Automatic generation of hardware FIR filters from a frequency domain specification

Nicolas Brisebarre, Florent de Dinechin, Silviu-Ioan Filip, Matei Istoan

Automatic generation of hardware FIR filters from a frequency domain specification

***Abstract*—Digital filtering operations are a prime example of the flexibility and opportunities offered by solutions using reconfigurable hardware. This article proposes an efficient solution for the automatic design and synthesis of finite impulse response (FIR) filters targeting FPGAs. Based on a novel approach to the filter coefficient quantization problem, it produces results which are faithful to a high-level frequency-domain specification. An automated design process is also proposed where user intervention is limited to a very small number of relevant input parameters. Computing the optimal value of the other parameters not only simplifies the user interface. The resulting architectures also outperform those generated by mainstream tools in accuracy, performance, and resource consumption.**

I. Introduction

This article presents a new open-source, integrated synthesis framework for finite impulse response (FIR) filters on FPGAs. Written in C++, its goal is to link state-of-the-art open-source tools to offer guaranteed-quality VHDL filters with best-of-class performance. This link is not trivial, because some of these tools work in the frequency domain, and some work in the time domain.

A. Digital filters: from specification to implementation

A digital filter is usually specified in the frequency domain (FD) by its frequency response, an example of which is provided in Figure 3. An actual FIR filter implementation, however, computes sums of products in the time domain (TD). An example hardware implementation is provided in Figure 1. To obtain such an implementation of a FIR filter out of its FD specification, the traditional approach can be summarized as a three-step process:

1) determine the filter length N and the real coefficients $\{h_k\}_{k=0}^{N-1}$ of an appropriate frequency response

$$H(\omega) = \sum_{k=0}^{N-1} h_k e^{-ik\omega}, \quad \omega \in [0, 2\pi); \tag{1}$$

2) quantize the obtained coefficients $\{h_k\}_{k=0}^{N-1}$ to some machine-representable formats (the values $\{\widetilde{h}_k\}_{k=0}^{N-1}$), while trying to minimize the degradation in quality of the frequency response from equation (1);
3) implement the time-domain computation

$$y_k = \sum_{i=0}^{N-1} \widetilde{h}_i x_{k-i}, \tag{2}$$

where x_i are the input signal values, given in some machine-representable format.

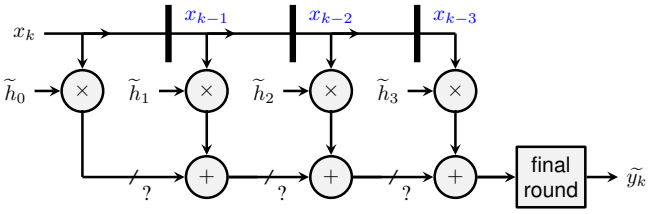

Fig. 1. A filter architecture

This article fuses several previous works that independently address these steps: equiripple designed filters using the Parks-McClellan algorithm [1], [2] for step 1, Euclidean lattice-based coefficient quantization [3] for step 2, and FloPoCo[1]-generated sum of products with constants (SPC) operators [4] for step 3.

B. Implementation parameter space

Each of these three steps can be controlled by some parameters, which can be roughly grouped into:

- functional, frequency-domain parameters and constraints (e.g. bands of frequency response);
- architectural, time-domain parameters and constraints: the fixed-point format of x_k (or input format); the format of y_k (output format); the format of the intermediate products and sums, represented by a ? on Figure 1;
- performance and resource consumption constraints.

Ideally, the quality of a filter is specified in the frequency domain. However, the architectural choices made in the time domain obviously also constrain this quality. For instance, however clever the filter design may be, the quantization noise added by the I/O formats will impose a limit to the filter quality. It is difficult to assess accurately the impact of such time-domain parameter choices in the frequency domain. The opposite is true, too: however accurate may be the inputs and outputs, the choice of N will limit the quality of the filter, for instance.

To address this difficulty, current mainstream design flows tend to expose all these parameters. To illustrate this, we consider in this work the versatile and widely used MATLAB™'s Signal Processing Toolbox, which goes from the specification to a synthesizable (VHDL or Verilog) hardware description (figure 2). Alternatives like GNURadio[2] or the Scipy `signal` package[3] do not support quantization or fixed-point synthesis.

