Performance-oriented Implementation of Hilbert Filters on FPGAs

Daniel Fortún, Carlos García de la Cueva, Jesús Grajal, Marisa López-Vallejo, Carlos López Barrio
Information Processing and Telecommunication Center (IPTC)
Universidad Politecnica de Madrid, Madrid, Spain
{daniel.forsanchez, carlos.gdelacueva, jesus.grajal, m.lopez.vallejo, cl.barrio}@upm.es

***Abstract*—FPGAs can implement digital signal processing algorithms in a fast and efficient way. In this work we present different hardware architectures for the implementation of the Hilbert transform on FPGAs. This transform is especially useful for the generation of analytic signals that are required by many applications (digital demodulators, PLLs, electronic warfare receivers, etc.). We have explored the implementation of several structures looking for architectures that allow real time processing of high input rates with a reasonable area overhead. A new architecture that provides efficient computation has been presented. Experimental results with Xilinx FPGAs show the feasibility of the proposal.**

***Index Terms*—FIR filter, Hilbert transform, FPGA, Electronic Warfare Receivers**

I. Introduction

Field-programmable gate arrays (FPGAs) large gate count makes possible the implementation of full systems on an FPGA. This is especially convenient for on-board digital signal processing systems, characterized by hard space constraints additionally to the conventional constraints on performance and power. FPGAs have shown their usefulness when designing high performance tasks with real-time requirements. FPGAs offer high levels of parallelism and reasonably high performance for a large set of applications in the signal processing domain, providing very efficient implementation of basic operations like multiply-add, wide vector operation, long pipelines, lookup-tables, on-chip memory or shift registers.

When dealing with signal processing systems, the implementation of several algorithms plays a key role, because of their usefulness and applicability. This is the case of the Fast Fourier Transform (FFT) [1], the Discret Cosine Transform (DCT) [2] or the Hilbert Transform [3]. In this work we will focus on the use of the Hilbert transform to generate an analytic signal from a real valued signal.

Nowadays ADCs provide conversion rates in the range of GHz which must be processed by the hardware that follows. This is not feasible for most implementation platforms, and more especially when dealing with programmable devices. A similar case may happen when a parallel input is provided and serialization is required then to feed for example a FIR filter. This may produce long latencies that severely affect the final performance of the whole system. In this work we have explored several implementations that provide some degree of concurrency in the computation of the Hilbert transform. Concurrent implementations allow several sub-sequences to be processed in parallel. Consequently, high input rates can be considered and a feasible clock cycle is available for the rest of the system.

This work has been funded by the project TOLERA2 TEC2015-65902.

There are many applications that benefit from the use of the Hilbert transform. Most of them use this transform to generate an analytic signal from a real valued signal. For the design of phase-locked loops (PLL), phase detectors are required. The CORDIC algorithm [4] is an efficient way to implement phase detection, but it requires the input be a complex analytic signal. Here the connection between I (in-phase) and Q (quadrature) components is given by the Hilbert transform. The implementation of FPGA-based PLLs has been proposed for very different fields of application, as they can be the phase measurement system of a heavy-ion synchrotron [5] or for synchronization in single-phase AC-DC converters [6]. In communication systems, most digital demodulators require the generation of complex data from real signals. Another case is an electronic warfare receiver, whose input is always a single channel and data that can be considered as real and it requires the generation of two output channels that are 90-deg out of phase, what is called I (in-phase) and Q (quadrature) channel conversion [3]. One digital approach to accomplished this conversion is the use of the discrete Hilbert transform.

The FPGAs provide a solid platform to implement Hilbert filters characterized by flexibility during the design flow and reasonable cost. However, their performance is far from that provided by VLSI circuits. To overcome this drawback parallel architectures must be used in FPGAs, expending area to provide better performance. In this context we analyze in this work the implementation of several architectures for the Hilbert filter, looking for a good area-performance trade-off. We have first studied the problem from the signal processing point of view, looking for the best algorithm choices to implement this transform, and after that we have searched for the most efficient implementation of those Hilbert filters on FPGAs.

