observed cases.

Although ASICs were not our primary target, we still obtain a substantial gain with our approach. It is very encouraging as we might expect that approaches designed for FPGAs also work well on ASICs. Hence, it would mean that the different methods we proposed for MCM and for which we confirmed significant gain in comparison with the state-of-the-art over FPGAs are also interesting for ASIC design.

8.6 Conclusion

In this chapter, we demonstrated optimal design of multiplierless second-order IIR filters w. r. t. the number of adders. We combined the classical steps, which consist in finding FxP filter coefficients and then their most efficient implementation according to a given metric,

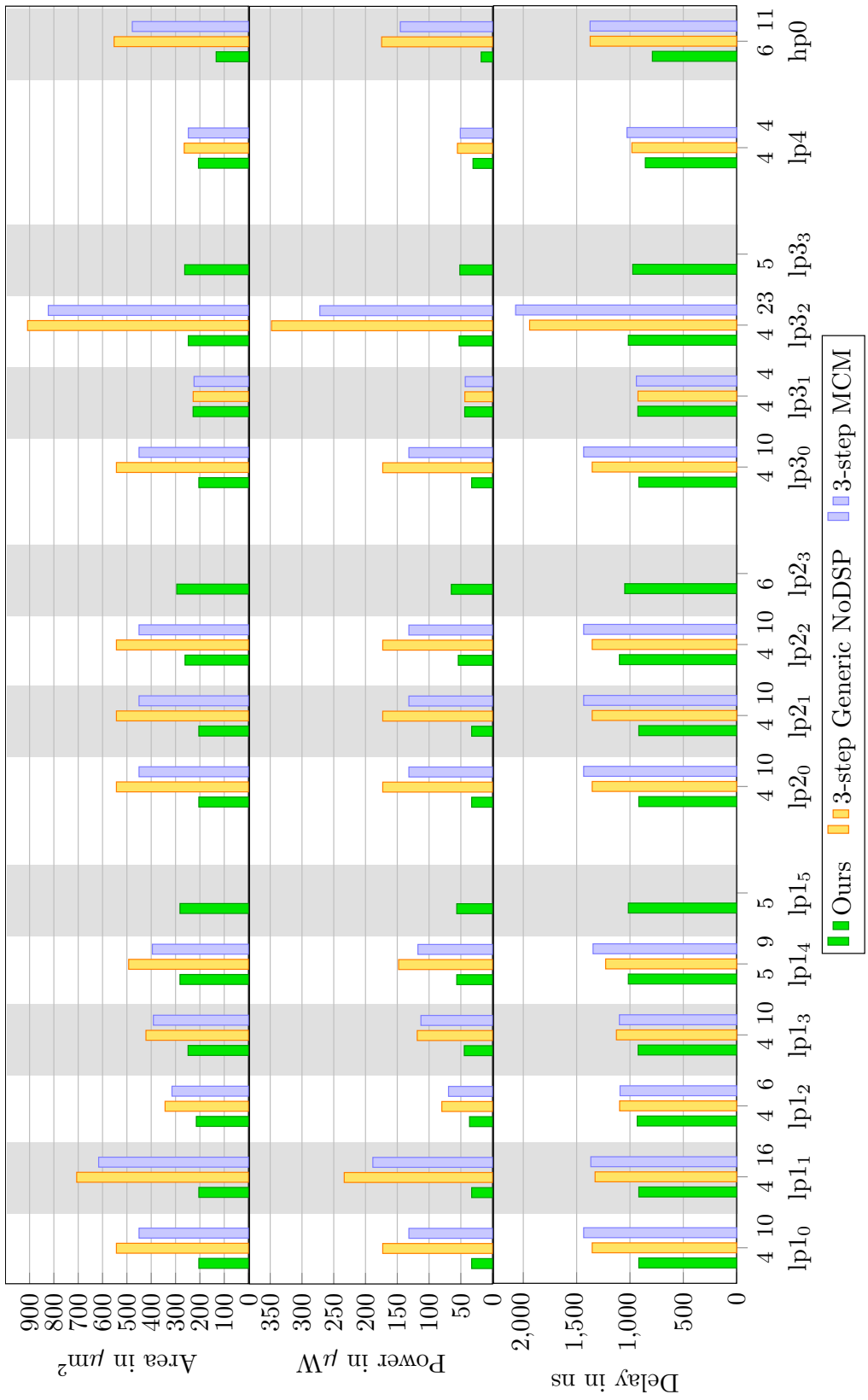


Figure 8.10: Benchmark results of resources, power and critical path delay, using I/O word length 16 bits, on ASICs. The x-axis features the benchmark name and the smallest possible coefficient word length for our designs (left) and 3-step approaches (right).

into a global optimization problem. We are able to guide the search for FxP coefficients which fit a set of specifications using directly the multiplierless implementation cost.

This work required to overcome multiples difficulties such as the nonlinear specifications or the bound requirement on coefficients. Ultimately, we proposed a linear model of the combined filter design, quantization and implementation step providing a convenient way for adjustments and extensions. We improved this model with several design space reduction techniques, including a novel symmetry breaking constraint, which we formally proved. Our approach permits to find in mere seconds filter coefficients which fully satisfy filter specifications without a back and forth process between filter design and quantization in addition to ensuring these coefficients have a low multiplierless implementation cost.

Combined with FloPoCo, we provide an automated tool which takes simple parameters as an input and outputs VHDL code for FPGAs. The synthesis results confirmed that a global optimization approach is superior to the multi-step FD + Q + I classical methods. After testing the tool on numerous benchmarks, we observed a significant hardware cost reduction obtained with our approach, on both FPGAs and ASICs.

Some efforts are still required to extend our method to higher order filters and we are confident that using mathematical modeling it should be possible to automatically find efficient design of cascaded second-order IIR filters. Every advance in structural methods, such as using direct form vs transposed direct form or using different metrics for multiplierless implementation, can be used to adjust the current set of variables and constraints. Indeed, although the number of adders is a reliable high level metric, optimizing the number full adders instead would be better and we are fairly optimistic on the fact that our method can be refined to minimize that criteria. In any case, this work shows that MCM, and MILP in general, is useful and efficient for practical applications.

Conclusion and Perspectives

End-of-the-World Switch. PLEASE
DO NOT TOUCH

Terry Pratchett (1948-2015),
Thief of Time (2001)

With this thesis, we contributed to intensify the scientific exchanges between operations research methods, computer arithmetic analysis and hardware design applications. Among the large variety of topics in hardware design, we have taken up the MCM problem which is an essential building block of many classical applications. At first we have redefined the terminology around the MCM problem to be able to easily differentiate MCM as a concept from the more specific MCM-Adders, MCM-Bits, tMCM or PMCM problems. Then, in this work we demonstrated that operations research, via mathematical modeling, brings to hardware applications valuable tools. In Chapter 4, we have presented the efficiency of mathematical modeling to solve a pertinent problem well-known for decades. First, we have presented our minimization model for the MCM-Adders problem and incorporated secondary metrics such as the adder depth. Then, we demonstrated the interest of tweaking solver parameters as, with our set of parameters compared to default values, it permitted to reduce solving times by 16.9%, on average.

Furthermore, we dived into hardware model counting the number of one-bit adders in Chapter 5. This lower-level metric actually reflects better the hardware cost and solving MCM-Bits instead of MCM-Adders permitted to reduce the LUT utilization by 7.8%, on average. To make these comparison, we included a VHDL code generation component to our toolbox.

We believe that Chapter 6 is really where the mixture takes. Although we are designing *arithmetic* operators, in a way we got around error analysis up to this point. In this chapter, an error analysis has been necessary to dive deeper in the hardware design possibilities. This error analysis is important for safety-critical applications in which hardware designers need to be able to trust that errors in the circuit are small enough. Furthermore, it facilitates the design of frugal hardware operators: using intermediate truncations, we reduce the power consumption by 22% in comparison to full-precision circuits, on average.

Finally, in Chapter 7, we demonstrated that mathematical modeling is not limited to modeling adders (block of LUTs), shifts (wires) and one-bit adders (LUTs) but can also be

used to model registers (FF) which are an essential component of hardware circuit. This permitted to drastically reduced the critical path delay, dividing it by 3 in some cases. Solving PMCM, instead of *a posteriori* pipelining a fixed adder graph, also reduced the one-bit adder and register cost by 33%, on average.

Throughout this thesis, we presented multiple models which have been implemented in our open-source Julia package `jMCM`[1]. Solving our models, we obtain adder graphs for which we can generate VHDL using our package `AdderGraphs`[2]. To provide an ultimate practicability, we believe that it would be extremely useful to embed these models in tools that hardware designers could trust. Currently, we are doing some work in that direction as all of our implementations are open-source and we have started the integration into FloPoCo.

Finally, as a breath of fresh air, we took a step back and demonstrated that MCM, solved with MILP, combines well with the design of bigger arithmetic operators such as digital filters. Digital filter algorithms usually involve multiple constant multiplications and with the design of second-order IIR filters we illustrated that MCM can be used at the core of a larger problem. This way, we performed the co-design of digital filter coefficients with their implementation using MCM. This permitted to obtain filters which power consumption has been divided by up to 10 in comparison to the state-of-the-art KCM. On average, we reduced the number of LUTs by 48% compared to the best results of other methods, excluding the one using DSPs. With these experiments, we have exhibited the efficiency of mathematical modeling to solve practical hardware design problems. This is very promising as it encourage us to pursue our work on other hardware problems.

Perspectives

The work presented in this thesis is a step towards using more operations research techniques in hardware design. Although we solved multiple flavors of the MCM problem, it still has many corners to explore.

Overall, we plan to **improve our toolbox on the MCM** problem and remove as many gray areas as possible:

- **Using DSPs.** We focused on FPGAs but left aside DSP blocks, these resources are valuable and could be used together with LUTs in order to dispatch the cost of multiple constant multiplications over various FPGAs elements. In short term, we plan to incorporate their usage within MILP models to replace costly constants with DSPs. In particular, it makes no doubt that combining DSPs and LUTs, as in [LPBG19], to make the most of both resources is a direction we need to go towards.

- **New modeling approaches.** We had an hammer, MILP, and we showed that MCM is close enough to a nail. However, we need to look at our own practice with critical thinking and maybe consider other mathematical modeling approaches or

[1] `https://github.com/remi-garcia/jMCM`
[2] `https://github.com/remi-garcia/AdderGraphs`

heuristics. In particular, in the mid-term future, we would like to experiment the constraint programming paradigm over the MCM-Adders problem to compare with MILP. Indeed, depending on the problem, MILP solvers or constraint programming solvers are better suited and, on the MCM-Adders problem, some limitations, such as the numerical instabilities issues, would be avoided with constraint programming. In any case, both methods require heavy computations and we believe that more heuristics can be developed for this problem. Currently, to our knowledge, available heuristics are primal heuristics which produce an initial solution but do not build on it. In the future, we would like to explore alternatives and propose heuristics that could improve solutions with a dedicated local search.

- **Finer-grained versatile hardware component modeling.** Overall, on the MCM topic, we went from a high-level metric, the number of adders, towards a lower-level metric, counting one-bit adders, including truncations. Ultimately, directly modeling LUTs, DSPs, FFs and wires would be a substantial realization. This would open many doors as this capability could then be applied to any function producing hardware arithmetic operators which are specifically and precisely designed for FPGAs. This is a long term goal that we keep in mind and that we wish to achieve in collaboration with hardware designers to have a better understanding of the hardware. As any problem we tackle with operations research tools, the domain-specific expertise is absolutely necessary. By misunderstanding the problem, we will surely trigger the wrong switch and, at best, lose valuable time.

This work on the MCM problem using MILP-based models also opens more **theoretical** doors which we want to explore in the future:

- **Complexity analysis.** We are interested whether the MCM problem is NP-hard, as it is conjectured in many previous papers [BH91, DM94, Gus08, AFM15, Kum18]? Thong and Nicolici [TN11] established that while many similar problems are NP-hard, this has not been proven for the MCM problem. Thong and Nicolici also noted the existing proofs on related problems do not hold for MCM. This represents an issue, as most of the techniques we imagine for solving MCM are built thinking that this problem is NP-hard. Proving it in a near future would comfort us that we are indeed going in the right direction. It would be a chance to raise the interest of researchers from computational complexity theory towards hardware design problems.

- **Statistical comparison.** In Chapter 2, we recalled that solvers are not deterministic. To our knowledge, the performance variability phenomenon has never been rigorously taken into account when comparing models together or with other approaches. However, to assert significant difference between approaches involving MILP-based model, we need statistical hypothesis testing. As we established, this is not trivial and we need to confront our practice with statisticians. In the future, providing a toolbox to ease this comparison process is one of our mid term goals.

Finally, we studied the MCM problem and one application through the IIR filter design. We believe that our approach to these **hardware problems** can be also used for other complete **applications** or larger hardware operators:

- **Generic multiplications.** Generic multiplications, for instance, have already benefited from MILP modeling [KKIZ17, BKdD21] and we believe that these models could be improved or extended to take into account more parameters such as variable precision, which can help to reduce the power consumption when the application does not need the highest precision all the time. We already have started to work on transprecision multipliers with Andreas Böttcher and Martin Kumm from Fulda University of Applied Sciences, which we plan to finalize in short term.

- **Floating-point function approximation.** Our work on the tMCM problem involved an interesting error analysis and we would like to keep working on problems which require optimized hardware with a guaranteed error bound. In particular, function approximation for their implementation on FPGA is a topic which interestingly aggregates computer arithmetic, hardware knowledge and optimization. Our goal is to investigate this topic in the mid-term future. An interesting theoretical component to this work would be to model FP numbers within MILP models. Indeed, we believe it might be feasible to model the mantissa and the exponent separately, as two integers, and, this way, to perform nonlinear operations in MILP. This could permit to embed in MILP models real$\times$real multiplication for the first time.

- **Towards new applications.** Over the long run, we are interested in a wider knowledge of possible applications. For instance, we are already working with Vincent Lostanlen on new digital filter operators for environmental acoustics and plan to finalize this work in near future. Our goal is to reduce the power consumption of hardware circuits that we intend to use to detect birds' presence [LSF$^+$18]. Other examples of possible applications are neural networks and cryptography which involve a large number of multiplications by constants similarly to the MCM problem. However, usually, they involve matrix or very large constant multiplications. This addresses similar, but new, problems which could probably benefit from our expertise and we plan to investigate these interesting applications.

Overall, we plan to continue paving the way towards optimized hardware operators.

Appendices

APPENDIX A

MILP Models

A.1 High-level models – Chapter 4

A.1.1 Satisfiability

$$c_0 = 1 \tag{C1.1}$$

$$c_a^{\text{nsh}} = c_{a,l}^{\text{sh,sg}} + c_{a,r}^{\text{sh,sg}} \qquad \forall a \in [\![1; N]\!] \tag{C1.2}$$

$$c_a^{\text{nsh}} = 2^{s_a} c_a \quad \text{if } \Psi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!], s \in [\![0; w]\!] \tag{C1.3}$$

$$\sum_{s=0}^{w} \Psi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!] \tag{C1.4}$$

$$\Phi_{a,0} + \Psi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!] \tag{C1.5}$$

$$c_a = 2c_a^{\text{odd}} + 1 \qquad \forall a \in [\![1; N]\!] \tag{C1.6}$$

$$c_{a,i} = c_k \quad \text{if } c_{a,i,k} = 1 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, k \in [\![0; a-1]\!] \tag{C1.7}$$

$$\sum_{k=0}^{a-1} c_{a,i,k} = 1 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \tag{C1.8}$$

$$c_{a,l}^{\text{sh}} = 2^s c_{a,l} \quad \text{if } \Phi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!], s \in [\![0; w]\!] \tag{C1.9}$$

$$\sum_{s=0}^{w} \Phi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!] \tag{C1.10}$$

$$c_{a,i}^{\text{sh,sg}} = -c_{a,i}^{\text{sh}} \quad \text{if } \sigma_{a,i} = 1 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \tag{C1.11}$$

$$c_{a,i}^{\text{sh,sg}} = c_{a,i}^{\text{sh}} \quad \text{if } \sigma_{a,i} = 0 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \tag{C1.12}$$

$$\sigma_{a,l} + \sigma_{a,r} \leq 1 \qquad \forall a \in [\![1; N]\!] \tag{C1.13}$$

$$c_a = C_j \quad \text{if } o_{a,j} = 1 \qquad \forall a \in [\![0; N]\!], j \in [\![1; |C|]\!] \tag{C1.14}$$

$$\sum_{a=0}^{N} o_{a,j} = 1 \qquad \forall j \in [\![1; |C|]\!] \tag{C1.15}$$

Variables:

$$c_a \in [\![1; 2^w]\!] \qquad \forall a \in [\![0; N]\!] \tag{C1.16}$$
$$c_a^{\text{nsh}} \in [\![1; 2^{w+1}]\!] \qquad \forall a \in [\![1; N]\!] \tag{C1.17}$$
$$c_a^{\text{odd}} \in \mathbb{N} \qquad \forall a \in [\![1; N]\!] \tag{C1.18}$$
$$c_{a,i} \in [\![0; 2^w]\!] \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \tag{C1.19}$$
$$c_{a,l}^{\text{sh}} \in [\![0; 2^{w+1}]\!] \qquad \forall a \in [\![1; N]\!] \tag{C1.20}$$
$$c_{a,i}^{\text{sh,sg}} \in [\![-2^{w+1}; 2^{w+1}]\!] \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \tag{C1.21}$$
$$\sigma_{a,i} \in \{0, 1\} \qquad \forall a \in [\![1; N]\!] \tag{C1.22}$$
$$c_{a,i,k} \in \{0, 1\} \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, \forall k \in [\![0; a-1]\!] \tag{C1.23}$$
$$\Phi_{a,s} \in \{0, 1\} \qquad \forall a \in [\![1; N]\!], s \in [\![0; w]\!] \tag{C1.24}$$
$$\Psi_{a,s} \in \{0, 1\} \qquad \forall a \in [\![1; N]\!], s \in [\![0; w]\!] \tag{C1.25}$$
$$o_{a,j} \in \{0, 1\} \qquad \forall a \in [\![1; N]\!], j \in [\![1; |C|]\!] \tag{C1.26}$$

A.1.2 Minimization

$$\min \sum_{a=1}^{N} u_a \tag{O2}$$

$$c_0 = 1 \tag{C2.1}$$
$$c_a^{\text{nsh}} = c_{a,l}^{\text{sh,sg}} + c_{a,r}^{\text{sh,sg}} \qquad \forall a \in [\![1; N]\!] \tag{C2.2}$$
$$c_a^{\text{nsh}} = 2^s c_a \quad \text{if } \Psi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!], s \in [\![0; w]\!] \tag{C2.3}$$
$$\sum_{s=0}^{w} \Psi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!] \tag{C2.4}$$
$$\Phi_{a,0} + \Psi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!] \tag{C2.5}$$
$$c_a = 2c_a^{\text{odd}} + 1 \qquad \forall a \in [\![1; N]\!] \tag{C2.6}$$
$$c_{a,i} = c_k \quad \text{if } c_{a,i,k} = 1 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, k \in [\![0; a-1]\!] \tag{C2.7}$$
$$\sum_{k=0}^{a-1} c_{a,i,k} = 1 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \tag{C2.8}$$
$$c_{a,l}^{\text{sh}} = 2^s c_{a,l} \quad \text{if } \Phi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!], s \in [\![0; w]\!] \tag{C2.9}$$
$$\sum_{s=0}^{w} \Phi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!] \tag{C2.10}$$
$$c_{a,i}^{\text{sh,sg}} = -c_{a,i}^{\text{sh}} \quad \text{if } \sigma_{a,i} = 1 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \tag{C2.11}$$

$$c_{a,i}^{\text{sh,sg}} = c_{a,i}^{\text{sh}} \quad \text{if } \sigma_{a,i} = 0 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \tag{C2.12}$$
$$\sigma_{a,l} + \sigma_{a,r} \leq 1 \qquad \forall a \in [\![1; N]\!] \tag{C2.13}$$
$$c_a = C_j \quad \text{if } o_{a,j} = 1 \qquad \forall a \in [\![0; N]\!], j \in [\![1; |C|]\!] \tag{C2.14}$$
$$\sum_{a=0}^{N} o_{a,j} = 1 \qquad \forall j \in [\![1; |C|]\!] \tag{C2.15}$$
$$u_{a-1} \geq u_a \qquad \forall a \in [\![1; N]\!] \tag{C2.16}$$
$$c_a = 0 \quad \text{if } u_a = 0 \qquad \forall a \in [\![1; N]\!] \tag{C2.17}$$
$$c_{a,i,k} = 0 \quad \text{if } u_a = 0 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, k \in [\![0; a-1]\!] \tag{C2.18}$$

