# Automatic Generation of Hardware FIR Filters From a Frequency Domain Specification

Silviu-Ioan Filip*, Matei Istoan†, Florent de Dinechin† and Nicolas Brisebarre‡
*Mathematical Institute, University of Oxford, Oxford, UK
Email: filips@maths.ox.ac.uk
†Université de Lyon, INRIA, INSA-Lyon, CITI, F-69621 Villeurbanne, France
Email: matei.istoan@inria.fr, florent.de-dinechin@insa-lyon.fr
‡CNRS, LIP, ÉNS Lyon, INRIA, Univ. C. Bernard Lyon 1, Lyon, France
Email: nicolas.brisebarre@ens-lyon.fr

***Abstract*—This article presents an open-source tool for the automatic design of reliable finite impulse response (FIR) filters, targeting FPGAs. It shows that user intervention can be limited to a very small number of relevant input parameters: a high-level frequency-domain specification, and input/output formats. All the other design parameters are computed automatically, using novel approaches to filter coefficient quantization and direct-form architecture implementation. Our tool guarantees a priori that the resulting architecture respects the specification, while attempting to minimize its cost. Our approach is evaluated on a range of examples and shown to produce designs that are very competitive with the state of the art, with very little design effort.**

## I. Introduction

This article addresses the synthesis of hardware or FPGA digital FIR filters starting from a frequency-domain (FD) specification. This pervasive problem has received much attention [1]–[6], and standard tools such as MATLAB assist designers in solving it. However, it remains difficult to achieve a reliable yet efficient filter implementation, due to the number of parameters of the problem.

We introduce a completely automatic design flow (illustrated in Figure 1) that

- only requires minimal user input: a FD filter specification, and input/output fixed-point formats,
- guarantees by construction that the obtained hardware matches the FD specification, and
- attempts to optimize the various filter design parameters at a minimal cost.

This flow is implemented on top of the open-source FloPoCo arithmetic core generator[1] and is available now from http://flopoco.gforge.inria.fr/FD2FIR. It embraces the FloPoCo philosophy of last-bit accuracy [7]. It also builds upon other recent open-source efforts that share a concern for numerical quality: a robust implementation of the Parks-McClellan algorithm and FD error evaluation [8] and Euclidean lattice-based coefficient quantization [9].

We address the simplest form of digital filters: direct form finite impulse response (FIR) filters. Many other techniques exist, from cascaded FIR filters to infinite impulse response (IIR) filters in a variety of structures. Direct-form FIR filters are an important basic block on which more complex filters can be built. The choice of direct form is also motivated by architectural considerations in the sequel. IIR filters are currently out of scope and considered for future work.

[1] http://flopoco.gforge.inria.fr/

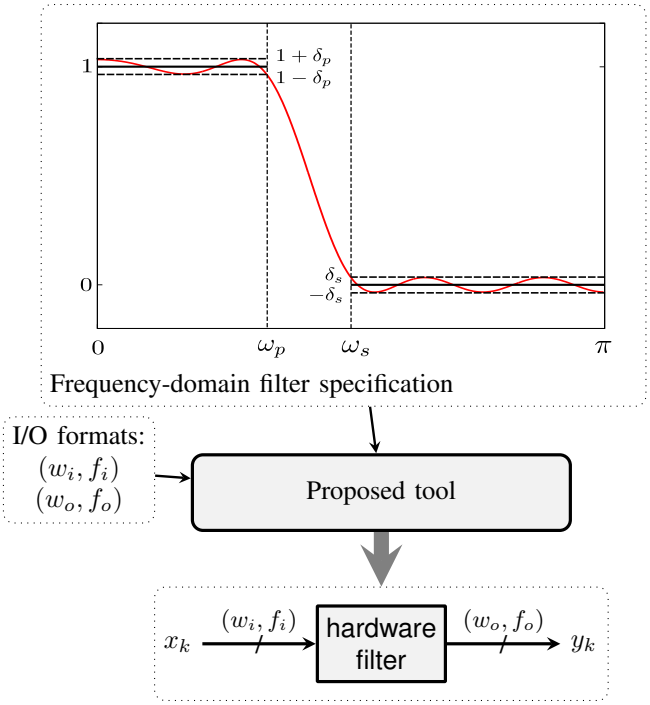


Fig. 1: Generating a hardware digital filter out of a minimal specification (here on a generic low-pass filter defined by passband frequency $\omega_p$, stopband frequency $\omega_s$, and bounds $\delta_p$ and $\delta_s$ on the passband ripple and stopband attenuation).