[1] http://flopoco.gforge.inria.fr
[2] http://gnuradio.org
[3] http://docs.scipy.org/doc/scipy/reference/signal.html

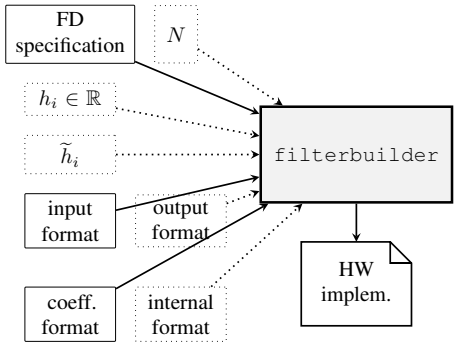

Fig. 2. MATLAB design flow with all the parameters. The dashed ones can be computed by the tool.

For this reason, we will not consider them further. However it is hoped that the present work could be a missing link from specification to FPGA implementation in GNURadio, complementary to FPGA back-end work [5], [6].

C. From specification to architecture in MATLAB

MATLAB offers two similar graphical user interfaces (GUI), `filterbuilder` and `fdatool`.

For the first step, one can either use a precomputed filter, or specify frequency constraints for the target filter and let the tool determine an adequate frequency response. MATLAB uses double-precision arithmetic for this step as a good enough approximation of real numbers (experiments in [2] show that for large filters, it is not good enough).

When this is done, one may launch the implementation step. The fixed-point formats for the inputs x_i must obviously be specified, as it defines the interface of the hardware component to be generated. The MATLAB flows also demands that the format of the coefficients $\widetilde{h}_i$ be specified. This (our second step) is a non-trivial task if we want to ensure that the filter with quantized coefficients still satisfies the frequency-domain constraint. However, it is left to the user. The `minimizecoeffwl` routine can help, but it does not seem to be integrated directly with either GUI tool, and as such must be used in an external manner.

For step 3, by default the design is carried out in such a way that the output precision of y_k is *maximal*, i.e. there is no rounding in the computation of (2). This is usually overkill, and necessary leads to an output format much larger than the input format, which is not a desirable goal in a signal processing filter. Therefore the output format can also be given, but then it must be specified together with the formats used to store intermediate addition and multiplication results in (2).

As Section IV will illustrate, this freedom in terms of parametrization makes the task of controlling *both* output precision and quantization quality difficult, especially when also trying to optimize for hardware resources. If an implementation does not match the specification, it is easy to detect. However, as soon as an implementation matches the specification, it is very difficult to convince oneself that it is the best one, i.e. the one that minimizes the cost in terms of resource consumption.

D. Objective and outline of the present work

Therefore, an overall objective of the present work is to deduce as many as possible of the architectural parameters from the functional ones, or, in other words, to automate as much as possible of the parameter space navigation.

We show in Section III how optimal values of most implementation parameters can be deduced from 1/ the FD filter specification and 2/ the desired input/output formats of the implementation. A nice side effect is a huge simplification of the user interface, since the user only requires to express these parameters. This has been implemented as an integrated open-source tool. Section IV shows that this tool leads to highly efficient architectures, thanks to the use of just the right precision in the internal computation. Before this, Section II briefly presents in more details the three steps and the state-of-the-art way to address them.

II. Background

Many filtering tasks require a frequency response $H(\omega)$ which has linear phase. For FIR filters used in practice, this means that $\{h_k\}_{k=0}^{N-1}$ are chosen to be symmetric or antisymmetric with respect to the middle coefficient(s) [7, Sec. 5.7]. For expository purposes, we only consider type I linear phase filters in the sequel (*i.e.*, symmetric coefficients and odd N), but our method work in the general case also.

A. Frequency-domain versus time-domain specification

Frequency domain specification of H is generally carried out in terms of a L^∞ formulation, a practice we also follow here. Given Ω, a compact subset of $[0, \pi]$, an ideal frequency response D, continuous on Ω, and a weight function W positive and continuous over Ω, we want to determine H such that the weighted error function $E(\omega) = W(\omega)(D(\omega) - H(\omega))$ has minimal uniform norm

$$\delta = \|E(\omega)\|_{\infty,\Omega} = \sup_{\omega\in\Omega} |E(\omega)| . \tag{3}$$

In the time domain, we can similarly define the error of an architecture computing $\widetilde{y}_k$ and supposed to compute y_k as $\varepsilon(x_k) = |\widetilde{y}_k - y_k|$, and our specification will be given as a bound $\bar{\varepsilon}$ on $\varepsilon(x_k)$.