Variables:

$$c_a \in [\![0; 2^w]\!] \qquad \forall a \in [\![0; N]\!] \tag{C2.19}$$
$$u_a \in \{0, 1\} \qquad \forall a \in [\![0; N]\!] \tag{C2.20}$$
$$c_a^{\text{nsh}} \in [\![1; 2^{w+1}]\!] \qquad \forall a \in [\![1; N]\!] \tag{C2.21}$$
$$c_a^{\text{odd}} \in \mathbb{N} \qquad \forall a \in [\![1; N]\!] \tag{C2.22}$$
$$c_{a,i} \in [\![0; 2^w]\!] \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \tag{C2.23}$$
$$c_{a,l}^{\text{sh}} \in [\![0; 2^{w+1}]\!] \qquad \forall a \in [\![1; N]\!] \tag{C2.24}$$
$$c_{a,i}^{\text{sh,sg}} \in [\![-2^{w+1}; 2^{w+1}]\!] \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \tag{C2.25}$$
$$\sigma_{a,i} \in \{0, 1\} \qquad \forall a \in [\![1; N]\!] \tag{C2.26}$$
$$c_{a,i,k} \in \{0, 1\} \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, \forall k \in [\![0; a-1]\!] \tag{C2.27}$$
$$\Phi_{a,s} \in \{0, 1\} \qquad \forall a \in [\![1; N]\!], s \in [\![0; w]\!] \tag{C2.28}$$
$$\Psi_{a,s} \in \{0, 1\} \qquad \forall a \in [\![1; N]\!], s \in [\![0; w]\!] \tag{C2.29}$$
$$o_{a,j} \in \{0, 1\} \qquad \forall a \in [\![1; N]\!], j \in [\![1; |C|]\!] \tag{C2.30}$$

A.1.3 MILP model with adder depth

$$\min \sum_{a=1}^{N} u_a \tag{O3}$$

$$c_0 = 1 \tag{C3.1}$$
$$c_a^{\text{nsh}} = c_{a,l}^{\text{sh,sg}} + c_{a,r}^{\text{sh,sg}} \qquad \forall a \in [\![1; N]\!] \tag{C3.2}$$
$$c_a^{\text{nsh}} = 2^s c_a \quad \text{if } \Psi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!], s \in [\![0; w]\!] \tag{C3.3}$$
$$\sum_{s=0}^{w} \Psi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!] \tag{C3.4}$$

$$\Phi_{a,0} + \Psi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C3.5)}$$

$$c_a = 2c_a^{\text{odd}} + 1 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C3.6)}$$

$$c_{a,i} = c_k \quad \text{if } c_{a,i,k} = 1 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, k \in [\![0; a-1]\!] \qquad \text{(C3.7)}$$

$$\sum_{k=0}^{a-1} c_{a,i,k} = 1 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \qquad \text{(C3.8)}$$

$$c_{a,l}^{\text{sh}} = 2^s c_{a,l} \quad \text{if } \Phi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!], s \in [\![0; w]\!] \qquad \text{(C3.9)}$$

$$\sum_{s=0}^{w} \Phi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C3.10)}$$

$$c_{a,i}^{\text{sh,sg}} = -c_{a,i}^{\text{sh}} \quad \text{if } \sigma_{a,i} = 1 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \qquad \text{(C3.11)}$$

$$c_{a,i}^{\text{sh,sg}} = c_{a,i}^{\text{sh}} \quad \text{if } \sigma_{a,i} = 0 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \qquad \text{(C3.12)}$$

$$\sigma_{a,l} + \sigma_{a,r} \leq 1 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C3.13)}$$

$$c_a = C_j \quad \text{if } o_{a,j} = 1 \qquad \forall a \in [\![0; N]\!], j \in [\![1; |C|]\!] \qquad \text{(C3.14)}$$

$$\sum_{a=0}^{N} o_{a,j} = 1 \qquad \forall j \in [\![1; |C|]\!] \qquad \text{(C3.15)}$$

$$u_{a-1} \geq u_a \qquad \forall a \in [\![1; N]\!] \qquad \text{(C3.16)}$$

$$ad_0 = 0 \qquad \text{(C3.17)}$$

$$ad_a \geq ad_{a,l} + 1 \qquad \forall a \in [\![1; \text{N}]\!] \qquad \text{(C3.18)}$$

$$ad_a \geq ad_{a,r} + 1 \qquad \forall a \in [\![1; \text{N}]\!] \qquad \text{(C3.19)}$$

$$ad_a \leq ad_{a,l} + 1 + N\,ad_a^b \qquad \forall a \in [\![1; \text{N}]\!] \qquad \text{(C3.20)}$$

$$ad_a \leq ad_{a,r} + 1 + N \times \left(1 - ad_a^b\right) \qquad \forall a \in [\![1; \text{N}]\!] \qquad \text{(C3.21)}$$

$$ad_{a,i} = ad_k \quad \text{if } c_{a,i,k} = 1, \qquad \forall a \in [\![1; \text{N}]\!], i \in \{l, r\}, k \in [\![0; a-1]\!] \qquad \text{(C3.22)}$$

$$ad_{\max} \geq ad_a \qquad \forall a \in [\![1; \text{N}]\!] \qquad \text{(C3.23)}$$

$$c_a = 0 \quad \text{if } u_a = 0 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C3.24)}$$

$$c_{a,i,k} = 0 \quad \text{if } u_a = 0 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, k \in [\![0; a-1]\!] \qquad \text{(C3.25)}$$

Variables:

$$c_a \in [\![0; 2^w]\!] \qquad \forall a \in [\![0; N]\!] \qquad \text{(C3.26)}$$

$$ad_a \in [\![0; N]\!] \qquad \forall a \in [\![0; N]\!] \qquad \text{(C3.27)}$$

$$ad_{\max} \in [\![0; N]\!] \qquad \text{(C3.28)}$$

$$ad_a^b \in \{0, 1\} \qquad \forall a \in [\![1; N]\!] \qquad \text{(C3.29)}$$

$$u_a \in \{0, 1\} \qquad \forall a \in [\![0; N]\!] \qquad \text{(C3.30)}$$

$$c_a^{\text{nsh}} \in [\![1; 2^{w+1}]\!] \qquad \forall a \in [\![1; N]\!] \qquad \text{(C3.31)}$$

$$c_a^{\text{odd}} \in \mathbb{N} \qquad \forall a \in [\![1; N]\!] \qquad \text{(C3.32)}$$

$$c_{a,i} \in [\![0; 2^w]\!] \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \qquad \text{(C3.33)}$$

$$ad_{a,i} \in [\![0; N]\!] \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \tag{C3.34}$$
$$c_{a,l}^{\text{sh}} \in [\![0; 2^{w+1}]\!] \qquad \forall a \in [\![1; N]\!] \tag{C3.35}$$
$$c_{a,i}^{\text{sh,sg}} \in [\![-2^{w+1}; 2^{w+1}]\!] \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \tag{C3.36}$$
$$\sigma_{a,i} \in \{0, 1\} \qquad \forall a \in [\![1; N]\!] \tag{C3.37}$$
$$c_{a,i,k} \in \{0, 1\} \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, \forall k \in [\![0; a-1]\!] \tag{C3.38}$$
$$\Phi_{a,s} \in \{0, 1\} \qquad \forall a \in [\![1; N]\!], s \in [\![0; w]\!] \tag{C3.39}$$
$$\Psi_{a,s} \in \{0, 1\} \qquad \forall a \in [\![1; N]\!], s \in [\![0; w]\!] \tag{C3.40}$$
$$o_{a,j} \in \{0, 1\} \qquad \forall a \in [\![1; N]\!], j \in [\![1; |C|]\!] \tag{C3.41}$$

+ Tightening constraints:

$$D_{\text{LB}}(C_j) \times o_{a,j} \leq ad_a \qquad \forall a \in [\![1; N]\!], j \in [\![1; |C|]\!] \tag{C3.42}$$
$$ad_{a-1} \leq ad_a + N(1 - u_a) \qquad \forall a \in [\![1; N]\!] \tag{C3.43}$$

+ User-given bound:

$$ad_{\max} \leq \overline{ad} \tag{C3.44}$$

+ Minimize adder depth:

$$\min \sum_{a=1}^{\text{N}} (\text{N}u_a) + ad_{\max}. \tag{O3}$$

A.2 Low-level models – Chapter 5

$$\min \sum_{a=1}^{N} \text{B}_a^u \tag{O4}$$

$$c_0 = 1 \tag{C4.1}$$
$$c_a^{\text{nsh}} = c_{a,l}^{\text{sh,sg}} + c_{a,r}^{\text{sh,sg}} \qquad \forall a \in [\![1; N]\!] \tag{C4.2}$$
$$c_a^{\text{nsh}} = 2^s c_a \quad \text{if } \Psi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!], s \in [\![0; w]\!] \tag{C4.3}$$
$$\sum_{s=0}^{w} \Psi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!] \tag{C4.4}$$
$$\Phi_{a,0} + \Psi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!] \tag{C4.5}$$
$$c_a = 2c_a^{\text{odd}} + 1 \qquad \forall a \in [\![1; N]\!] \tag{C4.6}$$
$$c_{a,i} = c_k \quad \text{if } c_{a,i,k} = 1 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, k \in [\![0; a-1]\!] \tag{C4.7}$$

$$\sum_{k=0}^{a-1} c_{a,i,k} = 1 \qquad \forall a \in [\![1;N]\!], i \in \{l,r\} \qquad \text{(C4.8)}$$

$$c_{a,l}^{\text{sh}} = 2^s c_{a,l} \quad \text{if } \Phi_{a,s} = 1 \qquad \forall a \in [\![1;N]\!], s \in [\![0;w]\!] \qquad \text{(C4.9)}$$

$$\sum_{s=0}^{w} \Phi_{a,s} = 1 \qquad \forall a \in [\![1;N]\!] \qquad \text{(C4.10)}$$

$$c_{a,i}^{\text{sh,sg}} = -c_{a,i}^{\text{sh}} \quad \text{if } \sigma_{a,i} = 1 \qquad \forall a \in [\![1;N]\!], i \in \{l,r\} \qquad \text{(C4.11)}$$

$$c_{a,i}^{\text{sh,sg}} = c_{a,i}^{\text{sh}} \quad \text{if } \sigma_{a,i} = 0 \qquad \forall a \in [\![1;N]\!], i \in \{l,r\} \qquad \text{(C4.12)}$$

$$\sigma_{a,l} + \sigma_{a,r} \leq 1 \qquad \forall a \in [\![1;N]\!] \qquad \text{(C4.13)}$$

$$c_a = C_j \quad \text{if } o_{a,j} = 1 \qquad \forall a \in [\![0;N]\!], j \in [\![1;|C|]\!] \qquad \text{(C4.14)}$$

$$\sum_{a=0}^{N} o_{a,j} = 1 \qquad \forall j \in [\![1;|C|]\!] \qquad \text{(C4.15)}$$

$$u_{a-1} \geq u_a \qquad \forall a \in [\![1;N]\!] \qquad \text{(C4.16)}$$

$$c_a = 0 \quad \text{if } u_a = 0 \qquad \forall a \in [\![1;N]\!] \qquad \text{(C4.17)}$$

$$c_{a,i,k} = 0 \quad \text{if } u_a = 0 \qquad \forall a \in [\![1;N]\!], i \in \{l,r\}, k \in [\![0;a-1]\!] \qquad \text{(C4.18)}$$

$$\text{B}_a = \text{msb}_a + 1 - s_a - \psi_a - g_a \qquad \forall a \in [\![1;N]\!] \qquad \text{(C4.19)}$$

$$\text{B}_a^u = \text{B}_a \quad \text{if } u_a = 1 \qquad \forall a \in [\![1;N]\!] \qquad \text{(C4.20)}$$

$$\text{B}_a^u = 0 \quad \text{if } u_a = 0 \qquad \forall a \in [\![1;N]\!] \qquad \text{(C4.21)}$$

$$\text{msb}_0 = w_{in} - 1 \qquad \text{(C4.22)}$$

$$\text{msb}_{a,i} = \text{msb}_k \quad \text{if } c_{a,i,k} = 1 \qquad \forall a \in [\![1;N]\!], i \in \{l,r\}, k \in [\![0;a-1]\!] \qquad \text{(C4.23)}$$

$$g_a \leq \text{msb}_a \qquad \forall a \in [\![1;N]\!] \qquad \text{(C4.24)}$$

$$g_a \leq 0 \quad \text{if } \sigma_{a,r} = 1 \qquad \forall a \in [\![1;N]\!] \qquad \text{(C4.25)}$$

$$g_a \leq s_{a,l} \quad \text{if } s_{a,l} \leq \text{msb}_{a,r} \qquad \forall a \in [\![1;N]\!] \qquad \text{(C4.26)}$$

$$g_a \leq s_{a,l} \quad \text{if } \sigma_{a,l} = 1 \qquad \forall a \in [\![1;N]\!] \qquad \text{(C4.27)}$$

$$s_{a,l} = \sum_{s=1}^{w} s\Phi_{a,s} \qquad \forall a \in [\![1;N]\!] \qquad \text{(C4.28)}$$

$$s_a = \sum_{s=-w}^{-1} s\Psi_{a,s} \qquad \forall a \in [\![1;N]\!] \qquad \text{(C4.29)}$$

$$\text{msb}_{a,l} + s_{a,l} + s_a + 1 \leq \text{msb}_a \quad \text{if } \psi_a = 1 \qquad \forall a \in [\![1;N]\!] \qquad \text{(C4.30)}$$

$$\text{msb}_{a,r} + s_a + 1 \leq \text{msb}_a \quad \text{if } \psi_a = 1 \qquad \forall a \in [\![1;N]\!] \qquad \text{(C4.31)}$$

$$2^{b+1} - 1 \geq (2^{w_{in}} - 1)\, c_a \quad \text{if } \text{msb}_{a,b}^B = 1 \qquad \forall a \in [\![1;N]\!], b \in [\![1;w+w_{in}-1]\!] \qquad \text{(C4.32)}$$

$$\sum_{b=1}^{w+w_{in}-1} \text{msb}_{a,b}^B = 1 \qquad \forall a \in [\![1;N]\!] \qquad \text{(C4.33)}$$

$$\sum_{b=1}^{w+w_{in}-1} b \times \mathrm{msb}_{a,b}^{B} = \mathrm{msb}_a \qquad \forall a \in [\![1;N]\!] \tag{C4.34}$$

Variables:

$$c_a \in [\![0;2^w]\!] \qquad \forall a \in [\![0;N]\!] \tag{C4.35}$$
$$u_a \in \{0,1\} \qquad \forall a \in [\![0;N]\!] \tag{C4.36}$$
$$c_a^{\mathrm{nsh}} \in [\![1;2^{w+1}]\!] \qquad \forall a \in [\![1;N]\!] \tag{C4.37}$$
$$c_a^{\mathrm{odd}} \in \mathbb{N} \qquad \forall a \in [\![1;N]\!] \tag{C4.38}$$
$$c_{a,i} \in [\![0;2^w]\!] \qquad \forall a \in [\![1;N]\!], i \in \{l,r\} \tag{C4.39}$$
$$c_{a,l}^{\mathrm{sh}} \in [\![0;2^{w+1}]\!] \qquad \forall a \in [\![1;N]\!] \tag{C4.40}$$
$$c_{a,i}^{\mathrm{sh,sg}} \in [\![-2^{w+1};2^{w+1}]\!] \qquad \forall a \in [\![1;N]\!], i \in \{l,r\} \tag{C4.41}$$
$$\sigma_{a,i} \in \{0,1\} \qquad \forall a \in [\![1;N]\!] \tag{C4.42}$$
$$c_{a,i,k} \in \{0,1\} \qquad \forall a \in [\![1;N]\!], i \in \{l,r\}, \forall k \in [\![0;a-1]\!] \tag{C4.43}$$
$$\Phi_{a,s} \in \{0,1\} \qquad \forall a \in [\![1;N]\!], s \in [\![0;w]\!] \tag{C4.44}$$
$$\Psi_{a,s} \in \{0,1\} \qquad \forall a \in [\![1;N]\!], s \in [\![0;w]\!] \tag{C4.45}$$
$$o_{a,j} \in \{0,1\} \qquad \forall a \in [\![1;N]\!], j \in [\![1;|C|]\!] \tag{C4.46}$$
$$\mathrm{msb}_a \in [\![0;w+w_{in}-1]\!] \qquad \forall a \in [\![0;N]\!] \tag{C4.47}$$
$$\mathrm{msb}_{a,i} \in [\![0;w+w_{in}-1]\!] \qquad \forall a \in [\![1;N]\!], i \in \{l,r\} \tag{C4.48}$$
$$\mathrm{msb}_{a,t}^{B} \in \{0,1\} \qquad \forall a \in [\![1;N]\!], t \in [\![1;w+w_{in}-1]\!] \tag{C4.49}$$
$$B_a \in [\![0;w+w_{in}]\!] \qquad \forall a \in [\![1;N]\!] \tag{C4.50}$$
$$B_a^u \in [\![0;w+w_{in}]\!] \qquad \forall a \in [\![1;N]\!] \tag{C4.51}$$
$$g_a \in [\![0;w+w_{in}]\!] \qquad \forall a \in [\![1;N]\!] \tag{C4.52}$$
$$\psi_a \in \{0,1\} \qquad \forall a \in [\![1;N]\!] \tag{C4.53}$$
$$s_{a,l} \in [\![0;w]\!] \qquad \forall a \in [\![1;N]\!] \tag{C4.54}$$
$$s_a \in [\![-w;0]\!] \qquad \forall a \in [\![1;N]\!] \tag{C4.55}$$

A.3 Model with truncations – Chapter 6

$$\min \sum_{a=1}^{N} \mathrm{B}_a^u \tag{O5}$$

$$c_0 = 1 \tag{C5.1}$$
$$c_a^{\mathrm{nsh}} = c_{a,l}^{\mathrm{sh,sg}} + c_{a,r}^{\mathrm{sh,sg}} \qquad \forall a \in [\![1;N]\!] \tag{C5.2}$$

$$c_a^{\text{nsh}} = 2^s c_a \quad \text{if } \Psi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!], s \in [\![0; w]\!] \qquad \text{(C5.3)}$$

$$\sum_{s=0}^{w} \Psi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C5.4)}$$

$$\Phi_{a,0} + \Psi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C5.5)}$$

$$c_a = 2c_a^{\text{odd}} + 1 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C5.6)}$$

$$c_{a,i} = c_k \quad \text{if } c_{a,i,k} = 1 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, k \in [\![0; a-1]\!] \qquad \text{(C5.7)}$$

$$\sum_{k=0}^{a-1} c_{a,i,k} = 1 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \qquad \text{(C5.8)}$$

$$c_{a,l}^{\text{sh}} = 2^s c_{a,l} \quad \text{if } \Phi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!], s \in [\![0; w]\!] \qquad \text{(C5.9)}$$

$$\sum_{s=0}^{w} \Phi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C5.10)}$$

$$c_{a,i}^{\text{sh,sg}} = -c_{a,i}^{\text{sh}} \quad \text{if } \sigma_{a,i} = 1 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \qquad \text{(C5.11)}$$

$$c_{a,i}^{\text{sh,sg}} = c_{a,i}^{\text{sh}} \quad \text{if } \sigma_{a,i} = 0 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \qquad \text{(C5.12)}$$

$$\sigma_{a,l} + \sigma_{a,r} \leq 1 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C5.13)}$$

$$c_a = C_j \quad \text{if } o_{a,j} = 1 \qquad \forall a \in [\![0; N]\!], j \in [\![1; |C|]\!] \qquad \text{(C5.14)}$$

$$\sum_{a=0}^{N} o_{a,j} = 1 \qquad \forall j \in [\![1; |C|]\!] \qquad \text{(C5.15)}$$

$$u_{a-1} \geq u_a \qquad \forall a \in [\![1; N]\!] \qquad \text{(C5.16)}$$

$$c_a = 0 \quad \text{if } u_a = 0 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C5.17)}$$

$$c_{a,i,k} = 0 \quad \text{if } u_a = 0 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, k \in [\![0; a-1]\!] \qquad \text{(C5.18)}$$

$$\text{B}_a = \text{msb}_a + 1 - s_a - \psi_a - g_a \qquad \forall a \in [\![1; N]\!] \qquad \text{(C5.19)}$$

$$\text{B}_a^u = \text{B}_a \quad \text{if } u_a = 1 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C5.20)}$$

$$\text{B}_a^u = 0 \quad \text{if } u_a = 0 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C5.21)}$$

$$\text{msb}_0 = w_{in} - 1 \qquad \text{(C5.22)}$$

$$\text{msb}_{a,i} = \text{msb}_k \quad \text{if } c_{a,i,k} = 1 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, k \in [\![0; a-1]\!] \qquad \text{(C5.23)}$$

$$g_a \leq \text{msb}_a \qquad \forall a \in [\![1; N]\!] \qquad \text{(C5.24)}$$

$$g_a \leq t_a^{\max} \quad \text{if } t_{a,l} \leq \text{msb}_{a,r} \wedge t_{a,r} \leq \text{msb}_{a,l} \qquad \forall a \in [\![1; N]\!] \qquad \text{(C5.25)}$$

$$g_a \leq t_{a,l} \quad \text{if } \sigma_{a,l} = 1 \wedge \sigma_{a,r} = 0 \qquad \forall a \in [\![1; N]\!], \qquad \text{(C5.26)}$$

