Towards Arithmetic-Centered Filter Design

(Invited Paper)

Florent de Dinechin[†], Silviu-Ioan Filip[‡], Martin Kumm[*], Anastasia Volkova[◇]
[‡]Univ Lyon, INSA Lyon, Inria, CITI, France [‡]Univ Rennes, Inria, CNRS, IRISA, France
[*]Fulda University of Applied Science, Germany [◇]Université de Nantes, CNRS, LS2N, France
florent.de-dinechin@insa-lyon.fr silviu.filip@inria.fr martin.kumm@cs.hs-fulda.de anastasia.volkova@univ-nantes.fr

***Abstract*—A hardware implementation can be defined to be faithful to the frequency specification of a linear time-invariant digital filter. Filter design and implementation then become a single global optimisation problem. To solve this problem, existing tools are reviewed, and the missing ones are framed.**

***Index Terms*—Digital filter design, computer arithmetic**

I. Introduction

A. From frequency specification to time-domain architecture

Digital filters are essential components of modern technology, from medical equipment and scientific instruments to radar and navigation systems, from Hi-Fi audio and radio-enabled personal electronics to Internet of Things devices. Filter design is therefore a core topic in digital signal processing and control theory, one that has received significant research interest for half a century. Digital filters can be implemented in software, or in hardware when performance and/or power efficiency are critical. Hardware digital filters are thus found in many Application-Specific Integrated Circuits (ASIC). Some application domains (5G/6G backbones, autonomous vehicles, edge computing) rely on hardware filters implemented on Field Programmable Gate Arrays (FPGAs).

A digital filter (Fig. 1, left) is a mathematical object that inputs and outputs discrete-time, real-valued signals (or vectors of such signals). Among digital filters, a widely useful class is that of Linear Time-Invariant (LTI) filters. Linearity and time-invariance enable a range of powerful analysis and synthesis techniques. In particular, LTI filters can be defined compactly by their impulse response in the time domain, or equivalently by a transfer function in the frequency domain. These two representations can be described using a finite set of real coefficients a_i and b_i (Fig. 2, middle).

An LTI filter is usually characterized by its frequency response (Fig. 2, top): it amplifies signals in certain frequency ranges, and attenuates them in others. The general problem addressed in this article is the construction of hardware (Fig. 2, bottom) that implements a given frequency response.

Indeed, hardware circuits can also be considered as mathematical objects that input and output discrete-time signals,

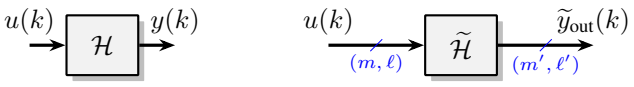

Fig. 1: An ideal filter $\mathcal{H} \in \boldsymbol{\mathcal{L}}$ and its implementation $\widetilde{\mathcal{H}} \in \boldsymbol{\mathcal{C}}$.

