Netherlands organization for applied scientific research

FEL

TNO Physics and Electronics Laboratory

FEL-89-C226

4

Alternative solutions for Hilbert filtering in a pulse compression radar

Netherlands
organization for
applied scientific
research

TNO-report

TNO Physics and Electronics
Laboratory

P.O. Box 96864
2509 JG The Hague
Oude Waalsdorperweg 63
The Hague, The Netherlands

Phone +31 70 26 42 21

report no. FEL-89-C226

copy no. 4

title

Alternative solutions for Hilbert filtering in a pulse compression radar

author(s): M.P.G. Otten

classification

title : -

abstract : -

report : -

no. of copies : 20

no. of pages : 31

appendices : 2

date : August 1989

report no. : FEL-89-C226
title : Alternative solutions for Hilbert filtering in a pulse compression radar

author(s) : M.P.G. Otten
institute : TNO Physics and Electronics Laboratory

date : August 1989

no. in pow '89 : 715.3

ABSTRACT (unclassified)

Digital radar signal processing is usually done on sampled data in a complex format. This complex form can be obtained by using a Hilbert transformer on real data. Very often, the processing stage following the conversion to complex data is pulse compression. In this case, the operations normally performed by the Hilbert filter can be incorporated in the pulse compression. Three methods for doing this have been devised, and their relative merits investigated. The first of these results in the most straightforward implementation, the second is more efficient in terms of required hardware, and the third is most suitable for adaptive applications (i.e. applications in which a new reference spectrum has to be determined relatively often). The arithmetical efficiency of all three solutions is about the same, and practically equal to that of pulse compression alone. Moreover, all three solutions can cope with high relative signal bandwidths, close to 100%.

rapport no. : FEL-89-C226
titel : Alternatieven voor Hilbert filtering in een radar met pulscompressie

auteur(s) : M.P.G. Otten
instituut : Fysisch en Elektronisch Laboratorium TNO

datum : augustus 1989
no. in iwp '89 : 715.3

SAMENVATTING (ongerubriceerd)

Digitale radarsignaalverwerking wordt in het algemeen uitgevoerd op data in complexe vorm. Een manier om reële data in complexe om te zetten is gebruik maken van een Hilbert filter. Veelal is de eerstvolgende stap de pulscompressie. In een dergelijk geval kan de Hilbert filtering met de pulscompressie gecombineerd worden. Hiertoe zijn drie methoden bedacht en op hun efficiëntie en toepasbaarheid beoordeeld. Eén methode levert de eenvoudigste implementatie op, een tweede methode is zuiniger in componenten (hardware), en een derde leent zich beter voor adaptieve toepassingen, d.w.z. toepassingen waarby het referentiespectrum relatief vaak opnieuw bepaald moet worden. De rekenkundige efficiëntie van alle drie de methoden is nagenoeg hetzelfde en gelijk aan die van enkel pulscompressie. Bovendien zijn alle drie de methoden geschikt voor bemonsterde signalen met hoge relative bandbreedte, tot dicht bij 100%.

CONTENTS

1 INTRODUCTION

Digital processing of radar signals in which amplitude and phase relations are essential is often most conveniently done when the sampled signals are in complex form. This can be achieved by deriving in-phase (I) and quadrature phase (Q) components from the real signal. Conversion of a modulated signal down to digital I/Q form can be done by using a phase detector for each channel, followed by A/D conversion. In analog implementations, matching of the channels in amplitude and phase can be a problem. Therefore, the Hilbert filter solution is sometimes preferred: the desired signal, on a suitable offset frequency, is digitized at a sampling frequency of at least twice the maximum frequency. This requires only one analog channel, and a single A/D converter. This sampled real signal can be transformed to (complex) I/Q pairs at half this sampling frequency by means of Hilbert filtering. The I-channel is derived directly from the real signal by deleting every other sample. The Q-channel is formed through Hilbert filtering of the real signal, and deleting every other sample. Optimum Hilbert filters can be derived for given approximation criteria of the transfer function. These criteria are amplitude spectrum ripple, and the relative bandwidth over which this is achieved. The filter length increases as the criteria get more stringent. Very often, the transformed signal is meant for further signal processing. The case described here concerns pulse compression - although other filtering applications may also benefit from this approach - by way of Fast Convolution, for high range resolution in radar. In such a case, the Hilbert filtering may sometimes be advantageously combined with the compression of the signal. When this is true, and how it is done, is described in this report.