### A. Digital FIR filters: from specification to implementation

A FIR filter implementation computes sums of products in the time domain (TD):

$$y_k = \sum_{i=0}^{N-1} h_i x_{k-i},$$

where $x_i$ and $y_i$ are respectively the input and output signal values, given in some machine-representable format.

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.

In fact, (2) is an ideal description of the computation; due to internal rounding the value returned by the implementation will usually be some $\widetilde{y}_k$ (Figure 4).

### B. Implementation parameter space

Each of these three steps involves several parameters. Functional, frequency-domain parameters and constraints (e.g. bands of frequency response) control the first step. The second and third steps depend on 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 (e.g. inside the dotted box of Figure 4). The filter order $N$ is a parameter shared by all the steps.

The main challenge for a filter designer is to ensure that certain error bounds are met in both the frequency and time domains, all while also trying to optimize for performance and resource consumption.

In order to do this, current mainstream design flows tend to expose all the aforementioned parameters. For instance, Figure 2 shows MATLAB's filter design interface, which goes from initial specification in the frequency and time domains to a synthesizable (VHDL or Verilog) hardware description.

### C. State of the art and positionning

If we focus on solutions that are able to generate hardware FIR filters starting from their FD specification, the most used tool is very probably MATLAB (`fdatool` or `filterbuilder`). It imposes a complex interface (Figure 2), if one wants to minimize the hardware cost while ensuring that the FD specification holds. Besides, to our knowledge, the quality of obtained architectures must be evaluated by simulation.

When targetting FPGAs, the mainstream solution is to use vendor tools (from Xilinx, Altera or Actel). They provide FIR IP cores that mainly address step 3, relying on Matlab/Simulink

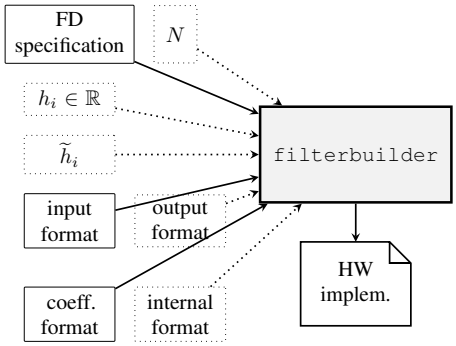


Fig. 2: MATLAB design flow with all the parameters. The dashed ones can be computed by the tool or input by the user. The external `minimizecoeffwl` function can also help compute the coefficient formats.

for coefficient generation and simulation [4]. This justifies that we compare only to MATLAB in Section III.

Open-source alternatives like GNURadio[2] or the Scipy `signal` package[3] essentially address step 1. It is hoped that the present work can establish a missing link to FPGA back-end work [10], [11].

There have been numerous academic works targetting FPGAs or ASICs. Unfortunately, none of these efforts (to our knowledge) describes an open-source tool. Among these, [1], [2], [3] address the three steps, but without error analysis and no quality guarantee: the first two steps are validated iteratively by simulation. The same holds for some recent work [6] whose main originality is to perform the computations in the Residue Number System.

Conversely, [5] is based on an error analysis that optimizes the formats subject to a prescribed output noise power, but it only addresses steps 2 and 3.

Therefore, the proposed approach is novel in two respects.

Firstly, all these previous approaches rely on approximate double-precision computations and/or simulation to evaluate the quality of steps 1 and 2. Conversely, in the present work, steps 1 and 2 use the implementation from [8], [9], which, in addition to computing extremely accurate quantized filters, can be used to report a guaranteed bound on the corresponding frequency domain error.