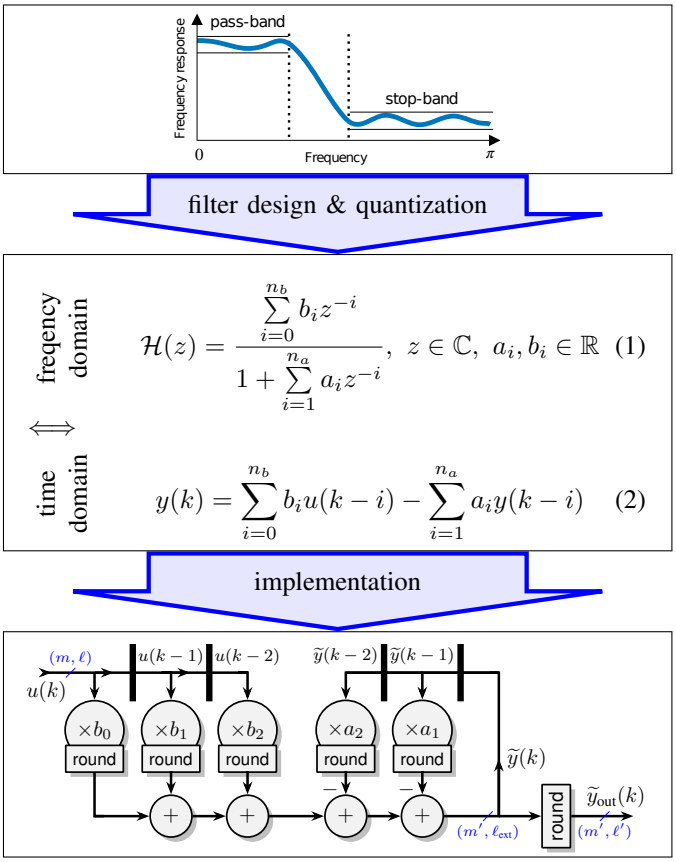

Fig. 2: From a frequency specification to an architecture.

although these signals are now also discrete in value (Fig. 1, right or Fig. 2, bottom). However, as illustrated by Fig. 3, most hardware circuits are *not* LTI filters. The inevitable presence of non-linear rounding errors in the hardware (as illustrated in Fig. 2, bottom) prevents the linearity property. Reconciling the world of LTI filters and the world of hardware circuits is the main challenge addressed by this article, with two objectives: 1) confidence in the hardware with respect to frequency specification and stability, and 2) performance.

II. Reconciling LTI filters and hardware

The starting point of this work is the classic filter design and implementation flow, as found for instance in the popular Matlab `filterDesigner` tool (previously called `fdatool`).

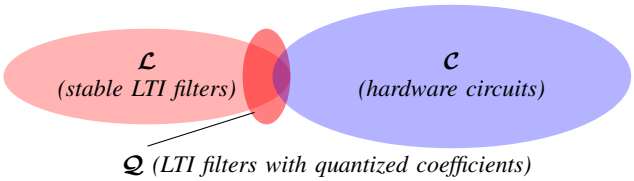

Fig. 3: LTI filters and their hardware implementation.

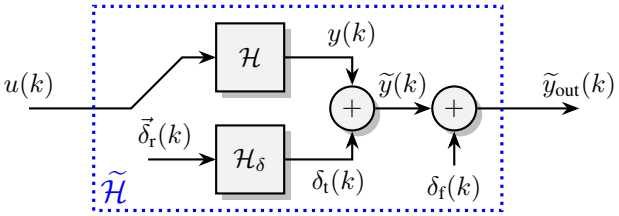

Fig. 4: Model [2] of a hardware filter $\widetilde{\mathcal{H}}$ faithful to the ideal filter $\mathcal{H}$.

Filter design (FD) consists in finding a filter adhering to a frequency response. Here, the filter is determined by the size parameters n_a and n_b, and the value of the coefficients a_i and b_i as defined in Fig. 2, middle. Mathematically, filter design consists in determining a polynomial or rational transfer function (1), using for instance a Remez-type algorithm [1] in the frequency domain. Here we are interested only in *causal* LTI filters (output only depends on the current input and past inputs/outputs) which are also *BIBO-stable* (bounded inputs yield bounded outputs). There is an infinite summation hidden in the recurrence (2), and the filter is stable iff this sum converges. Equivalently, the filter is stable if the poles of the transfer function (1) lie inside the unit circle in $\mathbb{C}$.

Two main classes of LTI filters exist: a Finite Impulse Response (FIR) filter corresponds to the simpler case $n_a = 0$ in Fig. 2. Its transfer function is a polynomial, and its architecture does not have any feedback loop. Also, FIR filters are stable by definition. Otherwise, the filter is an Infinite Impulse Response (IIR) one: its transfer function is a rational function, and its architecture will involve at least one feedback loop. In general, a large amount of different coefficient sets can realize a given frequency specification, offering a large design space.

Quantization (Q) consists in ensuring that the coefficients a_i and b_i are finitely representable (e. g., in fixed point), a prerequisite for a hardware implementation. Simply quantizing the coefficients obtained in the filter design step is likely to break the frequency response specification or even the stability (see $\mathcal{Q}$ in Fig. 3). A reason for this is that most interesting IIR filters are close to being unstable (their poles are close to the unit circle). This corresponds to a long-term memory effect in the feedback loop(s) of the architecture (Fig. 2, bottom).