2 HILBERT TRANSFORMATION AND PULSE COMPRESSION IN THE FREQUENCY DOMAIN

2.1 Basic theory

Suppose we have the real signal s, straight from the single A/D-converter, which is to be converted to the complex signal u, downsampled, and filtered by a matched filter, which is the pulse compressor. The imaginary part of u is given by:

$$\mathrm{Im}(u) = s * h_{ideal} \tag{1}$$

where h_{ideal} is the impulse response of an ideal Hilbert filter, which is infinite, and * denotes linear (aperiodic) convolution. It is well known that the ideal Hilbert filter frequency response is given by:

$$H_{ideal}(\exp(j\Omega)) = -j, \text{ for } 0 \le \Omega < \pi$$
$$H_{ideal}(\exp(j\Omega)) = +j, \text{ for } \pi \le \Omega < 2\pi \tag{2}$$

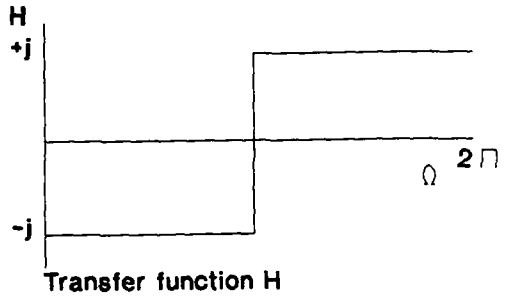
Any practical solution will have to involve an approximation of h_{ideal} by some finite h. The total transformation of s into u, forgetting about the sampling frequency reduction for now, is described by:

$$u = s + j \cdot s * h \overset{def}{=} s * g \tag{3a}$$

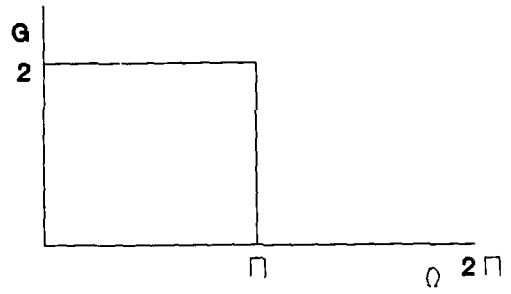
$$\text{where } g(n) = \delta(n) + j \cdot h(n) \tag{3b}$$

For want of a better word, the convolution with g(n) will be called a G-transformation, and its implementation a G-filter. The actual Hilbert filter is thus the non-trivial part of the G-filter.
The real signal s is looked upon as a complex signal, with only the real part non-zero. The Fourier transform of g(n), say G(k), is in fact such

that it simply 'cuts off' the upper half of the spectrum[1] of the pseudo-complex signal s, as is easily derived from (2) and (3), see also figure 2.1.



Transfer function H



Transfer function G

Figure 2.1 Ideal transfer functions H(Ω), and G(Ω)

Since s is actually real, it has a Hermitian spectrum. Removing one half of this corresponds to the desired conversion to a complex signal, and obviously the sampling frequency may then be reduced by a factor of 2. This amounts to deleting one out of every two samples. Figure 2.2 shows how, ideally, the signal spectrum is modified. The dotted part shows the spectral folding after sampling frequency reduction.

---

[1] or the negative half, depending on how one chooses to represent one cycle of the periodic spectrum.