Secondly, all these previous approaches (including MATLAB) produce multiplierless FIR architectures based on shift-and-add constant multipliers. Conversely, the present work uses the fused SOPC architecture of [7], where constant multipliers use an optimization of the KCM algorithm [12], and are merged into a single bit array [13]. In addition, the SOPC generator of [7] also takes care of the computation of optimal internal format respecting an output accuracy specification.

In both cases, our approach uses a-priori bounds, which allows us to design filters without needing to resort to simulation to evaluate their quality. This is both faster and more reliable.

[2] http://gnuradio.org
[3] http://docs.scipy.org/doc/scipy/reference/signal.html

### D. Outline of the paper

The details of the proposed methodology are given in Section II. Section III shows, on a range of examples, that it produces highly efficient architecture realizations compared to the state of the art. Section IV concludes and discusses research directions suggested by the present work.

## II. Integrated hardware FIR filter design

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) [14, Sec. 5.7]. We follow this convention here.

This section first discusses the specifications of frequency and time domain errors, and how they can be combined into an integrated design flow. It then outlines each step of the proposed approach.

### A. FD and TD error constraints

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

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

The overall TD error of an architecture is $\varepsilon_k = |\widetilde{y}_k - y_k|$, where $\widetilde{y}_k$ is the computed output and $y_k$ is defined by (2). The accuracy of the architecture will therefore be defined as a bound $\overline{\varepsilon}$ on $\varepsilon_k$. There is a deep relationship between $\overline{\varepsilon}$ and the output format: it is not possible to output more accuracy than the format can hold, and it makes little sense to output less accuracy either, especially in a hardware context where we pay for every bit. Therefore, following [7], the accuracy of the architecture is specified by the output format: if the least significant bit (LSB) of the output has weight $2^{-p}$, then the architecture must be such that $\forall k, |\widetilde{y}_k - y_k| < \overline{\varepsilon} = 2^{-p}$. Conversely, if an architecture is accurate to $\overline{\varepsilon} = 2^{-p}$, then its output format will have LSB $2^{-p}$.

This bound on the output accuracy means that a limit on the precision of the filter coefficients is also necessary, which will affect the quality of $H$ in the FD.

### B. Overview of the tool flow

Figure 3 shows the various steps of the algorithm that ensures that the implemented filter satisfies the given constraints (FD behavior and I/O accuracy), with minimal user intervention.

The first step (detailed in Section II-C) evaluates the minimal $N$ that allows for a real-coefficient filter respecting the FD domain specification.

The next step, detailed in Section II-D, determines the smallest internal coefficient formats that enable to match the time-domain error specification.