There is a deep relationship between this bound and the output format, based on the following observation. On the one hand, it is not possible to output more accuracy than the format can hold. On the other hand, it makes little sense to output less accuracy than the output format allows: In a hardware context where we pay for every bit, we want to only output bits that hold useful information. Therefore, following [4], in our approach, the accuracy of the architecture is specified by the output format.

However, this remains a time-domain accuracy specification, with respect to a time-domain ideal function. How we relate it to the frequency-domain accuracy specification will be the subject of Section III.

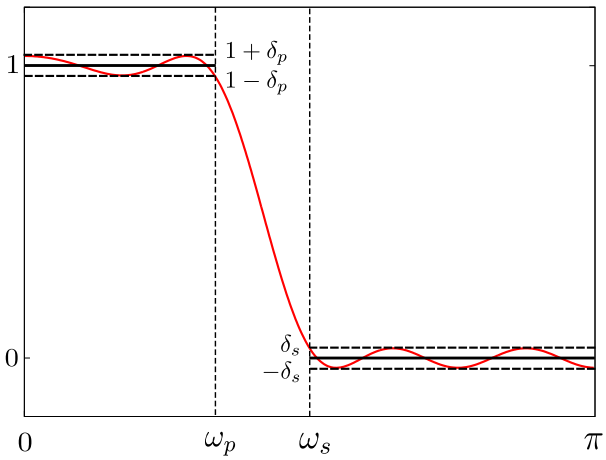

Fig. 3. Prototype lowpass minimax FIR filter

B. Minimax filter design via the exchange method

Finding the optimal set of real-valued taps $\{h_k\}_{k=0}^{N-1}$ which minimize δ in (3) is a classic Approximation Theory problem. The usual approach to solve it is to use the Remez exchange algorithm [8], which in Signal Processing is more commonly known as the Parks-McClellan algorithm [9]. It is a very robust approach. In our tool, we use the implementation from [2], which is capable of outputting extremely accurate filters.

Depending on the design requirements, we produce:

(a) filters of minimal length N satisfying some given error constraints on Ω;
(b) for a specified N, filters which ensure a certain target error on some part of Ω.

As an example, consider the design of a prototypical lowpass system (Figure 3) with passband $[0, \omega_p]$ and stopband $[\omega_s, \pi]$. It is defined on $\Omega = [0, \omega_p] \cup [\omega_s, \pi]$ with

$$D(\omega) = \begin{cases} 1, & \omega \in [0, \omega_p], \\ 0, & \omega \in [\omega_s, \pi]. \end{cases}$$

For a type (a) specification, the target is to minimize N, while the approximation error $|D - H|$ is upper bounded by δ_p on the passband and by δ_s on the stopband. Taking

$$W(\omega) = \begin{cases} \frac{\delta_s}{\delta_p}, & \omega \in [0, \omega_p], \\ 1, & \omega \in [\omega_s, \pi], \end{cases}$$

reduces the problem to: find the smallest N for which the minimax error $\delta \leqslant \delta_s$. The resulting H is shown in red in Figure 3. A simple way of determining it requires an initial guess for N. Kaiser [10] gave the formula

$$N \approx \frac{-10 \log_{10}(\delta_p \delta_s) - 13}{2.324(\omega_s - \omega_p)},$$

which was later refined in [11]. One iteratively modifies this degree and applies the exchange algorithm until $\delta \leqslant \delta_s$. More information can be found for instance in [12, Sec. 15.7–15.9].

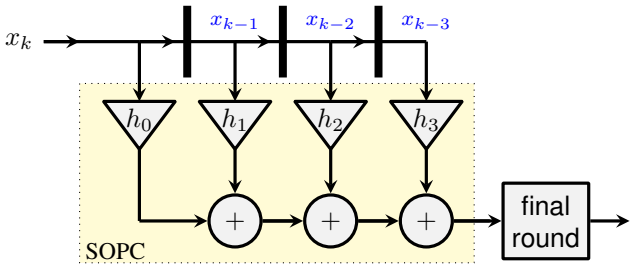

Fig. 4. Abstract filter architecture

When N is fixed, assume the goal is to bound δ_s by δ (the process for δ_p is analogous). By setting

$$W(\omega) = \begin{cases} w_p, & \omega \in [0, \omega_p], \\ 1, & \omega \in [\omega_s, \pi], \end{cases}$$

we can iteratively adjust w_p until, again, $\delta \leqslant \delta_s$.