$$g_a \leq t_{a,r} \quad \text{if } \sigma_{a,l} = 0 \wedge \sigma_{a,r} = 1 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C5.27)}$$

$$s_{a,l} = \sum_{s=1}^{w} s\Phi_{a,s} \qquad \forall a \in [\![1; N]\!] \qquad \text{(C5.28)}$$

$$s_a = \sum_{s=-w}^{-1} s\Psi_{a,s} \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.29)}$$

$$\mathrm{msb}_{a,l} + s_{a,l} + s_a + 1 \leq \mathrm{msb}_a \quad \text{if } \psi_a = 1 \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.30)}$$

$$\mathrm{msb}_{a,r} + s_a + 1 \leq \mathrm{msb}_a \quad \text{if } \psi_a = 1 \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.31)}$$

$$2^{b+1} - 1 \geq (2^{w_{in}} - 1)\, c_a + \varepsilon_a^{\mathrm{sup}} \quad \text{if } \mathrm{msb}_{a,b}^B = 1 \qquad \forall a \in [\![1;N]\!], b \in [\![1; w + w_{in} - 1]\!] \qquad \text{(C5.32)}$$

$$\sum_{b=1}^{w+w_{in}-1} \mathrm{msb}_{a,b}^B = 1 \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.33)}$$

$$\sum_{b=1}^{w+w_{in}-1} b \times \mathrm{msb}_{a,b}^B = \mathrm{msb}_a \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.34)}$$

$$\varepsilon_a^{\mathrm{inf,nsh}} = \varepsilon_{a,l}^{\mathrm{inf,sh}} + \varepsilon_{t_{a,l}} + \varepsilon_{a,r}^{\mathrm{inf,sh}} + \varepsilon_{t_{a,r}} \quad \text{if } \sigma_{a,l} = \sigma_{a,r} \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.35)}$$

$$\varepsilon_a^{\mathrm{inf,nsh}} = \varepsilon_{a,l}^{\mathrm{inf,sh}} + \varepsilon_{t_{a,l}} + \varepsilon_{a,r}^{\mathrm{sup,sh}} + \varepsilon_{t_{a,r}} \quad \text{if } \sigma_{a,r} = 1 \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.36)}$$

$$\varepsilon_a^{\mathrm{inf,nsh}} = \varepsilon_{a,l}^{\mathrm{sup,sh}} + \varepsilon_{t_{a,l}} + \varepsilon_{a,r}^{\mathrm{inf,sh}} + \varepsilon_{t_{a,r}} \quad \text{if } \sigma_{a,l} = 1 \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.37)}$$

$$\varepsilon_a^{\mathrm{sup,nsh}} = \varepsilon_{a,l}^{\mathrm{sup,sh}} + \varepsilon_{a,r}^{\mathrm{sup,sh}} \quad \text{if } \sigma_{a,l} = \sigma_{a,r} \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.38)}$$

$$\varepsilon_a^{\mathrm{sup,nsh}} = \varepsilon_{a,l}^{\mathrm{sup,sh}} + \varepsilon_{a,r}^{\mathrm{inf,sh}} \quad \text{if } \sigma_{a,r} = 1 \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.39)}$$

$$\varepsilon_a^{\mathrm{sup,nsh}} = \varepsilon_{a,l}^{\mathrm{inf,sh}} + \varepsilon_{a,r}^{\mathrm{sup,sh}} \quad \text{if } \sigma_{a,l} = 1 \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.40)}$$

$$\varepsilon_a^{\mathrm{inf,nsh}} = 2^s \varepsilon_a^{\mathrm{inf}} \quad \text{if } \Psi_{a,s} = 1 \qquad \forall a \in [\![1;N]\!], s \in [\![0;w]\!] \qquad \text{(C5.41)}$$

$$\varepsilon_a^{\mathrm{sup,nsh}} = 2^s \varepsilon_a^{\mathrm{sup}} \quad \text{if } \Psi_{a,s} = 1 \qquad \forall a \in [\![1;N]\!], s \in [\![0;w]\!] \qquad \text{(C5.42)}$$

$$\varepsilon_{a,l}^{\mathrm{inf,sh}} = 2^s \varepsilon_{a,l}^{\mathrm{inf}} \quad \text{if } \Phi_{a,s} = 1 \qquad \forall a \in [\![1;N]\!], s \in [\![0;w]\!] \qquad \text{(C5.43)}$$

$$\varepsilon_{a,l}^{\mathrm{sup,sh}} = 2^s \varepsilon_{a,l}^{\mathrm{sup}} \quad \text{if } \Phi_{a,s} = 1 \qquad \forall a \in [\![1;N]\!], s \in [\![0;w]\!] \qquad \text{(C5.44)}$$

$$\varepsilon_{a,r}^{\mathrm{inf,sh}} = \varepsilon_{a,r}^{\mathrm{inf}} \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.45)}$$

$$\varepsilon_{a,r}^{\mathrm{sup,sh}} = \varepsilon_{a,r}^{\mathrm{sup}} \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.46)}$$

$$\varepsilon_{a,i}^{\mathrm{inf}} = \varepsilon_k^{\mathrm{inf}} \quad \text{if } c_{a,i,k} = 1 \qquad \forall a \in [\![1;N]\!], i \in \{l,r\}, k \in [\![0;a-1]\!] \qquad \text{(C5.47)}$$

$$\varepsilon_{a,i}^{\mathrm{sup}} = \varepsilon_k^{\mathrm{sup}} \quad \text{if } c_{a,i,k} = 1 \qquad \forall a \in [\![1;N]\!], i \in \{l,r\}, k \in [\![0;a-1]\!] \qquad \text{(C5.48)}$$

$$z_{a,i} = z_k \quad \text{if } c_{a,i,k} = 1 \qquad \forall a \in [\![1;N]\!], i \in \{l,r\}, k \in [\![0;a-1]\!] \qquad \text{(C5.49)}$$

$$z_{a,l}^{\mathrm{sh}} = z_{a,l} + s_{a,l} \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.50)}$$

$$z_{a,r}^{\mathrm{sh}} = z_{a,r} \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.51)}$$

$$z_{a,l}^{\mathrm{nsh}} = z_{a,l}^{\mathrm{sh}} \quad \text{if } z_{a,l}^{\mathrm{nsh},B} = 1 \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.52)}$$

$$z_{a,l}^{\mathrm{nsh}} = t_{a,l} \quad \text{if } z_{a,l}^{\mathrm{nsh},B} = 0 \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.53)}$$

$$z_{a,r}^{\text{nsh}} = z_{a,r}^{\text{sh}} \quad \text{if } z_{a,r}^{\text{nsh},B} = 1 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C5.54)}$$

$$z_{a,r}^{\text{nsh}} = t_{a,r} \quad \text{if } z_{a,r}^{\text{nsh},B} = 0 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C5.55)}$$

$$z_a^{\text{nsh}} \leq z_{a,l}^{\text{nsh}} \qquad \forall a \in [\![1; N]\!] \qquad \text{(C5.56)}$$

$$z_a^{\text{nsh}} \leq z_{a,r}^{\text{nsh}} \qquad \forall a \in [\![1; N]\!] \qquad \text{(C5.57)}$$

$$z_a^{\text{nsh}} = z_a + s_a \qquad \forall a \in [\![1; N]\!], s \in [\![0; w]\!] \qquad \text{(C5.58)}$$

$$z_0 = 0 \qquad \text{(C5.59)}$$

$$\sum_{b=0}^{w+w_{in}} t_{a,i,b} = 1 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \qquad \text{(C5.60)}$$

$$\sum_{b=0}^{w+w_{in}} z_{a,i,b} = 1 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \qquad \text{(C5.61)}$$

$$\sum_{b=0}^{w+w_{in}} b \times t_{a,i,b} = t_{a,i} \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \qquad \text{(C5.62)}$$

$$\sum_{b=0}^{w+w_{in}} b \times z_{a,i,b} = z_{a,i}^{\text{sh}} \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \qquad \text{(C5.63)}$$

$$\varepsilon_{a,i}^{T} = 2^{b-1}, \quad \text{if } t_{a,i,b} = 1 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, b \in [\![1; w + w_{in}]\!] \qquad \text{(C5.64)}$$

$$\varepsilon_{a,i}^{T} = 0, \quad \text{if } t_{a,i,0} = 1 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \qquad \text{(C5.65)}$$

$$\varepsilon_{a,i}^{Z} = 2^{b-1}, \quad \text{if } z_{a,i,b} = 1 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, b \in [\![1; w + w_{in}]\!] \qquad \text{(C5.66)}$$

$$\varepsilon_{a,i}^{Z} = 0, \quad \text{if } z_{a,i,0} = 1 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \qquad \text{(C5.67)}$$

$$\varepsilon_{t_{a,i}} \geq \varepsilon_{a,i}^{T} - \varepsilon_{a,i}^{Z} \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \qquad \text{(C5.68)}$$

$$\varepsilon_a^{\text{inf}} \leq \overline{\varepsilon} \qquad \forall a \in [\![1; N]\!] \qquad \text{(C5.69)}$$

$$\varepsilon_a^{\text{sup}} \leq \overline{\varepsilon} \qquad \forall a \in [\![1; N]\!] \qquad \text{(C5.70)}$$

$$\varepsilon_0^{\text{inf}} = 0 \qquad \text{(C5.71)}$$

$$\varepsilon_0^{\text{sup}} = 0 \qquad \text{(C5.72)}$$

$$t_{a,i} \geq z_{a,i}^{\text{sh}} \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \qquad \text{(C5.73)}$$

$$t_a^{\max} = t_{a,l} \quad \text{if } t_a^B = 1 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C5.74)}$$

$$t_a^{\max} = t_{a,r} \quad \text{if } t_a^B = 0 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C5.75)}$$

$$t_{a,l} = z_{a,l}^{\text{sh}} \quad \text{if } \sigma_{a,r} = 1 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C5.76)}$$

$$t_{a,r} = z_{a,r}^{\text{sh}} \quad \text{if } \sigma_{a,l} = 1 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C5.77)}$$

Variables:

$$c_a \in [\![0; 2^w]\!] \qquad \forall a \in [\![0; N]\!] \qquad \text{(C5.78)}$$

$$u_a \in \{0,1\} \qquad \forall a \in [\![0;N]\!] \qquad \text{(C5.79)}$$
$$c_a^{\text{nsh}} \in [\![1;2^{w+1}]\!] \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.80)}$$
$$c_a^{\text{odd}} \in \mathbb{N} \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.81)}$$
$$c_{a,i} \in [\![0;2^{w}]\!] \qquad \forall a \in [\![1;N]\!], i \in \{l,r\} \qquad \text{(C5.82)}$$
$$c_{a,l}^{\text{sh}} \in [\![0;2^{w+1}]\!] \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.83)}$$
$$c_{a,i}^{\text{sh,sg}} \in [\![-2^{w+1};2^{w+1}]\!] \qquad \forall a \in [\![1;N]\!], i \in \{l,r\} \qquad \text{(C5.84)}$$
$$\sigma_{a,i} \in \{0,1\} \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.85)}$$
$$c_{a,i,k} \in \{0,1\} \qquad \forall a \in [\![1;N]\!], i \in \{l,r\}, \forall k \in [\![0;a-1]\!] \qquad \text{(C5.86)}$$
$$\Phi_{a,s} \in \{0,1\} \qquad \forall a \in [\![1;N]\!], s \in [\![0;w]\!] \qquad \text{(C5.87)}$$
$$\Psi_{a,s} \in \{0,1\} \qquad \forall a \in [\![1;N]\!], s \in [\![0;w]\!] \qquad \text{(C5.88)}$$
$$o_{a,j} \in \{0,1\} \qquad \forall a \in [\![1;N]\!], j \in [\![1;|C|]\!] \qquad \text{(C5.89)}$$
$$\text{msb}_a \in [\![0;w+w_{in}-1]\!] \qquad \forall a \in [\![0;N]\!] \qquad \text{(C5.90)}$$
$$\text{msb}_{a,i} \in [\![0;w+w_{in}-1]\!] \qquad \forall a \in [\![1;N]\!], i \in \{l,r\} \qquad \text{(C5.91)}$$
$$\text{msb}_{a,t}^{B} \in \{0,1\} \qquad \forall a \in [\![1;N]\!], t \in [\![1;w+w_{in}-1]\!] \qquad \text{(C5.92)}$$
$$B_a \in [\![0;w+w_{in}]\!] \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.93)}$$
$$B_a^u \in [\![0;w+w_{in}]\!] \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.94)}$$
$$g_a \in [\![0;w+w_{in}]\!] \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.95)}$$
$$\psi_a \in \{0,1\} \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.96)}$$
$$s_{a,l} \in [\![0;w]\!] \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.97)}$$
$$s_a \in [\![-w;0]\!] \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.98)}$$
$$\varepsilon_a^{\text{inf,nsh}} \in [\![0;2^{w+w_{in}}]\!] \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.99)}$$
$$\varepsilon_{a,i}^{\text{inf,sh}} \in [\![0;2^{w+w_{in}}]\!] \qquad \forall a \in [\![1;N]\!], i \in \{l,r\} \qquad \text{(C5.100)}$$
$$\varepsilon_a^{\text{inf}} \in [\![0;2^{w+w_{in}}]\!] \qquad \forall a \in [\![0;N]\!] \qquad \text{(C5.101)}$$
$$\varepsilon_{a,i}^{\text{inf}} \in [\![0;2^{w+w_{in}}]\!] \qquad \forall a \in [\![1;N]\!], i \in \{l,r\} \qquad \text{(C5.102)}$$
$$\varepsilon_a^{\text{sup,nsh}} \in [\![0;2^{w+w_{in}}]\!] \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.103)}$$
$$\varepsilon_{a,i}^{\text{sup,sh}} \in [\![0;2^{w+w_{in}}]\!] \qquad \forall a \in [\![1;N]\!], i \in \{l,r\} \qquad \text{(C5.104)}$$
$$\varepsilon_a^{\text{sup}} \in [\![0;2^{w+w_{in}}]\!] \qquad \forall a \in [\![0;N]\!] \qquad \text{(C5.105)}$$
$$\varepsilon_{a,i}^{\text{sup}} \in [\![0;2^{w+w_{in}}]\!] \qquad \forall a \in [\![1;N]\!], i \in \{l,r\} \qquad \text{(C5.106)}$$
$$\varepsilon_{t_{a,i}} \in [\![0;2^{w+w_{in}}]\!] \qquad \forall a \in [\![1;N]\!], i \in \{l,r\} \qquad \text{(C5.107)}$$
$$z_{a,i}^{\text{sh}} \in [\![0;w+w_{in}]\!] \qquad \forall a \in [\![1;N]\!], i \in \{l,r\} \qquad \text{(C5.108)}$$
$$z_{a,i}^{\text{nsh}} \in [\![0;w+w_{in}]\!] \qquad \forall a \in [\![1;N]\!], i \in \{l,r\} \qquad \text{(C5.109)}$$
$$z_a^{\text{nsh}} \in [\![0;w+w_{in}]\!] \qquad \forall a \in [\![1;N]\!] \qquad \text{(C5.110)}$$

$$z_{a,i} \in [\![0; w + w_{in}]\!] \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \qquad \text{(C5.111)}$$
$$z_a \in [\![0; w + w_{in}]\!] \qquad \forall a \in [\![0; N]\!] \qquad \text{(C5.112)}$$
$$z_{a,i}^{\text{nsh},B} \in \{0, 1\} \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \qquad \text{(C5.113)}$$
$$z_{a,i,b} \in \{0, 1\} \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, b \in [\![0; w + w_{in}]\!] \qquad \text{(C5.114)}$$
$$t_{a,i,b} \in \{0, 1\} \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, b \in [\![0; w + w_{in}]\!] \qquad \text{(C5.115)}$$
$$\varepsilon_{a,i}^{Z} \in [\![0; 2^{w+w_{in}}]\!] \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \qquad \text{(C5.116)}$$
$$\varepsilon_{a,i}^{T} \in [\![0; 2^{w+w_{in}}]\!] \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \qquad \text{(C5.117)}$$
$$t_{a,i} \in [\![0; w + w_{in}]\!] \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \qquad \text{(C5.118)}$$
$$t_a^{\max} \in [\![0; w + w_{in}]\!] \qquad \forall a \in [\![1; N]\!] \qquad \text{(C5.119)}$$
$$t_a^{B} \in \{0, 1\} \qquad \forall a \in [\![1; N]\!] \qquad \text{(C5.120)}$$

A.4 Pipelined adder graph – Chapter 7

A.4.1 High-level model – PMCM-Adders

$$\min \sum_{a=0}^{\mathrm{N}} r_a \qquad \text{(O6)}$$

$$c_0 = 1 \qquad \text{(C6.1)}$$
$$c_a^{\text{nsh}} = c_{a,l}^{\text{sh,sg}} + c_{a,r}^{\text{sh,sg}} \qquad \forall a \in [\![1; N]\!] \qquad \text{(C6.2)}$$
$$c_a^{\text{nsh}} = 2^s c_a \quad \text{if } \Psi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!], s \in [\![0; w]\!] \qquad \text{(C6.3)}$$
$$\sum_{s=0}^{w} \Psi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C6.4)}$$
$$\Phi_{a,0} + \Psi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C6.5)}$$
$$c_a = 2c_a^{\text{odd}} + 1 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C6.6)}$$
$$c_{a,i} = c_k \quad \text{if } c_{a,i,k} = 1 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, k \in [\![0; a-1]\!] \qquad \text{(C6.7)}$$
$$\sum_{k=0}^{a-1} c_{a,i,k} = 1 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \qquad \text{(C6.8)}$$
$$c_{a,l}^{\text{sh}} = 2^s c_{a,l} \quad \text{if } \Phi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!], s \in [\![0; w]\!] \qquad \text{(C6.9)}$$
$$\sum_{s=0}^{w} \Phi_{a,s} = 1 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C6.10)}$$

$$c_{a,i}^{\text{sh,sg}} = -c_{a,i}^{\text{sh}} \quad \text{if } \sigma_{a,i} = 1 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \tag{C6.11}$$

$$c_{a,i}^{\text{sh,sg}} = c_{a,i}^{\text{sh}} \quad \text{if } \sigma_{a,i} = 0 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \tag{C6.12}$$

$$\sigma_{a,l} + \sigma_{a,r} \leq 1 \qquad \forall a \in [\![1; N]\!] \tag{C6.13}$$

$$c_a = C_j \quad \text{if } o_{a,j} = 1 \qquad \forall a \in [\![0; N]\!], j \in [\![1; |C|]\!] \tag{C6.14}$$

$$\sum_{a=0}^{N} o_{a,j} = 1 \qquad \forall j \in [\![1; |C|]\!] \tag{C6.15}$$

$$u_{a-1} \geq u_a \qquad \forall a \in [\![1; N]\!] \tag{C6.16}$$

$$c_a = 0 \quad \text{if } u_a = 0 \qquad \forall a \in [\![1; N]\!] \tag{C6.17}$$

$$c_{a,i,k} = 0 \quad \text{if } u_a = 0 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, k \in [\![0; a-1]\!] \tag{C6.18}$$

$$p_a \geq p_{a,l} + 1 \qquad \forall a \in [\![1; \mathrm{N}]\!] \tag{C6.19}$$

$$p_a \geq p_{a,r} + 1 \qquad \forall a \in [\![1; \mathrm{N}]\!] \tag{C6.20}$$

$$p_a \leq p_{a,l} + 1 + \mathrm{N} p_a^b \qquad \forall a \in [\![1; \mathrm{N}]\!] \tag{C6.21}$$

$$p_a \leq p_{a,r} + 1 + \mathrm{N} \times \left(1 - p_a^b\right) \qquad \forall a \in [\![1; \mathrm{N}]\!] \tag{C6.22}$$

$$p_{a,i} = p_k \quad \text{if } c_{a,i,k} = 1 \qquad \forall a \in [\![1; \mathrm{N}]\!], i \in \{l, r\}, k \in [\![0; a-1]\!] \tag{C6.23}$$

$$p_{\max} \geq p_a + 1 \qquad \forall a \in [\![1; \mathrm{N}]\!] \tag{C6.24}$$

$$p_k \leq N \times \overline{p}_{a,p} + p - 1 \quad \text{if } c_{k,i,a} = 1 \qquad \forall p \in [\![1; N+1]\!], a \in [\![0; N-1]\!], \forall k \in [\![a+1; N]\!], i \in \{l, r\} \tag{C6.25}$$

$$p_{\max} \leq (N+1) \times \overline{p}_{a,p} + p - 1 \quad \text{if } o_{a,j} = 1 \qquad \forall s \in [\![1; N+1]\!], a \in [\![0; N]\!], j \in [\![1; |C|]\!] \tag{C6.26}$$

$$\overline{p}_a \geq p \times \overline{p}_{a,p} \qquad \forall p \in [\![1; N+1]\!], a \in [\![0; N-1]\!] \tag{C6.27}$$

$$r_a = \overline{p}_a - p_a \qquad \forall a \in [\![0; N]\!] \tag{C6.28}$$

Variables:

$$c_a \in [\![0; 2^w]\!] \qquad \forall a \in [\![0; N]\!] \tag{C6.29}$$

$$u_a \in \{0, 1\} \qquad \forall a \in [\![0; N]\!] \tag{C6.30}$$

$$c_a^{\text{nsh}} \in [\![1; 2^{w+1}]\!] \qquad \forall a \in [\![1; N]\!] \tag{C6.31}$$

$$c_a^{\text{odd}} \in \mathbb{N} \qquad \forall a \in [\![1; N]\!] \tag{C6.32}$$

$$c_{a,i} \in [\![0; 2^w]\!] \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \tag{C6.33}$$

$$c_{a,l}^{\text{sh}} \in [\![0; 2^{w+1}]\!] \qquad \forall a \in [\![1; N]\!] \tag{C6.34}$$

$$c_{a,i}^{\text{sh,sg}} \in [\![-2^{w+1}; 2^{w+1}]\!] \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \tag{C6.35}$$

$$\sigma_{a,i} \in \{0, 1\} \qquad \forall a \in [\![1; N]\!] \tag{C6.36}$$

$$c_{a,i,k} \in \{0, 1\} \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, \forall k \in [\![0; a-1]\!] \tag{C6.37}$$

$$\Phi_{a,s} \in \{0, 1\} \qquad \forall a \in [\![1; N]\!], s \in [\![0; w]\!] \tag{C6.38}$$

$$\Psi_{a,s} \in \{0,1\} \qquad \forall a \in [\![1;N]\!], s \in [\![0;w]\!] \qquad \text{(C6.39)}$$
$$o_{a,j} \in \{0,1\} \qquad \forall a \in [\![1;N]\!], j \in [\![1;|C|]\!] \qquad \text{(C6.40)}$$
$$p_a \in [\![0;N]\!] \qquad \forall a \in [\![0;N]\!] \qquad \text{(C6.41)}$$
$$p_{a,i} \in [\![0;N]\!] \qquad \forall a \in [\![1;N]\!], i \in \{l,r\} \qquad \text{(C6.42)}$$
$$p_{\max} \in [\![0;N+1]\!] \qquad \text{(C6.43)}$$
$$\overline{p}_{a,p} \in \{0,1\} \qquad \forall a \in [\![0;N]\!], p \in [\![1;N+1]\!] \qquad \text{(C6.44)}$$
$$\overline{p}_a \in [\![0;N+1]\!] \qquad \forall a \in [\![0;N]\!] \qquad \text{(C6.45)}$$
$$\overline{r}_a \in [\![0;N+1]\!] \qquad \forall a \in [\![0;N]\!] \qquad \text{(C6.46)}$$