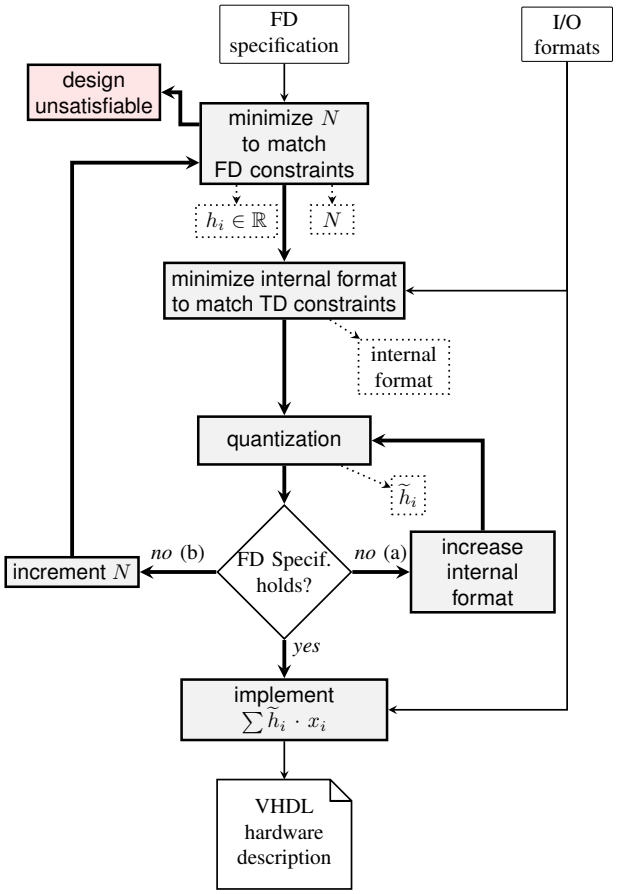


Fig. 3: A simplified view of our design flow

This provides enough information to round the initial real coefficients to this internal format. This quantization step is detailed in Section II-E.

However, the internal format decision was taken out of TD considerations: it is therefore possible that the filter with quantized coefficients no longer match the FD specification. In this case the algorithm re-evaluates previous choices in two ways:

(a) increase the coefficient and internal formats until the FD constraints are satisfied;
(b) increment $N$ and restart the design process.

One of these choices will lead to the smallest architecture. Currently, we attempt both choices and pick the one which seems to be the most resource efficient. Experiments in Section III will illustrate this trade-off.

Finally, once the quantized coefficients have been determined, all the parameters of the architecture are known, and we proceed to generate synthesizable VHDL code (see Section II-F).

Let us now detail each of these steps.

### C. Minimizing $N$ to match FD constraints

Finding the optimal set of real-valued taps $\{h_k\}_{k=0}^{N-1}$ which minimize $\delta$ in (3) is a classic approximation theory problem. The usual approach to solve it is to use the Remez exchange algorithm [15], [16]. It is a very robust approach. In our tool,

we use the `firpm` implementation from [8], which is capable of outputting extremely accurate filters.

As an example, consider the design of a prototypical lowpass system (Figure 1) defined on $\Omega = [0, \omega_p] \cup [\omega_s, \pi]$, with

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

Here, the objective is to minimize $N$, while the approximation error $|D - H|$ is upper bounded by $\delta_p$ on the passband and by $\delta_s$ 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 $\delta \leqslant \delta_s$. We solve this problem using the strategy presented in [17, Sec. 15.7–15.9]. The resulting $H$ is shown in red in Figure 1.

### D. Minimizing the internal format to match TD constraints

Once $N$ is known, a worst-case rounding error analysis [7, Sec.III] can determine the minimal internal format that will ensure that the TD error bound is respected in the worst case. Essentially, we know it is possible to compute each product $h_i x_i$ with arbitrary accuracy, say with a rounding error smaller than $\bar{\varepsilon}$ – this will of course require the LSB of the internal format to be smaller than $\log_2(\bar{\varepsilon})$. The summation in this fixed-point internal format will entail no rounding error. In this case, the accumulation of all these rounding errors in the sum will be bounded by $N\bar{\varepsilon}$. The rounding of the internal format to the output precision $2^{-p}$ will entail another error bounded by $2^{-p-1}$. The TD error constraint for last-bit accuracy to this output format is therefore

$$N\bar{\varepsilon} + 2^{-p-1} \;<\; 2^{-p} \quad . \tag{4}$$

This constraint can always be satisfied by choosing a large enough internal precision, but the value of $\bar{\varepsilon}$ depends on the multiplier technique used: see for instance [7] for the technique used in this work.

Once the internal format is chosen, we may quantize the coefficients to this internal format to check that the FD constraints are still satisfied.

### E. A robust quantization scheme

To address the coefficient quantization issue, we use the recent approach described in [9], based on the LLL algorithm [18]. 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 [9, Sec. 5].

