# Filter optimization for real time digital processing of radiofrequency signals: application to oscillator metrology

A. Hugeat*†, J. Bernard†, G. Goavec-Mérou*, P.-Y. Bourgeois*, J.-M. Friedt*
*FEMTO-ST, Time & Frequency department, Besançon, France
†FEMTO-ST, Computer Science department DISC, Besançon, France
Email: {pyb2,jmfriedt}@femto-st.fr

***Abstract*—Software Defined Radio (SDR) provides stability, flexibility and reconfigurability to radiofrequency signal processing. Applied to oscillator characterization in the context of ultrastable clocks, stringent filtering requirements are defined by spurious signal or noise rejection needs. Since real time radiofrequency processing must be performed in a Field Programmable Array to meet timing constraints, we investigate optimization strategies to design filters meeting rejection characteristics while limiting the hardware resources required and keeping timing constraints within the targeted measurement bandwidths. The presented technique is applicable to scheduling any sequence of processing blocks characterized by a throughput, resource occupation and performance tabulated as a function of configuration characateristics, as is the case for filters with their coefficients and resolution yielding rejection and number of multipliers.**

***Index Terms*—Software Defined Radio, Mixed-Integer Linear Programming, Finite Impulse Response filter**

## I. Digital signal processing of ultrastable clock signals

Analog oscillator phase noise characteristics are classically performed by downconverting the radiofrequency signal using a saturated mixer to bring the radiofrequency signal to baseband, followed by a Fourier analysis of the beat signal to analyze phase fluctuations close to carrier. In a fully digital approach, the radiofrequency signal is digitized and numerically downconverted by multiplying the samples with a local numerically controlled oscillator (Fig. 1) [1].

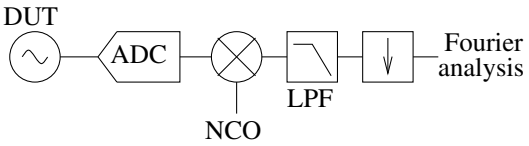



Fig. 1: Fully digital oscillator phase noise characterization: the Device Under Test (DUT) signal is sampled by the radiofrequency grade Analog to Digital Converter (ADC) and downconverted by mixing with a Numerically Controlled Oscillator (NCO). Unwanted signals and noise aliases are rejected by a Low Pass Filter (LPF) implemented as a cascade of Finite Impulse Response (FIR) filters. The signal is then decimated before a Fourier analysis displays the spectral characteristics of the phase fluctuations.

As with the analog mixer, the non-linear behavior of the downconverter introduces noise or spurious signal aliasing as well as the generation of the frequency sum signal in addition to the frequency difference. These unwanted spectral characteristics must be rejected before decimating the data stream for the phase noise spectral characterization [2]. The characteristics introduced between the downconverter and the decimation processing blocks are core characteristics of an oscillator characterization system, and must reject out-of-band signals below the targeted phase noise – typically in the sub -170 dBc/Hz for ultrastable oscillator we aim at characterizing. The filter blocks will use most resources of the Field Programmable Gate Array (FPGA) used to process the radiofrequency datastream: optimizing the performance of the filter while reducing the needed resources is hence tackled in a systematic approach using optimization techniques. Most significantly, we tackle the issue by attempting to cascade multiple Finite Impulse Response (FIR) filters with tunable number of coefficients and tunable number of bits representing the coefficients and the data being processed.

## II. Finite impulse response filter

We select FIR filters for their unconditional stability and ease of design. A FIR filter is defined by a set of weights $b_k$ applied to the inputs $x_k$ through a convolution to generate the outputs $y_k$

$$y_n = \sum_{k=0}^{N} b_k x_{n-k} \tag{1}$$

As opposed to an implementation on a general purpose processor in which word size is defined by the processor architecture, implementing such a filter on an FPGA offers more degrees of freedom since not only the coefficient values and number of taps must be defined, but also the number of bits defining the coefficients and the sample size. For this reason, and because we consider pipeline processing (as opposed to First-In, First-Out FIFO memory batch processing) of radiofrequency signals, High Level Synthesis (HLS) languages [3] are not considered but the problem is tackled at the Very-high-speed-integrated-circuit Hardware Description Language (VHDL) level. Since latency is not an issue in a openloop phase noise characterization instrument, the large numbre of taps in the FIR, as opposed to the shorter Infinite Impulse Response (IIR) filter, is not considered as an issue as would be in a closed loop system.

The coefficients are classically expressed as floating point values. However, this binary number representation is not efficient for fast arithmetic computation by an FPGA. Instead, we select to quantify these floating point values into integer values. This quantization will result in some precision loss.

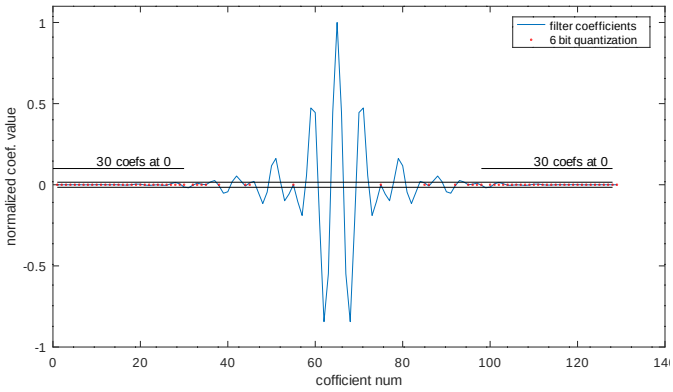



Fig. 2: Impact of the quantization resolution of the coefficients: the quantization is set to 6 bits – with the horizontal black lines indicating $\pm 1$ least significant bit – setting the 30 first and 30 last coefficients out of the initial 128 band-pass filter coefficients to 0 (red dots).

The tradeoff between quantization resolution and number of coefficients when considering integer operations is not trivial. As an illustration of the issue related to the relation between number of fiter taps and quantization, Fig. 2 exhibits a 128-coefficient FIR bandpass filter designed using floating point numbers (blue). Upon quantization on 6 bit integers, 60 of the 128 coefficients in the beginning and end of the taps become null, making the large number of coefficients irrelevant: processing resources are hence saved by shrinking the filter length. This tradeoff aimed at minimizing resources to reach a given rejection level, or maximizing out of band rejection for a given computational resource, will drive the investigation on cascading filters designed with varying tap resolution and tap length, as will be shown in the next section. Indeed, our development strategy closely follows the skeleton approach [4], [5], [6] in which basic blocks are defined and characterized before being assembled [7] in a complete processing chain. In our case, assembling the filter blocks is a simpler block combination process since we assume a single value to be processed and a single value to be generated at each clock cycle. The FIR filters will not be considered to decimate in the current implementation: the decimation is assumed to be located after the FIR cascade at the moment.

## III. Methodology description

Our objective is to develop a new methodology applicable to any Digital Signal Processing (DSP) chain obtained by assembling basic processing blocks, with hardware and manufacturer independence. Achieving such a target requires defining an abstract model to represent some basic properties of DSP blocks such as performance (i.e. rejection or ripples in the bandpass for filters) and resource occupation. These abstract properties, not necessarily related to the detailed hardware implementation of a given platform, will feed a scheduler solver aimed at assembling the optimum target, whether in terms of maximizing performance for a given arbitrary resource occupation, or minimizing resource occupation for a given performance. In our approach, the solution of the solver is then synthesized using the dedicated tool provided by each platform manufacturer to assess the validity of our abstract resource occupation indicator, and the result of running the DSP chain on the FPGA allows for assessing the performance

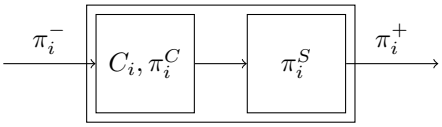



Fig. 3: A single filter is composed of a FIR (on the left) and a Shifter (on the right)

of the scheduler. We emphasize that all solutions found by the solver are synthesized and executed on hardware at the end of the analysis.