A.4.2 Low-level model – PMCM-Bits

$$\min \sum_{a=0}^{\mathrm{N}} C_a^r \qquad \text{(O7)}$$

$$c_0 = 1 \qquad \text{(C7.1)}$$
$$c_a^{\mathrm{nsh}} = c_{a,l}^{\mathrm{sh,sg}} + c_{a,r}^{\mathrm{sh,sg}} \qquad \forall a \in [\![1;N]\!] \qquad \text{(C7.2)}$$
$$c_a^{\mathrm{nsh}} = 2^s c_a \quad \text{if } \Psi_{a,s} = 1 \qquad \forall a \in [\![1;N]\!], s \in [\![0;w]\!] \qquad \text{(C7.3)}$$
$$\sum_{s=0}^{w} \Psi_{a,s} = 1 \qquad \forall a \in [\![1;N]\!] \qquad \text{(C7.4)}$$
$$\Phi_{a,0} + \Psi_{a,s} = 1 \qquad \forall a \in [\![1;N]\!] \qquad \text{(C7.5)}$$
$$c_a = 2c_a^{\mathrm{odd}} + 1 \qquad \forall a \in [\![1;N]\!] \qquad \text{(C7.6)}$$
$$c_{a,i} = c_k \quad \text{if } c_{a,i,k} = 1 \qquad \forall a \in [\![1;N]\!], i \in \{l,r\}, k \in [\![0;a-1]\!] \qquad \text{(C7.7)}$$
$$\sum_{k=0}^{a-1} c_{a,i,k} = 1 \qquad \forall a \in [\![1;N]\!], i \in \{l,r\} \qquad \text{(C7.8)}$$
$$c_{a,l}^{\mathrm{sh}} = 2^s c_{a,l} \quad \text{if } \Phi_{a,s} = 1 \qquad \forall a \in [\![1;N]\!], s \in [\![0;w]\!] \qquad \text{(C7.9)}$$
$$\sum_{s=0}^{w} \Phi_{a,s} = 1 \qquad \forall a \in [\![1;N]\!] \qquad \text{(C7.10)}$$
$$c_{a,i}^{\mathrm{sh,sg}} = -c_{a,i}^{\mathrm{sh}} \quad \text{if } \sigma_{a,i} = 1 \qquad \forall a \in [\![1;N]\!], i \in \{l,r\} \qquad \text{(C7.11)}$$
$$c_{a,i}^{\mathrm{sh,sg}} = c_{a,i}^{\mathrm{sh}} \quad \text{if } \sigma_{a,i} = 0 \qquad \forall a \in [\![1;N]\!], i \in \{l,r\} \qquad \text{(C7.12)}$$
$$\sigma_{a,l} + \sigma_{a,r} \leq 1 \qquad \forall a \in [\![1;N]\!] \qquad \text{(C7.13)}$$
$$c_a = C_j \quad \text{if } o_{a,j} = 1 \qquad \forall a \in [\![0;N]\!], j \in [\![1;|C|]\!] \qquad \text{(C7.14)}$$
$$\sum_{a=0}^{N} o_{a,j} = 1 \qquad \forall j \in [\![1;|C|]\!] \qquad \text{(C7.15)}$$

$$u_{a-1} \geq u_a \qquad \forall a \in [\![1; N]\!] \qquad \text{(C7.16)}$$

$$c_a = 0 \quad \text{if } u_a = 0 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C7.17)}$$

$$c_{a,i,k} = 0 \quad \text{if } u_a = 0 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, k \in [\![0; a-1]\!] \qquad \text{(C7.18)}$$

$$p_a \geq p_{a,l} + 1 \qquad \forall a \in [\![1; \mathrm{N}]\!] \qquad \text{(C7.19)}$$

$$p_a \geq p_{a,r} + 1 \qquad \forall a \in [\![1; \mathrm{N}]\!] \qquad \text{(C7.20)}$$

$$p_a \leq p_{a,l} + 1 + \mathrm{N} p_a^b \qquad \forall a \in [\![1; \mathrm{N}]\!] \qquad \text{(C7.21)}$$

$$p_a \leq p_{a,r} + 1 + \mathrm{N} \times \left(1 - p_a^b\right) \qquad \forall a \in [\![1; \mathrm{N}]\!] \qquad \text{(C7.22)}$$

$$p_{a,i} = p_k \quad \text{if } c_{a,i,k} = 1 \qquad \forall a \in [\![1; \mathrm{N}]\!], i \in \{l, r\}, k \in [\![0; a-1]\!] \qquad \text{(C7.23)}$$

$$p_{\max} \geq p_a + 1 \qquad \forall a \in [\![1; \mathrm{N}]\!] \qquad \text{(C7.24)}$$

$$p_k \leq N \times \overline{p}_{a,p} + p - 1 \quad \text{if } c_{k,i,a} = 1 \qquad \forall p \in [\![1; N+1]\!], a \in [\![0; N-1]\!], \forall k \in [\![a+1; N]\!], i \in \{l, r\} \qquad \text{(C7.25)}$$

$$p_{\max} \leq (N+1) \times \overline{p}_{a,p} + p - 1 \quad \text{if } o_{a,j} = 1 \qquad \forall s \in [\![1; N+1]\!], a \in [\![0; N]\!], j \in [\![1; |C|]\!] \qquad \text{(C7.26)}$$

$$\overline{p}_a \geq p \times \overline{p}_{a,p} \qquad \forall p \in [\![1; N+1]\!], a \in [\![0; N-1]\!] \qquad \text{(C7.27)}$$

$$r_a = \overline{p}_a - p_a \qquad \forall a \in [\![0; N]\!] \qquad \text{(C7.28)}$$

$$B_a^r = \mathrm{msb}_a + 1 \qquad \forall a \in [\![0; N]\!] \qquad \text{(C7.29)}$$

$$\mathrm{msb}_0 = w_{in} - 1 \qquad \text{(C7.30)}$$

$$2^{b+1} - 1 \geq (2^{w_{in}} - 1)\, c_a \quad \text{if } \mathrm{msb}_{a,b}^B = 1 \qquad \forall a \in [\![1; N]\!], b \in [\![1; w + w_{in} - 1]\!] \qquad \text{(C7.31)}$$

$$\sum_{b=1}^{w+w_{in}-1} \mathrm{msb}_{a,b}^B = 1 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C7.32)}$$

$$\sum_{b=1}^{w+w_{in}-1} b \times \mathrm{msb}_{a,b}^B = \mathrm{msb}_a \qquad \forall a \in [\![1; N]\!] \qquad \text{(C7.33)}$$

$$\sum_{b=1}^{N} b \times r_{a,b}^B = r_a \qquad \forall a \in [\![0; N]\!] \qquad \text{(C7.34)}$$

$$\sum_{b=1}^{N} r_{a,b}^B = 1 \qquad \forall a \in [\![0; N]\!] \qquad \text{(C7.35)}$$

$$C_a^r = b \times B_a^r \quad \text{if } r_{a,b}^B = 1 \qquad \forall a \in [\![0; N]\!], b \in [\![1; N]\!] \qquad \text{(C7.36)}$$

Variables:

$$c_a \in [\![0; 2^w]\!] \qquad \forall a \in [\![0; N]\!] \qquad \text{(C7.37)}$$

$$u_a \in \{0, 1\} \qquad \forall a \in [\![0; N]\!] \qquad \text{(C7.38)}$$

$$c_a^{\mathrm{nsh}} \in [\![1; 2^{w+1}]\!] \qquad \forall a \in [\![1; N]\!] \qquad \text{(C7.39)}$$

$$c_a^{\mathrm{odd}} \in \mathbb{N} \qquad \forall a \in [\![1; N]\!] \qquad \text{(C7.40)}$$

$$c_{a,i} \in [\![0; 2^w]\!] \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \qquad \text{(C7.41)}$$

$$c_{a,l}^{\text{sh}} \in [\![0; 2^{w+1}]\!] \qquad \forall a \in [\![1; N]\!] \qquad \text{(C7.42)}$$

$$c_{a,i}^{\text{sh,sg}} \in [\![-2^{w+1}; 2^{w+1}]\!] \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \qquad \text{(C7.43)}$$

$$\sigma_{a,i} \in \{0, 1\} \qquad \forall a \in [\![1; N]\!] \qquad \text{(C7.44)}$$

$$c_{a,i,k} \in \{0, 1\} \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, \forall k \in [\![0; a-1]\!] \qquad \text{(C7.45)}$$

$$\Phi_{a,s} \in \{0, 1\} \qquad \forall a \in [\![1; N]\!], s \in [\![0; w]\!] \qquad \text{(C7.46)}$$

$$\Psi_{a,s} \in \{0, 1\} \qquad \forall a \in [\![1; N]\!], s \in [\![0; w]\!] \qquad \text{(C7.47)}$$

$$o_{a,j} \in \{0, 1\} \qquad \forall a \in [\![1; N]\!], j \in [\![1; |C|]\!] \qquad \text{(C7.48)}$$

$$p_a \in [\![0; N]\!] \qquad \forall a \in [\![0; N]\!] \qquad \text{(C7.49)}$$

$$p_{a,i} \in [\![0; N]\!] \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \qquad \text{(C7.50)}$$

$$p_{\max} \in [\![0; N+1]\!] \qquad \text{(C7.51)}$$

$$\overline{p}_{a,p} \in \{0, 1\} \qquad \forall a \in [\![0; N]\!], p \in [\![1; N+1]\!] \qquad \text{(C7.52)}$$

$$\overline{p}_a \in [\![0; N+1]\!] \qquad \forall a \in [\![0; N]\!] \qquad \text{(C7.53)}$$

$$\overline{r}_a \in [\![0; N+1]\!] \qquad \forall a \in [\![0; N]\!] \qquad \text{(C7.54)}$$

$$B_a^r \in [\![0; w + w_{in}]\!] \qquad \forall a \in [\![0; N]\!] \qquad \text{(C7.55)}$$

$$C_a^r \in [\![0; N \times (w + w_{in})]\!] \qquad \forall a \in [\![0; N]\!] \qquad \text{(C7.56)}$$

$$\text{msb}_a \in [\![0; w + w_{in} - 1]\!] \qquad \forall a \in [\![0; N]\!] \qquad \text{(C7.57)}$$

$$\text{msb}_{a,t}^B \in \{0, 1\} \qquad \forall a \in [\![1; N]\!], t \in [\![1; w + w_{in} - 1]\!] \qquad \text{(C7.58)}$$

$$r_{a,b}^B \in \{0, 1\} \qquad \forall a \in [\![0; N]\!], b \in [\![1; N]\!] \qquad \text{(C7.59)}$$

\+ Critical path:

$$\text{B}_a = \text{msb}_a + 1 - s_a - \psi_a - g_a \qquad \forall a \in [\![1; N]\!] \qquad \text{(C7.60)}$$

$$\text{B}_a^u = \text{B}_a \quad \text{if } u_a = 1 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C7.61)}$$

$$\text{B}_a^u = 0 \quad \text{if } u_a = 0 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C7.62)}$$

$$B_{\max} \geq \text{B}_a^u \qquad \forall a \in [\![1; N]\!] \qquad \text{(C7.63)}$$

$$\text{msb}_{a,i} = \text{msb}_k \quad \text{if } c_{a,i,k} = 1 \qquad \forall a \in [\![1; N]\!], i \in \{l, r\}, k \in [\![0; a-1]\!] \qquad \text{(C7.64)}$$

$$g_a \leq \text{msb}_a \qquad \forall a \in [\![1; N]\!] \qquad \text{(C7.65)}$$

$$g_a \leq 0 \quad \text{if } \sigma_{a,r} = 1 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C7.66)}$$

$$g_a \leq s_{a,l} \qquad \forall a \in [\![1; N]\!] \qquad \text{(C7.67)}$$

$$s_{a,l} = \sum_{s=1}^{w} s\Phi_{a,s} \qquad \forall a \in [\![1; N]\!] \qquad \text{(C7.68)}$$

$$s_a = \sum_{s=-w}^{-1} s\Psi_{a,s} \qquad \forall a \in [\![1; N]\!] \qquad \text{(C7.69)}$$

$$\text{msb}_{a,l} + s_{a,l} + s_a + 1 \leq \text{msb}_a \quad \text{if } \psi_a = 1 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C7.70)}$$

$$\text{msb}_{a,r} + s_a + 1 \leq \text{msb}_a \quad \text{if } \psi_a = 1 \qquad \forall a \in [\![1; N]\!] \qquad \text{(C7.71)}$$

Variables (Critical path):

$$\mathrm{msb}_{a,i} \in [\![0; w + w_{in} - 1]\!] \qquad \forall a \in [\![1; N]\!], i \in \{l, r\} \tag{C7.72}$$
$$B_a \in [\![0; w + w_{in}]\!] \qquad \forall a \in [\![1; N]\!] \tag{C7.73}$$
$$B_a^u \in [\![0; w + w_{in}]\!] \qquad \forall a \in [\![1; N]\!] \tag{C7.74}$$
$$B_{\max} \in [\![0; w + w_{in}]\!] \qquad \forall a \in [\![1; N]\!] \tag{C7.75}$$
$$g_a \in [\![0; w + w_{in}]\!] \qquad \forall a \in [\![1; N]\!] \tag{C7.76}$$
$$\psi_a \in \{0, 1\} \qquad \forall a \in [\![1; N]\!] \tag{C7.77}$$
$$s_{a,l} \in [\![0; w]\!] \qquad \forall a \in [\![1; N]\!] \tag{C7.78}$$
$$s_a \in [\![-w; 0]\!] \qquad \forall a \in [\![1; N]\!] \tag{C7.79}$$

New objective function (Critical path):

$$\min \sum_{a=1}^{\mathrm{N}} (\mathrm{N}u_a) + ad_{\max}. \tag{O7}$$

APPENDIX B

Benchmark description

Throughout this thesis, we solve all our models on benchmarks from image processing (11 instances), that have already been used to compare MCM algorithms [KZFC12, KFM+13, Kum16, Kum18], and on the whole FIRsuite project "From Publication" (75 instances) [FIR23], which is a collection of digital filter designs and is a direct application of the MCM problem. For all these instances, we report their positive odd coefficients, removing zeros and ones which are not interesting for the MCM problem.

B.1 Image processing benchmark

Benchmark	Word length	Target constants
`Gaussian 3`	8	3 21 159
`Gaussian 5`	11	23 343 1267
`Highpass 5`	7	3 5 7 121
`Highpass 9`	7	3 5 7 11 125
`Highpass 15`	9	3 5 7 9 11 13 15 17 19 21 23 507
`Laplacian 3`	7	5 21 107
`Lowpass 5`	7	11 33 35 53 103
`Lowpass 9`	9	5 7 25 31 63 65 67 73 97 117 165 303
`Lowpass 15`	11	5 7 13 17 19 21 27 41 43 45 53 61 79 93 101 103 113 133 137 199 331 333 613 1097 1197
`Unsharp 3-1`	7	3 11 69
`Unsharp 3-2`	11	43 171 1109

B.2 FIRsuite

Benchmark	Word length	Target constants
Bull91 32	7	3 5 7 9 15 35 45 49 107 127
Chang06 3	10	139 283 815
Chen99 15	11	13 223 897 2017
Chen99 28a	9	3 7 9 21 27 43 69 97 261
Chen99 28b	11	5 9 13 19 23 27 97 119 1151
Chenyao01 28a	8	3 5 11 15 25 33 121 177
Chenyao01 28b	10	3 5 11 15 25 39 121 133 1023
Dempster02 25	12	35 133 199 291 327 331 355 499 505 699 1943 2987 3395
Dempster04 a3	8	5 117 195
Dempster04 b3	8	13 43 215
Dempster04 c3	11	947 973 1243
Dempster04 d3	10	105 479 939
Dempster04 e3	14	3787 9943 15567
Dempster04 f3	14	3981 6243 14603
Dempster04 g4	13	303 3643 6729 7479
Dempster04 h4	14	10083 12095 12975 15103
Dempster04 i5	12	63 407 1823 3059 3817
Dempster04 j5	11	19 841 911 935 1561
Dempster04 k5	12	407 569 831 1115 2473
Dempster04 l5	12	947 973 1243 1991 3651
Goodman77 a3	0	
Goodman77 b3	0	
Goodman77 c7	4	9
Goodman77 d7	5	3 19
Goodman77 e11	7	3 25 75
Goodman77 f11	8	9 11 13 173
Goodman77 g11	8	7 53 151
Goodman77 h15	9	3 29 33 245 401
Goodman77 i19	12	9 29 429 1277 2521
Jain91 11	5	3 5 17
Jang02 11	3	3 5 7
Jheng04 29a	9	5 7 11 17 25 31 71 73 389
Jheng04 29b	6	5 9 13 15 17 19 33 39 47
Jheng04 30	7	3 5 7 11 29 39 43 123
Johansson08 28	10	3 9 33 45 57 73 139 143 167 273 363 571
Johansson08 30	10	37 47 53 57 67 81 93 133 165 179 185 211 223 253 261 283 367 375 409 441 447 495 497 647 915 961 963 975

Johansson08 a5	10	9 387 745 805
Johansson08 b5	10	57 155 189 431 611
Kwentus97 11	7	3 25 75
Kwentus97 15	10	5 277 621
Kwentus97 47	14	21 27 29 55 187 207 381 623 913 977 1147 1295 16359
Lim83a 36	5	3 5 7 9 31
Lim83 121	14	3 7 9 17 19 21 23 35 37 39 41 45 51 55 59 61 65 69 71 75 77 89 109 125 131 141 143 151 165 167 193 195 267 273 277 323 375 399 541 545 585 761 1071 1515 1653 1739 2823 5927 6747 7343 10267
Lim83 36	4	3 5 7 9 15
Lim83 37	7	3 5 7 9 127
Lim83 63	9	3 5 7 9 15 19 21 23 29 31 33 37 41 59 61 71 99 119 195 321 351 431
Limakt08 121	14	3 5 7 9 11 17 19 21 23 25 31 33 35 37 41 49 61 65 69 73 79 89 119 131 151 161 167 189 197 271 281 379 413 533 537 541 641 869 1685 2993 5921 14671
Limpasko99 121	14	3 7 9 17 19 21 23 35 37 39 41 45 51 55 59 61 65 69 71 75 77 89 109 125 131 141 143 151 165 167 193 195 267 273 277 323 375 399 435 541 545 585 741 761 827 1071 1515 1687 2567 2823 14687
Limyu07 121	14	3 5 7 9 11 13 15 19 29 31 53 57 59 89 113 125 129 131 133 139 145 183 241 263 271 339 355 375 475 491 493 553 595 713 789 1215 1275 1343 1369 1371 6115 6655 9305
Martinez02 4	10	105 621 815 831
Nielsen89 67a	15	3 9 11 13 27 39 45 51 95 113 153 217 263 323 379 411 431 471 987 993 1523 2347 3413 3753 10693 32765
Nielsen89 67b	15	3 5 11 13 19 25 27 39 51 95 113 145 161 215 247 307 379 411 431 469 527 761 943 993 1173 6827 16383 21387
Potkonjak96 4	10	815 621 831 105
Rosa04 49	9	3 5 7 11 17 33 35 37 87 167 207
Samueli89 60	13	7 11 17 21 23 27 29 31 47 49 57 59 67 79 125 129 133 161 241 251 263 529 1217 2055 2239 4737
Shahein11 b	9	3 5 7 9 13 15 23 25 45 95
Shahein11 s2	11	3 5 11 17 19 23 27 31 33 35 53 63 91 109 145 165 281 305 321
Shi11 a	10	3 7 21 23 25 45 67 75 79 171 201 509 833 969
Shi11 g1	7	3 19
Shi11 l2	10	3 5 9 13 15 19 31 35 49 51 79 85 203 339 371 499 911
Shi11 s1 a	7	3 5 13 17

Shi11 s1 b	7	3 5 7 15 65
Shi11 s2	10	3 5 13 21 23 39 49 59 65 87 93 267 413 437 449 575 757
Shi11 x1	10	7 105 113 509
Shi11 y1	10	17 21 23 205 497 527
Shi11 y2	11	3 9 11 15 43 91 101 137 171 255
Vinod03 15	11	97 101 391 1289
Vinod03 26a	6	3 7 9 11 19 31 47
Vinod03 26b	14	89 223 611 621 1027 2201 2503 3067 3645 4999 6053 8139 14433
Xu07 15	12	5 69 553 2483
Xu07 28	11	3 11 23 25 35 53 79 407
Yeung04 40	19	5 13 15 63 115 157 177 283 453 579 961 2113 3215 3401 3549 6981 31381 58565 116589 251821
Yli01 30	10	3 7 9 23 25 37 543
Yoshino90 64	13	5 9 11 15 17 19 23 39 41 51 53 61 63 65 67 85 93 107 133 141 167 215 243 391 623
Zahosam89 25	8	3 5 7 17 23 123

APPENDIX C

Full results

C.1 Optimization results – Chapter 4

In the following tables, we provide full results for the different approaches we used to solve MCM-Adders. First, we compare the quality of the results in terms of number of adders. Second, we compare the adder count, the adder depth and solving times of our methods to solve the MCM-Adders problem taking the adder depth into account. Finally, we compare solving times between our model with/without branching priority.