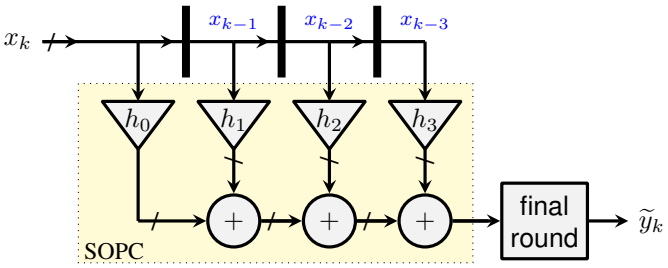


Fig. 4: Direct-form abstract filter architecture

### F. Sum of product generation

The last step is based on the automation of sum-of-product-by-constant (SOPC) present in FloPoCo [7] (a schematic view is given in Figure 4). This work introduced two architectural innovations: the optimization of the KCM constant multiplication algorithm [12] to fixed-point format, and the use of a bit heap [13] to fuse the summations inside the KCMs into the summation of (2).

## III. Examples and Results

In this section, we compare our proposed tool with MATLAB's mainstream design flow with `filterbuilder` (we use version R2014B).

Unless otherwise stated, FPGA syntheses were performed with Xilinx ISE 14.7, using the Virtex6 xc6vcx75t-2ff484 FPGA as a target device, with design goals set to balanced. FPGA resource consumption is measured in look-up tables (LUT) and registers (Reg.), and performance is measured as maximum operating frequency at a given latency. Results are after place and route.

### A. A detailed high-pass example

As a first working example, let us use the default filter specification that is proposed by `filterbuilder`. 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 consist of 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.

*1) MATLAB's overwhelming choice:* The first two parts of Table I are obtained using MATLAB's `filterbuilder`. They show how performance and resource utilization depend on the coefficient and internal formats provided by the user.

The following part is obtained using MATLAB's `minimizecoeffwl` routine, which tries to minimize the coefficient wordlengths, while still respecting the initial FD constraints. We had to leave the graphical interface for that.

All these trials illustrate the complexity incurred by the vast design parameter space and shows that both formats should be

TABLE I: Virtex6 synthesis results for generated high-pass FIR filters.

| N | coeff. (w, f) | internal (w, f) | cost LUT + Reg. | performance cycles @ freq. |
|---|---|---|---|---|
| **Using MATLAB's graphical interface defaults** | | | | |
| 45 | (16, 16) | (32, 31) | 5463L + 754R | 2 @ 91 MHz |
| **Using MATLAB with naive trial and error** ($N = 45$) | | | | |
| 45 | (16,15) | (16, 15)† | 5504L + 766R<br>5336L + 1415R | 2 @ 88 MHz<br>8 @ 138 MHz |
| | (16,15) | (33, 31) | 5180L + 779R<br>4713L + 2048R | 2 @ 95 MHz<br>8 @ 157 MHz |
| | (31, 31) | (16, 15)† | 12251L + 759R<br>12203L + 1471R | 2 @ 65 MHz<br>8 @ 93 MHz |
| **Using MATLAB with `minimizecoeffwl`** | | | | |
| 49 | (10, 10) | (26, 25) | 3087L + 811R | 2 @ 104 MHz |
| 45 | (12, 12) | (28, 27) | 4169L + 757R | 2 @ 98 MHz |
| **Using the proposed tool** | | | | |
| 43 | (22, 21) | (23,21) | 3449L + 717R | 2 @ 207 MHz |

†: Error analysis of this architecture shows that it is accurate to $2^{-10}$.

optimized jointly. Such an exploration of the parameter space takes time. Besides, it is non trivial to obtain from MATLAB the implementation accuracy data (or an equivalent signal/noise measure).

*2) Running the proposed algorithm:* The last line of Table I presents data about the filter generated using our approach in less than one minute.

On this example, the minimal $N$ satisfying the FD specification (when the filter has real coefficients) is found at $N = 43$. For `filterbuilder` it was $N = 45$.

Then the tool finds that the internal format requires 21 fractional bits. A filter with coefficients quantized to this format is then found: its passband ripple is smaller than 0.9 dB and its stopband attenuation is larger than 61 dB, matching the FD specification.