In this demonstration, we focus on only two operations: filtering and shifting the number of bits needed to represent the data along the processing chain. We have chosen these basic operations because shifting and the filtering have already been studied in the literature [8], [9], [10], [11] providing a framework for assessing our results. Furthermore, filtering is a core step in any radiofrequency frontend requiring pipelined processing at full bandwidth for the earliest steps, including for time and frequency transfer or characterization [12], [13], [1].

Addressing only two operations allows for demonstrating the methodology but should not be considered as a limitation of the framework which can be extended to assembling any number of skeleton blocks as long as performance and resource occupation can be determined. Hence, in this paper we will apply our methodology on simple DSP chains: a white noise input signal is generated using a Pseudo-Random Number (PRN) generator or by sampling a wideband (125 MS/s) 14-bit Analog to Digital Converter (ADC) loaded by a 50 $\Omega$ resistor. Once samples have been digitized at a rate of 125 MS/s, filtering is applied to qualify the processing block performance – practically meeting the radiofrequency frontend requirement of noise and bandwidth reduction by filtering and decimating. Finally, bursts of filtered samples are stored for post-processing, allowing to assess either filter rejection for a given resource usage, or validating the rejection when implementing a solution minimizing resource occupation.

The first step of our approach is to model the DSP chain. Since we aim at only optimizing the filtering part of the signal processing chain, we have not included the PRN generator or the ADC in the model: the input data size and rate are considered fixed and defined by the hardware. The filtering can be done in two ways, either by considering a single monolithic FIR filter requiring many coefficients to reach the targeted noise rejection ratio, or by cascading multiple FIR filters, each with fewer coefficients than found in the monolithic filter.

After each filter we leave the possibility of shifting the filtered data to consume less resources. Hence in the case of cascaded filter, we define a stage as a filter and a shifter (the shift could be omitted if we do not need to divide the filtered data).

### A. Model of a FIR filter

A cascade of filters is composed of $n$ FIR stages. In stage $i$ ($1 \leq i \leq n$) the FIR has $C_i$ coefficients and each coefficient is an integer value with $\pi_i^C$ bits while the filtered data are shifted by $\pi_i^S$ bits. We define also $\pi_i^-$ as the size of input data and $\pi_i^+$ as the size of output data. The figure 3 shows a filtering stage.

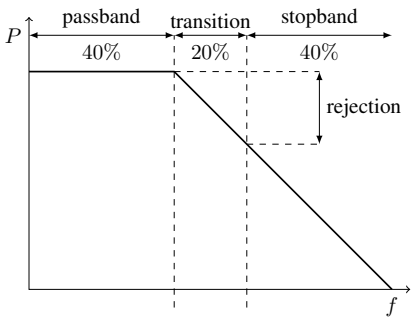



Fig. 4: Shape of the filter transmitted power $P$ as a function of frequency $f$: the passband is considered to occupy the initial 40% of the Nyquist frequency range, the stopband the last 40%, allowing 20% transition width.

FIR $i$ has been characterized through numerical simulation as able to reject $F(C_i, \pi_i^C)$ dB. This rejection has been computed using GNU Octave software FIR coefficient design functions (`firls` and `fir1`). For each configuration $(C_i, \pi_i^C)$, we first create a FIR with floating point coefficients and a given $C_i$ number of coefficients. Then, the floating point coefficients are discretized into integers. In order to ensure that the coefficients are coded on $\pi_i^C$ bits effectively, the coefficients are normalized by their absolute maximum before being scaled to integer coefficients. At least one coefficient is coded on $\pi_i^C$ bits, and in practice only $b_{C_i/2}$ is coded on $\pi_i^C$ bits while the others are coded on much fewer bits.

With these coefficients, the `freqz` function is used to estimate the magnitude of the filter transfer function. Comparing the performance between FIRs requires however defining a unique criterion. As shown in figure 4, the FIR magnitude exhibits two parts: we focus here on the transitions width and the rejection rather than on the bandpass ripples as emphasized in [9], [8]. Throughout this demonstration, we arbitrarily set a bandpass of 40% of the Nyquist frequency and a bandstop from 60% of the Nyquist frequency to the end of the band, as would be typically selected to prevent aliasing before decimating the dataflow by 2. The method is however generalized to any filter shape as long as it is defined from the initial modeling steps: Fig. 6 as described below is indeed unique for each filter shape.

In the transition band, the behavior of the filter is left free, we only define the passband and the stopband characteristics. Initial considered criteria include the mean value of the stopband rejection which yields unacceptable results since notches overestimate the rejection capability of the filter. An intermediate criterion considered the maximal rejection within the stopband, to which the sum of the absolute values within the passband is subtracted to avoid filters with excessive ripples, normalized to the bin width to remain consistent with the passband criterion (dBc/Hz units in all cases). In this case, cascading too many filters with individual excessive ($>$ 1 dB) passband ripples led to unacceptable ($>$ 10 dB) final ripple levels, especially close to the transition band. Hence, the final criterion considers the minimal rejection in the stopband to which the the maximal amplitude in the passband (maximum value minus the minimum value) is substracted, with a 1 dB threshold on the latter quantity over which the filter is discarded. With this criterion, we meet the expected

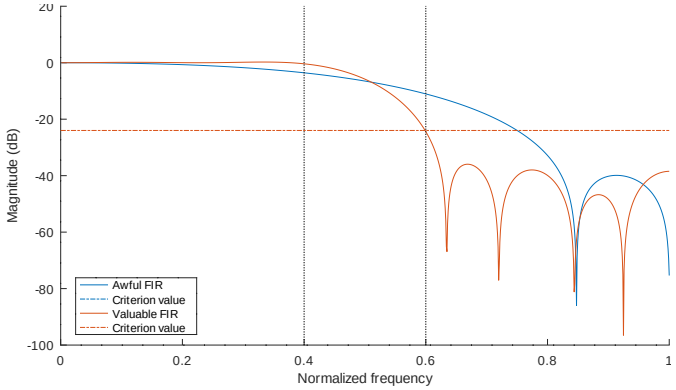



Fig. 5: Selected filter qualification criterion computed as the maximum rejection in the stopband minus the maximal ripple amplitude in the passband with a $>$ 1 dB threshold above which the filter is discarded: comparison between monolithic filter (blue, rejected in this case) and cascaded filters (red).

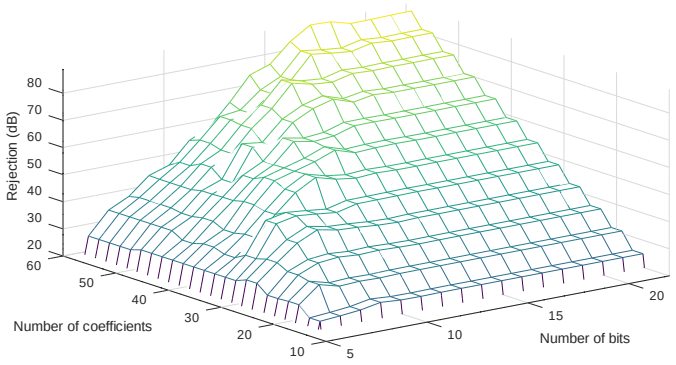



Fig. 6: Filter rejection as a function of number of coefficients and number of bits : this lookup table will be used to identify which filter parameters – number of bits representing coefficients and number of coefficients – best match the targeted transfer function. Filters with fewer than 10 taps or with coefficients coded on fewer than 5 bits are discarded due to excessive ripples in the passband.

rejection capability of low pass filters as shown in figure 5. The best filter has a correct rejection estimation and the worst filter is discarded based on the excessive passband ripple criterion.