Table C.1: Number of adders obtained with each method within the available solving time. Results followed by a star (*) are not proven optimal w. r. t. the number of adders. MILP_I and MILP_M correspond to our model with indicator and big-M constraints, respectively.

Bench			Loop		Min	
	CSD	RPAG	MILP_I	MILP_M	MILP_I	MILP_M
`Bull91 32`	14*	10*	10	10	10	10
`Chang06 3`	10*	6*	6	6	6	6
`Chen99 15`	8*	6*	5	5	5	5
`Chen99 28a`	16*	10*	9	9	9	9
`Chen99 28b`	15*	10*	9	9	9	9
`Chenyao01 28a`	11*	8*	8	8	8	8
`Chenyao01 28b`	12*	9*	9	9	9	9
`Dempster02 25`	37*	20*	-	-	18*	18*
`Dempster04 a3`	7*	5*	4	4	4	4
`Dempster04 b3`	8*	5*	4	4	4	4
`Dempster04 c3`	11*	7*	6	6	6	6
`Dempster04 d3`	9*	5*	5	5	5	5
`Dempster04 e3`	15*	9*	-	-	9*	9*
`Dempster04 f3`	14*	9*	-	-	8	8*

Dempster04 g4	17*	10*	-	-	10*	8*
Dempster04 h4	16*	10*	-	-	8	10*
Dempster04 i5	16*	10*	-	-	9*	9*
Dempster04 j5	16*	9*	-	-	8	8*
Dempster04 k5	18*	10*	-	-	10*	10*
Dempster04 l5	18*	11*	-	-	11*	11*
Gaussian 3	5*	4*	4	4	4	4
Gaussian 5	10*	6*	5	5	5	5
Goodman77 a3	-	-	-	-	-	-
Goodman77 b3	-	-	-	-	-	-
Goodman77 c7	1*	1*	1	1	1	1
Goodman77 d7	3*	2*	2	2	2	2
Goodman77 e11	6*	4*	3	3	3	3
Goodman77 f11	9*	5*	5	5	5	5
Goodman77 g11	7*	5*	4	4	4	4
Goodman77 h15	9*	7*	-	7	7	7
Goodman77 i19	14*	9*	-	-	7*	7*
Highpass 15	14*	12*	12	12	12	12
Highpass 5	5*	4*	4	4	4	4
Highpass 9	7*	5*	5	5	5	5
Jain91 11	3*	3*	3	3	3	3
Jang02 11	3*	3*	3	3	3	3
Jheng04 29a	15*	10*	9	9	9	9
Jheng04 29b	11*	9*	9	9	9	9
Jheng04 30	14*	8*	8	8	8	8
Johansson08 28	21*	14*	-	12	12	12
Johansson08 30	52*	33*	-	-	33*	33*
Johansson08 a5	11*	7*	6	6	6	6
Johansson08 b5	14*	7*	7	7	7	7
Kwentus97 11	6*	4*	3	3	3	3
Kwentus97 15	8*	5*	4	4	4	4
Kwentus97 47	31*	20*	-	-	20*	19*
Laplacian 3	6*	4*	3	3	3	3
Lim83a 36	5*	5*	5	5	5	5
Lim83 121	101*	54*	-	-	54*	54*
Lim83 36	5*	5*	5	5	5	5
Lim83 37	5*	5*	5	5	5	5
Lim83 63	33*	23*	22	-	22	22
Limakt08 121	76*	43*	-	-	43*	43*
Limpasko99 121	97*	54*	-	-	54*	54*
Limyu07 121	90*	45*	-	-	45*	45*
Lowpass 15	51*	27*	-	-	27*	26*

Lowpass 5	11*	7*	6	6	6	6
Lowpass 9	17*	13*	12	12	12	12
Martinez02 4	13*	6*	6	6	6	6
Nielsen89 67a	67*	32*	-	-	32*	32*
Nielsen89 67b	61*	35*	-	-	35*	35*
Potkonjak96 4	13*	6*	6	6	6	6
Rosa04 49	18*	11*	11	11	11	11
Samueli89 60	39*	28*	-	-	27	26
Shahein11 b	16*	10*	10	10	10	10
Shahein11 s2	30*	19*	-	19	19	19
Shi11 a	30*	16*	-	-	14	14
Shi11 g1	3*	2*	2	2	2	2
Shi11 l2	31*	18*	-	17	17	17
Shi11 s1 a	4*	4*	4	4	4	4
Shi11 s1 b	5*	5*	5	5	5	5
Shi11 s2	35*	18*	-	-	18*	17
Shi11 x1	7*	5*	5	5	5*	5
Shi11 y1	12*	7*	6	6	6	6
Shi11 y2	18*	11*	10	10	10	10
Unsharp 3-1	5*	4*	4	4	4	4
Unsharp 3-2	11*	6*	5	5	5	5
Vinod03 15	10*	8*	6	6	6	6
Vinod03 26a	9*	7*	7	7	7	7
Vinod03 26b	46*	20*	-	-	20*	20*
Xu07 15	11*	6*	5	5	5	6*
Xu07 28	17*	9*	8	8	8	8
Yeung04 40	68*	28*	-	-	28*	28*
Yli01 30	10*	8*	-	8	8*	8
Yoshino90 64	39*	27*	-	-	26*	27*
Zahosam89 25	8*	6*	6	6	6	6

Table C.2: Number of adders and adder depth obtained with each method within the available solving time. MCM-Adders, BiObj. and Tighten correspond, respectively, to the basis model with big-M constraints, the lexicographic bi-objective one and the model with tightening constraints. Solving times are given in seconds and TO stands for timed out.

Benchmark	MCM-Adders			BiObj.			Tighten		
	#A	AD	time	#A	AD	time	#A	AD	time
Bull91 32	10	2	1	10	2	102	10	2	92
Chang06 3	6	3	233	6	3	606	6	3	780

Chen99 15	5	3	1	5	3	24	5	3	9
Chen99 28a	9	5	3	9	3	TO	9	3	TO
Chen99 28b	9	5	13	9	3	TO	9	3	TO
Chenyao01 28a	8	2	1	8	2	6	8	2	7
Chenyao01 28b	9	2	1	9	2	1	9	2	15
Dempster02 25	18	7	TO	-	-	TO	20	3	TO
Dempster04 a3	4	3	1	4	3	2	4	3	1
Dempster04 b3	4	4	1	4	3	1	4	3	2
Dempster04 c3	6	4	62	6	4	3550	6	4	9580
Dempster04 d3	5	3	2	5	3	18	5	3	44
Dempster04 e3	9	3	TO	-	0	TO	-	0	TO
Dempster04 f3	8	5	TO	-	0	TO	9	4	TO
Dempster04 g4	8	4	TO	-	0	TO	-	0	TO
Dempster04 h4	10	3	TO	-	0	TO	-	0	TO
Dempster04 i5	9	3	TO	10	3	TO	10	4	TO
Dempster04 j5	8	4	TO	8	4	TO	9	3	TO
Dempster04 k5	10	3	TO	-	-	TO	-	-	TO
Dempster04 l5	11	3	TO	11	4	TO	-	-	TO
Gaussian 3	4	2	1	4	2	1	4	2	1
Gaussian 5	5	4	5	5	4	210	5	4	154
Goodman77 a3	-	-	TO	-	-	TO	-	-	TO
Goodman77 b3	-	-	TO	-	-	TO	-	-	TO
Goodman77 c7	1	1	1	-	-	TO	-	-	TO
Goodman77 d7	2	2	1	-	-	TO	-	-	TO
Goodman77 e11	3	3	1	3	3	1	3	3	1
Goodman77 f11	5	3	1	5	3	5	5	3	15
Goodman77 g11	4	3	1	4	3	2	4	3	3
Goodman77 h15	7	2	614	7	2	3002	7	2	738
Goodman77 i19	7	4	TO	8	5	TO	8	3	TO
Highpass 15	12	2	1	12	2	4	12	2	1
Highpass 5	4	2	1	4	2	1	4	2	1
Highpass 9	5	2	1	5	2	1	5	2	1
Jain91 11	3	1	1	3	1	1	3	1	1
Jang02 11	3	1	1	3	1	1	3	1	1
Jheng04 29a	9	3	5	9	3	TO	9	3	TO
Jheng04 29b	9	2	1	9	2	2	9	2	1
Jheng04 30	8	2	1	8	2	2	8	2	1
Johansson08 28	12	4	2414	13	3	TO	14	3	TO
Johansson08 30	33	3	TO	-	-	TO	33	3	TO
Johansson08 a5	6	3	23	6	3	257	6	3	851
Johansson08 b5	7	3	83	7	3	1044	7	3	4449
Kwentus97 11	3	3	1	3	3	1	3	3	1

Kwentus97 15	4	3	1	4	3	8	4	3	6
Kwentus97 47	19	3	TO	20	4	TO	20	2	TO
Laplacian 3	3	3	1	3	3	1	3	3	1
Lim83a 36	5	1	1	5	1	1	5	1	1
Lim83 121	54	3	TO	-	-	TO	54	3	TO
Lim83 36	5	1	1	5	1	1	5	1	1
Lim83 37	5	1	1	5	1	1	5	1	1
Lim83 63	22	4	73	22	3	TO	22	3	TO
Limakt08 121	43	3	TO	-	-	TO	43	3	TO
Limpasko99 121	54	3	TO	-	-	TO	54	3	TO
Limyu07 121	45	3	TO	-	-	TO	45	3	TO
Lowpass 15	26	4	TO	-	-	TO	27	3	TO
Lowpass 5	6	3	1	6	3	52	6	3	51
Lowpass 9	12	3	6	12	3	TO	13	3	TO
Martinez02 4	6	3	3	6	3	56	6	3	108
Nielsen89 67a	32	3	TO	-	-	TO	-	-	TO
Nielsen89 67b	35	3	TO	-	-	TO	-	-	TO
Potkonjak96 4	6	3	13	6	3	65	6	3	43
Rosa04 49	11	2	1	11	2	30	11	2	89
Samueli89 60	26	5	4108	-	-	TO	-	-	TO
Shahein11 b	10	2	1	10	2	1	10	2	5
Shahein11 s2	19	2	1	19	3	TO	19	2	1
Shi11 a	14	7	3634	15	4	TO	16	4	TO
Shi11 g1	2	2	1	-	-	TO	-	-	TO
Shi11 l2	17	4	37	18	4	TO	18	3	TO
Shi11 s1 a	4	2	1	4	2	1	4	2	1
Shi11 s1 b	5	1	1	5	1	1	5	1	1
Shi11 s2	17	4	25299	-	-	TO	18	3	TO
Shi11 x1	5	2	1	5	2	1	5	2	1
Shi11 y1	6	4	3	6	4	881	6	4	326
Shi11 y2	10	4	40	10	4	TO	11	3	TO
Unsharp 3-1	4	2	1	4	2	1	4	2	1
Unsharp 3-2	5	4	2	5	3	23	5	3	57
Vinod03 15	6	3	73	6	3	1493	6	3	328
Vinod03 26a	7	2	1	7	2	1	7	2	1
Vinod03 26b	20	3	TO	-	-	TO	-	-	TO
Xu07 15	6	3	TO	5	3	27	5	4	TO
Xu07 28	8	5	3	8	3	72	8	3	8
Yeung04 40	28	3	TO	-	-	TO	28	3	TO
Yli01 30	8	2	8782	8	2	TO	8	2	23858
Yoshino90 64	27	2	TO	27	3	TO	-	-	TO

Zahosam89 25	6	2	1	6	2	1	6	2	1

Table C.3: Solving time (in seconds) to prove optimality using the model in Appendix A.1.2 without and with branching priority (BP) parameter. TO means that the time limit of 28800 seconds has been reached without proving optimality.

Benchmark	No BP	BP
Bull91 32	1	1
Chang06 3	233	9
Chen99 15	1	1
Chen99 28a	3	2
Chen99 28b	13	3
Chenyao01 28a	1	1
Chenyao01 28b	1	1
Dempster02 25	TO	TO
Dempster04 a3	1	1
Dempster04 b3	1	1
Dempster04 c3	62	22
Dempster04 d3	2	1
Dempster04 e3	TO	TO
Dempster04 f3	TO	TO
Dempster04 g4	TO	TO
Dempster04 h4	TO	TO
Dempster04 i5	TO	3943
Dempster04 j5	TO	TO
Dempster04 k5	TO	TO
Dempster04 l5	TO	TO
Gaussian 3	1	1
Gaussian 5	5	3
Goodman77 a3	TO	TO
Goodman77 b3	TO	TO
Goodman77 c7	1	1
Goodman77 d7	1	1
Goodman77 e11	1	1
Goodman77 f11	1	1
Goodman77 g11	1	1
Goodman77 h15	614	66
Goodman77 i19	TO	147
Highpass 15	1	1
Highpass 5	1	1
Highpass 9	1	1
Jain91 11	1	1
Jang02 11	1	1
Jheng04 29a	5	2

Jheng04 29b	1	1
Jheng04 30	1	1
Johansson08 28	2414	2239
Johansson08 30	TO	TO
Johansson08 a5	23	9
Johansson08 b5	83	2
Kwentus97 11	1	1
Kwentus97 15	1	1
Kwentus97 47	TO	TO
Laplacian 3	1	1
Lim83a 36	1	1
Lim83 121	TO	TO
Lim83 36	1	1
Lim83 37	1	1
Lim83 63	73	68
Limakt08 121	TO	12113
Limpasko99 121	TO	TO
Limyu07 121	TO	TO
Lowpass 15	TO	TO
Lowpass 5	1	1
Lowpass 9	6	3
Martinez02 4	3	1
Nielsen89 67a	TO	TO
Nielsen89 67b	TO	TO
Potkonjak96 4	13	1
Rosa04 49	1	1
Samueli89 60	4108	TO
Shahein11 b	1	1
Shahein11 s2	1	1
Shi11 a	3634	3318
Shi11 g1	1	1
Shi11 l2	37	921
Shi11 s1 a	1	1
Shi11 s1 b	1	1
Shi11 s2	25299	9335
Shi11 x1	1	1
Shi11 y1	3	1
Shi11 y2	40	2
Unsharp 3-1	1	1
Unsharp 3-2	2	2
Vinod03 15	73	7
Vinod03 26a	1	1

Vinod03 26b	TO	TO
Xu07 15	TO	TO
Xu07 28	3	2
Yeung04 40	TO	TO
Yli01 30	8782	1943
Yoshino90 64	TO	TO
Zahosam89 25	1	1

C.2 Optimization results – Chapter 5

In the following table, we provide full optimization results comparing our models for the MCM-Adders and MCM-Bits problems with various input word lengths.

Table C.4: Number of adders, #A, and one-bit adders, #A_b, obtained with MCM-Adders vs MCM-Bits within the available solving time.

Benchmark	8-bit input				16-bit input				32-bit input			
	MCM-A.		MCM-B.		MCM-A.		MCM-B.		MCM-A.		MCM-B.	
	#A	#A_b	#A	#A_b	#A	#A_b	#A	#A_b	#A	#A_b	#A	#A_b
Bull91 32	**10**	109	**10**	**92**	**10**	189	**10**	**172**	**10**	349	**10**	**332**
Chang06 3	**6**	73	**6**	**62**	**6**	121	**6**	**110**	**6**	217	**6**	**206**
Chen99 15	**5**	**68**	**5**	**68**	**5**	**108**	**5**	**108**	**5**	**188**	**5**	**188**
Chen99 28a	**9**	106	10	**85**	**9**	178	**9**	**155**	**9**	322	**9**	**299**
Chen99 28b	**9**	115	10	**100**	**9**	187	**9**	**173**	**9**	331	**9**	**317**
Chenyao01 28a	**8**	82	**8**	**74**	**8**	146	**8**	**138**	**8**	274	**8**	**266**
Chenyao01 28b	**9**	98	**9**	**90**	**9**	170	**9**	**162**	**9**	314	**9**	**306**
Dempster02 25	**18**	254	20	**214**	**18**	398	20	**385**	**18**	**686**	20	703
Dempster04 a3	**4**	51	5	**44**	**4**	83	**4**	**76**	**4**	147	**4**	**140**
Dempster04 b3	**4**	51	**4**	**44**	**4**	83	**4**	**76**	**4**	147	**4**	**140**
Dempster04 c3	**6**	87	7	**76**	**6**	135	**6**	**122**	**6**	231	**6**	**218**
Dempster04 d3	**5**	**72**	**5**	**72**	**5**	**112**	**5**	**112**	**5**	**192**	**5**	**192**
Dempster04 e3	**9**	129	**9**	**117**	**9**	202	**9**	**190**	**9**	346	**9**	**334**
Dempster04 f3	**8**	**119**	9	122	8	183	**0**	**0**	**8**	**311**	9	324
Dempster04 g4	**8**	**108**	10	124	**8**	**172**	10	207	**8**	**300**	10	365
Dempster04 h4	**10**	**149**	**10**	**149**	**10**	**229**	**10**	**229**	**10**	389	**10**	**373**
Dempster04 i5	**9**	136	10	**130**	**9**	**208**	10	213	**9**	**352**	10	373
Dempster04 j5	**8**	**102**	9	105	**8**	**166**	**8**	**166**	**8**	**294**	9	312
Dempster04 k5	**10**	141	**10**	**140**	**10**	221	**10**	**210**	**10**	381	**10**	**377**
Dempster04 l5	**11**	154	**11**	**139**	11	242	**10**	**204**	**11**	418	**11**	**402**
Gaussian 3	**4**	43	**4**	**40**	**4**	75	**4**	**72**	**4**	139	**4**	**136**