In [7] an in-depth analysis of common methods for the analytic signal generation is performed. The goal of this analysis is the use of the Hilbert transform for PLLs measuring phase. IIR filters are used, but in this work FIR structures are preferred because, due to the nature of IIR filters, the group delay is not constant.

978-1-7281-0171-2/18/$31.00 ©2018 IEEE

The structure of this paper is the following. Next section describes the mathematical foundation of the Hilbert transform. Section IV describes the proposed architectures, from the baseline to the most efficient in terms of performance and area. Finally, experimental results are presented and some conclusions are drawn.

II. Analytic signal generation

Given a real-valued input function $x[n]$, its analytic signal can be obtained as follows:

$$\widetilde{x}[n] = x[n] + j\mathcal{H}\{x[n]\} \tag{1}$$

where $\mathcal{H}\{\cdot\}$ is the Hilbert transform operator.

The resulting complex signal ($\widetilde{x}[n]$) has the property that no spectral components exist for negative frequencies. This is, $\widetilde{X}(e^{j\omega}) = 0$, $-\pi < \omega < 0$ in which $\widetilde{X}(e^{j\omega})$ is the Discrete-Time Fourier Transform of $\widetilde{x}[n]$.

This latter statement leads to two different approaches to generate the analytic signal: The straightforward implementation (section III-A), which boils down to apply the Hilbert Transform definition (described in section II-A); and the other alternatives (sections III-D - III-C), which directly filter out the negative frequency components.

A. Hilbert Transform definition

The Hilbert Transform of a continuous time signal $x(t)$ is defined as the convolution of $x(t)$ and a function $h(t) = 1/\pi t$ [8]:

$$\mathcal{H}[x(t)] = x(t) * h(t) = x(t) * \frac{1}{\pi t} \tag{2}$$

In the frequency domain the Hilbert transform may be expressed as:

$$\mathcal{H}^h(f) = X(f) \times H(f) = X(f) \times (-j\ sgn(f)) \tag{3}$$

since the Fourier transform of $h(t)$ is $H(f) = -j\ sgn(f)$.

The discrete Hilbert transform can be obtained by using a finite impulse filter (FIR) scheme. The equivalent digital filter $h[n] = h(nT_s)$ can be computed by using its Discrete-Time Fourier transform:

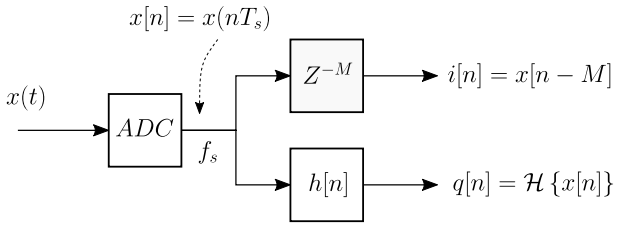
$$H\left(e^{j\omega}\right) = \sum_{-\infty}^{\infty} h(nTs)e^{-j\omega n} = H_r\left(e^{j\omega}\right) + jH_i\left(e^{j\omega}\right) \tag{4}$$

$$H_r\left(e^{j\omega}\right) = h(0) + \sum_{n=1}^{\infty}[h(-nT_s) + h(nTs)]cos(\omega n) \tag{5}$$

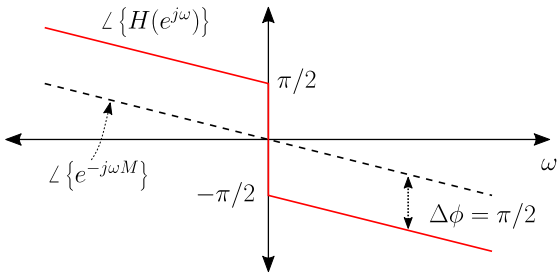
$$H_i\left(e^{j\omega}\right) = \sum_{n=1}^{\infty}[h(-nT_s) - h(nTs)]sen(\omega n) \tag{6}$$

where $\omega = 2\pi fT_s$, $T_s = 1/f_s$ and f_s corresponds to the sampling frequency.