Note that long-term memory can also be achieved using an FIR filter with a correspondingly large n_b: a filter designer often has a choice between a compact but potentially unstable IIR filter, or a stable but bulky FIR one. Apart from that, FIR filters are also widely used because they allow a strict linear phase response whereas IIR filters can only approximate this.

Implementation (I) consists in generating a valid hardware description (in VHDL, Verilog, or more recently C++ suitable for High-Level Synthesis) using the quantized coefficients.

In hardware, the result of a product such as $a_i y(k-i)$ requires in general more bits to be represented exactly than $y(k)$, and this is traditionally controlled by a run-time quantization of intermediate time-domain results, using some rounding scheme represented by the round boxes in Fig. 2. In IIR filters, such quantization is necessary, otherwise the infinite summation due to the feedback loop would entail an infinite growth of the number of bits needed to represent $y(k)$. This is why, in Fig. 2 (bottom), the value fed back to the multipliers is a finitely-representable approximate value $\widetilde{y}(k)$, not $y(k)$ itself as in (2).

In addition to this fundamental reason to round the intermediate results, another good reason is that truncated (constant) multipliers can be implemented using fewer resources than exact ones, in contrast to fixed-point additions. This is the reason why the multipliers are rounded in Fig. 2. An alternative would be to compute the sum of products exactly, and round it only once to $\widetilde{y}(k)$, but in general it would be more costly.

The difference between $y(k)$ and its approximation by the hardware is modeled as time-domain *rounding errors*, to be distinguished from the coefficient quantization. These errors are themselves reinjected in the feedback loop where they are amplified or attenuated by the multipliers, accumulating ad infinitum. Fortunately, this error propagation can be modeled [2] as in Fig. 4. By linearity, the rounding errors $\vec{\delta}_{\text{r}}(k)$ can be dissociated from the intermediate values computed by an ideal filter (2). Their amplification can then be modeled by a virtual filter $\mathcal{H}_\delta$, and bounded using the worst-case peak gain (WCPG) metric [3]. The intermediate data $\widetilde{y}(k)$ requires in general more bits than should be output, but this is solved with the final round box and its associated error $\delta_{\text{f}}(k)$. This worst-case error analysis provides stronger guarantees than traditional statistical quantization noise models [4].

Still, these rounding errors are not linear, therefore **a circuit with rounding errors is, strictly speaking, no longer an LTI filter**. In particular, it is no longer possible to define exactly its frequency response, its transfer function, or the poles of the latter: the mere question of knowing if the implemented filter respects the initial frequency specification is ill-posed. This fundamental problem is the root cause of many potentially catastrophic effects: loss of BIBO stability, limit cycles (when the input signal vanishes, the output signal does not), etc.

There is a small set $\mathcal{L} \cap \mathcal{C}$ of circuits that are also stable LTI filters (Fig. 3). Their coefficients must be finitely representable, and the hardware must evaluate (2) (or a mathematically equivalent formula) exactly (without any rounding). This limits this set to FIR filters (and a handful of trivial IIR filters) with more output bits than input bits, which is not typical in actual applications.

As rounding errors are unavoidable, a first contribution of the arithmetic community to filter design is to ensure that their impact is controlled, thanks to the following two definitions.

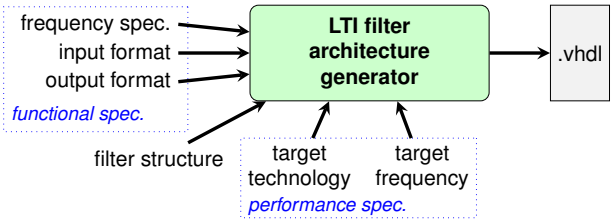

Fig. 5: A simple interface to a hardware filter design tool.