C. A robust quantization scheme

To address the coefficient quantization issue, we use the recent approach described in [3], based on the LLL algorithm [13]. It inputs the value of the quantum $2^p, p \in \mathbb{Z}$, and considers filters with coefficients of the form $\frac{m_k}{2^p}, m_k \in \mathbb{Z}$. Among these, it quickly finds a filter with minimal (or close to minimal) FD error norm $\sup_{\omega \in \Omega} \left| W(\omega)(D(\omega) - \widetilde{H}(\omega)) \right|$.

This technique does not directly provide the optimal coefficient format, but it is fast enough to be iterated over with increasing values of p until the target FD error is matched.

The main advantages of this quantization scheme are its scalability to large degrees and the excellent quality of the results it produces, when compared to other quantization methods [3, Sec. 5].

D. Sum of product generation

The third step, in the present work, is based on the automation of sum-of-product-by-constant (SOPC) introduced in [4] (a schematic view is given in Figure 4). This work introduced two architectural innovations: the optimization of the KCM constant multiplication algorithm [14] to fixed-point format, and the use of a bit heap [15] to fuse the summations inside the KCMs into the summation of (2). However, the main focus of this work is to compute optimal values of all the architectural parameters that ensure last-bit accuracy of the architecture. This is achieved thanks to an automated error analysis that we have adapted to the present work.

An innovation of [4] that we don't exploit here is that constants could be input to the tool as arbitrary real numbers: quantization was part of the architecture generation. As the sequel will show, the present approach needs to check the FD specification on quantized coefficients, so their quantization must be done before architecture generation. Of course, quantized coefficients are still real numbers, so the architecture generation of [4] works just as well on already quantized coefficients.

III. Integrated hardware FIR filter design

This section presents the integrated toolflow that we implemented. We have stated the objective of simplifying the user interface, however it is important that the designer stays in control of the implementation quality. A discussion of this issue in III-A identifies to two use-cases, each with its distinct interface. The following sections, III-B and III-C, present the algorithms that implement these two minimal interfaces.

A. What is the minimal specification?

The tools presented in section II will enable us to compute optimal values of the coefficient formats and the internal formats: there is no reason why we should burden the designer with these parameters.

However, we still want to leave him the control of the quality of the generated architecture. More precisely, the quality of the frequency-domain response is an inherent trade-off between accuracy of the approximation in the frequency domain (controlled by the parameter N) and the accuracy in the time domain (controlled by the output precision as exposed in II-A).

Because of this trade-off, if one of these controls is supplied, we can deduce the optimal value of the other.

This leads to two use cases:

1) The user provides N. What is the smallest relevant output format, such that the implemented filter satisfies the target FD specification?
2) The user provides the output format. What is the smallest N such that the implemented filter satisfies the target FD specification?

The second use case is probably more intuitive in a hardware context, when I/O formats are part of the specification.

Our approach to address these questions is detailed in the remainder of this section. While not following the simple linear flow from Section I-A, the three main steps introduced there are still present in our proposed flow. Step 1 corresponds to the *FD design* and *minimize N for FD design* boxes from Figures 5 and 6, Step 2 to the *quantization* box, whereas Step 3 is identified by the *implement* $\sum \widetilde{h}_i \cdot x_i$ box.

B. Computing a relevant output format from the value of N

The first question leads to the design flow highlighted in Figure 5. The FD specification, target length N and input format are the only required parameters from the user. Step 1 finds a filter with real coefficients $\{h_i\}_{i=0}^{n-1}$, satisfying the FD specification. Then the loop involving the quantization block determines the smallest coefficient format for which its corresponding quantization $\left\{\widetilde{h}_i\right\}_{i=0}^{n-1}$ still remains consistent with the FD specification. This verification is carried out by the quantization tool from Section II-C, which computes $\left\|\widetilde{E}\right\|_\Omega$ for this purpose. Before passing to the implementation phase (Step 3 of the classic flow), we determine the smallest relevant output format based on the input and quantized coefficient format previously computed. Once this intermediary step is performed, we can generate the implementation. During this phase, the internal computational formats are determined automatically to ensure our target output accuracy.