From Eq. 3 we can deduct that the frequency domain representation of the Hilbert transform has only imaginary part, therefore $H_r(e^{j\omega}) = 0$ and $H_i(e^{j\omega}) \neq 0$. Thus, $h(nT_s)$ must fulfill the following conditions:



(a) Diagram of the Hilbert transformer.



(b) Phase evolution of $H(e^{j\omega})$ and Z^{-M}.

Fig. 1: Hilbert transformer scheme.

$$h(0) = 0 \tag{7}$$

$$h(-nT_s) = -h(nT_s) \tag{8}$$

and consequently

$$H\left(e^{j\omega}\right) = H_i\left(e^{j\omega}\right) = \sum_{n=1}^{\infty} b_n \times sen(2\pi fnT_s) \tag{9}$$

where b_n are the Fourier series coefficients corresponding to H_i:

$$b_n = \frac{1}{n\pi}[-2 + 2 \cdot cos(n\pi)] = \begin{cases} 0 & \text{if } n \text{ is even} \\ \frac{-4}{n\pi} & \text{if } n \text{ is odd} \end{cases} \tag{10}$$

The coefficients of the filter become then

$$b_n = h(-nT_s) - h(nT_s) = -2 \cdot h(nT_s) \tag{11}$$

$$h(nT_s) = -b_n/2 \tag{12}$$

III. Implementation alternatives

A. Straightforward method

Equation 10 and 12 represent the coefficients ($h[n]$) of an infinite-length Hilbert filter. For a digital filter to be feasible, the number of coefficients must be finite, which is equivalent to multiply a rectangular window to the input signal, for instance $n \in [-M, M]$ (filter length $L = 2M + 1$). Additionally, the filter must be causal, and therefore the summation cannot start from a negative value. For this reason $h[n] = 0\ \forall\ n < 0$. Consequently, a time shift of M samples must be applied.

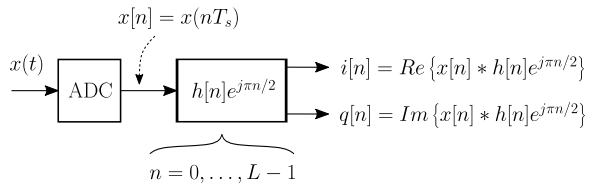


Fig. 2: Diagram of the complex filter structure.

As a result, $h[n]$ can be characterized by two parameters, phase (τ_ϕ) and group (τ_g) delay (both expressed in samples):

$$\tau_g = -d\left(\phi(\omega)\right)/d\omega = M \quad (13)$$

$$\tau_\phi = -\phi(\omega)/\omega = \frac{\pi/2 \cdot sign(w)}{w} + M \quad (14)$$

where $\phi(\omega) = \angle\left\{H\left(e^{j\omega}\right)\right\} = -\pi/2 \cdot sign(\omega) - \omega M$. However, it is more common to represent τ_ϕ in radians: $\tau_\phi(rad) = \omega \cdot \tau_\phi(sp)$.

A signal filtered with $h[n]$ will suffer a time delay of $\tau_g = M$ samples and a phase shift of $\tau_\phi = \pi/2 + \omega \cdot M$ radians. It is important to be aware that group delay is an effect that arises due to the causality condition. To compensate this issue, a time delay of M samples is applied to the real part (in-phase component) of the analytic signal (Eq. 1) as shown in Fig.1a. Phase evolution of $H(e^{jw})$ and z^{-M} are depicted in Fig.1b, where it is easy to see that the phase difference between the two branches is $\pi/2$.

B. Complex filter implementation [9]

This structure is composed of a low-pass filter which is translated to $f = f_s/4$ ($\omega = \pi/2$) by a complex exponential (Fig. 2). The bandwidth of the low-pass filter is adjusted so that the stop band extends from $\pm f_s/4$ (Fig. 3a). Thus, after frequency translation the stop band covers the whole of the negative frequency spectrum (Fig. 3b).