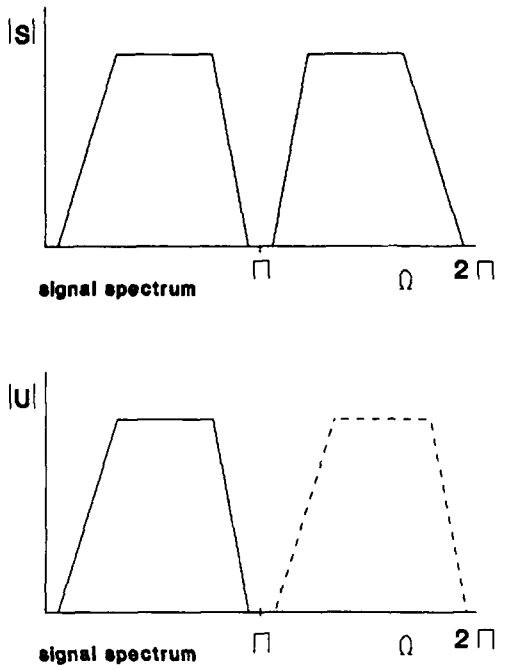


Figure 2.2 Transformation of a real signal to a complex signal

This cutting off, however, is not perfect because of the fact that g(n) has to be finite. Also, in the notation of (2) and (3), g(n) would have to be non-causal, so an actual filter would have a time shifted version of g(n) as its impulse response.
In the case of pulse compression, the signal u(n) is to be convolved with a complex function $r_c(n)$, which is the reversed-in-time complex conjugate of the complex signal waveform to be detected. The digitized frequency offset waveform corresponds to a real sequence r(n). The complex reference $r_c(n)$ is obtained by using the G-filter on the real reference as well.
For reasonably long $r_c(n)$, an efficient way of doing the convolution is by multiplication in the frequency domain, making use of FFT's for the

transformations, so-called Fast Convolution. We will not elaborate on that now, since it is a well established, and well documented procedure. Figure 2.3 shows a block diagram of the 'conventional' implementation of Hilbert filtering followed by pulse compression. The downsampling is indicated by the symbol R↓ (rate reduction). Thin lines indicate real data paths, bold lines complex data paths.

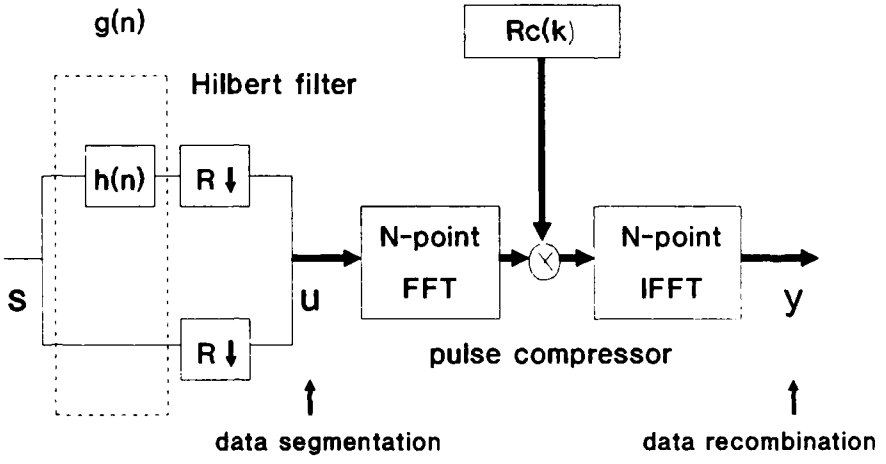


Figure 2.3 Hilbert filtering followed by pulse compression

So in general, the desired signal y(n) is:

$$y = (s*g) * (r*g) = s * (r*g*g) \qquad (4)$$

Now we can do Fast Convolution, with r*g*g (call this r_g) as the new convolution function, without explicitly forming u by G-filtering. This is the basic idea of the proposed approach.

The sequence of operations that have to be carried out is thus:

- formation of the reference spectrum $R(k) \cdot G(k)^2 = R_g(k)$ (this has to be done only once), where R(k) is the spectrum of the real reference.
- segmentation of the input signal s
- Fourier transformation of each segment
- multiplication in the frequency domain
- inverse transformation
- recombination of segment with previous segments

The segmentation/recombination is done according to the overlap-add or overlap-save method. The next section deals with the choice of FFT-lenghts, the segment lengths, and what approximations may be made to simplify things.