The generated filter is comparable in resource and performance to filters generated using `minimizecoeffwl`, and much better than those obtained using MATLAB's graphical interface.

The better frequency of our architecture essentially comes from a better implementation of the summation (using a bit heap in our approach, where MATLAB generates rows of additions).

Still, the comparison of resource and performance shows where there is room for improvement in our approach. An expert user of MATLAB is able to take advantage of smaller quantizations of the coefficients that eventually lead to smaller multipliers. This suggests that the flow depicted on Figure 3 could be improved to look for solutions in this direction (the smaller quantizations can be obtained using [9]). This will be discussed in more details in the conclusion.

### B. A low-pass example

There exists a trade-off between minimizing $N$ and minimizing the coefficient size, already illustrated by the two lines of Table I for MATLAB with the `minimizecoeffwl` approach. In our tool, it is managed by the two arrows labeled *no* (a) and *no* (b) from Figure 3. As an artificial example that illustrates this trade-off, let us consider a low pass specification with ideal response

$$D(\omega) = \begin{cases} 1, & \omega \in [0, 0.1\pi] \quad \text{(pass band)}, \\ 0, & \omega \in [0.9\pi, \pi] \quad \text{(stop band)}. \end{cases}$$

The maximal pass band ripple is 0.5 dB ($\delta_p \leqslant 0.0287$) and the stop band attenuation limit is 120 dB ($\delta_s \leqslant 10^{-6}$). The input format is fixed to $(w_i, f_i) = (13, 12)$ and the output to $f_o = 12$ fractional bits. We obtain a minimal type I filter with $N = 9$ unquantized coefficients, pass band ripple $0.46$ dB and stop band attenuation of $120.671$ dB. These values are very close to the tolerances, having the effect that by following (a) in Figure 3, we need to quantize the coefficients with a signed format $(20, 19)$ in order to assure that the tolerances still hold. On the other hand, if we follow (b) and construct a type II filter with $N = 10$, we will obtain a real-valued coefficient filter with a pass band ripple of $0.081$ dB and stop band attenuation of $135.846$ dB. These values are much more flexible for performing quantization. Indeed, optimizing the coefficients using a type $(17, 16)$ format suffices. However, when synthesizing both filters with our tool, the $N = 9$ filter is more resource efficient (see Table II). Again, for comparison, the result using `minimizecoeffwl` and `filterbuilder` is also shown, here with a clear advantage to our tool.

TABLE II: Virtex6 synthesis results for the lowpass example.

| N | coeff. (w, f) | internal (w, f) | cost LUT + Reg. | performance cycles @ freq. |
|---|---|---|---|---|
| **Using MATLAB with `minimizecoeffwl`** | | | | |
| 10 | (15,15) | (29,27) | 860L + 490R | 6 @ 143 MHz |
| **Using the proposed tool** | | | | |
| 9 | (20,19) | (21,19) | 464L + 131R | 2 @ 193 MHz |
| 10 | (17,16) | (18,16) | 570L + 139R | 2 @ 179 MHz |

### C. A band-stop example

The last example is a three-band filter, for which MATLAB's `minimizecoeffwl` tool does not work. It is a band stop filter with passbands $[0, 0.4\pi]$ and $[0.85\pi, \pi]$, and stopband $[0.5\pi, 0.75\pi]$. The maximum passband ripple is set to 0.1dB ($\delta_p \leqslant 0.575$) and the stopband attenuation to 60dB ($\delta_s \leqslant 10^{-3}$). The input format is $(w_i, f_i) = (16, 15)$ and the output has $f_o = 15$ fractional bits.

Table III compares the results obtained by our tool with those obtained with MATLAB's `filterbuilder`. Without `minimizecoeffwl`, we used naive rounding of the coefficients to the smallest format that allows to satisfy the FD specification: $(16, 15)$ (found by trial and error), and the internal format chosen by MATLAB. The filter designed by our tool has better performance and resource consumption.