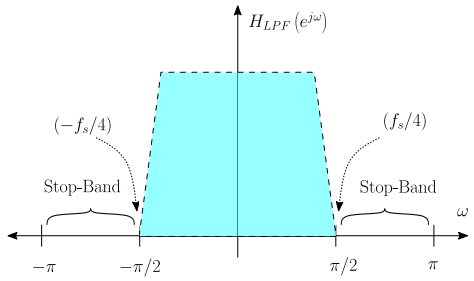
C. Special Sampling Scheme [3]

Another way to generate the analytic signal can be derived from a special sampling scheme. This option performs almost the same signal processing stages as structure III-B but in the reverse order. First, the signal is demodulated with a complex exponential of frequency $f_s/4$, and then it is passed through a low-pass filter to eliminate the image frequency components.

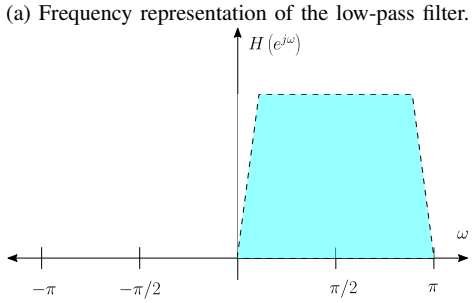
What makes this option computationally efficient is the fact that it takes full advantage of complex demodulation. Real and imaginary parts of the complex exponential (of frequency $f_s/4$) have zeros interspersed (the real part has zeros in the even indices and imaginary part in the odd ones, or vice-versa) and remaining samples are 1 or -1:

$$Re\left\{e^{-j\pi n/2}\right\} = 1, 0, -1, 0, 1, 0, -1, \ldots \quad (15)$$

$$Im\left\{e^{-j\pi n/2}\right\} = 0, -1, 0, 1, 0, -1, 0, \ldots \quad (16)$$



(a) Frequency representation of the low-pass filter.



(b) Complex filter spectrum $\left(H\left(e^{j\Omega}\right) = H_{LPF} \cdot e^{j\pi n/2}\right)$.

Fig. 3: Spectral scheme of the complex Hilbert filter.

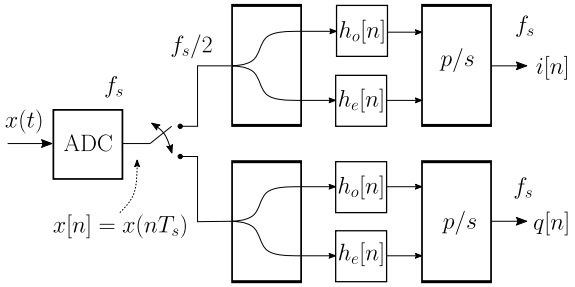


Fig. 4: Diagram of the Special Sampling Scheme technique.

Therefore, the received signal can be divided into two branches, odd samples go to the in-phase branch (real part) and the even ones to the quadrature component (imaginary part). This is performed through the switch depicted in fig. 4 after the ADC block.

After this, real and imaginary components go through a low-pass filter ($h[n]$), which can be divided into its even ($h_e[n]$) and odd ($h_o[n]$) indices to get rid of the interspersed zeros.

$$h_e[n] = h[2n]$$
$$h_o[n] = h[2n+1]$$

where $n = 0, \ldots, \lfloor (L-1)/2 \rfloor$ and L is the filter length of $h[n]$. After filtering, samples merge into the in-phase and quadrature components using a parallel to serial converter.

D. Analytic Signal via FFT [10]

This alternative generates the analytic signal by processing the input function ($x[n]$) in the frequency domain. First of all,

the DFT (*Discrete Fourier Transform*) of the input real-valued signal is computed. Then, negative frequency components are set to zero, positive frequencies are scaled by a factor of 2, and $f = 0$ and $f = f_s/2$ remain unchanged. After this, the analytic signal is calculated through the IDFT (*Inverse Discrete Fourier Transform*) of this latter result (see Fig. 5).