Gaussian 5	**5**	**57**	**5**	**57**	**5**	**97**	**5**	**97**	**5**	**177**	**5**	**177**
Goodman77 c7	**1**	**8**	**1**	**8**	**1**	**16**	**1**	**16**	**1**	**32**	**1**	**32**
Goodman77 d7	**2**	**16**	**2**	**16**	**2**	**32**	**2**	**32**	**2**	**64**	**2**	**64**
Goodman77 e11	**3**	33	**3**	**31**	**3**	57	**3**	**55**	**3**	105	**3**	**103**
Goodman77 f11	**5**	53	**5**	**45**	**5**	93	**5**	**87**	**5**	173	**5**	**165**
Goodman77 g11	**4**	58	**4**	**40**	**4**	90	**4**	**72**	**4**	154	**4**	**136**
Goodman77 h15	**7**	76	**7**	**65**	**7**	132	**7**	**121**	**7**	244	**7**	**233**
Goodman77 i19	**7**	**82**	9	92	**7**	**138**	8	150	**7**	**250**	8	275
Highpass 15	**12**	122	**12**	**105**	**12**	218	**12**	**201**	**12**	410	**12**	**393**
Highpass 5	**4**	42	**4**	**39**	**4**	74	**4**	**71**	**4**	138	**4**	**135**
Highpass 9	**5**	50	**5**	**47**	**5**	90	**5**	**87**	**5**	170	**5**	**167**
Jain91 11	**3**	**24**	**3**	**24**	**3**	**48**	**3**	**48**	**3**	**96**	**3**	**96**
Jang02 11	**3**	27	**3**	**24**	**3**	51	**3**	**48**	**3**	99	**3**	**96**
Jheng04 29a	**9**	102	10	**88**	**9**	174	**9**	**163**	**9**	318	**9**	**307**
Jheng04 29b	**9**	87	**9**	**80**	**9**	159	**9**	**152**	**9**	303	**9**	**296**
Jheng04 30	**8**	88	**8**	**69**	**8**	152	**8**	**133**	**8**	280	**8**	**261**
Johansson08 28	**12**	**148**	14	150	**12**	**244**	14	262	**12**	**436**	14	486
Johansson08 30	**33**	413	**33**	**338**	**33**	677	**33**	**602**	**33**	1205	**33**	**1130**
Johansson08 a5	**6**	69	**6**	**67**	**6**	117	**6**	**115**	**6**	213	**6**	**211**
Johansson08 b5	**7**	**89**	**7**	**89**	**7**	**145**	**7**	**145**	**7**	**257**	**7**	**257**
Kwentus97 11	**3**	35	**3**	**31**	**3**	59	**3**	**55**	**3**	107	**3**	**103**
Kwentus97 15	**4**	46	5	**44**	**4**	**78**	**4**	**78**	**4**	**142**	**4**	**142**
Kwentus97 47	**19**	241	20	**213**	**19**	393	20	**379**	19	697	**0**	**0**
Laplacian 3	**3**	34	**3**	**31**	**3**	58	**3**	**55**	**3**	106	**3**	**103**
Lim83a 36	**5**	48	**5**	**42**	**5**	88	**5**	**82**	**5**	168	**5**	**162**
Lim83 121	**54**	699	**54**	**551**	**54**	1131	**54**	**996**	**54**	1995	**54**	**1844**
Lim83 36	**5**	47	**5**	**40**	**5**	87	**5**	**80**	**5**	167	**5**	**160**
Lim83 37	**5**	50	**5**	**47**	**5**	90	**5**	**87**	**5**	170	**5**	**167**
Lim83 63	**22**	261	**22**	**198**	**22**	437	23	**387**	**22**	789	23	**755**
Limakt08 121	**43**	512	**43**	**401**	**43**	856	**43**	**749**	**43**	1544	**43**	**1439**
Limpasko99 121	**54**	682	**54**	**509**	**54**	1114	**54**	**945**	**54**	1978	**54**	**1827**
Limyu07 121	**45**	580	**45**	**490**	**45**	941	**45**	**889**	**45**	1661	**45**	**1599**
Lowpass 15	**26**	316	27	**250**	**26**	524	27	**466**	**26**	940	27	**898**
Lowpass 5	**6**	67	**6**	**60**	**6**	115	**6**	**108**	**6**	211	**6**	**204**
Lowpass 9	**12**	130	13	**128**	**12**	**226**	13	232	**12**	**418**	13	440
Martinez02 4	**6**	83	**6**	**73**	**6**	131	**6**	**121**	**6**	227	**6**	**217**
Nielsen89 67a	**32**	440	**32**	**381**	**32**	696	**32**	**651**	**32**	1208	**32**	**1163**
Nielsen89 67b	**35**	441	**35**	**384**	**35**	721	**35**	**676**	**35**	1281	**35**	**1248**
Potkonjak96 4	**6**	83	**6**	**73**	**6**	131	**6**	**121**	**6**	227	**6**	**217**
Rosa04 49	**11**	105	**11**	**99**	**11**	193	**11**	**187**	**11**	369	**11**	**363**
Samueli89 60	**26**	309	28	**273**	**26**	518	28	**500**	**26**	**934**	28	947
Shahein11 b	**10**	109	**10**	**83**	**10**	189	**10**	**163**	**10**	349	**10**	**323**

Shahein11 s2	**19**	206	**19**	**178**	**19**	358	**19**	**330**	**19**	662	**19**	**634**
Shi11 a	**14**	178	16	**156**	**14**	290	16	**284**	**14**	**514**	16	540
Shi11 g1	**2**	**16**	**2**	**16**	**2**	**32**	**2**	**32**	**2**	**64**	**2**	**64**
Shi11 l2	**17**	187	18	**165**	**17**	323	18	**309**	**17**	**595**	18	597
Shi11 s1 a	**4**	34	**4**	**32**	**4**	66	**4**	**64**	**4**	130	**4**	**128**
Shi11 s1 b	**5**	47	**5**	**40**	**5**	87	**5**	**80**	**5**	167	**5**	**160**
Shi11 s2	**17**	210	18	**178**	**17**	346	18	**322**	**17**	618	18	**610**
Shi11 x1	**5**	62	**5**	**56**	**5**	102	**5**	**96**	**5**	182	**5**	**176**
Shi11 y1	**6**	79	7	**72**	**6**	**127**	**6**	**127**	**6**	**223**	**6**	**223**
Shi11 y2	**10**	117	**10**	**93**	**10**	197	**10**	**173**	**10**	357	**10**	**333**
Unsharp 3-1	**4**	**32**	**4**	**32**	**4**	**64**	**4**	**64**	**4**	**128**	**4**	**128**
Unsharp 3-2	**5**	57	**5**	**49**	**5**	97	**5**	**89**	**5**	177	**5**	**171**
Vinod03 15	**6**	75	**6**	**69**	**6**	123	**6**	**117**	**6**	219	**6**	**213**
Vinod03 26a	**7**	70	**7**	**59**	**7**	126	**7**	**115**	**7**	238	**7**	**227**
Vinod03 26b	**20**	293	**20**	**286**	**20**	453	**20**	**445**	**20**	773	**20**	**764**
Xu07 15	6	75	**5**	**63**	6	124	**5**	**103**	6	220	**5**	**183**
Xu07 28	**8**	93	**8**	**79**	**8**	157	**8**	**143**	**8**	285	**8**	**271**
Yeung04 40	28	428	**0**	**0**	**28**	652	**28**	**614**	**28**	1100	**28**	**1083**
Yli01 30	**8**	88	**8**	**69**	**8**	152	**8**	**133**	**8**	280	**8**	**261**
Yoshino90 64	**27**	305	**27**	**241**	**27**	521	**27**	**457**	**27**	953	**27**	**889**
Zahosam89 25	**6**	63	**6**	**55**	**6**	111	**6**	**103**	**6**	207	**6**	**199**

C.3 Hardware results – Chapter 5

In the following table, we provide full optimization results comparing our models for the MCM-Adders and MCM-Bits problems with 8-bit inputs.

Table C.5: Hardware results for 8-bit inputs.

Benchmark	MCM-Adders			MCM-Bits		
	LUTs	Delay (ns)	Power (mW)	LUTs	Delay (ns)	Power (mW)
Bull91 32	109	**1.972**	**10**	**98**	3.239	11
Chang06 3	73	**2.619**	8	**64**	3.208	**7**
Chen99 15	**66**	**2.591**	**8**	**66**	**2.591**	**8**
Chen99 28a	105	4.054	12	**86**	**2.38**	**9**
Chen99 28b	117	3.819	13	**104**	**2.56**	**11**
Chenyao01 28a	82	**1.971**	**8**	**76**	2.484	10
Chenyao01 28b	97	**1.938**	**9**	**90**	2.53	10
Dempster02 25	253	5.427	27	**224**	**4.584**	**24**
Dempster04 a3	52	**2.434**	6	**46**	**2.434**	**5**
Dempster04 b3	49	3.191	**6**	**44**	**3.168**	**6**

Dempster04 c3	85	**3.334**	**9**	**77**	4.548	10
Dempster04 d3	**70**	**2.56**	**7**	**70**	**2.56**	**7**
Dempster04 e3	129	**2.682**	**11**	**118**	4.051	12
Dempster04 f3	**120**	3.743	**11**	122	**3.187**	**11**
Dempster04 g4	**107**	3.312	**12**	124	**3.275**	13
Dempster04 h4	**142**	**2.96**	**14**	**142**	**2.96**	**14**
Dempster04 i5	131	**2.856**	**12**	**128**	3.953	14
Dempster04 j5	**102**	**3.179**	**12**	105	3.837	13
Dempster04 k5	**138**	**2.668**	**11**	139	3.93	14
Dempster04 l5	150	**2.763**	**14**	**141**	4.767	17
Gaussian 3	44	**1.833**	5	**41**	1.936	**4**
Gaussian 5	**58**	3.166	**6**	**58**	**3.039**	**6**
Goodman77 c7	**8**	**1.025**	**2**	**8**	**1.025**	**2**
Goodman77 d7	**17**	**1.846**	**3**	**17**	**1.846**	**3**
Goodman77 e11	34	2.469	**5**	**33**	**2.4**	**5**
Goodman77 f11	55	**2.301**	**6**	**46**	3.023	**6**
Goodman77 g11	54	2.682	6	**40**	**2.457**	**5**
Goodman77 h15	77	**2.083**	**7**	**67**	2.915	8
Goodman77 i19	**83**	**3.202**	**10**	95	3.823	**10**
Highpass 15	122	**1.933**	**11**	**108**	2.4	**11**
Highpass 5	42	**1.922**	**5**	**41**	2.501	**5**
Highpass 9	51	1.93	**6**	**48**	**1.848**	**6**
Jain91 11	**25**	**1.267**	**4**	**25**	**1.267**	**4**
Jang02 11	28	**1.205**	**3**	**25**	1.934	4
Jheng04 29a	101	**2.461**	**10**	**91**	2.464	**10**
Jheng04 29b	87	**2.021**	**8**	**82**	3.199	9
Jheng04 30	89	**1.974**	**10**	**74**	3.115	**10**
Johansson08 28	**148**	**3.053**	**16**	155	3.287	17
Johansson08 30	408	**2.66**	44	**346**	4.452	**42**
Johansson08 a5	71	2.531	9	**69**	**2.343**	**8**
Johansson08 b5	91	**2.552**	**10**	**88**	4.482	11
Kwentus97 11	35	2.51	**5**	**33**	**2.4**	**5**
Kwentus97 15	**47**	**2.613**	**6**	48	4.029	7
Kwentus97 47	237	**2.722**	**20**	**213**	3.81	21
Laplacian 3	35	2.429	**4**	**33**	**2.378**	**4**
Lim83a 36	49	**1.239**	**5**	**44**	2.374	6
Lim83 121	685	**2.788**	80	**578**	5.011	**74**
Lim83 36	48	**1.405**	**5**	**42**	2.38	6
Lim83 37	51	**1.428**	**5**	**48**	2.052	6
Lim83 63	258	**3.191**	27	**208**	4.426	**23**
Limakt08 121	508	**2.909**	57	**422**	4.55	**52**
Limpasko99 121	668	**2.684**	75	**532**	5.224	**69**

Limyu07 121	568	**2.858**	**63**	**500**	5.776	64
Lowpass 15	310	**3.263**	33	**259**	3.871	**29**
Lowpass 5	68	**2.449**	8	**64**	2.468	**7**
Lowpass 9	135	**2.6**	**14**	**134**	4.57	16
Martinez02 4	81	**2.693**	**9**	**74**	3.741	10
Nielsen89 67a	428	**3.231**	**41**	**386**	4.507	42
Nielsen89 67b	427	**2.825**	**41**	**386**	3.882	**41**
Potkonjak96 4	81	**2.722**	**9**	**74**	3.741	10
Rosa04 49	108	**2.029**	**11**	**102**	3.144	**11**
Samueli89 60	312	**3.826**	35	**283**	4.064	**31**
Shahein11 b	108	**1.996**	**10**	**89**	3.046	**10**
Shahein11 s2	207	**2.114**	**20**	**187**	3.152	**20**
Shi11 a	179	5.257	21	**160**	**3.256**	**17**
Shi11 g1	**17**	**1.846**	**3**	**17**	**1.846**	**3**
Shi11 l2	189	**3.156**	21	**172**	3.798	**19**
Shi11 s1 a	35	**1.852**	5	**34**	2.014	**4**
Shi11 s1 b	48	**1.381**	**5**	**42**	2.448	6
Shi11 s2	210	**3.24**	23	**184**	3.396	**21**
Shi11 x1	62	**1.977**	**6**	**58**	3.045	7
Shi11 y1	80	**2.9**	**10**	**75**	3.069	**10**
Shi11 y2	119	3.203	**12**	**99**	**3.13**	**12**
Unsharp 3-1	**33**	**2.066**	**4**	**33**	**2.066**	**4**
Unsharp 3-2	58	3.205	**6**	**51**	**3.162**	**6**
Vinod03 15	75	2.516	**8**	**70**	**2.416**	**8**
Vinod03 26a	70	**2.008**	**8**	**61**	3.245	**8**
Vinod03 26b	289	**2.726**	**28**	**285**	3.985	29
Xu07 15	76	**2.527**	8	**65**	3.357	**7**
Xu07 28	92	3.758	11	**83**	**3.174**	**10**
Yeung04 40	422	**2.974**	**41**	**395**	6.694	44
Yli01 30	89	**1.881**	**9**	**73**	2.986	**9**
Yoshino90 64	302	**2.169**	**28**	**251**	3.765	**28**
Zahosam89 25	64	**1.862**	**7**	**58**	2.542	**7**

APPENDIX D

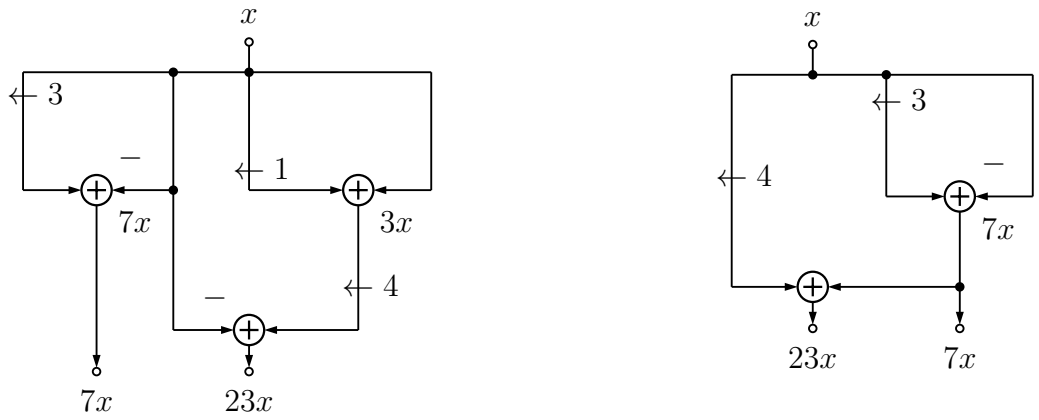
Résumé long en français

Introduction

Le monde moderne bénéficie de l'informatique à bien des égards. Même si nous pensons d'abord aux ordinateurs personnels et aux *smartphones*, les appareils électroniques qui effectuent des calculs intégrés sont omniprésents. Par exemple, les prothèses auditives ou les montres connectées sont des ordinateurs dédiés à une tâche spécifique. Les systèmes intégrés peuvent également être utilisés pour des tâches moins courantes, comme la détection de chants d'oiseaux à partir d'enregistrements audio. Ces dispositifs, qui peuvent fonctionner sur batteries ou avec de petits panneaux solaires, doivent généralement produire des résultats instantanés. Par exemple, les aides auditives fonctionnent sur de très petites batteries et son fonctionnement ne doit produire aucun délai.

Il en résulte un besoin spécifique de circuits optimisés dédiés et les concepteurs de matériel consacrent beaucoup de travail et d'efforts dans cette direction. Cependant, face à des problèmes et des systèmes de plus en plus complexes, il n'est plus réaliste d'optimiser manuellement la conception des circuits. Des outils de haut niveau ont été développés pour simplifier la conception, mais la mise en œuvre des noyaux arithmétiques est toujours laissée aux ingénieurs *hardware*. Par conséquent, les ingénieurs travaillant dans le domaine du traitement du signal, par exemple, doivent également être des experts en circuits intégrés. Pour simplifier leur travail, des outils automatiques qui guident également les décisions architecturales spécifiques sont nécessaires. Notre travail porte sur l'automatisation de la mise en œuvre des noyaux arithmétiques et des opérateurs, ce qui implique souvent la résolution de problèmes combinatoires.

Pour résoudre ces problèmes, nous utiliserons des techniques issues du domaine de la recherche opérationnelle (RO). En particulier, nous aborderons les problèmes de conception de matériel en utilisant la modélisation mathématique avec l'objectif de modéliser le matériel, en particulier les Field-Programmable Gate Arrays (FPGAs). Nous utiliserons spécifiquement l'approche de programmation linéaire en variables mixtes (MILP) pour améliorer l'état de l'art en ce qui concerne les problèmes de conception de filtres et de multiplications par plusieurs constantes. Cela conduira à des implémentations plus efficaces



(a) Adder graph for target constants 7 and 23.

(b) Optimal for target constants 7 and 23.

Figure D.1: Example of adder graphs for target constants 7 and 23.

réduisant le délai, la consommation d'énergie et l'utilisation des ressources sur FPGA. Idéalement, cette thèse facilitera l'utilisation des techniques de RO par les ingénieurs *hardware* pour la résolution de problèmes de conception. En plus de générer un matériel moins couteux, notre objectif est de rapprocher les communautés de conception de matériel et de RO.

Contexte et problème

La représentation des nombres en virgule flottante (FP), qui s'inspire de la notation scientifique, ne dépend pas de l'application pour laquelle elle est utilisée. Cela fait de l'arithmétique FP le choix le plus classique pour les applications génériques. Cependant, pour des applications spécifiques, l'arithmétique à virgule fixe (FxP) est souvent préférée. Au prix d'un travail supplémentaire pour représenter les nombres FxP avec le nombre de bits strictement nécessaire, il est possible d'implémenter des algorithmes plus efficaces et moins coûteux sur le plan matériel. Ainsi, chaque calcul/opérateur peut être spécifiquement conçu pour utiliser exactement les ressources nécessaires et pas plus. Dans le cadre de notre travail, nous tirerons parti de la représentation FxP et construirons chaque circuit avec précision.

La méthode standard pour construire un circuit consiste à le décrire à l'aide d'un langage de description de matériel (HDL). Bien que les outils de synthèse de haut niveau (HLS) soulagent les concepteurs de matériel d'une partie fastidieuse du travail de conception, les opérateurs arithmétiques efficaces sont encore souvent conçus en HDL. Cependant, l'expertise requise pour le faire efficacement peut être bloquante, révélant le besoin d'une automatisation pour la génération de noyaux arithmétiques. En particulier, dans cette thèse, nous aborderons la question de la génération automatique de l'opérateur de multiplication par plusieurs constantes (MCM pour *Multiple Constant Multiplication*).

Trouver l'implémentation la plus efficace de l'opérateur MCM est une tâche difficile. Une approche simple consisterait à utiliser des multiplicateurs génériques. Cependant,

ceux-ci sont coûteux car ils ne tirent pas profit de la valeur des constantes, étant conçus pour des variables. Dans notre cas, il est préférable de construire des circuits dédiés et nous remplaçons généralement les multiplications par des décalages de la chaîne de bits, qui sont des multiplications par des puissances de deux, et des additions/soustractions. C'est ce que l'on appelle l'approche *shift-and-add*. Par exemple, il est possible de calculer la multiplication d'une variable avec les constantes 7 et 23 en utilisant l'*adder graph* représenté dans la figure D.1a.

Dans cette thèse, nous aborderons donc le problème MCM qui peut être simplement défini comme suit : "étant donné un ensemble de constantes cibles avec lesquelles opérer des multiplications, trouver la meilleure implémentation en utilisant des décalages et des additions". Nous devrons définir le concept de "meilleure implémentation" par le biais de variables *proxy* car il n'existe généralement pas de modèles formels pour les FPGAs. Afin d'éviter d'effectuer une synthèse pour chaque graphe d'additionneurs pour en évaluer le coût, nous introduisons des variables de substitution qui serviront à estimer ce coût plus simplement. La méthode la plus classique consiste donc à compter le nombre d'additionneurs dans les adder graphs. En utilisant cette variable de substitution, nous pouvons prédire que l'adder graph représenté dans la figure D.1b, qui nécessite deux additionneurs, sera meilleur que celui de la figure D.1a, après la synthèse.