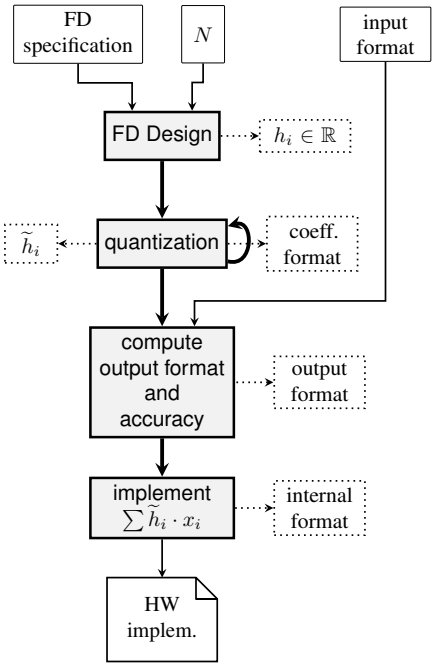

Fig. 5. First proposed design flow, with the parameters computed at each step

C. Computing the optimal filter out of the I/O format specification

Figure 6 deals with the second use case. The input and output formats and FD quality bounds are given. It might happen that there is no solution with these constraints, in which case we alert the user. The first step, again, determines an acceptable real-coefficient filter with minimal N (see Section II-B). This is to limit the implemented filter complexity as much as possible. The next box determines the minimal coefficient and internal formats needed to satisfy the TD constraints. Based on this coefficient format, we perform coefficient quantization. Since the TD constraints hold for the quantized coefficients (the coefficient format was chosen to guarantee this), the next step in the design flow is to ensure that the FD specifications still hold as well. If this is not the case, we can do one of the following (or a combination of the two):

1) increase the coefficient and internal formats until the FD constraints are satisfied;
2) increment N and restart the design process from Step 1.

The choice between these steps is a trade-off. We already know from Step 1 that the filter can satisfy the FD specification. What remains to be determined is the necessary coefficient and internal format. When we observe that looping with the first choice does not have any relevant impact on the results, we turn to the second choice and increase the number of coefficients N, which could guarantee a faster convergence towards a set of formats satisfying both TD and FD specifications. Once we arrive at an acceptable result, we can implement the filter.

An alternative approach to the one in Figure 6 is to perform the quantization before determining the required internal format

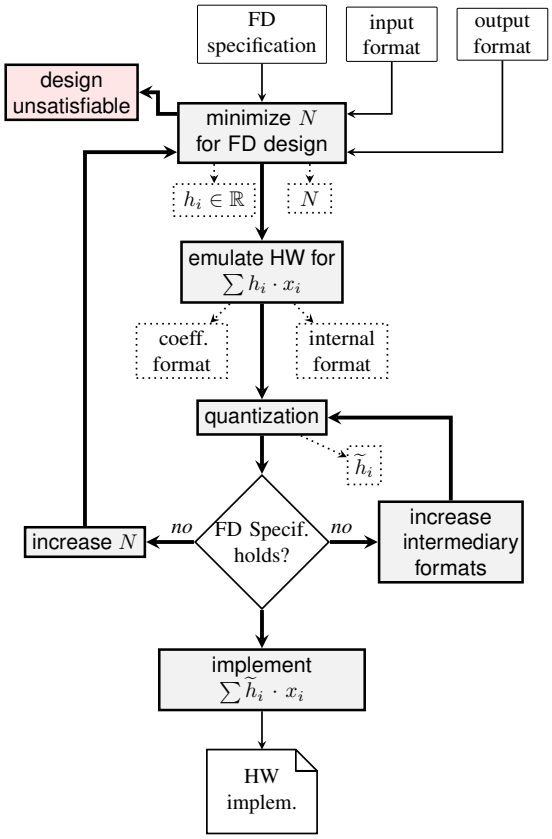

Fig. 6. Second proposed design flow

(the emulate hardware box). Thus, we would first fix the coefficient format, and then determine a corresponding internal format. The rest of the design process remains the same.