2.2 General implementation considerations

2.2.1 How it is done

The suggested approach seems pretty straightforward; however, there are some snags:

- The sampling frequency reduction: In the conventional approach both s*g and r*g can be downsampled, since the spectrum has been halved. Now, this also goes for r*g*g in (4), which is complex, but not for the real signal s. These two sequences cannot be convolved while they have different sampling frequencies, so that the only solution is to postpone the sampling frequency reduction to a later stage. This can be done most effectively when the multiplication of S(k) and $R(k) \cdot G(k)^2$ has been carried out. By carrying out the inverse transformation on only half this spectrum, discarding the part which is (practically) zero, the desired sampling frequency is automatically obtained.
- The formation of the reference spectrum, R_g: In order to mimic the situation involving an actual Hilbert filter, one would have to design the optimum Hilbert filter, according to accuracy requirements (spectrum ripple, giving rise to paired echos in the time domain), and signal characteristics (the normalised bandwidth). Fourier transformation of

the resulting sequence g(n) gives G(k). But, we know that G(k) approaches a simple window function. Since it is already assumed that the ripple is sufficiently small, why not forget about it altogether and use for G(k):

$$G_w(k) = 1, \quad \text{for } 0 \le k < \tfrac{1}{2}N_{fft} - 1$$
$$G_w(k) = 0, \quad \text{for } \tfrac{1}{2}N_{fft} \le k < N_{fft} - 1 \qquad (5)$$

where N_{fft} is the FFT-length.
With this approximation, R_g is formed by taking the Fourier transform of r(n), and discarding the upper half. Two things have been disregarded now:

1. The time sequence corresponding to $G_w(k)$ is non-causal, otherwise an additional linear phase factor would have been present in $G_w(k)$.
2. While the twice G-filtered signal r_g, would have the length of r plus twice the Hilbert-filter length, the time sequence corresponding to the ideal window function has a length equal to the number of frequency samples, i.e. the FFT-length. By multiplying by R(k), we have in fact determined a circular convolution of r and g, which has therefore also a length equal to the FFT length.

The result of this is that $r_{w,g}$, the inverse transform of $R(k) \cdot G_w(k)$, has a sloping leading edge, occurring at the <u>end</u> of the FFT block, because of circular wrap-around, and non-causality of g_w, and a trailing edge. Figure 2.4 shows this in an exaggerated manner. In this plot, a phase modulated signal was assumed, which has constant amplitude.

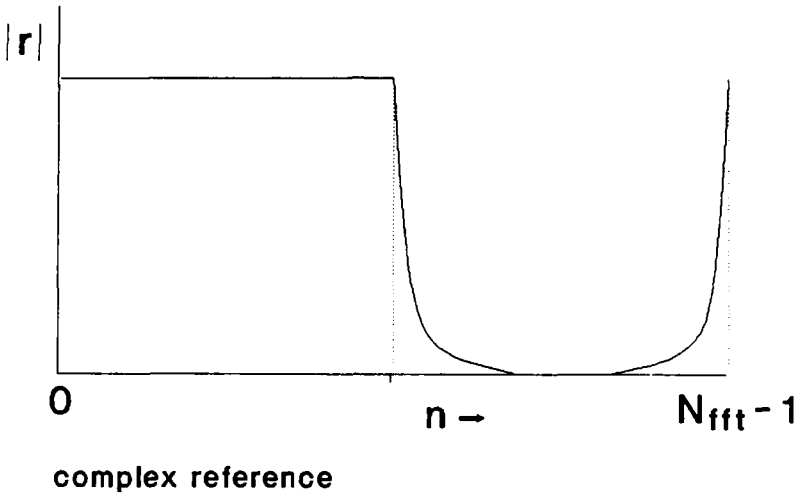


Figure 2.4 Reference sequence after windowing in the frequency domain

In an implementation using an actual Hilbert filter, there also occur leading and trailing edges when the complex reference is formed. It is common practice, however, to cut the excess length off, since this part is not essential for proper compression of the modulated pulses: the effect is only noticeable in the lower sidelobes of a compressed pulse. A similar approach can be adopted in the implementation considered here, although the situation is not exactly the same. Because of the periodicity of the convolution of r and g_w, the trailing and leading edges in fact extend over the whole data block, thus also over the part we must retain for compression. However, this part of the edges is usually so small, that cutting off everything outside the 'main' part of the transformed replica, yields a very good approximation of the desired complex reference function. To summarise, the complex reference could be formed by the following steps:

- append zeros to the real reference till the necessary FFT length is reached
- take the (complex) FFT of the real reference.
- set the upper half of the spectrum equal to zero
- take the inverse FFT of the spectrum, including zeros
- set the second half of the resulting time sequence to zero
- to obtain the spectrum necessary for Fast Convolution: take again the FFT of this sequence, including zeros