Plusieurs articles traitent du problème MCM, certains avec des heuristiques [Ber86, DM94, ACFM12, KZFC12] et d'autres avec des approches optimales [Gus08, AGF10, Kum16, Kum18]. Malgré la simplicité apparente du problème, des améliorations sont encore possibles. En effet, nous constatons deux limitations principales des méthodes actuelles. Premièrement, les méthodes optimales nécessitent beaucoup de temps de calcul et, avec des outils et des connaissances en RO, ce temps peut certainement être réduit. Deuxièmement, ces méthodes reposent sur le nombre d'additionneurs et nous montrerons qu'il est possible de définir des variables de substitution plus proches du matériel final. En outre, dans certains cas, un débit élevé est nécessaire et l'introduction de registres dans les adder graph, le *pipelining*, est une méthode courante [Par99, KZ11, KZFC12] pour augmenter le débit. Cela induit une augmentation de l'utilisation des ressources et nous aborderons également le problème de la minimisation du coût des adder graphs dits "pipelinés".

Étant donné que l'opérateur MCM est un élément de base des filtres numériques, certains travaux combinent la conception de filtres à réponse impulsionnelle finie (FIR) avec MCM [KVF23]. Le design de filtres à réponse impulsionnelle infinie (IIR), dont la conception est non linéaire, n'a pas encore été réalisé en MILP.

Objectif

Notre objectif principal est de fournir un modèle à grain fin du matériel et de l'englober dans des modèles mathématiques. Tout d'abord, nous corrigerons et améliorerons le modèle de l'état de l'art basé sur l'approche MILP [Kum18] qui résout le problème MCM en prenant en compte le nombre d'additionneurs. Cette métrique de haut niveau, bien qu'utile en pratique, peut être affinée et, dans notre travail, nous nous intéresserons à une métrique de

plus bas niveau : le nombre d'"additionneurs un bit". De cette manière, nous proposons un outil pour fournir automatiquement de meilleurs adder graphs.

Pour réduire davantage le coût d'implémentation, nous nous intéresseront également aux graphes d'additions tronquées, c'est-à-dire les adder graphs dans lesquels nous avons tronqué des bits dans le chemin de données pour réduire le coût des additionneurs. Cela induit une certaine erreur mais nous garantissons que celle-ci sera limitée, par construction, en deçà d'une valeur donnée *a priori* par l'utilisateur. Nous nous attaquerons à cette difficulté en proposant un modèle d'erreur permettant de propager correctement les erreurs dans les adder graphs. Enfin, l'un des objectifs de cette thèse est de montrer la polyvalence de la modélisation mathématique. Nous le ferons en recherchant des adder graphs pipelinés à un coût minimal en ajoutant à nos modèles MILP de nouvelles contraintes.

Enfin, nos travaux porteront sur la combinaison de l'opérateur MCM avec la conception de filtres IIR de second ordre. Notre objectif est de chercher les coefficients du filtre en prenant en compte leur implémentation matériel. Nous proposons donc une méthode de co-conception des coefficients du filtre avec l'implémentation via le problème MCM. De cette façon, nous présenterons une application typique du problème MCM et nous montrerons qu'il peut être abordé globalement en utilisant le modèle pour MCM comme une sous-partie d'un modèle plus général.

Nous chercherons à mettre à disposition toutes nos approches au sein d'outils de génération de code. L'objectif est de fournir automatiquement un code HDL aux concepteurs de matériel pour qu'ils l'utilisent dans des circuits complets.

Plan de la thèse

Cette thèse est organisée dans l'ordre des problèmes présentés ci-dessus. Naturellement, nous commençons par des métriques de haut niveau et, étape par étape, nous approfondissons vers les métriques plus proches du matériel. Nous concluons ensuite notre travail par une application. Le document est divisé en trois parties, chacune subdivisée en chapitres.

Partie I. Dans le chapitre 1, nous commençons par fournir les notions principales de l'arithmétique en virgule fixe. Ensuite, nous présentons notre cible, le FPGA, et les problèmes que nous cherchons à résoudre, MCM et la conception de filtres. Dans le chapitre 2, nous fournissons des connaissances de base sur la modélisation mathématique et l'approche de programmation linéaire en variables mixtes. Ensuite, dans le chapitre 3, nous décrivons l'objectif global cette thèse : fournir des générateurs de code efficaces.

Partie II. Cette deuxième partie est consacrée à nos solutions au problème MCM, celles-ci sont basées sur des modèles de plus en plus proche du matériel. Tout d'abord, dans le chapitre 4, nous présentons notre travail sur MCM-Adders. Ensuite, dans les chapitres 5 et 6, nous abordons la métrique de bas niveau de l'additionneur à un bit, puis les troncatures. Ces chapitres sont plus techniques et contiennent les détails de notre analyse d'erreur. Enfin, nous proposons une solution pour le problème PMCM dans le chapitre 7.

Partie III. Dans cette dernière partie, nous présentons une application qui implique le problème MCM. Avec cette dernière contribution, qui est chronologiquement la première, nous présentons la co-conception de filtres IIR de second ordre et de MCM.

Publications

Les travaux réalisés dans le cadre de cette thèse ont été publiés dans les revues internationales et les conférences évaluées par les pairs suivantes :

[GV23a] Rémi Garcia and Anastasia Volkova. Multiple Constant Multiplication: From Target Constants to Optimized Pipelined Adder Graphs. In *33rd International Conference on Field-Programmable Logic and Applications (FPL)*, Gothenburg, Sweden, September 2023. doi: 10.1109/FPL60245.2023.00027

[GV23b] Rémi Garcia and Anastasia Volkova. Toward the Multiple Constant Multiplication at Minimal Hardware Cost. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 70(5):1976–1988, 2023. doi: 10.1109/TCSI.2023.3241859

[GVK22a] Rémi Garcia, Anastasia Volkova, and Martin Kumm. Truncated Multiple Constant Multiplication with Minimal Number of Full Adders. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, Austin, Texas, United States, May 2022. doi: 10.1109/ISCAS48785.2022.9937441

[GVK+22b] Rémi Garcia, Anastasia Volkova, Martin Kumm, Alexandre Goldsztejn, and Jonas Kühle. Hardware-aware Design of Multiplierless Second-Order IIR Filters with Minimum Adders. *IEEE Transactions on Signal Processing*, pages 70:1673–1686, 2022. doi: 10.1109/TSP.2022.3161158

Conclusion

Avec cette thèse, nous avons contribué à intensifier les échanges scientifiques entre les méthodes de recherche opérationnelle, l'analyse arithmétique des ordinateurs et les applications de conception de matériel. Parmi la grande variété de sujets relatifs à la conception de matériel, nous avons abordé le problème MCM qui est un élément essentiel de nombreuses applications classiques. Dans un premier temps, nous avons redéfini la terminologie relative au problème MCM afin de pouvoir facilement différencier MCM, en tant que concept, des problèmes plus spécifiques MCM-Adders, MCM-Bits, tMCM ou PMCM. Ensuite, dans ce travail, nous avons démontré que la recherche opérationnelle, via la modélisation mathématique, apporte aux applications matérielles des outils précieux. Dans le chapitre 4, nous avons présenté l'efficacité de la modélisation mathématique pour résoudre un problème pertinent bien connu depuis des décennies. Tout d'abord, nous avons présenté notre modèle de minimisation pour le problème MCM-Adders et incorporé des métriques secondaires telles que la profondeur de l'adder graph. Ensuite, nous avons

démontré l'intérêt d'ajuster les paramètres du solveur car, avec des paramètres correctement choisis, comparé aux valeurs par défaut, nous avons pu réduire les temps de résolution de 16.9%, en moyenne.

Dans un second temps, nous nous sommes plongés dans le modèle matériel en comptant le nombre d'additionneurs à un bit dans le chapitre 5. Cette métrique de bas niveau reflète mieux le coût du matériel et la résolution de MCM-Bits au lieu de MCM-Adders a permis de réduire l'utilisation de la LUT de 7.8%, en moyenne. Pour effectuer ces comparaisons, nous avons inclus un composant de génération de code VHDL dans notre toolbox.

Nous pensons que le chapitre 6 est vraiment là où le mélange prend. Bien que nous concevions des opérateurs *arithmétiques*, nous avons d'une certaine manière contourné l'analyse d'erreur jusqu'ici. Dans ce chapitre, une analyse des erreurs a été nécessaire pour approfondir les possibilités de conception matérielle. Cette analyse d'erreur est importante pour les applications critiques, d'un point de vue sécurité, dans lesquelles les concepteurs de matériel doivent pouvoir être sûrs que les erreurs dans le circuit sont suffisamment faibles. En outre, cette analyse facilite la conception d'opérateurs matériels frugaux : en utilisant des troncatures intermédiaires, nous réduisons la consommation d'énergie de 22% par rapport aux circuits en *full*-précision, en moyenne.

Enfin, dans le chapitre 7, nous avons démontré que la modélisation mathématique ne se limite pas à la modélisation des additionneurs (blocs de LUTs), des décalages (bit-shifts) et des additionneurs à un bit (LUT), mais qu'elle peut également être utilisée pour modéliser les registres (FF) qui sont un composant essentiel du circuit matériel. Cela a permis de réduire considérablement le chemin critique, le divisant par 3 dans certains cas. La résolution de PMCM, au lieu de pipeliner *a posteriori* un adder graph fixé, a également réduit le coût de l'adder graph de 33%, en moyenne.

Tout au long de cette thèse, nous avons présenté plusieurs modèles qui ont été implémentés en open-source dans notre *package* Julia `jMCM`[1]. En résolvant nos modèles, nous obtenons des adder graphs pour lesquels nous pouvons générer du VHDL à l'aide de notre package `AdderGraphs`[2]. Nous pensons qu'il serait extrêmement utile d'intégrer ces modèles dans des outils auxquels les concepteurs de matériel pourraient faire confiance et nous travaillons actuellement dans ce sens en rendant tous nos outils open-source.

Enfin, nous avons pris du recul et démontré que MCM, résolu avec l'approche MILP, se combine bien avec la conception d'opérateurs arithmétiques plus larges tels que les filtres numériques. Les algorithmes utilisés dans les filtres numériques impliquent généralement plusieurs multiplications par des constantes et, avec la conception de filtres IIR du second ordre, nous avons montré que MCM peut être utilisé au cœur d'un problème plus vaste. De cette manière, nous avons réalisé la co-conception des coefficients de filtres numériques avec leur implémentation à l'aide de MCM. Cela a permis d'obtenir des filtres dont la consommation d'énergie a été divisé par 10 dans un certains cas par rapport à l'état de l'art KCM. En moyenne, nous avons réduit le nombre de LUTs de 48% par rapport aux meilleurs résultats des autres méthodes, à l'exception de celle utilisant les DSPs. Grâce

[1] `https://github.com/remi-garcia/jMCM`

[2] `https://github.com/remi-garcia/AdderGraphs`

à ces expériences, nous avons démontré l'efficacité de la modélisation mathématique pour résoudre des problèmes pratiques de conception de matériel. Ces résultats sont donc très prometteurs et nous encouragent à poursuivre nos travaux sur d'autres problèmes de conception de matériel.

Perspectives

Le travail présenté dans cette thèse est un nouveau pas vers l'utilisation de plus de techniques de recherche opérationnelle dans la conception de matériel. Bien que nous ayons résolu plusieurs variantes du problème MCM, il reste encore de nombreux aspects à explorer. Dans l'ensemble, nous prévoyons d'**améliorer notre boîte à outils sur le problème MCM** et d'éliminer autant de zones d'ombre que possible :

- **Utiliser les DSP.** Nous nous sommes concentrés sur les FPGAs tout en laissant de côté les blocs DSP, ces ressources sont précieuses et pourraient être utilisées à côté des LUTs afin de répartir le coût des multiplications par constantes sur divers éléments. À court terme, nous prévoyons d'incorporer leur utilisation dans les modèles MILP pour remplacer les constantes coûteuses en LUTs par des DSPs. En particulier, il ne fait aucun doute que la combinaison des DSPs et des LUTs, comme dans [LPBG19], pour tirer le meilleur parti des deux ressources est une direction vers laquelle nous devons nous diriger.

- **Nouvelles approches de modélisation.** Nous avions un marteau, le MILP, et nous avons montré que MCM est suffisamment proche d'un clou. Cependant, nous devons examiner notre propre pratique avec un esprit critique et peut-être envisager d'autres approches de modélisation mathématique ou l'utilisation d'heuristiques. En particulier, à moyen terme, nous aimerions expérimenter la programmation par contraintes sur le problème MCM-Adders pour le comparer au MILP. En effet, selon le problème, les solveurs MILP ou les solveurs de programmation par contraintes sont plus ou moins adaptés et, sur le problème MCM-Adders, certaines limitations, telles que les problèmes d'instabilité numérique, seraient évitées avec la programmation par contraintes. Quoi qu'il en soit, les deux méthodes nécessitent des calculs lourds et nous pensons que des heuristiques peuvent être développées pour ce problème. Actuellement, à notre connaissance, les heuristiques disponibles sont des heuristiques primales qui produisent une solution initiale mais ne la développent pas. À l'avenir, nous aimerions explorer des alternatives et proposer des heuristiques qui pourraient améliorer les solutions à l'aide d'une recherche locale dédiée.

- **Modélisation de composants matériels polyvalents à grain fin** Globalement, sur le sujet MCM, nous sommes passés d'une métrique de haut niveau, le nombre d'additionneurs, à une métrique de plus bas niveau, comptant les additionneurs d'un bit, y compris les troncatures. En fin de compte, la modélisation directe des LUT, des DSP, des FF et des fils constituerait une réalisation substantielle. Cela ouvrirait de nombreuses portes, car cette capacité pourrait alors être appliquée à n'importe

quelle fonction produisant des opérateurs arithmétiques matériels spécifiquement et précisément conçus pour les FPGA. C'est un objectif à long terme que nous gardons à l'esprit et que nous souhaitons atteindre en collaboration avec les concepteurs de matériel afin de mieux comprendre le matériel. Comme pour tout problème abordé à l'aide d'outils de recherche opérationnelle, l'expertise spécifique au domaine est absolument nécessaire. En comprenant mal le problème, nous déclencherons certainement le mauvais interrupteur et, au mieux, nous perdrons un temps précieux.

Ce travail sur le problème MCM, à l'aide de modèles basés sur le MILP, pose également d'autres des questions théoriques que nous souhaitons explorer à l'avenir :

- **Analyse de la complexité.** Nous souhaitons savoir si le problème MCM est NP-hard, comme cela a été conjecturé dans de nombreux articles [BH91, DM94, Gus08, AFM15, Kum18] ? Thong et Nicolici [TN11] ont établi que si plusieurs problèmes similaires sont NP-hard, cela n'a pas été prouvé pour le problème MCM. Thong et Nicolici ont également noté que les preuves existantes pour des problèmes similaires ne sont pas valables pour MCM. Cela représente un problème car la plupart des techniques que nous utilisons pour résoudre MCM sont construites en supposant que ce problème est NP-hard. Le prouver dans un futur proche nous conforterait dans l'idée que nous allons dans la bonne direction. Ce serait l'occasion d'éveiller l'intérêt pour les problèmes de conception de matériel des chercheurs travaillant sur des problématique de complexité.

- **Comparaison statistique.** Dans le chapitre 2, nous avons rappelé que les solveurs ne sont pas déterministes. À notre connaissance, le phénomène de *performance variability* n'a jamais été rigoureusement pris en compte dans la comparaison de modèles entre eux, ni avec d'autres approches. Cependant, pour affirmer une différence significative entre les approches impliquant un modèle basé sur le MILP, nous avons besoin de tests statistiques. Comme nous l'avons établi, ce n'est pas trivial et nous devons confronter notre pratique à des expérimentations rigoureuses. À l'avenir, l'un de nos objectifs est de fournir une boîte à outils pour faciliter ce processus de comparaison.

Enfin, nous avons étudié le problème MCM et une application à travers la conception du filtre IIR. Nous pensons que notre approche de ces **problèmes matériels** peut également être utilisée pour d'autres **applications** complètes ou pour des opérateurs matériels différents :

- **Multiplications génériques.** Les multiplications génériques, par exemple, ont déjà bénéficié d'une modélisation MILP [KKIZ17, BKdD21] et nous pensons que ces modèles pourraient être améliorés ou étendus pour prendre en compte davantage de paramètres tels que la précision variable, qui peut aider à réduire la consommation d'énergie lorsque l'application n'a pas besoin de la précision la plus élevée en permanence. Nous avons déjà commencé à travailler sur les multiplicateurs à précision variable avec Andreas Böttcher et Martin Kumm de *Fulda University of Applied Sciences*, et nous prévoyons de finaliser ces travaux à court terme.

- **Approximation des fonctions en virgule flottante.** Notre travail sur le problème tMCM a impliqué une analyse d'erreur intéressante et nous aimerions continuer à travailler sur des problèmes qui nécessitent un matériel optimisé avec une borne d'erreur garantie. En particulier, l'approximation de fonctions pour leur implémentation sur FPGA est un sujet qui regroupe de manière intéressante l'arithmétique informatique, la connaissance du matériel et l'optimisation. Une composante théorique intéressante de ce travail serait de modéliser les nombres FP dans les modèles MILP. En effet, nous pensons qu'il serait possible de modéliser la mantisse et l'exposant séparément, comme deux entiers, et, de cette façon, d'effectuer des opérations non linéaires dans le MILP. Cela pourrait permettre d'intégrer pour la première fois dans des modèles MILP la multiplication réelle.

- **Vers de nouvelles applications.** À long terme, nous souhaitons élargir notre connaissance des applications possibles. Par exemple, nous travaillons déjà avec Vincent Lostanlen sur de nouveaux opérateurs de filtres numériques pour l'acoustique environnementale et nous prévoyons de finaliser ce travail dans un avenir proche. Notre objectif est de réduire la consommation d'énergie des circuits matériels que nous pourrions utiliser pour détecter la présence d'oiseaux [LSF+18]. D'autres exemples d'applications possibles sont les réseaux de neurones et la cryptographie, qui impliquent un grand nombre de multiplications par des constantes, de manière similaire au problème MCM. Contrairement à MCM, ils impliquent généralement des multiplications de matrices ou de très grandes constantes.

Dans l'ensemble, nous prévoyons de continuer à travailler pour l'optimisation des opérateurs matériels et l'amélioration des méthodes de recherche opérationnelle.

Bibliography

[ACFM12] Levent Aksoy, Eduardo Costa, Paulo Flores, and José Monteiro. Optimization Algorithms for the Multiplierless Realization of Linear Transforms. *ACM Transactions on Design Automation of Electronic Systems*, 17(1):1–27, January 2012.

[AFM15] Levent Aksoy, Paulo Flores, and José Monteiro. Exact and Approximate Algorithms for the Filter Design Optimization Problem. *IEEE Transactions on Signal Processing*, 63(1):142–154, January 2015.

[AGF10] Levent Aksoy, Ece Olcay Güneş, and Paulo Flores. Search algorithms for the multiple constant multiplications problem: Exact and approximate. *Microprocessors and Microsystems*, 34(5):151–162, August 2010.

[Ant18] Andreas Antoniou. *Digital Filters: Analysis, Design, and Signal Processing Applications*. McGraw-Hill Education, New York, 2018.

[BBC+21] Ksenia Bestuzheva, Mathieu Besançon, Wei-Kun Chen, Antonia Chmiela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros Gleixner, Leona Gottwald, Christoph Graczyk, Katrin Halbig, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Thorsten Koch, Marco Lübbecke, Stephen J. Maher, Frederic Matter, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Daniel Rehfeldt, Steffan Schlein, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Boro Sofranac, Mark Turner, Stefan Vigerske, Fabian Wegscheider, Philipp Wellner, Dieter Weninger, and Jakob Witzig. The SCIP Optimization Suite 8.0. Technical report, Optimization Online, December 2021.

[BBF+16] Pietro Belotti, Pierre Bonami, Matteo Fischetti, Andrea Lodi, Michele Monaci, Amaya Nogales-Gómez, and Domenico Salvagnin. On handling indicator constraints in mixed integer programming. *Computational Optimization and Applications*, 65(3):545–566, May 2016.

[BdDI+13] Nicolas Brunie, Florent de Dinechin, Matei Iştoan, Guillaume Sergent, Kinga Illyes, and Bogdan Popa. Arithmetic core generation using bit heaps. In *2013*

23rd International Conference on Field programmable Logic and Applications. IEEE, September 2013.

[BEL08] Alain Billionnet, Sourour Elloumi, and Amélie Lambert. Linear Reformulations of Integer Quadratic Programs. In Hoai An Le Thi, Pascal Bouvry, and Tao Pham Dinh, editors, *Modelling, Computation and Optimization in Information Systems and Management Sciences*, pages 43–51, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[Ber86] Robert Bernstein. Multiplication by integer constants. *Software: Practice and Experience*, 16(7):641–652, July 1986.

[BH91] David R. Bull and David H. Horrocks. Primitive operator digital filters. *IEE Proceedings G - Circuits, Devices and Systems*, 138(3):401–412, June 1991.