Definition 1. *A hardware circuit $\mathcal{C}$ is faithful to an LTI filter $\mathcal{H}$ iff for any input signal, the difference between the output of $\mathcal{C}$ and the output of $\mathcal{H}$ is at most one ulp (unit in the last place) of the output of $\mathcal{C}$.*

Definition 2. *A hardware filter $\mathcal{C} \in \boldsymbol{\mathcal{C}}$ is faithful to a frequency specification iff there exists an LTI filter $\mathcal{H} \in \boldsymbol{\mathcal{L}}$ that fulfills the frequency specification and $\mathcal{C}$ is faithful to $\mathcal{H}$.*

Definition 1 is simply the application of a classical notion of computer arithmetic to digital filters, although ensuring it is non-trivial [2]. Definition 2 is, to our knowledge, original. It has several advantages:

1) It does not make the circuits linear, but it does ensure that non-linearities are harmless in practice. For instance, if the ideal filter $\mathcal{H}$ is stable, the amplitude of possible instability or limit cycles in $\mathcal{C}$ will be less than one ulp: they belong to the last-bit noise that filter designers must live with.

2) It enables the minimalistic interface to a filter design tool depicted in Fig. 5. Tools such as Matlab's `filterDesigner` expose many other parameters: n_a, n_b, the quantization parameters of the a_i and b_i, the values of these coefficients, the internal formats such as that of $\widetilde{y}(k)$, etc. A wrong choice of these parameters may entail an unstable implementation. We claim that most of these parameters are implementation technicalities that should be determined automatically. There should only remain a handful of knobs controlling, e. g., cost/performance trade-offs. One important knob is the filter structure which can be seen as a performance specification (degree of parallelism) or a functional specification (e. g., to ensure linear phase). It can be a user parameter for now, with the long-term goal to explore it automatically.

3) It enables the definition of *hardware filter design as an optimization problem*, building upon existing literature that has formalized and solved relevant sub-problems.

III. Hardware filter design as an optimization problem

An optimization problem is defined by a parameter space, a set of constraints, and an objective function based on cost and performance metrics.

The **parameter space** includes the filter order (n_a and n_b), and the (quantized) coefficients (a_i and b_i), but also several other parameters, including two which are extremely important in the construction of an architecture.

The first is the *arithmetic* to be used internally, in particular for constant multiplication. Two families of methods exist [5], one based on shift-and-add [6]–[9], the other one on precomputed tables [10], [11]. Each method offers the possibility of sharing intermediate results in *multiple constant multiplication* to improve area, performance, or power. This defines a wide spectrum of low-level filter realizations that depend on the coefficient values in a non-trivial way.

The second important parameter is the *filter type and structure*: for both FIR and IIR filters, there is a range of possible realizations [4]. Each realization corresponds to some algebraic rewriting of (2), and the direct form of Fig. 2 (bottom) is among the simplest. Other well-studied realizations include lattice wave filters [12], filter cascades [13], etc. A long-term purpose here would be to improve the qualitative and experimental know-how of filter designers ("this structure is more stable than that one in this context") with quantitative assessments of the cost of a stable-by-design implementation using each candidate structure.

The **constraints** of the problem include those capturing the *frequency-domain specification*. These constraints will be rigorously expressed and satisfied even in the presence of rounding errors in the FD+Q software, thanks to safe arbitrary-precision techniques [1], [3]; We propose here to complement them with *time-domain accuracy* constraints, ensuring that the implemented filter will be faithful according to Definition 2.

The **cost functions** may include high-level metrics, typically the number of adders in a shift-and-add filter. However, with rounded intermediate results, all the adders are not of the same size, and finer metrics taking this into account should be used, providing accurate gate-level models of area, performance and power consumption [14]–[16].

IV. Methods and tools: past, present & future

A review of these three aspects shows that the state of the art is solving increasingly relevant subproblems, but that a tool addressing our complete optimization problem is still missing.

A. Solutions to sub-problems

The combination of the FD & Q steps have been studied since the 1960's [17], and can even be regarded as solved for certain practical instances of fixed-point FIR design [18], [19].

A large body of work exists for the I step. Some methods explore filter structures. The structural adders in FIR filters can be rearranged to reduce their word size [20]. A FIR filter may be factored into a cascade of sub-sections [13]. Several approaches to the optimization of the internal word sizes exist [21, Ch. 4].

Other hardware optimizations in the I step target the arithmetic hardware. The implementation of multiple constant multiplications (MCM) can be optimized using shift-and-add [6]–[9] or using precomputed tables [10], [11].