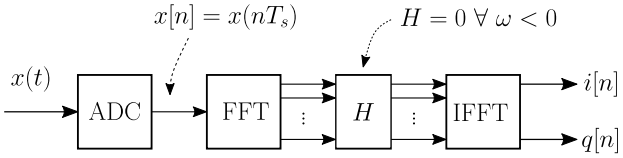


Fig. 5: FFT+IFFT analytic signal generation scheme.

E. Optimized Hardware Architecture for the Hilbert Transformer

One important design requirement in implementing a Hilbert filter on FPGAs is to reduce the operation speed of $h[n]$. If the input data is sampled at f_s, the filter must operate at the same speed for real time operation.

Nevertheless, half the filter coefficients are zeros (see Eq. 10). Taking these zeros into consideration the filter operation speed can be reduced to $f_s/2$ at the cost of duplicating the number of required filters. The approach is discussed below.

We can write $q[n]$ in Fig.1a as a function of the input sequence ($x[n]$) and non-zero filter coefficients of $h[n]$ (odd terms, see Eq. 10) as follows:

$$\begin{aligned} q[0] &= x[0]h[1] \\ q[1] &= x[1]h[1] \\ q[2] &= x[2]h[1] + x[0]h[3] \\ q[3] &= x[3]h[1] + x[1]h[3] \\ &\dots \end{aligned}$$

It is interesting to note that odd-outputs contain only odd-inputs of $x[n]$, and even-outputs contain only even-inputs. Therefore, $x[n]$ can be separated into its odd and even samples (fig. 6). Each branch is filtered by $h'[n]$, which is $h[n]$ after eliminating the zero coefficients.

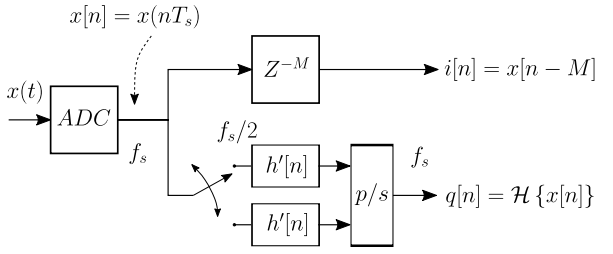


Fig. 6: Diagram of the modified Hilbert transformer.

TABLE I: Comparison of the different Hilbert alternatives.

	Baseline	Complex Filter	Special Sampling	Optimized Architecture
Multipliers	$(L+1)/4$	$2L$	$2L$	$(L+1)/2$
Data rate	f_s	f_s	$f_s/2$	$f_s/2$

F. Comparison of the presented alternatives

The different implementation alternatives presented before allow a broad range of FPGA architectures. The generation of analytic signals via FFT is interesting in terms of performance, but it comes at the cost of high area overhead for the implementation of the FFT. This possibility will be therefore left out of our study.

In table I we compare all the remaining alternatives in terms of multiplications per clock cycle and operation speed of filters. Results are expressed as a function of the filter length (L) and the ADC sampling frequency (f_s). Given that the main goal of this work is to look for the best implementation in terms of performance and area efficiency we will focus on the implementations that allow a fast sampling frequency, which are *Optimized architecture* and *Special Sampling*.

We must discard the implementation using a complex filter (FIR or IIR), not only because it does not allow a fast sampling, but also due to the its high area cost because of the complex multipliers.

Finally, it is clear that the architecture proposed here, based on approximating the transform by a FIR filter that generates the imaginary part and delays the real one outperforms the conventional *Special Sampling* because for the same clock speed it requires a significantly reduced amount of multipliers.

IV. FPGA Implementation of Hilbert Filters

A. Baseline FIR architecture

Our baseline for the Hilbert filter is the first implementation described in section III-A (the so-called *Baseline*). The implementation, depicted in figure 7 for $L = 59$, consists of two sub-components: the FIR Hilbert filter itself, for the generation of the imaginary part, Q, and a shift register that produces the delay of the Hilbert filter for the real part (I). As was described in section III-A this hardware implementation takes benefit from two important facts: the even coefficients are zero and there is odd symmetry ($b_n = -b_{-n}$). Since even coefficients are zero we can skip their multiplication and consequently the number of multipliers needed for this architecture is $L/2$, with L being the filter length.