Thanks to the latter criterion which will be used in the remainder of this paper, we are able to automatically generate multiple FIR taps and estimate their rejection. Figure 6 exhibits the rejection as a function of the number of coefficients and the number of bits representing these coefficients. The curve shaped as a pyramid exhibits optimum configurations sets at the vertex where both edges meet. Indeed for a given number of coefficients, increasing the number of bits over the edge will not improve the rejection. Conversely when setting the a given number of bits, increasing the number of coefficients will not improve the rejection. Hence the best coefficient set are on the vertex of the pyramid. Notice that the word length and number of coefficients do not start at 1: filters with too few coefficients or too little tap word size are rejected by the excessive ripple constraint of the criterion. Hence, the size of the pyramid is significantly reduced by discarding these filters and so is the solution search space.

Although we have an efficient criterion to estimate the rejection of one set of coefficients (taps), we have a problem

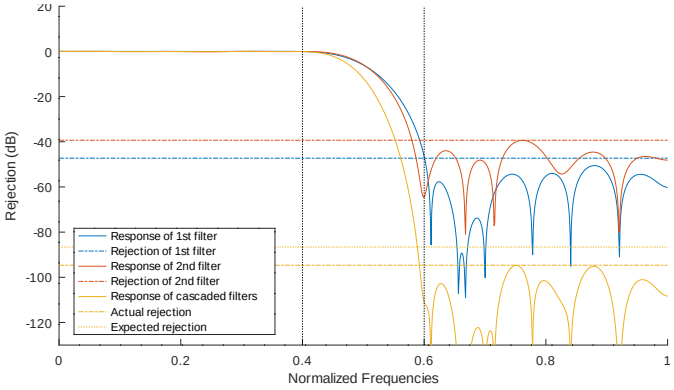



Fig. 7: Transfer function of individual filters and after cascading the two filters, demonstrating that the selected criterion of maximum rejection in the bandstop (horizontal lines) is met. Notice that the cascaded filter has better rejection than summing the bandstop maximum of each individual filter.

when we cascade filters and estimate the criterion as a sum two or more individual criteria. If the FIR filter coefficients are the same between the stages, we have:

$$F_{total} = F_1 + F_2$$

But selecting two different sets of coefficient will yield a more complex situation in which the previous relation is no longer valid as illustrated on figure 7. The red and blue curves are two different filters with maximums and notches not located at the same frequency offsets. Hence when summing the transfer functions, the resulting rejection shown as the dashed yellow line is improved with respect to a basic sum of the rejection criteria shown as a the dotted yellow line. Thus, estimating the rejection of filter cascades is more complex than taking the sum of all the rejection criteria of each filter. However since the individual filter rejection sum underestimates the rejection capability of the cascade, this upper bound is considered as a conservative and acceptable criterion for deciding on the suitability of the filter cascade to meet design criteria.

Finally in our case, we consider that the input signal are fully known. The resolution of the input data stream are fixed and still the same for all experiments in this paper.

Based on this analysis, we address the estimate of resource consumption (called silicon area – in the case of FPGAs this means processing cells) as a function of filter characteristics. As a reminder, we do not aim at matching actual hardware configuration but consider an arbitrary silicon area occupied by each processing function, and will assess after synthesis the adequation of this arbitrary unit with actual hardware resources provided by FPGA manufacturers. The sum of individual processing unit areas is constrained by a total silicon area representative of FPGA global resources. Formally, variable $a_i$ is the area taken by filter $i$ (in arbitrary unit). Variable $r_i$ is the rejection of filter $i$ (in dB). Constant $\mathcal{A}$ is the total available area. We model our problem as follows:

$$\text{Maximize} \sum_{i=1}^{n} r_i$$

$$\sum_{i=1}^{n} a_i \leq \mathcal{A} \tag{2}$$

$$a_i = C_i \times (\pi_i^C + \pi_i^-), \qquad \forall i \in [1,n] \tag{3}$$

$$r_i = F(C_i, \pi_i^C), \qquad \forall i \in [1,n] \tag{4}$$

$$\pi_i^+ = \pi_i^- + \pi_i^C - \pi_i^S, \qquad \forall i \in [1,n] \tag{5}$$

$$\pi_{i-1}^+ = \pi_i^-, \qquad \forall i \in [2,n] \tag{6}$$

$$\pi_i^+ \geq 1 + \sum_{k=1}^{i} \left(1 + \frac{r_j}{6}\right), \qquad \forall i \in [1,n] \tag{7}$$

$$\pi_1^- = \Pi^I \tag{8}$$

Equation 2 states that the total area taken by the filters must be less than the available area. Equation 3 gives the definition of the area used by a filter, considered as the area of the FIR since the Shifter is assumed not to require significant resources. We consider that the FIR needs $C_i$ registers of size $\pi_i^C + \pi_i^-$ bits to store the results of the multiplications of the input data with the coefficients. Equation 4 gives the definition of the rejection of the filter thanks to the tabulated function $F$ that we defined previously. The Shifter does not introduce negative rejection as we will explain later, so the rejection only comes from the FIR. Equation 5 states the relation between $\pi_i^+$ and $\pi_i^-$. The multiplications in the FIR add $\pi_i^C$ bits as most coefficients are close to zero, and the Shifter removes $\pi_i^S$ bits. Equation 6 states that the output number of bits of a filter is the same as the input number of bits of the next filter. Equation 7 ensures that the Shifter does not introduce negative rejection. Indeed, the results of the FIR can be right shifted without compromising the quality of the rejection until a threshold. Each bit of the output data increases the maximum rejection level by 6 dB. We add one to take the sign bit into account. If equation 7 was not present, the Shifter could shift too much and introduce some noise in the output data. Each supplementary shift bit would cause an additional 6 dB rejection rise. A totally equivalent equation is: $\pi_i^S \leq \pi_i^- + \pi_i^C - 1 - \sum_{k=1}^{i} \left(1 + \frac{r_j}{6}\right)$. Finally, equation 8 gives the number of bits of the global input.

This model is non-linear since we multiply some variable with another variable and it is even non-quadratic, as the cost function $F$ does not have a known linear or quadratic expression. To linearize this problem, we introduce $p$ FIR configurations. This variable $p$ is defined by the user, and represents the number of different set of coefficients generated (remember, we use `firls` and `fir1` functions from GNU Octave) based on the targeted filter characteristics and implementation assumptions (estimated number of bits defining the coefficients). Hence, $C_{ij}$ and $\pi_{ij}^C$ become constants and we define $1 \leq j \leq p$ so that the function $F$ can be estimated (Look Up Table) for each configurations thanks to the rejection criterion. We also define the binary variable $\delta_{ij}$ that has value 1 if stage $i$ is in configuration $j$ and 0 otherwise. The new equations are as follows:

$$a_i = \sum_{j=1}^{p} \delta_{ij} \times C_{ij} \times (\pi_{ij}^C + \pi_i^-), \quad \forall i \in [1,n] \tag{9}$$

$$r_i = \sum_{j=1}^{p} \delta_{ij} \times F(C_{ij}, \pi_{ij}^C), \quad \forall i \in [1,n] \tag{10}$$

$$\pi_i^+ = \pi_i^- + \left( \sum_{j=1}^{p} \delta_{ij} \pi_{ij}^C \right) - \pi_i^S, \quad \forall i \in [1,n] \tag{11}$$

$$\sum_{j=1}^{p} \delta_{ij} \leq 1, \quad \forall i \in [1,n] \tag{12}$$

Equations 9, 10 and 11 replace respectively equations 3, 4 and 5. Equation 12 states that for each stage, a single configuration is chosen at most.