This may seem a rather long way to go, just to get the complex reference spectrum; note that the extra inverse and forward FFT are necessary to make the truncation in the time domain possible. It is worthwhile considering whether this is really necessary. If the truncation is not done and the data segment length is not reduced, a wrap-around error is introduced. But, as shown in appendix A.1, this error is usually quite acceptable, so that we may then simply take half the spectrum of the real reference as our reference spectrum. On the other hand, if the application involves only a fixed reference, a slight complication in forming the reference is not a problem; the problem would sooner lie in the fact that, by truncating in the time domain, the spectrum broadens again, so that it is not half zero anymore. In the next paragraph, an implementation will be derived, in which this last drawback is eliminated.

The length of the FFT's can now be determined. The above described procedure yields a reference time sequence having basically the same length as the original real signal replica. Although as far as aliasing is concerned, the sampling frequency could be halved (figure 2.2), we cannot do this at this stage, for reasons mentioned earlier. The Fast Convolution therefore requires FFT lengths, which are twice the real reference length. Obviously, this is also the length required to form the complex reference. As for the inverse FFT's, two cases must be distinguished:

1. If the reference is formed the 'simple' way, half the spectrum is zero by definition, so the inverse FFT can be done with half length, thus yielding the reduced sampling rate at the output.

2. If the reference is formed by truncation in the time domain, the spectrum is not half zero, and should not be discarded. Half length inverse FFT is still possible, if, after each multiplication with the input signal spectrum over the full length, the upper half of the resulting spectrum is added to the lower half (see appendix A.2).

An actual implementation of the proposed scheme, using the half zero reference spectrum, would look like figure 2.5.

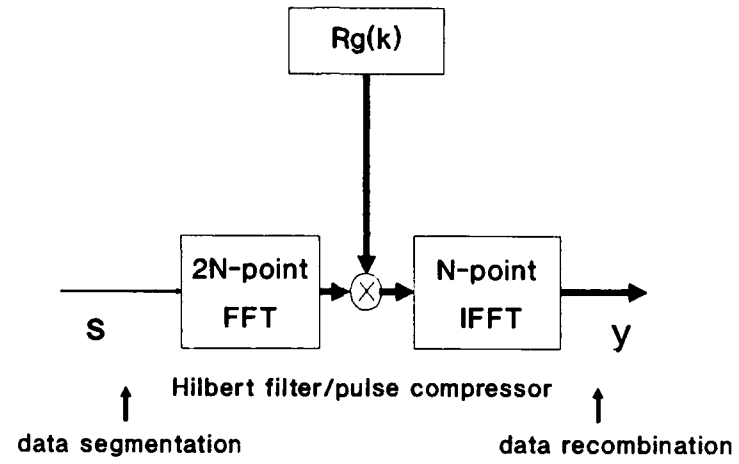


Figure 2.5 Pulse compression with implicit Hilbert transformation. Implementation 1a.

Several other implementations will be discussed; this one we will denote by 'implementation 1a'. Implementation 1b is the variation on 1a, where the reference is truncated in the time domain, and the upper and lower halves of the multiplied spectra are added during convolution. Note that the only reason to do this is to avoid wrap-around error, which is usually not necessary.

2.2.2 FFT-length reduction

If the separate Hilbert filter/pulse compression set-up is compared to the combined set-up, it is clear that the latter approach has got rid of the Hilbert filter, however, the input (forward) FFT length has doubled, compared to the original pulse compressor. This might be a problem, for instance when available hardware components do not allow this FFT-length increase. This double length can be avoided: two methods will be discussed.