Taking advantage of the odd symmetry of coefficients, $b_n = -b_{-n}$, we can obtain a more efficient implementation of this filter. In particular, for the case of Xilinx FPGAs where embedded DSP are available, we can use the pre-adder included in the DSPs in the following way:

$$x(i) \times b(L-i) + x(L-i) \times b(i) = b(i) \times [x(i) - x(L-i)] \quad (17)$$

given that $b(i) = -b(L-i)$, and $x(i)$ being the input signal. An example of the implementation of this area-efficient architecture is shown if figure 8 for the particular case of $L = 59$

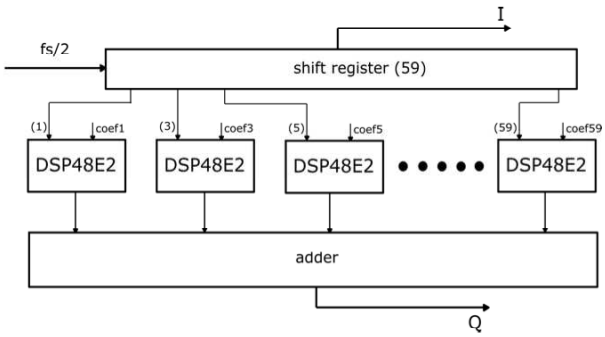


Fig. 7: Baseline FIR architecture.

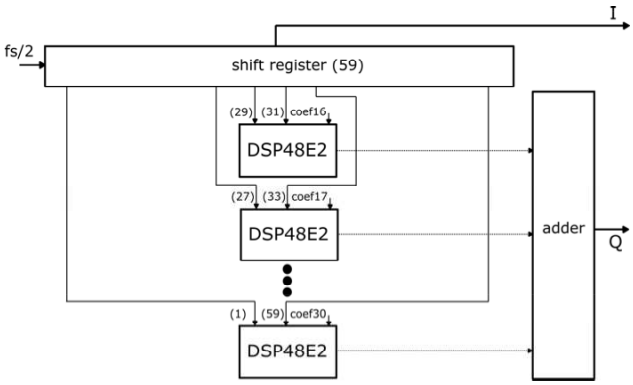


Fig. 8: Optimized baseline FIR architecture.

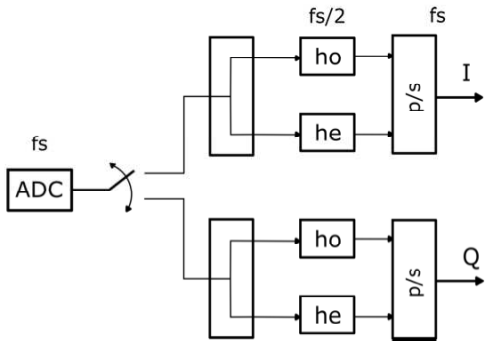


Fig. 9: *Special sampling* architecture.

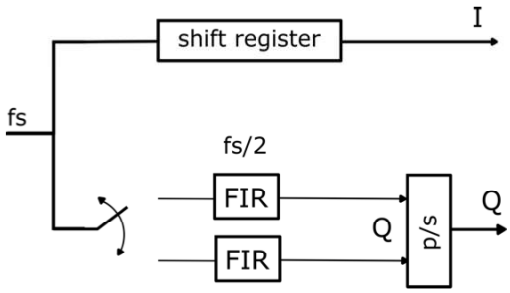


Fig. 10: Proposed *Optimized* architecture.