Both approaches are based on heuristics. Determining which one performs better in most cases is not yet clear and is subject of ongoing work.

IV. Examples and Results

The goal of this section is to highlight, in detail, the design flow and robustness of our approach, when compared to other mainstream design suites.

A. Working example and experimental setup

As a working example, the default high-pass filter specification of the `filterbuilder` command in MATLAB™ R2014B illustrates well the scope of our tool. It is a high-pass FD specification with $\Omega = [0, 0.45\pi] \cup [0.55\pi, \pi]$ and

$$D(\omega) = \begin{cases} 0, & \omega \in [0, 0.45\pi] \quad \text{(stop band)} \\ 1, & \omega \in [0.55\pi, \pi] \quad \text{(pass band)} \end{cases}$$

The error bounds are specified as a passband ripple below 1dB ($\delta_p \leqslant 0.0575$) and a stopband attenuation of at least 60dB ($\delta_s \leqslant 10^{-3}$). We also fix the input format to $(w_i, f_i) = (16, 15)$ in Matlab's fixed-point format notation: total width of 16 bits, out of which 15 fractional bits. The output has $f_o = 15$ fractional bits.

In order to compare the performance of our tool with respect to Matlab's `filterbuilder` from the hardware point of view, we generated and synthesized several variants of this filter.

Syntheses were performed using Xilinx ISE 14.7, using the Virtex6 xc6vcx75t-2ff484 FPGA as a target device, with design goals set to balanced. The performance of the designs is measured as FPGA resource consumption (number of look-up tables (LUT) and registers (Reg.)) and maximum operating frequency at a given latency.

In order to evaluate the accuracy of the generated architectures, we also emulated them (both `filterbuilder`-generated and ours) using the MPFR[4] multiple-precision library, on 10^6 randomly-generated inputs. We measured the maximum absolute difference between the result computed by the architecture (in its internal format, just before the final rounding to the $(16, 15)$ format), and a very large precision simulation of the real filter (before any quantization). We call this maximum error "implementation accuracy" in the sequel.

B. Steps 1 and 2

We are in the second scenario of Section III (with the design flow given by Figure 6) and start by determining a minimal N for which the FD specification remains satisfied when the filter has real coefficients. The `filterbuilder` result is a type I filter with $N = 45$ coefficients. Our result is a type I filter with $N = 43$, which, already, gives a slightly more efficient design. With coefficients quantized to 2^{-21}, its passband ripple is smaller than 0.9 dB and a stopband attenuation larger than 61 dB, so it matches the FD specification.

C. MATLAB's overwhelming choice

Table I shows how the performance and the resource utilization depend on the three fixed-point formats that appear in Figures 2, 5, and 6. It is a good illustration of the complexity incurred by the vast design parameter space.

Here we assume that the output format is identical to the input format, and fixed to (16,15). It remains to jointly optimize the coefficient format and the internal computation format, for an efficient hardware implementation. The upper part of this table is obtained using `filterbuilder`. The table reports results where these formats are set to the input format, then to larger ones. The phrase "full prec" denotes a format on which all the multiplications and additions will be computed exactly (no rounding). The phrase "smart prec" denotes that we reported in `filterbuilder` the format found by our tool. We show in the sequel how it can be achieved by trial and error in MATLAB.

The first 8 lines of the table keep the coefficient format fixed at $(16, 15)$, while gradually augmenting the precision of the internal computations. It can be seen that the implementation accuracy reaches a limit at $(22, 21)$, after which incrementing the internal computation precision only increases the resource consumption. The implementation accuracy is also much lower

[4] http://www.mpfr.org/

than what is required, the designs providing only 9 meaningful bits, as opposed to 15, as requested in the specifications.

The following two groups of results provide a different scenario. The internal computation format is fixed, at $(16, 15)$ and then at $(23, 21)$, while the coefficient quantization format is varied. As in the previous scenario, the output accuracy reaches a limit. The format $(23, 21)$ is chosen (by error analysis) so as to avoid the accumulation of errors due to the internal computations. The second half of the scenario also shows that once we have enough information in the quantized coefficients, the implementation accuracy requirement can be satisfied.