1. The FFT length can be reduced by making use of the decomposition, which is the basis of the Fast Fourier Transform itself.
The input FFT length is $N_{fft} = 2 \cdot N_{ref}$, so in formula:

$$S(k) = \sum_{n=0}^{N_{fft}-1} s(n) \cdot \exp[-j(2\pi/N_{fft})n \cdot k] =$$

$$S(k) = \sum_{m=0}^{N_{ref}-1} s(2m) \cdot \exp[-j(2\pi/N_{ref})m \cdot k] +$$

$$\exp[-j\pi k/N_{ref}] \cdot \sum_{m=0}^{N_{ref}-1} s(2m+1) \cdot \exp[-j(2\pi/N_{ref})m \cdot k] \qquad (6)$$

$$0 \leq k \leq N_{fft}-1$$

Now, we are only going to use the values for k up to $\frac{1}{2}N_{fft}-1$, so that (half) the spectrum is given by:

$$S(k) = F_k(s_{even}) + \exp[-j\pi k/N_{ref}] \cdot F_k(s_{odd}) \qquad (7)$$

where F_k is the k^{th} sample of the N_{ref}-point FFT of the indicated sequence: s_{even} and s_{odd} are formed by the even and odd samples of the input data block. The exponential factor implies an extra multiplication in the frequency domain: an efficient way of doing this would be to store a second reference function, namely one which has already been

multiplied by this exponential factor. This way, even and odd batches would have to be alternately multiplied by the 'clean' reference, and 'pre-multiplied' reference respectively. If a provision is made to alternate these references quickly, no extra multipliers are required. Better still, the approach of truncating the reference in the time domain can be used here, without increasing the complexity of the Fast Convolution. If R_g' is the $2N_{ref}$-point spectrum of the truncated complex reference, reference spectra for convolution are given by: (see appendix A.2.)

$$R_e(k) = R_g'(k)+R_g'(N+k) \qquad (8a)$$

$$R_o(k) = [R_g'(k)-R_g'(N+k)]\cdot\exp[-j\pi k/N_{ref}] \qquad (8b)$$

Figure 2.6 shows a block diagram of this set-up. The symbol 'T' denotes a one sample delay.
This set-up will be called implementation 2.

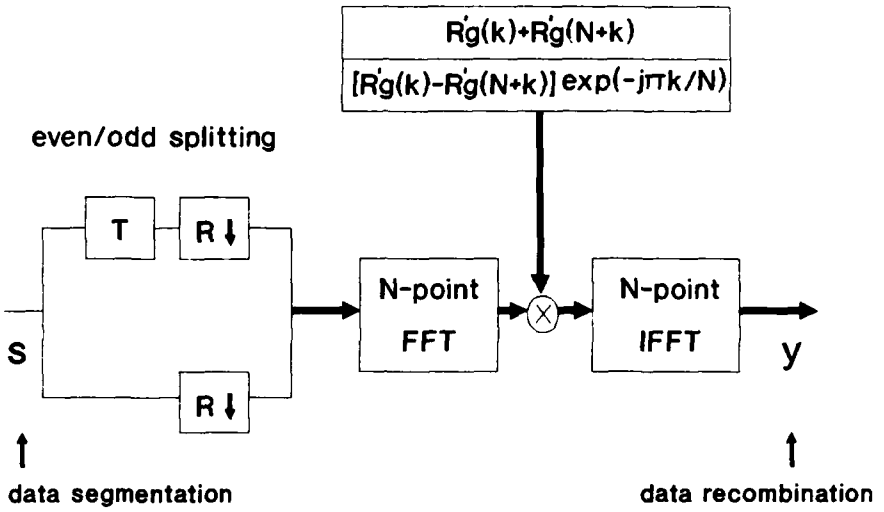


Figure 2.6 FFT-length reduction by even/odd separation.
Implementation 2.

This way, the Hilbert filter has been eliminated at the cost of only a minor modification of the original pulse compressor!
This works very well for a non-adaptive pulse compressor. For an adaptive system, an easier way of obtaining the reference spectra would be preferable. The following approach allows this.
2. The FFT-length can be reduced by cutting the real reference in half, and performing the convolution with both halves separately. Let's call the lower and upper halves (in the time domain) r_L and r_U respectively. We want to compute:

$$y = s*g*r*g = s*(r*g*g) = s*(r_L*g*g) + s*(r_U*g*g) \qquad (9)$$

Now we assume that the approximations that were considered before are still valid for both convolutions with half the reference. This means

that the forward FFT's now have to be of length N_{ref} and the inverse FFT's of length $\frac{1}{2}N_{ref}$.