B. Performance-oriented architectures

As mentioned before, using the *Special Sampling* architecture is a good solution to have the filter working at a slower clock rate and allowing a faster sampling rate. This implementation alternative is based on splitting odd and even sub-components as shown in figure 9. When analyzing the equations of the results provided by the Hilbert filter it can be seen that the odd outputs of the filter contain only terms computed with odd inputs and the even outputs contain only terms coming from even inputs. We use $ho(t)$ and $he(t)$ to name the odd and even terms of $h(t)$. If $h(n)$ has L coefficients, $ho(t)$ and $he(t)$ will contain half of them. The output of I and Q can be subdivided into two channels and each channel requires a low pass filter, as depicted in figure 9. A total of four FIR filters are required. Clearly, the price we pay with this implementation is area and a higher complexity, becoming therefore more complicated to optimize or speed up the implementation.

The *Optimized architecture* we propose, shown in figure 10, takes advantage of using FIR filters, which have no dependency on past samples. Therefore we can perform computations related to two clock cycles at once. And this can be carried out with only one duplication of the FIRs (instead of the four replicas required by the *Special Sampling* architecture). The FPGA implementation requires therefore double number of DSPs than the baseline. The shift register holds all samples and feeds the first FIR filter with odd samples. The output combines both results to generate the output Q. The I branch is obtained in the same way it is performed in the baseline architecture, after the shift register.

V. Experimental results

We have designed a Hilbert filter with 59 coefficients following a FIR architecture. Even coefficients are zero, while the odd coefficients are non-zero. This filter is used to generate the analytic signal required by an automatic modulation classifier for electronic warfare applications [11].

The straight-forward implementation of this filter requires 30 DSPs, one for each multiplication of each non-zero coefficient, and a shift register, which introduces the input sample in each clock cycle and has a length of 59 samples. The first position of the shift register is occupied by the most recent sample. This is entered in the first DSP to multiply it by the first coefficient. The second space is not entered in any DSP because the second coefficient is zero, thus saving FPGA resources. The third space is multiplied by the third filter coefficient (second non-null). This sequence continues until the 59 coefficients are completed. To obtain Q, the output of all DSPs is added. For branch I, nothing new must be implemented, since the necessary delay is already implemented in the shift register, obtaining I directly from its 30th position.

The proposed architecture requires a duplication of the number of DSPs and the shift register must be extended to 60 samples in length. Now the samples enter two by two. This

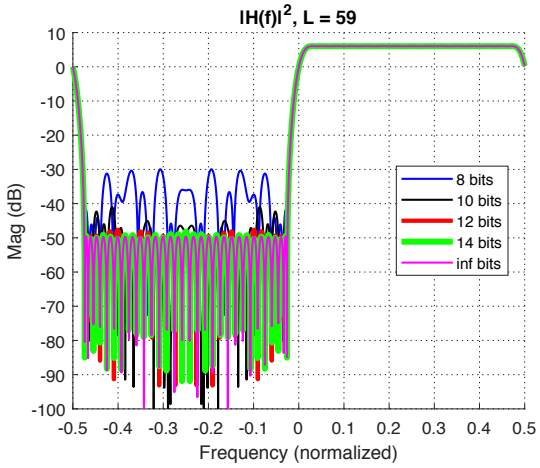


Fig. 11: Frequency response of the FIR filter for different bit widths.

way the odd samples (1,3,5,7,...) will be in the even positions and the even samples (2,4,6,8,...) in the odd ones. We need 30 DSPs to multiply the odd gaps of the shift register, and 30 DSPs to multiply the even gaps, thus getting two outputs at the same clock cycle.

By duplicating the number of filters we can reduce the frequency by half. This can be done as many times as necessary to adapt the filter to the entry frequency, allowing a lower working frequency at the FPGA. The increase in resources regarding FFs and LUTs is not hight, and it is related to the extension of the shift register to hold as many samples as new filters have been added. However, the increase in DSPs is critical, and since it is a scarce resource it is necessary to find a balance between resources used and working frequency. A higher FPGA frequency will require a better optimization of critical paths through pipeline and duplication of resources.

An important aspect to consider is the frequency response of the filter for different bit widths (shown in Figure 11). From the results shown there it is clear that 14 bits are enough to keep the rejection band of the filter close to the theoretical response (*inf* in the figure).