The problem remains quadratic at this stage since in the constraint 9 we multiply $\delta_{ij}$ and $\pi_i^-$. However, since $\delta_{ij}$ is a binary variable we can linearize this multiplication. The following formula shows how to linearize this situation in general case with $y$ a binary variable and $x$ a real variable ($0 \leq x \leq X^{max}$):

$$m = x \times y \implies \begin{cases} m \geq 0 \\ m \leq y \times X^{max} \\ m \leq x \\ m \geq x - (1-y) \times X^{max} \end{cases}$$

So if we bound up $\pi_i^-$ by 128 bits which is the maximum data size whose estimation is assumed on hardware characteristics, the Gurobi (www.gurobi.com) optimization software will be able to linearize for us the quadratic problem so the model is left as is. This model has $O(np)$ variables and $O(n)$ constraints.

Two problems will be addressed using the workflow described in the next section: on the one hand maximizing the rejection capability of a set of cascaded filters occupying a fixed arbitrary silicon area (section V) and on the second hand the dual problem of minimizing the silicon area for a fixed rejection criterion (section V-A). In the latter case, the objective function is replaced with:

$$\text{Minimize} \sum_{i=1}^{n} a_i$$

We adapt our constraints of quadratic program to replace equation 2 with equation 13 where $\mathcal{R}$ is the minimal rejection required.

$$\sum_{i=1}^{n} r_i \geq \mathcal{R} \tag{13}$$

## IV. Design workflow

In this section, we describe the workflow to compute all the results presented in sections V and V-A. Figure 8 shows the global workflow and the different steps involved in the computation of the results.

The filter solver is a C++ program that takes as input the maximum area $\mathcal{A}$, the number of stages $n$, the size of the input signal $\Pi^I$, the FIR configurations $(C_{ij}, \pi_{ij}^C)$ and the function $F$. It creates the quadratic programs and uses the Gurobi solver

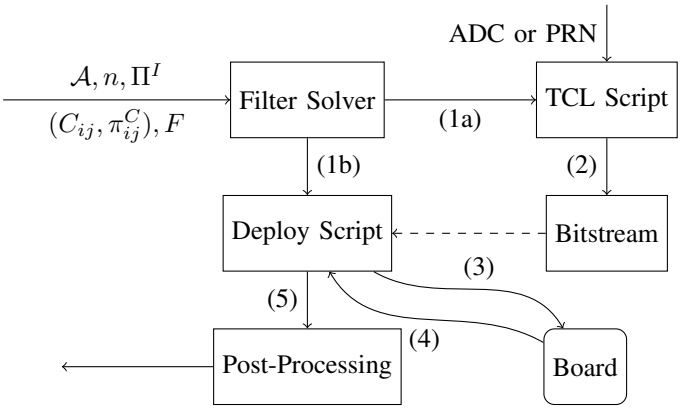



Fig. 8: Design workflow from the input parameters to the results allowing for a fully automated optimal solution search.

to estimate the optimal results. Then it produces two scripts: a TCL script ((1a) on figure 8) and a deploy script ((1b) on figure 8).

The TCL script describes the whole digital processing chain from the beginning (the raw signal data) to the end (the filtered data) in a language compatible with proprietary synthesis software, namely Vivado for Xilinx and Quartus for Intel/Altera. The raw input data generated from a 20-bit Pseudo Random Number (PRN) generator inside the FPGA and $\Pi^I$ is fixed at 16 bits. Then the script builds each stage of the chain with a generic FIR task that comes from a skeleton library. The generic FIR is highly configurable with the number of coefficients and the size of the coefficients. The coefficients themselves are not stored in the script. As the signal is processed in real-time, the output signal is stored as consecutive bursts of data for post-processing, mainly assessing the consistency of the implemented FIR cascade transfer function with the design criteria and the expected transfer function.

The TCL script is used by Vivado to produce the FPGA bitstream ((2) on figure 8). We use the 2018.2 version of Xilinx Vivado and we execute the synthesized bitstream on a Redpitaya board fitted with a Xilinx Zynq-7010 series FPGA (xc7z010clg400-1) and two LTC2145 14-bit 125 MS/s ADC, loaded with 50 Ω resistors to provide a broadband noise source. The board runs the Linux kernel and surrounding environment produced from the Buildroot framework available at https://github.com/trabucayre/redpitaya/: configuring the Zynq FPGA, feeding the FIR with the set of coefficients, executing the simulation and fetching the results is automated.

The deploy script uploads the bitstream to the board ((3) on figure 8), flashes the FPGA, loads the different drivers, configures the coefficients of the FIR filters. It then waits for the results and retrieves the data to the main computer ((4) on figure 8).

Finally, an Octave post-processing script computes the final results thanks to the output data ((5) on figure 8). The results are normalized so that the Power Spectrum Density (PSD) starts at zero and the different configurations can be compared.

## V. Maximizing the rejection at fixed silicon area

This section presents the output of the filter solver *i.e.* the computed configurations for each stage, the computed rejection and the computed silicon area. Such results allow

TABLE I: Configurations $(C_i, \pi_i^C, \pi_i^S)$, rejections and areas (in arbitrary units) for MAX/500

| $n$ | $i=1$ | $i=2$ | $i=3$ | $i=4$ | $i=5$ | Rejection | Area |
|---|---|---|---|---|---|---|---|
| 1 | (21, 7, 0) | - | - | - | - | 32 dB | 483 |
| 2 | (3, 5, 18) | (33, 10, 0) | - | - | - | 48 dB | 492 |
| 3 | (3, 5, 18) | (19, 7, 1) | (15, 7, 0) | - | - | 56 dB | 493 |
| 4 | (3, 5, 18) | (19, 7, 1) | (15, 7, 0) | - | - | 56 dB | 493 |
| 5 | (3, 5, 18) | (19, 7, 1) | (15, 7, 0) | - | - | 56 dB | 493 |

TABLE II: Configurations $(C_i, \pi_i^C, \pi_i^S)$, rejections and areas (in arbitrary units) for MAX/1000

| $n$ | $i=1$ | $i=2$ | $i=3$ | $i=4$ | $i=5$ | Rejection | Area |
|---|---|---|---|---|---|---|---|
| 1 | (37, 11, 0) | - | - | - | - | 56 dB | 999 |
| 2 | (15, 8, 17) | (35, 11, 0) | - | - | - | 80 dB | 990 |
| 3 | (3, 13, 26) | (31, 9, 1) | (27, 9, 0) | - | - | 92 dB | 999 |
| 4 | (3, 5, 18) | (19, 7, 1) | (19, 7, 0) | (19, 7, 0) | - | 98 dB | 994 |
| 5 | (3, 5, 18) | (19, 7, 1) | (19, 7, 0) | (19, 7, 0) | - | 98 dB | 994 |

for understanding the choices made by the solver to compute its solutions.