The last scenario in Table I first shows the results of leaving the decisions concerning the used formats (internal computations and coefficient quantization) up to `filterbuilder`. The notations *full precision* here denote that the tool determines the used formats. The first line provides best implementation accuracy (more than 24 bits), but at a very high cost. This is wasteful, because 8 of these these 24 bits can not be expressed in the output format.

To alleviate this problem, we then force the tool to use a smaller coefficient quantization format $(22, 21)$, denoted *smart precision* in the table). This precision could be determined by error analysis so that there are no errors propagated in the final results at the given output precision. This choice does increase the efficiency of the design, as can be seen in the table. We next tried to improve the design even further, by reducing the format of the internal computations as well. Strangely, due to implementation choices made in `filterbuilder`, the resource consumption is, *higher* than compared to using full precision computations and the design is *less* efficient.

Such an exploration of the parameter space takes time. It is non trivial to obtain from MATLAB the implementation accuracy data (or an equivalent signal/noise measure). Our approach replaces it with a-priori error analyses (respectively detailed in [2]–[4]) that are both safer and faster.

D. Automating the choice leads to better performance

The last two lines of Table I present data about the filter generated using our approach in less than one minute. It is better than MATLAB-generated filters in resource and performance, while achieving the last-bit accuracy target.

The last line of the MATLAB table and the line obtained by our tool use the same parameter set. They are therefore particularly interesting to compare. The accuracy is indeed comparable. The slightly better accuracy in our approach probably comes from the a-priori error analysis that integrates the effect of all rounding errors, including the final rounding to the target format.

The comparison of resource and performance justifies the architectural choices made by FloPoCo. MATLAB uses integer shift-and-add multipliers that compute 38 bits, half of which are then rounded out. Our FixRealKCM, conversely, just computes the right number of bits sufficient to achieve the desired accuracy. Besides, we perform the operations on a format (w, f) with $f = 21$, but the total width w depends on the coefficient. Finally, the additions are performed using a bit heap in our approach, and using rows of additions in the MATLAB-generated code. Which of these optimizations contributes more to our good results is difficult to evaluate.

V. Conclusion

This paper has presented two new heuristic approaches (Sections III-B and III-C) for automating the fixed-point FIR filter design process for FPGAs. A real-size filter implementation of guaranteed quality is obtained within seconds. Apart from requiring minimal user intervention, this tool produces results which are more accurate, almost twice as fast, and more than twice more resource-effficient than what can be generated with similar mainstream tools.

This result is obtained thanks to a pervasive concern to minimize the size of computations in the resulting architecture, coupled to strict error evaluation techniques to ensure the numerical quality. The main issue that this work has raised is the relationship between quality specification in the frequency domain and quality specification in the time domain. We have in both domains well understood and well implemented error analysis procedures. We could design efficient heuristics that ensure that the design meets the specification in both domains. However, we still don't really transfer the error analysis computed in one domain to the other. Improving on this could allow a better balance of both error contributions to the overall filter quality, hence even better efficiency.

Short-term work also focuses on extending this approach to infinite impulse response filters.

References

[1] J. H. McClellan, T. W. Parks, and L. Rabiner, "A computer program for designing optimum FIR linear phase digital filters," *IEEE Transactions on Audio and Electroacoustics*, vol. 21, no. 6, pp. 506–526, Dec 1973.

[2] S.-I. Filip, "A robust and scalable implementation of the Parks-McClellan algorithm for designing FIR filters," *ACM Transactions on Mathematical Software*, p. to appear, 2016.

[3] N. Brisebarre, S.-I. Filip, and G. Hanrot, "A lattice basis reduction approach for the design of quantized FIR filters," *submitted*, 2016.

[4] F. De Dinechin, M. Istoan, and A. Massouri, "Sum-of-product architectures computing just right," in *IEEE 25th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, June 2014, pp. 41–47.

[5] J. Lotze, S. A. Fahmy, J. Noguera, L. Doyle, and R. Esser, "An FPGA-based cognitive radio framework," in *Irish Signals and Systems Conference (ISSC 2008)*. IET, Jun. 2008, pp. 138–143.

[6] R. Marlow, C. Dobson, and P. Athanas, "An enhanced and embedded GNU radio flow," in *Field Programmable Logic and Applications (FPL)*. IEEE, Sep. 2014.