Table II presents the utilization of resources for the alternatives under analysis for this 59-stage Hilbert filter, with 14-bit widths in an FPGA Virtex Ultrascale XCVU13P. The implementation can run at a clock frequency of 480 MHz (post place & route).

A. Implementation details

In terms of implementation, the shift register has been designed by means of an array of vectors, whose length can be varied depending on the desired band rejection. Each shift register position corresponds to the entry of at least one DSP. For a working case with four filters each position corresponds to the entry of two DSPs and so on.

TABLE II: FPGA resources used by the different Hilbert alternatives.

	Baseline	Special Sampling	Optimized Architecture
LUT	871	1771	973
FF	917	943	874
DSP	15	104	30
# Filters	1	4	2

To save DSPs their outputs are added by means of combinational logic. Finally, to avoid timing issues pipelining is used, taking five clock cycles to add the outputs of each filter.

The DSP used is the model DSP48E2 of Xilinx. We use three of the four available data entries. Two of them correspond to the pre-adder with the samples that follow Eq. 17. The pre-adder output feeds the multiplier as well as the third data entry. For a better functioning of the system all internal registers of the DSP have been activated, causing a delay of four clock cycles in the corresponding operations.

VI. Conclusions

In this paper, the implementation of Hilbert filters has been discussed to be used as generators of analytic signals. This kind of signals is required by many different applications (digital demodulators, PLLs, etc.). Different alternatives to the implementation of these filters have been analyzed in depth looking for a high performance implementation on FPGAs. A new approximation to implement Hilbert filters has been proposed that allows to double the sampling frequency of the system with a reasonable area cost. Experimental results have shown the validity of the approach.

References

[1] M. Garrido, J. Grajal, M. Sanchez, and O. Gustafsson, "Pipelined radix-2^k feedforward fft architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 1, pp. 23–32, 2013.

[2] S. An and C. Wang, "Recursive algorithm, architectures and fpga implementation of the two-dimensional discrete cosine transform," *IET Image Processing*, vol. 2, no. 6, pp. 286–294, 2008.

[3] J. B. Tsui, *Digital techniques for wideband receivers*. SciTech Publishing, 2004, vol. 2.

[4] J. E. Volder, "The cordic trigonometric computing technique," *IRE Transactions on electronic computers*, no. 3, pp. 330–334, 1959.

[5] M. Kumm, H. Klingbeil, and P. Zipf, "An fpga-based linear all-digital phase-locked loop," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 9, pp. 2487–2497, Sept 2010.

[6] P. Lamo, F. Lpez, A. Pigazo, and F. J. Azcondo, "An efficient fpga implementation of a quadrature signal-generation subsystem in srf plls in single-phase pfcs," *IEEE Transactions on Power Electronics*, vol. 32, no. 5, pp. 3959–3969, May 2017.

[7] M. Kumm and M. S. Sanjari, "Digital hilbert transformers for fpga-based phase-locked loops," in *Field Programmable Logic and Applications, 2008. FPL 2008. International Conf. on*. IEEE, 2008, pp. 251–256.

[8] A. V. Oppenheim and R. W. Schafer, *Discrete-time signal processing: Pearson new International Edition*. Pearson Higher Ed, 2013.

[9] A. Reilly, G. Frazer, and B. Boashash, "Analytic signal generation-tips and traps," *IEEE Transactions on Signal Processing*, vol. 42, no. 11, pp. 3241–3245, 1994.

[10] L. Marple, "Computing the discrete-time" analytic" signal via fft," *IEEE Transactions on signal processing*, vol. 47, no. 9, pp. 2600–2603, 1999.

[11] V. Iglesias, J. Grajal, P. Royer, M. A. Sanchez, M. Lopez-Vallejo, and O. A. Yeste-Ojeda, "Real-time low-complexity automatic modulation classifier for pulsed radar signals," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 51, no. 1, pp. 108–126, 2015.