The experimental setup is composed of three cases. The raw input is generated by a Pseudo Random Number (PRN) generator, which fixes the input data size $\Pi^I$. Then the total silicon area $\mathcal{A}$ has been fixed to either 500, 1000 or 1500 arbitrary units. Hence, the three cases have been named: MAX/500, MAX/1000, MAX/1500. The number of configurations $p$ is 1133, with $C_i$ ranging from 3 to 60 and $\pi^C$ ranging from 2 to 22. In each case, the quadratic program has been able to give a result up to five stages ($n = 5$) in the cascaded filter.

Table I shows the results obtained by the filter solver for MAX/500. Table II shows the results obtained by the filter solver for MAX/1000. Table III shows the results obtained by the filter solver for MAX/1500.

By analyzing these tables, we can first state that we reach an optimal solution for each case : $n = 3$ for MAX/500, and $n = 4$ for MAX/1000 and MAX/1500. Moreover the cascaded filters always exhibit better performance than the monolithic solution. It was an expected result as it has been previously observed that many small filters are better than a single large filter [9], [8], [10], despite such conclusions being hardly used in practice due to the lack of tools for identifying individual filter coefficients in the cascaded approach.

Second, the larger the silicon area, the better the rejection. This was also an expected result as more area means a filter of better quality with more coefficients or more bits per coefficient.

Then, we also observe that the first stage can have a larger shift than the other stages. This is explained by the fact that the solver tries to use just enough bits for the computed rejection after each stage. In the first stage, a balance between a strong rejection with a low number of bits is targeted. Equation 7 gives the relation between both values.

Finally, we note that the solver consumes all the given silicon area.

TABLE III: Configurations $(C_i, \pi_i^C, \pi_i^S)$, rejections and areas (in arbitrary units) for MAX/1500

| $n$ | $i=1$ | $i=2$ | $i=3$ | $i=4$ | $i=5$ | Rejection | Area |
|---|---|---|---|---|---|---|---|
| 1 | (47, 15, 0) | - | - | - | - | 71 dB | 1457 |
| 2 | (19, 6, 15) | (51, 14, 0) | - | - | - | 102 dB | 1489 |
| 3 | (15, 9, 18) | (31, 8, 0) | (27, 9, 0) | - | - | 116 dB | 1488 |
| 4 | (3, 9, 22) | (31, 9, 1) | (27, 9, 0) | (19, 7, 0) | - | 125 dB | 1500 |
| 5 | (3, 9, 22) | (31, 9, 1) | (27, 9, 0) | (19, 7, 0) | - | 125 dB | 1500 |

The following graphs present the rejection for real data on the FPGA. In all the following figures, the solid line represents the actual rejection of the filtered data on the FPGA as measured experimentally and the dashed line are the noise levels given by the quadratic solver. The configurations are those computed in the previous section.

Figure 9a shows the rejection of the different configurations in the case of MAX/500. Figure 9b shows the rejection of the different configurations in the case of MAX/1000. Figure 9c shows the rejection of the different configurations in the case of MAX/1500.

In all cases, we observe that the actual rejection is close to the rejection computed by the solver.

We compare the actual silicon resources given by Vivado to the resources in arbitrary units. The goal is to check that our arbitrary units of silicon area models well enough the real resources on the FPGA. Especially we want to verify that, for a given number of arbitrary units, the actual silicon resources do not depend on the number of stages $n$. Most significantly, our approach aims at remaining far enough from the practical logic gate implementation used by various vendors to remain platform independent and be portable from one architecture to another.

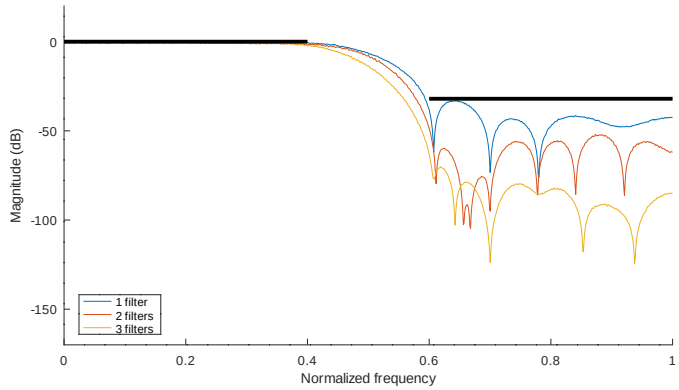
Table IV shows the resources usage in the case of MAX/500, MAX/1000 and MAX/1500 *i.e.* when the maximum allowed silicon area is fixed to 500, 1000 and 1500 arbitrary units. We have taken care to extract solely the resources used by the FIR filters and remove additional processing blocks including FIFO and Programmable Logic (PL – FPGA) to Processing System (PS – general purpose processor) communication.

TABLE IV: Resource occupation following synthesis of the solutions found for the problem of maximizing rejection for a given resource allocation. The last column refers to available resources on a Zynq-7010 as found on the Redpitaya.

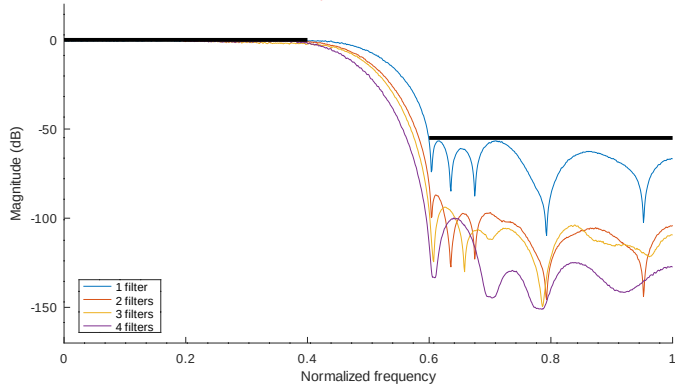
| $n$ | | MAX/500 | MAX/1000 | MAX/1500 | *Zynq 7010* |
|---|---|---|---|---|---|
| 1 | LUT | 249 | 453 | 627 | *17600* |
| | BRAM | 1 | 1 | 1 | *120* |
| | DSP | 21 | 37 | 47 | *80* |
| 2 | LUT | 2253 | 474 | 691 | *17600* |
| | BRAM | 2 | 2 | 2 | *120* |
| | DSP | 0 | 50 | 70 | *80* |
| 3 | LUT | 1329 | 2006 | 3158 | *17600* |
| | BRAM | 3 | 3 | 3 | *120* |
| | DSP | 15 | 30 | 42 | *80* |
| 4 | LUT | 1329 | 1600 | 2260 | *17600* |
| | BRAM | 3 | 4 | 4 | *120* |
| | DPS | 15 | 38 | 49 | *80* |
| 5 | LUT | 1329 | 1600 | 2260 | *17600* |
| | BRAM | 3 | 4 | 4 | *120* |
| | DPS | 15 | 38 | 49 | *80* |

In case $n = 2$ for MAX/500, Vivado replaces the DSPs by Look Up Tables (LUTs). We assume that, when the filter coefficients are small enough, or when the input size is small enough, Vivado optimizes resource consumption by selecting multiplexers to implement the multiplications instead of a DSP. In this case, it is quite difficult to compare the whole silicon budget.