## IV. Conclusion and Future Work

This paper introduces a tool that automates the design of fixed-point FIR filters for FPGAs. A practical size filter implementation of guaranteed quality is obtained within seconds. Apart from requiring minimal user intervention, our

TABLE III: Virtex6 synthesis results for the band-stop example.

| N | coeff. (w, f) | internal (w, f) | cost LUT + Reg. | performance cycles @ freq. |
|---|---|---|---|---|
| **Using MATLAB** | | | | |
| 59 | (16,15) | (33,30) | 5117L + 2847R | 8 @ 127 MHz |
| **Using the proposed tool** | | | | |
| 59 | (21,20) | (23,20) | 3911L + 1020R | 2 @ 147 MHz |

tool produces results which are at least as resource-efficient as what is obtainable using similar mainstream tools, all while being faster.

This result is achieved thanks to strict error evaluation techniques to ensure the numerical quality of the results, coupled with pervasive concern to compute just right in the resulting architecture.

An open question remains: expressing and exploiting the relationship between quality specification in the frequency domain and quality specification in the time domain. In both domains, we have error analysis procedures that are well understood and well implemented. We could design efficient heuristics that ensure that the design meets its 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.

Another weakness of the current approach is that it doesn't really attempt to minimize the coefficient sizes, as MATLAB's `minimizecoeffwl` does. This is due to the choice of using internal computation format as the coefficient quantization format. Considering that MATLAB is able to determine filters with much smaller coefficients (e.g. 10 bits in Table I), we should disconnect the coefficient format and the internal format. The smallest coefficient format could be found by a dichotomy loop around the quantization step. This will reduce the output size of the KCM tables, hence resource consumption. It will also reduce the bitheap size, hence the circuit critical path.

Another research direction is to expose more design freedom to expert users. We want to do so 1/ without jeopardizing reliability, and 2/ while keeping the interface simple. For instance, it seems important to provide a knob that defines the weight between arrows (a) and (b) in Figure 3.

Short-term work also focuses on extending this approach to infinite impulse response filters, and to more filter structures.

## References


[1] P. T. Yang, R. Jain, T. Yoshino, W. Gass, and A. Shah, "A functional silicon compiler for high speed FIR digital filters," in *Acoustics, Speech, and Signal Processing, International Conference on (ICASSP)*, Apr 1990, pp. 1329–1332.

[2] M. Ishikawa, M. Edahiro, T. Yoshimura, T. Miyazaki, S. I. Aikoh, T. Nishitani, K. Mitsuhashi, and M. Furuichi, "Automatic layout synthesis for FIR filters using a silicon compiler," in *Circuits and Systems, IEEE International Symposium on*, May 1990, pp. 2588–2591 vol.4.

[3] R. Jain, P. T. Yang, and T. Yoshino, "FIRGEN: a computer-aided design system for high performance FIR filter integrated circuits," *IEEE Transactions on Signal Processing*, vol. 39, no. 7, pp. 1655–1668, Jul 1991.

[4] J. Hwang and J. Ballagh, "Building custom FIR filters using system generator," in *Proc. International Conference on Field-Programmable Logic and Applications (FPL)*, 2002, pp. 1101–1104.

[5] S. C. Chan, K. M. Tsui, and S. H. Zhao, "A methodology for automatic hardware synthesis of multiplier-less digital filters with prescribed output accuracy," in *IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, Dec 2006, pp. 61–64.

[6] R. Smyk, "FIReWORK: FIR filters hardware structures auto-generator," *Journal of Applied Computer Science*, vol. 21, no. 1, pp. 135–149, 2013.

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

[8] 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, Aug. 2016. [Online]. Available: https://hal.inria.fr/hal-01136005

[9] N. Brisebarre, S.-I. Filip, and G. Hanrot, "A lattice basis reduction approach for the design of finite wordlength FIR filters," *submitted*, 2016. [Online]. Available: https://hal.inria.fr/hal-01308801

[10] 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)*. IET, Jun. 2008, pp. 138–143.

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

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

[13] 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.

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

[15] 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.

[16] 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.

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

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