However, the result of these optimizations in the I step depend strongly (and in a non-monotonous way) on the coefficient values, which are fixed in the earlier FD and Q steps. The way to go is therefore a joint optimization of the FD+Q+I steps, finding the set of fixed-point coefficients that minimize the overall cost of a MCM implementation

while respecting the frequency specification. We are almost there for FIR filters (in direct or transposed form), either using a custom branch and bound algorithm [22] or integer linear programming (ILP) [23]. Another combined FD+Q+I optimization for FIR filters focuses on power consumption [16], by searching for shift-and-add solutions with low adder depth using a combination of branch and bound and linear programming. However, all these works only find filters in $\mathcal{L} \cap \mathcal{C}$, thus don't respect proper rounding which restricts their practicality. For IIR filters, FD+Q+I optimization is currently limited to second-order filters [24]. Finally, none of these approaches can claim optimality in terms of bit-level cost.

Conversely, a combined Q+I optimization [2] with bit-level cost models (albeit only for the easier table-based constant multipliers) also misses global optimality, since it requires a separate FD step.

B. Existing tools, and tools to build

In order to implement the tool from Fig. 5, several state-of-the-art open-source components may be used. For solving ILP problems, numerous generic solvers are available, and the ScaLP library[1] provides a versatile C++ interface with a unified API. For the exploration of various filter structures and their error analysis, the Python-based FiXiF toolbox[2] offers an efficient and universal internal representation. It also includes a reliable implementation of the WCPG [3]. As a hardware backend, the FloPoCo generator[3] offers the necessary arithmetic, and already relies on ScaLP and WCPG along with other state-of-the-art tools such as Sollya and MPFR.

Early prototypes[4][5] [23], [24] of Fig. 5 have already been built upon these components among others (the IIR optimization problem [24] is modeled in Julia). We expect that the progress towards our ideal tool will require yet more components, in particular solvers beyond ILP.

V. Conclusion

This position paper has overviewed the state of the art in arithmetic approaches for filter design, defined how hardware filters can be faithful to a frequency specification, and sketched a global optimization problem that now remains to be completely formalized, then solved. This will probably require several incremental steps from the current state of the art, and the incremental development of more software tools.

The joint FD+Q+I optimization process we envision will improve the confidence in the generated hardware, with unstable hardware filters or limit cycle oscillations becoming a thing of the past. It will also significantly improve performance: preliminary results [24] for second-oder IIR filters on FPGAs show an average 42% improvement in area and 21% improvement in delay over filters obtained with the classic multi-step approach.

[1]https://digidev.digi.e-technik.uni-kassel.de/scalp

[2]https://github.com/fixif

[3]https://flopoco.org

[4]https://gitlab.com/filteropt/firopt

[5]https://gitlab.univ-nantes.fr/volkova-a/jiir2hw

Acknowledgements: Many thanks to Gabriel Charlet, Jean-Marie Gorce and Thibault Hilaire for their time discussing these subjects. This work was partially funded by the ANR Imprenum project.

References

[1] S.-I. Filip, "A robust and scalable implementation of the Parks-McClellan algorithm for designing FIR filters," *ACM Transactions on Mathematical Software*, vol. 43, no. 1, pp. 1–24, 2016.

[2] A. Volkova, M. Istoan, F. De Dinechin, and T. Hilaire, "Towards hardware IIR filters computing just right: Direct form I case study," *IEEE Transactions on Computers*, vol. 68, no. 4, pp. 597–608, 2018.

[3] A. Volkova, T. Hilaire, and C. Lauter, "Arithmetic approaches for rigorous design of reliable Fixed-Point LTI filters," *IEEE Transactions on Computers*, vol. 69, no. 4, pp. 489–504, 2019.

[4] A. Antoniou, *Digital Signal Processing: Signals, Systems, and Filters*. McGraw-Hill Education, 2005.

[5] F. de Dinechin, S.-I. Filip, M. Kumm, and L. Forget, "Table-based versus shift-and-add constant multipliers for FPGAs," in *IEEE Symposium on Computer Arithmetic (ARITH)*. IEEE, 2019, pp. 151–158.