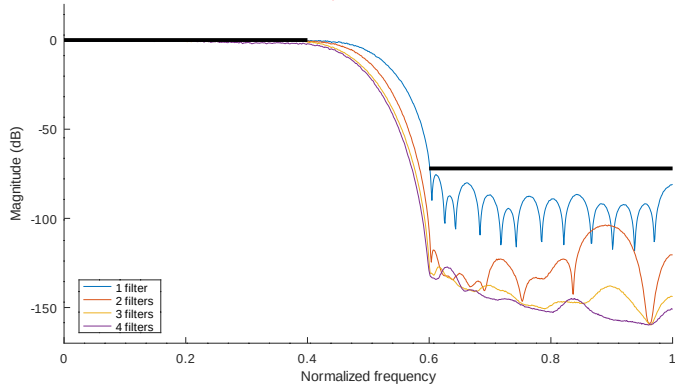
However, a rough estimation can be made with a simple equivalence: looking at the first column (MAX/500), where the number of LUTs is quite stable for $n \geq 2$, we can deduce that a DSP is roughly equivalent to 100 LUTs in terms of




(a) Filter transfer functions for varying number of cascaded filters solving the MAX/500 problem of maximizing rejection for a given resource allocation (500 arbitrary units).




(b) Filter transfer functions for varying number of cascaded filters solving the MAX/1000 problem of maximizing rejection for a given resource allocation (1000 arbitrary units).




(c) Filter transfer functions for varying number of cascaded filters solving the MAX/1500 problem of maximizing rejection for a given resource allocation (1500 arbitrary units).

Fig. 9: Solutions for the MAX/500, MAX/1000 and MAX/1500 problems of maximizing rejection for a given resource allocation. The filter shape constraint (bandpass and bandstop) is shown as thick horizontal lines on each chart.

silicon area use. With this equivalence, our 500 arbitrary units correspond to 2500 LUTs, 1000 arbitrary units correspond to 5000 LUTs and 1500 arbitrary units correspond to 7300 LUTs. The conclusion is that the orders of magnitude of our arbitrary unit map well to actual hardware resources. The relatively small differences can probably be explained by the optimizations done by Vivado based on the detailed map of available processing resources.

We now present the computation time needed to solve the quadratic problem. For each case, the filter solver software is executed on a Intel(R) Xeon(R) CPU E5606 clocked at 2.13 GHz. The CPU has 8 cores that are used by Gurobi to solve the quadratic problem. Table V shows the time needed to solve the quadratic problem when the maximal area is fixed to 500, 1000 and 1500 arbitrary units.

TABLE V: Time needed to solve the quadratic program with Gurobi

| $n$ | Time (MAX/500) | Time (MAX/1000) | Time (MAX/1500) |
|---|---|---|---|
| 1 | 0.01 s | 0.02 s | 0.03 s |
| 2 | 0.1 s | 1 s | 2 s |
| 3 | 5 s | 27 s | 351 s ($\approx$ 6 min) |
| 4 | 4 s | 141 s ($\approx$ 3 min) | 1134 s ($\approx$ 18 min) |
| 5 | 6 s | 630 s ($\approx$ 10 min) | 49400 s ($\approx$ 13 h) |

As expected, the computation time seems to rise exponentially with the number of stages. When the area is limited, the design exploration space is more limited and the solver is able to find an optimal solution faster. We also notice that the solution with $n$ greater than the optimal value takes more time to be found than the optimal one. This can be explained since the search space is larger and we need more time to ensure that the previous solution (from the smaller value of $n$) still remains the optimal solution.

### A. *Minimizing resource occupation at fixed rejection*

This section presents the results of the complementary quadratic program aimed at minimizing the area occupation for a targeted rejection level.

The experimental setup is composed of four cases. The raw input is the same as in the previous section, from a PRN generator, which fixes the input data size $\Pi^I$. Then the targeted rejection $\mathcal{R}$ has been fixed to either 40, 60, 80 or 100 dB. Hence, the three cases have been named: MIN/40, MIN/60, MIN/80 and MIN/100. The number of configurations $p$ is the same as previous section.

Table VI shows the results obtained by the filter solver for MIN/40. Table VII shows the results obtained by the filter solver for MIN/60. Table VIII shows the results obtained by the filter solver for MIN/80. Table IX shows the results obtained by the filter solver for MIN/100.

TABLE VI: Configurations $(C_i, \pi_i^C, \pi_i^S)$, rejections and areas (in arbitrary units) for MIN/40

| $n$ | $i=1$ | $i=2$ | $i=3$ | $i=4$ | $i=5$ | Rejection | Area |
|---|---|---|---|---|---|---|---|
| 1 | (27, 8, 0) | - | - | - | - | 41 dB | 648 |
| 2 | (3, 5, 18) | (27, 8, 0) | - | - | - | 42 dB | 360 |
| 3 | (3, 5, 18) | (27, 8, 0) | - | - | - | 42 dB | 360 |
| 4 | (3, 5, 18) | (27, 8, 0) | - | - | - | 42 dB | 360 |
| 5 | (3, 5, 18) | (27, 8, 0) | - | - | - | 42 dB | 360 |

From these tables, we can first state that almost all configurations reach the targeted rejection level or even better thanks

TABLE VII: Configurations $(C_i, \pi_i^C, \pi_i^S)$, rejections and areas (in arbitrary units) for MIN/60

| $n$ | $i=1$ | $i=2$ | $i=3$ | $i=4$ | $i=5$ | Rejection | Area |
|---|---|---|---|---|---|---|---|
| 1 | (39, 13, 0) | - | - | - | - | 60 dB | 1131 |
| 2 | (15, 6, 16) | (23, 9, 0) | - | - | - | 60 dB | 675 |
| 3 | (3, 5, 18) | (15, 6, 2) | (23, 8, 0) | - | - | 60 dB | 543 |
| 4 | (3, 5, 18) | (15, 6, 2) | (23, 8, 0) | - | - | 60 dB | 543 |
| 5 | (3, 5, 18) | (15, 6, 2) | (23, 8, 0) | - | - | 60 dB | 543 |

TABLE VIII: Configurations $(C_i, \pi_i^C, \pi_i^S)$, rejections and areas (in arbitrary units) for MIN/80

| $n$ | $i=1$ | $i=2$ | $i=3$ | $i=4$ | $i=5$ | Rejection | Area |
|---|---|---|---|---|---|---|---|
| 1 | (55, 16, 0) | - | - | - | - | 81 dB | 1760 |
| 2 | (15, 8, 17) | (35, 11, 0) | - | - | - | 80 dB | 990 |
| 3 | (3, 7, 20) | (31, 9, 1) | (19, 7, 0) | - | - | 80 dB | 783 |
| 4 | (3, 7, 20) | (31, 9, 1) | (19, 7, 0) | - | - | 80 dB | 783 |
| 5 | (3, 7, 20) | (31, 9, 1) | (19, 7, 0) | - | - | 80 dB | 783 |

TABLE IX: Configurations $(C_i, \pi_i^C, \pi_i^S)$, rejections and areas (in arbitrary units) for MIN/100

| $n$ | $i=1$ | $i=2$ | $i=3$ | $i=4$ | $i=5$ | Rejection | Area |
|---|---|---|---|---|---|---|---|
| 1 | - | - | - | - | - | - | - |
| 2 | (27, 9, 15) | (35, 11, 0) | - | - | - | 100 dB | 1410 |
| 3 | (3, 5, 18) | (35, 11, 1) | (27, 9, 0) | - | - | 100 dB | 1147 |
| 4 | (3, 5, 18) | (15, 6, 2) | (27, 9, 0) | (19, 7, 0) | - | 100 dB | 1067 |
| 5 | (3, 5, 18) | (15, 6, 2) | (27, 9, 0) | (19, 7, 0) | - | 100 dB | 1067 |

to our underestimate of the cascade rejection as the sum of the individual filter rejection. The only exception is for the monolithic case ($n = 1$) in MIN/100: no solution is found for a single monolithic filter reach a 100 dB rejection. Furthermore, the area of the monolithic filter is twice as big as the two cascaded filters (675 and 1131 arbitrary units v.s 990 and 1760 arbitrary units for 60 and 80 dB rejection respectively). More generally, the more filters are cascaded, the lower the occupied area.