[BHH+21] Michael L. Bynum, Gabriel A. Hackebeil, William E. Hart, Carl D. Laird, Bethany L. Nicholson, John D. Siirola, Jean-Paul Watson, and David L. Woodruff. *Pyomo–optimization modeling in python*, volume 67. Springer Science & Business Media, third edition, 2021.

[BKdD21] Andreas Böttcher, Martin Kumm, and Florent de Dinechin. Resource Optimal Truncated Multipliers for FPGAs. In *2021 IEEE 28th Symposium on Computer Arithmetic (ARITH)*. IEEE, June 2021.

[BLS18] Timo Berthold, Andrea Lodi, and Domenico Salvagnin. Ten years of feasibility pump, and counting. *EURO Journal on Computational Optimization*, 7(1):1–14, November 2018.

[Boo51] Andrew D. Booth. A Signed Binary Multiplication Technique. *The Quarterly Journal of Mechanics and Applied Mathematics*, 4(2):236–240, 1951.

[BT97] Dimitris Bertsimas and John Tsitsiklis. *Introduction to linear optimization*. Athena Scientific, Belmont, Mass, 1997.

[BT05] Nicolas Boullis and Arnaud Tisserand. Some Optimizations of Hardware Multiplication by Constant Matrices. *IEEE Transactions on Computers*, 54(10):1271–1282, October 2005.

[Cha94] Kenneth David Chapman. Fast integer multipliers fit in FPGAs. *Electronic Design News*, (10):80, May 1994.

[CPL20] CPLEX. CPLEX User's Manual, 2020.

[CVKF03] Joan Carletta, Robert Veillette, Frederick Krach, and Zhengwei Fang. Determining appropriate precisions for signals in fixed-point IIR filters. In *Proceedings 2003. Design Automation Conference (IEEE Cat. No.03CH37451)*, pages 656–661, June 2003.

[dDFKF19] Florent de Dinechin, Silviu-Ioan Filip, Martin Kumm, and Luc Forget. Table-Based versus Shift-And-Add Constant Multipliers for FPGAs. In *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*, pages 151–158. IEEE, June 2019.

[dDIM14] Florent de Dinechin, Matei Iştoan, and Abdelbassat Massouri. Sum-of-product architectures computing just right. In *2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*. IEEE, June 2014.

[DDK00] S. S. Demirsoy, A. Dempster, and I. Kale. Transition analysis on FPGA for multiplier-block based FIR filter structures. In *ICECS 2000. 7th IEEE International Conference on Electronics, Circuits and Systems (Cat. No.00EX445)*. IEEE, December 2000.

[DDK02] Suleyman S. Demirsoy, Andrew G. Dempster, and Izzet Kale. Power analysis of multiplier blocks. In *2002 IEEE International Symposium on Circuits and Systems. Proceedings (Cat. No.02CH37353)*, volume 1, pages 297–300, 2002.

[dDP11] Florent de Dinechin and Bogdan Pasca. Designing Custom Arithmetic Data Paths with FloPoCo. *IEEE Design and Test of Computers*, 28(4):18–27, July 2011.

[DHL17] Iain Dunning, Joey Huchette, and Miles Lubin. JuMP: A Modeling Language for Mathematical Optimization. *SIAM Review*, 59(2):295–320, May 2017.

[DIN+18] Eva Darulova, Anastasia Izycheva, Fariha Nasir, Fabian Ritter, Heiko Becker, and Robert Bastian. Daisy - Framework for Analysis and Optimization of Numerical Programs (Tool Paper). In Dirk Beyer and Marieke Huisman, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 270–287, Cham, 2018. Springer International Publishing.

[DM94] Andrew G. Dempster and Malcolm D. Macleod. Constant integer multiplication using minimum adders. *IEE Proceedings - Circuits, Devices and Systems*, 141(5):407–413, October 1994.

[DM95] Andrew G. Dempster and Malcolm D. Macleod. Comparison of IIR filter structure complexities using multiplier blocks. In *Proceedings of ISCAS'95 - International Symposium on Circuits and Systems*, volume 2, pages 858–861, April 1995.

[DM04] Andrew G. Dempster and Malcolm D. Macleod. Using all signed-digit representations to design single integer multipliers using subexpression elimination. In *2004 IEEE International Symposium on Circuits and Systems (IEEE Cat. No.04CH37512)*. IEEE, May 2004.

[EL04] Miloš D. Ercegovac and Tomás Lang. *Digital Arithmetic*. Elsevier, 2004.

[EN84] Stephen E. Edgell and Sheila M. Noon. Effect of violation of normality on the t test of the correlation coefficient. *Psychological Bulletin*, 95(3):576–583, May 1984.

[Esp18] Daniel Espinoza. Avoiding numerical issues in optimization models. Online webinar, January 2018. `https://www.gurobi.com/resource/numerical-issues-webinar/` - Web page visited in April 2023.

[FGL05] Matteo Fischetti, Fred Glover, and Andrea Lodi. The feasibility pump. *Mathematical Programming*, 104(1):91–104, March 2005.

[FIR23] FIRsuite. Suite of Constant Coefficient FIR Filters, 2023. Accessed: August., 2023.

[FLP+18] Franz Franchetti, Tze Meng Low, Doru Thom Popovici, Richard M. Veras, Daniele G. Spampinato, Jeremy R. Johnson, Markus Puschel, James C. Hoe, and Jose M. F. Moura. SPIRAL: Extreme Performance Portability. *Proceedings of the IEEE*, 106(11):1935–1968, November 2018.

[GDJ10] Rui Guo, Linda S. DeBrunner, and Kenny Johansson. Truncated MCM using pattern modification for FIR filter implementation. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 3881–3884. IEEE, May 2010.

[Glo75] Fred Glover. Improved Linear Integer Programming Formulations of Nonlinear Integer Problems. *Management Science*, 22(4):455–460, December 1975.

[Gom58] Ralph E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275–278, 1958.

[GP11] Eric Goubault and Sylvie Putot. Static Analysis of Finite Precision Computations. In *Lecture Notes in Computer Science*, pages 232–247. Springer Berlin Heidelberg, 2011.

[Gur20] Gurobi Optimization. Gurobi Optimizer Reference Manual, 2020.

[Gus07] Oscar Gustafsson. Lower Bounds for Constant Multiplication Problems. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 54(11):974–978, November 2007.

[Gus08] Oscar Gustafsson. Towards optimal multiple constant multiplication: A hypergraph approach. In *2008 42nd Asilomar Conference on Signals, Systems and Computers*, pages 1805–1809, October 2008.

[GV23a] Rémi Garcia and Anastasia Volkova. Multiple Constant Multiplication: From Target Constants to Optimized Pipelined Adder Graphs. In *33rd International Conference on Field-Programmable Logic and Applications (FPL)*, Gothenburg, Sweden, September 2023.

[GV23b] Rémi Garcia and Anastasia Volkova. Toward the Multiple Constant Multiplication at Minimal Hardware Cost. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 70(5):1976–1988, 2023.

[GVK22a] Rémi Garcia, Anastasia Volkova, and Martin Kumm. Truncated Multiple Constant Multiplication with Minimal Number of Full Adders. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, Austin, Texas, United States, May 2022. IEEE.

[GVK+22b] Rémi Garcia, Anastasia Volkova, Martin Kumm, Alexandre Goldsztejn, and Jonas Kühle. Hardware-aware Design of Multiplierless Second-Order IIR Filters with Minimum Adders. *IEEE Transactions on Signal Processing*, 70:1673–1686, 2022.

[GWD09] Rui Guo, Lei Wang, and Linda S. DeBrunner. A novel FIR filter implementation using truncated MCM technique. In *2009 Conference Record of the Forty-Third Asilomar Conference on Signals, Systems and Computers*, pages 718–722. IEEE, November 2009.

[Ham50] Richard W. Hamming. Error Detecting and Error Correcting Codes. *Bell System Technical Journal*, 29(2):147–160, April 1950.

[Har96] Richard I. Hartley. Subexpression sharing in filters using canonic signed digit multipliers. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 43(10):677–688, 1996.

[HK93] R. Hettich and K. O. Kortanek. Semi-Infinite Programming: Theory, Methods, and Applications. *SIAM Review*, 35(3):380–429, September 1993.

[IEE05] IEEE Standard for Verilog Hardware Description Language, 2005.

[IEE19] IEEE Standard for VHDL Language Reference Manual. IEEE Computer Society, 2019.

[JGW07] Kenny Johansson, Oscar Gustafsson, and Lars Wanhammar. Bit-Level Optimization of Shift-and-Add Based FIR Filters. In *2007 14th IEEE International Conference on Electronics, Circuits and Systems*, pages 713–716. IEEE, December 2007.

[JN86] William K. Jenkins and Majid Nayeri. Adaptive filters realized with second order sections. In *ICASSP '86. IEEE International Conference on Acoustics, Speech, and Signal Processing*. Institute of Electrical and Electronics Engineers, 1986.

[KBPV22] Thorsten Koch, Timo Berthold, Jaap Pedersen, and Charlie Vanaret. Progress in mathematical programming solvers from 2001 to 2020. *EURO Journal on Computational Optimization*, 10:100031, 2022.

[KFCV03] Frederick Krach, B. Frackelton, Joan Carletta, and Robert Veillette. FPGA-based implementation of digital control for a magnetic bearing. In *Proceedings of the 2003 American Control Conference, 2003.* IEEE, 2003.

[KFM+13] Martin Kumm, Diana Fanghänel, Konrad Möller, Peter Zipf, and Uwe Meyer-Baese. FIR filter optimization for video processing on FPGAs. *EURASIP Journal On Advances in Signal Processing*, 2013(1), May 2013.

[KGGZ16] Martin Kumm, Oscar Gustafsson, Mario Garrido, and Peter Zipf. Optimal Single Constant Multiplication Using Ternary Adders. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 65(7):928–932, July 2016.

[KKIZ17] Martin Kumm, Johannes Kappauf, Matei Iştoan, and Peter Zipf. Resource Optimal Design of Large Multipliers for FPGAs. In *2017 IEEE 24th Symposium on Computer Arithmetic (ARITH)*, pages 131–138, London, UK, 2017. IEEE.

[KN13] Ed Klotz and Alexandra M. Newman. Practical guidelines for solving difficult mixed integer linear programs. *Surveys in Operations Research and Management Science*, 18(1):18–32, October 2013.

[Kum16] Martin Kumm. *Multiple Constant Multiplication Optimizations for Field Programmable Gate Arrays*. Springer Fachmedien Wiesbaden, Wiesbaden, 2016.

[Kum18] Martin Kumm. Optimal Constant Multiplication Using Integer Linear Programming. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 65(5):567–571, May 2018.

[KVF23] Martin Kumm, Anastasia Volkova, and Silviu-Ioan Filip. Design of Optimal Multiplierless FIR Filters With Minimal Number of Adders. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(2):658–671, February 2023.

[KZ11] Martin Kumm and Peter Zipf. High speed low complexity FPGA-based FIR filters using pipelined adder graphs. In *2011 International Conference on Field-Programmable Technology*. IEEE, December 2011.

[KZFC12] Martin Kumm, Peter Zipf, Mathias Faust, and Chip-Hong Chang. Pipelined adder graph optimization for high speed multiple constant multiplication. In *2012 IEEE International Symposium on Circuits and Systems*. IEEE, May 2012.

[Lef01] Vincent Lefèvre. Multiplication by an Integer Constant. Research Report RR-4192, INRIA, May 2001.

[Lei97] Henri Leich. Toolbox for the design of IIR digital filters. In *Proceedings of 13th International Conference on Digital Signal Processing*, volume 2, pages 621–624, July 1997.

[LM20] Vitaly Lagoon and Amit Metodi. Deriving Optimal Multiplication-by-Constant Circuits With A SAT-based Constraint Engine. ModRef 2020 – The 19th workshop on Constraint Modelling and Reformulation, September 2020.

[Lop14] Benoit Lopez. *Implémentation optimale de filtre linéaire en arithmétique virgule fixe*. phdthesis, Université Pierre et Marie Curie - Paris VI, November 2014.

[LP83] Yong Ching Lim and Sydney R. Parker. FIR filter design over a discrete powers-of-two coefficient space. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 31(3):583–591, 1983.

[LPBG19] Martin Langhammer, Bogdan Pasca, Gregg Baeckler, and Sergey Gribok. Extracting INT8 Multipliers from INT18 Multipliers. In *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, September 2019.

[LSF+18] Vincent Lostanlen, Justin Salamon, Andrew Farnsworth, Steve Kelling, and Juan Pablo Bello. Birdvox-Full-Night: A Dataset and Benchmark for Avian Flight Call Detection. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, April 2018.

[LT13] Andrea Lodi and Andrea Tramontani. Performance Variability in Mixed-Integer Programming. In *Theory Driven by Influential Applications*, pages 1–12. INFORMS, September 2013.

[Man06] Olvi Leon Mangasarian. Absolute value programming. *Computational Optimization and Applications*, 36(1):43–53, November 2006.

[MB14] Uwe Meyer-Baese. *Digital Signal Processing with Field Programmable Gate Arrays*. Springer Berlin Heidelberg, May 2014.

[MBCCD06] Uwe Meyer-Baese, Jiajia Chen, Chip Hong Chang, and Andrew G. Dempster. A Comparison of Pipelined RAG-n and DA FPGA-based Multiplierless Filters. In *APCCAS 2006 - 2006 IEEE Asia Pacific Conference on Circuits and Systems*. IEEE, December 2006.

[NN94] Yurii Nesterov and Arkadii Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. Society for Industrial and Applied Mathematics, January 1994.

[OK92] Muhittin Oral and Ossama Kettani. A Linearization Procedure for Quadratic and Cubic Mixed-Integer Problems. *Operations Research*, 40(1-supplement-1):S109–S116, February 1992.

[OS89] Alan Oppenheim and Ronald W. Schäfer. *Discrete-time signal processing*. Prentice Hall, Englewood Cliffs, N.J, 1989.

[Par99] Keshab K. Parhi. *VLSI digital signal processing systems*. Wiley, 1999.

[Ped19] Volnei A. Pedroni. *Circuit Design with VHDL*. MIT Press, 2019.

[PSC96] Miadrag Potkonjak, Mani B. Srivastava, and Anantha 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*, 15(2):151–165, February 1996.

[PV08] Paolo Prandoni and Martin Vetterli. *Signal Processing for Communications*. EPFL Press, 2008.

[RDJ99] Anand Raghunathan, Sujit Dey, and Niraj K. Jha. Register transfer level power optimization with emphasis on glitch analysis and reduction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(8):1114–1131, August 1999.

[SP95] Neil James Alexander Sloane and Simon Plouffe. *The Encyclopedia of Integer Sequences*. Academic Press, 1995.

[Spi80] D. J. Spiegelhalter. An omnibus test for normality for small samples. *Biometrika*, 67(2):493–496, 1980.

[SRZ12] Omid Sarbishei, Katarzyna Radecka, and Zeljko Zilic. Analytical Optimization of Bit-Widths in Fixed-Point LTI Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(3):343–355, March 2012.

[SSKZ18] Patrick Sittel, Thomas Schönwälder, Martin Kumm, and Peter Zipf. ScaLP: A Light-Weighted (MI)LP Library. In *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV)*, pages 1–10, 2018.

[Stu08] Student. The Probable Error of a Mean. *Biometrika*, 6(1):1, March 1908.

[SV99] Zdeněk Smékal and Robert Vích. Optimized models of IIR digital filters for fixed-point digital signal processor. In *ICECS'99. Proceedings of ICECS '99. 6th IEEE International Conference on Electronics, Circuits and Systems (Cat. No.99EX357)*, volume 1, pages 145–148, September 1999.

[SW65] S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3-4):591–611, December 1965.

[SY11] Dong Shi and Ya Jun Yu. Design of Linear Phase FIR Filters With High Probability of Achieving Minimum Number of Adders. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 58(1):126–136, January 2011.

[TN10] Jason Thong and Nicola Nicolici. A novel optimal single constant multiplication algorithm. In *Design Automation Conference*, pages 613–616, June 2010.

[TN11] Jason Thong and Nicola Nicolici. An Optimal and Practical Approach to Single Constant Multiplication. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(9):1373–1386, 2011.

[VBVBT02] Gunter Vanuytsel, Patrick Boets, Leo Van Biesen, and Serge Temmerman. Efficient hybrid optimization of fixed-point cascaded IIR filter coefficients. In *IMTC/2002. Proceedings of the 19th IEEE Instrumentation and Measurement Technology Conference (IEEE Cat. No.00CH37276)*, volume 1, pages 793–797, May 2002.

[VHL15] Anastasia Volkova, Thibault Hilaire, and Christoph Lauter. Reliable Evaluation of the Worst-Case Peak Gain Matrix in Multiple Precision. In *2015 IEEE 22nd Symposium on Computer Arithmetic*. IEEE, June 2015.

[VIDDH19] Anastasia Volkova, Matei Iştoan, Florent De Dinechin, and Thibault Hilaire. Towards Hardware IIR Filters Computing Just Right: Direct Form I Case Study. *IEEE Transactions on Computers*, 68(4):597–608, April 2019.

[VLH17] Anastasia Volkova, Christoph Lauter, and Thibault Hilaire. Reliable Verification of Digital Implemented Filters Against Frequency Specifications. In *2017 IEEE 24th Symposium on Computer Arithmetic (ARITH)*. IEEE, July 2017.

[VP07] Yevgen Voronenko and Markus Püschel. Multiplierless multiple constant multiplication. *ACM Transactions on Algorithms*, 3(2):11, May 2007.

[Wal06] Toby Walsh. General Symmetry Breaking Constraints. In *Principles and Practice of Constraint Programming - CP 2006*, pages 650–664. Springer Berlin Heidelberg, 2006.

[Wel47] Bernard Lewis Welch. The Generalization of "Student's" Problem When Several Different Population Varlances Are Involved. *Biometrika*, 34(1-2):28–35, 1947.

[WLL10] Yu Wang, Bin Li, and Zhengdong Li. Fixed-point digital IIR filter design using multi-objective optimization evolutionary algorithm. In *2010 IEEE Youth Conference on Information, Computing and Telecommunications*, pages 174–177, November 2010.

[Wol20] Laurence A. Wolsey. *Integer Programming*. Wiley & Sons, Incorporated, John, 2020.

[ZT88] Quangfu Zhao and Yoshiaki Tadokoro. A simple design of FIR filters with powers-of-two coefficients. *IEEE Transactions on Circuits and Systems*, 35(5):566–570, 1988.

COLLEGE ED MATHEMATIQUES
DOCTORAL STIC
PAYS DE LA LOIRE

Nantes Université

Titre : Optimisation pour l'implémentation de circuits arithmétiques sans multiplication

Mot clés : multiplication par constantes, recherche opérationnelle, programmation linéaire en nombres entiers, filtre numérique

Résumé : Les calculs embarqués sont omniprésents et ont besoin d'implémentations efficaces avec des contraintes fortes sur les ressources disponibles. De telles implémentations sont chronophages et requièrent une expertise combinée entre circuits électroniques et arithmétique des ordinateurs.

Cette thèse étudie la conception automatique d'opérateurs de multiplication par plusieurs constantes (MCM), indispensables pour les calculs numériques sur Field-Programmable Gate Array. Derrière une interface simple à utiliser, l'outil proposé dans cette thèse se base sur la modélisation mathématique permettant une exploration automatique d'un grand espace de conception. Une des contributions majeures de cette thèse est l'estimation plus fine du coût matériel pour des opérateurs arithmétiques de base et une formulation sous forme d'un problème d'optimisation. Différentes techniques de recherche opérationnelle ont été explorées, incluant l'utilisation de plans coupants ou de contraintes de cassage de symétries.

Ce travail propose différentes variantes de MCM : de l'utilisation de troncatures internes vérifiant une borne d'erreur de sortie du circuit, à l'intégration de couches de pipelines. Finalement, l'évaluation des méthodes proposée démontre des gains significatifs en consommation de ressources, notamment utilisé pour la conception de filtres numériques.

Title: Towards optimized multiplierless arithmetic circuits

Keywords: multiplierless, multiple constant multiplication, digital filter, operations research, mixed-integer linear programming

Abstract: Embedded computing is ubiquitous and requires efficient implementations with tight constraints on available resources. Such implementations are time-consuming and require combined expertise between electronic circuits and computer arithmetic.

This thesis studies the automatic design of multiple constant multiplication (MCM) operators, essential for implementation on Field-Programmable Gate Arrays. Behind an easy-to-use interface, the tool proposed in this thesis is based on mathematical modeling, enabling automatic exploration of a large design space. One of the major contributions of this thesis is the finer estimation of the hardware cost for basic arithmetic operators and a formulation in the form of an optimization problem. Various operations research techniques have been explored, including the use of cutting planes or symmetry breaking constraints.

This work proposes different variants of MCM: from the use of internal truncations verifying a circuit output error bound, to the integration of pipeline layers. Finally, evaluation of the proposed methods shows significant gains in resource consumption, particularly when used for digital filter design.