[6] Y. Voronenko and M. Püschel, "Multiplierless Multiple Constant Multiplication," *ACM Trans. on Algorithms*, vol. 3, no. 2, pp. 1–38, 2007.

[7] O. Gustafsson, "A Difference Based Adder Graph Heuristic for Multiple Constant Multiplication Problems," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2007, pp. 1097–1100.

[8] L. Aksoy, E. da Costa, P. Flores, and J. Monteiro, "Exact and approximate algorithms for the optimization of area and delay in multiple constant multiplications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 6, pp. 1013–1026, 2008.

[9] M. Kumm, "Optimal Constant Multiplication using Integer Linear Programming," in *IEEE Int. Symp. on Circ. and Systems (ISCAS)*, 2018.

[10] K. D. Chapman, "Fast Integer Multipliers Fit in FPGAs," *Electronic Design News*, 1994.

[11] M. Faust and C.-H. Chang, "Bit-parallel Multiple Constant Multiplication using Look-Up Tables on FPGA," *IEEE International Symposium of Circuits and Systems (ISCAS)*, pp. 657–660, 2011.

[12] A. Fettweis, "Wave digital filters: Theory and practice," *Proceedings of the IEEE*, vol. 74, no. 2, pp. 270–327, 1986.

[13] A. Mehrnia and A. N. Willson, "Optimal factoring of FIR filters," *IEEE Trans. on Signal Processing*, vol. 63, no. 3, pp. 647–661, Feb. 2015.

[14] K. Johansson, O. Gustafsson, and L. Wanhammar, "A Detailed Complexity Model for Multiple Constant Multiplication and an Algorithm to Minimize the Complexity," *European Conference on Circuit Theory and Design*, vol. 3, pp. III/465–III/468 vol. 3, 2005.

[15] X. Lou, Y. J. Yu, and P. K. Meher, "Fine-Grained Critical Path Analysis and Optimization for Area-Time Efficient Realization of Multiple Constant Multiplications," *IEEE Transactions on Circuits and Systems I*, vol. 62, no. 3, pp. 863–872, Jul. 2015.

[16] W. B. Ye, X. Lou, and Y. J. Yu, "Design of Low Power Multiplierless Linear-Phase FIR Filters," *IEEE Access*, pp. 23 466–23 472, 2017.

[17] J. Knowles and E. Olcayto, "Coefficient Accuracy and Digital Filter Response," *IEEE Trans. on Circ. Th.*, vol. 15, no. 1, pp. 31–41, 1968.

[18] D. M. Kodek, "LLL algorithm and the optimal finite wordlength FIR design," *IEEE Trans. on Sig. Proc.*, vol. 60, no. 3, pp. 1493–1498, 2012.

[19] N. Brisebarre, S.-I. Filip, and G. Hanrot, "A lattice basis reduction approach for the design of finite wordlength FIR filters," *IEEE Transactions on Signal Processing*, vol. 66, no. 10, pp. 2673–2684, 2018.

[20] J. Chen, C.-H. Chang, J. Ding, R. Qiao, and M. Faust, "Tap delay-and-accumulate cost aware coefficient synthesis algorithm for the design of area-power efficient FIR filters," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. PP, no. 99, pp. 1–11, 2017.

[21] G. A. Constantinides, P. Y. Cheung, and W. Luk, *Synthesis and optimization of DSP algorithms*. Kluwer, 2004.

[22] L. Aksoy, P. Flores, and J. Monteiro, "Exact and Approximate Algorithms for the Filter Design Optimization Problem," *Signal Processing, IEEE Transactions on*, vol. 63, no. 1, pp. 142–154, 2015.

[23] M. Kumm, A. Volkova, and S.-I. Filip, "Design of Optimal Multiplierless FIR Filters with Minimal Number of Adders," *preprint*, 2019. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02392522

[24] R. Garcia, A. Goldsztejn, J. Kühle, M. Kumm, and A. Volkova, "Optimal design of multiplierless second-order infinite impulse response digital filters," *preprint*, 2021. [Online]. Available: https://hal.archives-ouvertes.fr/hal-03208221