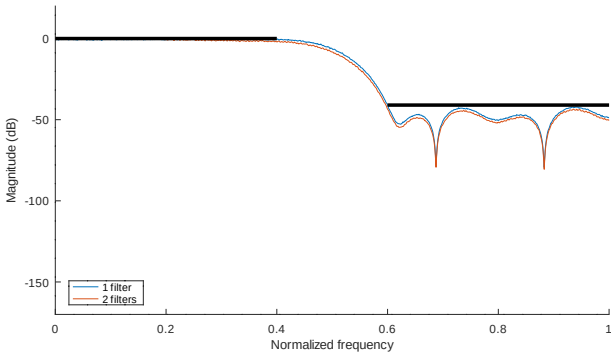
Like in previous section, the solver chooses always a little filter as first filter stage and the second one is often the biggest filter. This choice can be explained as in the previous section, with the solver using just enough bits not to degrade the input signal and in the second filter selecting a better filter to improve rejection without having too many bits in the output data.

For each case, we found an optimal solution with $n < 5$: for MIN/40 $n = 2$, for MIN/60 and MIN/80 $n = 3$ and for MIN/100 $n = 4$. In all cases, the solutions when $n$ is greater than this optimal $n$ remain identical to the optimal one.

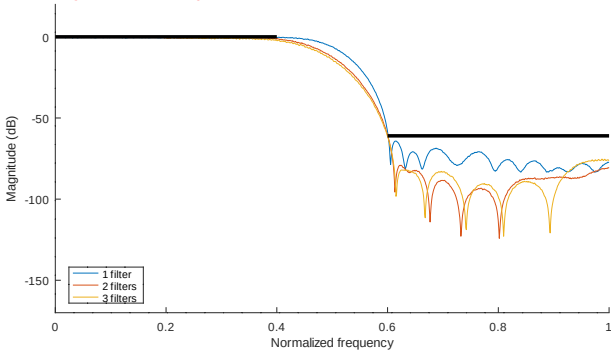
The following graphs present the rejection for real data on the FPGA. In all the following figures, the solid line represents the actual rejection of the filtered data on the FPGA as measured experimentally and the dashed line is the noise level given by the quadratic solver.

Figure 10a shows the rejection of the different configurations in the case of MIN/40. Figure 10b shows the rejection of the different configurations in the case of MIN/60. Figure 10c shows the rejection of the different configurations in the case of MIN/80. Figure 10d shows the rejection of the different configurations in the case of MIN/100.

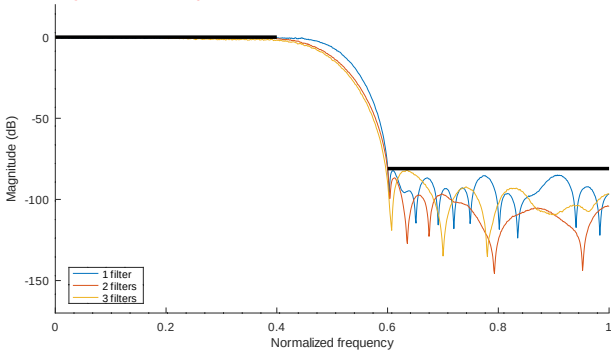
We observe that all rejections given by the quadratic solver are close to the experimentally measured rejection. All curves prove that the constraint to reach the target rejection is respected with both monolithic (except in MIN/100 which has no monolithic solution) or cascaded filters.



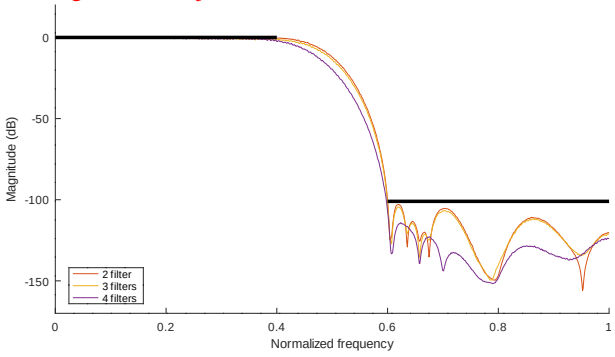

(a) Filter transfer functions for varying number of cascaded filters solving the MIN/40 problem of minimizing resource allocation for reaching a 40 dB rejection.




(b) Filter transfer functions for varying number of cascaded filters solving the MIN/60 problem of minimizing resource allocation for reaching a 60 dB rejection.




(c) Filter transfer functions for varying number of cascaded filters solving the MIN/80 problem of minimizing resource allocation for reaching a 80 dB rejection.




(d) Filter transfer functions for varying number of cascaded filters solving the MIN/100 problem of minimizing resource allocation for reaching a 100 dB rejection.

Fig. 10: Solutions for the MIN/40, MIN/60, MIN/80 and MIN/100 problems of reaching a given rejection while minimizing resource allocation. The filter shape constraint (bandpass and bandstop) is shown as thick horizontal lines on each chart.

TABLE X: Resource occupation. The last column refers to available resources on a Zynq-7010 as found on the Redpitaya.

| $n$ | | MIN/40 | MIN/60 | MIN/80 | MIN/100 | *Zynq 7010* |
|---|---|---|---|---|---|---|
| 1 | LUT | 343 | 334 | 772 | - | *17600* |
| | BRAM | 1 | 1 | 1 | - | *120* |
| | DSP | 27 | 39 | 55 | - | *80* |
| 2 | LUT | 1664 | 2329 | 474 | 620 | *17600* |
| | BRAM | 2 | 2 | 2 | 2 | *120* |
| | DSP | 0 | 15 | 50 | 62 | *80* |
| 3 | LUT | 1664 | 3114 | 1884 | 2873 | *17600* |
| | BRAM | 2 | 3 | 3 | 3 | *120* |
| | DSP | 0 | 0 | 22 | 27 | *80* |
| 4 | LUT | 1664 | 3114 | 2570 | 4318 | *17600* |
| | BRAM | 2 | 3 | 4 | 4 | *120* |
| | DPS | 0 | 15 | 19 | 19 | *80* |
| 5 | LUT | 1664 | 3114 | 2570 | 4318 | *17600* |
| | BRAM | 2 | 3 | 4 | 4 | *120* |
| | DPS | 0 | 0 | 19 | 19 | *80* |

Table IV shows the resource usage in the case of MIN/40, MIN/60; MIN/80 and MIN/100 *i.e.* when the target rejection is fixed to 40, 60, 80 and 100 dB. We have taken care to extract solely the resources used by the FIR filters and remove additional processing blocks including FIFO and PL to PS communication.

If we keep the previous estimation of cost of one DSP in terms of LUT (1 DSP $\approx$ 100 LUT) the real resource consumption decreases as a function of the number of stages in the cascaded filter according to the solution given by the quadratic solver. Indeed, we have always a decreasing consumption even if the difference between the monolithic and the two cascaded filters is less than expected.