[7] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*. Prentice Hall, 2010.

[8] E. Remes, "Sur le calcul effectif des polynomes d'approximation de Tchebichef," *Comptes rendus hebdomadaires des séances de l'Académie des Sciences*, no. 199, pp. 337–340, 1934.

[9] T. W. Parks and J. H. McClellan, "Chebyshev Approximation for Nonrecursive Digital Filters with Linear Phase," *IEEE Transactions on Circuit Theory*, vol. 19, no. 2, pp. 189–194, March 1972.

[10] J. Kaiser, "Nonrecursive digital filter design using the I_0-sinh window function," in *Proc. 1974 IEEE International Symposium on Circuits & Systems*, 1974, pp. 20–23.

[11] J. Shen and G. Strang, "The asymptotics of optimal (equiripple) filters," *IEEE Transactions on Signal Processing*, vol. 47, no. 4, pp. 1087–1098, Apr. 1999.

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

TABLE I
SYNTHESES AND ACCURACY MEASUREMENTS OF A FIR FILTER GENERATED USING MATLAB AND OUR TOOL.

I/O Format (w, f)	Coeff. Format (w, f)	Internal Format (w, f)	Resource Utilization and Performance				Implementation Accuracy (max. abs. error)
			LUTs	Reg.	Latency	Frequency (MHz)	
Using MATLAB ($N = 45$)							
(16,15)	(16,15)	(16, 15)	5504 5336	766 1415	2 8	88.39 138.38	1.9397e-3 ($2^{-9.009}$)
		(22, 21)	5658 5551	738 1691	2 8	80.62 138.79	1.1393e-3 ($2^{-9.777}$)
		(26, 25)	5693 5619	741 1889	2 8	80.53 138.27	1.1451e-3 ($2^{-9.770}$)
		full prec. (33, 31)	5180 4713	779 2048	2 8	95.32 156.80	1.1231e-3 ($2^{-9.798}$)
	(16, 15)	(16, 15)	5504 5336	766 1415	2 8	88.39 138.38	1.9397e-3 ($2^{-9.009}$)
	(22, 21)		8420 8295	760 1402	2 8	78.87 116.13	9.6687e-4 ($2^{-10.014}$)
	(26, 25)		10210 10065	751 1408	2 8	68.81 100.39	9.6988e-4 ($2^{-10.009}$)
	full prec. (31, 31)		12251 12203	759 1471	2 8	64.65 92.74	9.5673e-4 ($2^{-10.029}$)
	(16, 15)	(23, 21)	5727 5551	758 1691	2 8	77.82 138.79	1.1618e-3 ($2^{-9.749}$)
	(22, 21)		8712 8503	775 1701	2 8	72.24 118.52	2.6128e-5 ($2^{-15.224}$)
	(26, 25)		10419 10458	743 1696	2 8	64.59 104.04	1.5934e-5 ($2^{-15.937}$)
	full prec. (31, 31)		12493 12440	794 1756	2 8	60.29 92.80	1.5229e-5 ($2^{-16.002}$)
	full prec. (31, 31)	full prec. (33, 31)	13911 13826	775 2198	2 8	66.06 91.93	4.2366e-8 ($2^{-24.492}$)
	smart prec. (22, 21)	full prec. (38, 36)	7870 7341	783 2351	2 8	83.27 123.39	1.3245e-5 ($2^{-16.204}$)
	smart prec. (22, 21)	smart prec. (23, 21)	8712 8503	775 1701	2 8	72.24 118.52	2.6128e-5 ($2^{-15.224}$)
Using our tool ($N = 43$)							
(16,5)	*(22,21)*	*(23,21)*	3437 3312	630 710	1 2	75.78 209.05	1.5152e-5 ($2^{-16.010}$)

[13] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász, "Factoring polynomials with rational coefficients," *Math. Ann.*, vol. 261, pp. 515–534, 1982.

[14] K. Chapman, "Fast integer multipliers fit in FPGAs (EDN 1993 design idea winner)," *EDN magazine*, no. 10, p. 80, May 1993.

[15] N. Brunie, F. de Dinechin, M. Istoan, G. Sergent, K. Illyes, and B. Popa, "Arithmetic core generation using bit heaps," in *Field-Programmable Logic and Applications*, Sep. 2013.