Finally, table XI shows the computation time to solve the quadratic program.

TABLE XI: Time to solve the quadratic program with Gurobi

| $n$ | Time (MIN/40) | Time (MIN/60) | Time (MIN/80) | Time (MIN/100) |
|---|---|---|---|---|
| 1 | 0.04 s | 0.01 s | 0.01 s | - |
| 2 | 2.7 s | 2.4 s | 2.4 s | 0.8 s |
| 3 | 4.6 s | 7 s | 7 s | 18 s |
| 4 | 3 s | 22 s | 70 s | 220 s ($\approx$ 3 min) |
| 5 | 5 s | 122 s | 200 s | 384 s ($\approx$ 5 min) |

The time needed to solve this configuration is significantly shorter than the time needed in the previous section. Indeed the worst time in this case is only 5 minutes, compared to 13 hours in the previous section: this problem is more easily solved than the previous one.

To conclude, we compare our monolithic filters with the FIR Compiler provided by Xilinx in the Vivado software suite (v.2018.2). For each experiment we use the same coefficient set and we compare the resource consumption, having checked that the transfer functions are indeed the same with both implementations. Table XII exhibits the results. The FIR Compiler never uses BRAM while our filter implementation uses one block. This difference is explained be our wish to have a dynamically reconfigurable FIR filter whose coefficients can be updated from the processing system without having to update the FPGA design. With the FIR compiler, the coefficients are defined during the FPGA design so that changing coefficients required generating a new design. The difference with the LUT consumption is also attributed to the reconfigurability logic. However the DSP consumption, the scarcest resource, is the same between the Xilinx FIR Compiler end our FIR block: we hence conclude that our solutions are as good as the Xilinx implementation.

TABLE XII: Resource consumption compared between the FIR Compiler from Xilinx and our FIR block

| | Xilinx | | | Our FIR block | | |
|---|---|---|---|---|---|---|
| | LUT | BRAM | DSP | LUT | BRAM | DSP |
| MAX/500 | 177 | 0 | 21 | 249 | 1 | 21 |
| MAX/1000 | 306 | 0 | 37 | 453 | 1 | 37 |
| MAX/1500 | 418 | 0 | 47 | 627 | 1 | 47 |
| MIN/40 | 225 | 0 | 27 | 347 | 1 | 27 |
| MIN/60 | 322 | 0 | 39 | 334 | 1 | 39 |
| MIN/80 | 482 | 0 | 55 | 772 | 1 | 55 |

## VI. Conclusion

We have proposed a new approach to optimize a set of signal processing blocks whose performances and resource consumption has been tabulated, and applied this methodology to the practical case of implementing cascaded FIR filters inside a FPGA. This method aims to be hardware independent and focuses an a high-level of abstraction. We have modeled the FIR filter operation and the impact of data shift. Thanks to this model, we have created a quadratic program to select the optimal FIR taps to reach a targeted rejection. Individual filter taps have been identified using commonly available tools and the emphasis is on FIR assembly rather than individual FIR coefficient identification.

Our experimental results are very promising in providing a rational approach to selecting the coefficients of each FIR filter in the context of a performance target for a chain of such filters. The FPGA design that is produced automatically by the proposed workflow is able to filter an input signal as expected, validating experimentally our model and our approach. The quadratic program can be adapted it to an other problem based on assembling skeleton blocks.

Considering that all area and rejection considerations could be explored within a reasonable computation duration, and that no improvement is observed when cascading more than four filters, we consider that this particular problem has been exhaustively investigated and optimal solutions found in all cases.

A perspective is to model and add the decimators to the processing chain to have a classical FIR filter and decimator. The impact of the decimator is not trivial, especially in terms of silicon area usage for subsequent stages since some hardware optimization can be applied in this case.

The software used to demonstrate the concepts developed in this paper is based on the CPU-FPGA co-design framework available at https://github.com/oscimp/oscimpDigital.

## Acknowledgement

This work is supported by the ANR Programme d'Investissement d'Avenir in progress at the Time and Frequency Departments of the FEMTO-ST Institute (Oscillator IMP, First-TF and Refimeve+), and by Région de Franche-Comté. The authors would like to thank E. Rubiola, F. Vernotte, and G. Cabodevila for support and fruitful discussions.

## REFERENCES


[1] J. A. Sherman and R. Jördens, "Oscillator metrology with software defined radio," *Review of Scientific Instruments*, vol. 87, no. 5, p. 054711, 2016.

[2] C. Andrich, A. Ihlow, J. Bauer, N. Beuster, and G. Del Galdo, "High-precision measurement of sine and pulse reference signals using software-defined radio," *IEEE Transactions on Instrumentation and Measurement*, vol. 67, no. 5, pp. 1132–1141, May 2018.

[3] S. J. Kasbah, I. W. Damaj, and R. A. Haraty, "Multigrid solvers in reconfigurable hardware," *Journal of Computational and Applied Mathematics*, vol. 213, no. 1, pp. 79–94, 2008.

[4] D. Crookes, K. Alotaibi, A. Bouridane, P. Donachy, and A. Benkrid, "An environment for generating FPGA architectures for image algebra-based algorithms," in *Image Processing, 1998. ICIP 98. Proceedings. 1998 International Conference on*. IEEE, 1998, pp. 990–994.

[5] D. Crookes, K. Benkrid, A. Bouridane, K. Alotaibi, and A. Benkrid, "Design and implementation of a high level programming environment for FPGA-based image processing," *IEE Proceedings-Vision, Image and Signal Processing*, vol. 147, no. 4, pp. 377–384, 2000.

[6] K. Benkrid, D. Crookes, and A. Benkrid, "Towards a general framework for FPGA based image processing using hardware skeletons," *Parallel Computing*, vol. 28, no. 7, pp. 1141–1154, 2002.

[7] K. Benkrid, S. Belkacemi, and A. Benkrid, "Hide: A hardware intelligent description environment," *Microprocessors and Microsystems*, vol. 30, no. 6, pp. 283–300, 2006.

[8] Y.-C. Lim, R. Yang, and B. Liu, "The design of cascaded fir filters," in *1996 IEEE International Symposium on Circuits and Systems. Circuits and Systems Connecting the World. ISCAS 96*, vol. 2, May 1996, pp. 181–184 vol.2.

[9] Y. C. Lim and B. Liu, "Design of cascade form fir filters with discrete valued coefficients," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 36, no. 11, pp. 1735–1739, Nov 1988.

[10] C. Young and D. L. Jones, "Improvement in finite wordlength fir digital filter design by cascading," in *[Proceedings] ICASSP-92: 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 5, March 1992, pp. 109–112 vol.5.

[11] L. M. Smith, "Decomposition of fir digital filters for realization via the cascade connection of subfilters," *IEEE Transactions on Signal Processing*, vol. 46, no. 6, pp. 1681–1684, June 1998.

[12] A. C. Olaya, S. Micalizio, M. Ortolano, C. Calosso, E. Rubiola, and J. Friedt, "Digital electronics based on red pitaya platform for coherent fiber links," in *2016 European Frequency and Time Forum (EFTF)*. IEEE, 2016, pp. 1–4.

[13] A. C. C. Olaya, C. E. Calosso, J.-M. Friedt, S. Micalizio, and E. Rubiola, "Phase noise and frequency stability of the red-pitaya internal pll," *IEEE transactions on ultrasonics, ferroelectrics, and frequency control*, vol. 66, no. 2, pp. 412–416, 2019.