Suppose the input s is split up into batches denoted by s_m, of $\frac{1}{2}N_{ref}$ samples each, and overlap-add is applied. Then, the convolution result with r_L, say $y_{L,m}$ overlaps with $y_{L,m-1}$ and $y_{L,m+1}$ by $\frac{1}{4}N_{ref}$ complex samples (downsampling taken into account). Added to this should be the convolution result with r_U: since r_U is shifted in time relative to r_L by $\frac{1}{2}N_{ref}$ real samples, the complex output must be shifted by $\frac{1}{4}N_{ref}$ complex samples. In other words, $y_{U,m}$ must be added to $y_{L,m+1}$. Since these two results are seen to coincide in time, it is possible to add them in the frequency domain, thus saving one complex (inverse) FFT. The following procedure per batch then results (the $\frac{1}{2}N_{ref}$-point reference spectra are now $R_{g,L}(k)$ and $R_{g,U}(k)$):

- take the N_{ref}-point FFT of input batch s_m and s_{m+1}, each containing $\frac{1}{2}N_{ref}$ real samples plus zeros. This can be done by <u>one</u> complex FFT.
- multiply half of $S_m(k)$ by $R_{g,U}(k)$, and half of $S_{m+1}(k)$ by $R_{g,L}(k)$
- add both spectra and take the $\frac{1}{2}N_{ref}$-point inverse FFT.

There is a slight complication: per set of output samples, <u>one</u> new batch of real input samples is required. If the complex forward FFT is used to transform two real batches at the same time, which is most efficient, an extra data buffer is needed for the 'second' transformed batch of data. So, for every complex N-point forward FFT, 2 $\frac{1}{2}$N-point complex IFFT's are carried out.

Figure 2.7 shows this: implementation 3.

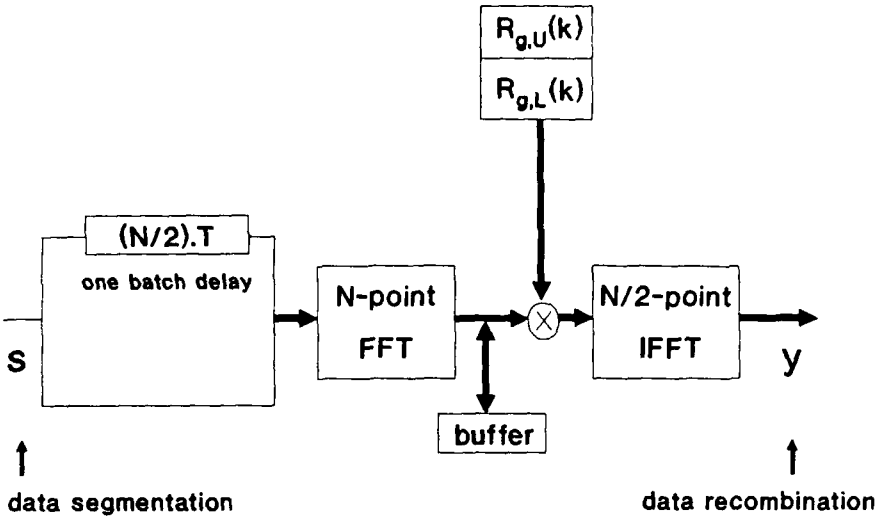


Figure 2.7 FFT-length reduction by splitting the reference.
Implementation 3.

This way, the main thing that has been achieved is the reduction of FFT-lengths, while the reference spectra are obtained in a simple manner: by taking N_{ref}-point FFT's of both halves of the real reference.

2.3 Efficiency

In this section, a comparison of the 4 implementations proposed is made, based mainly on efficiency criteria.
The efficiencies can be evaluated on the basis of arithmetical requirements or (hardware) component requirements. Furthermore, as we have seen, the procedure for forming the reference may be quite simple or relatively complicated, so that we should distinguish between cases which do or do not require repeated adaptation of the reference.
The simplest way of evaluating arithmetical efficiency is to count the number of multiplications required per output sample, assuming then that they are the most time consuming operations.
In a sequential implementation, as in conventional software, or a hardware implementation using only one FFT processor, this number of operations is also an indication of throughput, i.e. the number of

samples processed per unit time. Alternatively, a hardware implementation might have a processing channel, consisting of forward FFT processor, a set of multipliers, and an inverse FFT processor. In this case, the throughput depends on the 'slowest' elements in the chain, most likely the FFT processors, which do most of the work. The 3 implementations discussed do not all have a perfect 'balance' between the forward and inverse FFT's, and for a given maximum FFT-length, they do not produce the same number of processed samples per batch of data, so the throughput in a set-up with separate FFT-processors is not necessarily determined by the total arithmetical efficiency.
Obviously, the 'efficiency' is a very application dependent quantity. Therefore,the comparison will be presented in the form of a listing of 6 items:

1. The number of complex multiplications per processed output sample (arithmetical efficiency)
2. Relative throughput in a set-up with separate forward and inverse FFT processors. This should be interpreted as follows:
   The number of processed samples per batch is determined; it is assumed that the actual throughput (data rate) is determined by the slowest element, i.e. the longest FFT. The data rate thus achieved is divided by the throughput of a conventional pulse compressor, operating on equivalent complex data. So, a relative throughput of 1 means that, for a given FFT-speed, the data rate is the same as for an ordinary pulse compressor.
3. The FFT-lengths. The required lenghts of the forward and inverse FFT's are given: these are probably most of interest in hardware implementations.
4. Adaptivity. It is simply stated, whether the implementation is suitable for adaptive pulse compression or not. If the formation of the reference spectra requires extra FFT's or multipliers, it is considered unsuitable for adaptive applications (which does not mean that the reference cannot be changed!)

5. Wrap-around error. Although this is not an efficiency criterion, it is listed because the implementations do differ in this respect. It can be shown (appendix A.1) that in the most commonly encountered circumstances, the wrap-around error is negligible, and therefore of no concern. In the most general circumstances, however, it should be verified, that for a particular application the error may be neglected. If not, the choice of implementation is limited.
6. This item contains any complementary remark particular to the implementation.

Since conventional pulse compression by Fast Convolution serves as a reference, let's analyse this first.

We assume that we have a real signal replica of N_{ref} samples. By Hilbert transformation, downsampling and truncation, this is transformed into a $\frac{1}{2}N_{ref}$-point complex reference. So, Fast Convolution can be carried using N_{ref}-point complex FFT's, producing $\frac{1}{2}N_{ref}$ processed samples per batch (relative throughput 1 by definition).

Pulse compression with a $\frac{1}{2}N_{ref}$ complex replica requires, per output sample (the number of complex multiplications in a complex N-point FFT is $(\frac{1}{2}N)\cdot{}^2\log N$, assuming $N=2^n$):

$$\begin{aligned} N_{c,pc} &= (2\cdot\tfrac{1}{2}N_{ref}\cdot{}^2\log N_{ref} + N_{ref})/(\tfrac{1}{2}N_{ref}) \\ &= 2\cdot{}^2\log N_{ref} + 2 \end{aligned} \qquad (10)$$

Table 2.8 now lists the 6 items for plain pulse compression, and the 4 combined pulse compression/Hilbert transformation implementations. For notational brevity, N_{ref} is denoted simply by N, and $^2\log$ by log.

Table 2.8 Efficiencies and requirements for combined implementations, relative to conventional pulse compression

Impl.nr	N_c	Relative thruput	FFT-lengths	wrap-around error	adaptive	remarks
pulse compr.	2logN+2	1 (def.)	N,N	none	yes	Hilbert filter required
1a	2logN+3	≈1*	2N,N	small	yes	-
1b	2logN+5	≈1*	2N,N	none	no	-
2	2logN+4	1	N,N	none	no	2 ref.spectra
3	2logN+3	1	N,½N	small	yes	2 ref.spectra + buffer

* If it is assumed that a 2N-point FFT takes a little bit longer than 2 N-point FFT's, the throughput is slightly less than 1 (logN/log2N).

Some additional remarks are in order. Obviously, the whole idea is to get rid of the separate Hilbert filter. The size of such a filter depends on the signal relative bandwidth, for a specified spectrum ripple. It was verified by simulation, that the combined implementations yield good results for bandwidths close to 100%. This is best explained by considering implementation 2, which has no wrap-around error. In order to avoid <u>any</u> error, a sequence g(n) should be designed as follows: the maximum length that could be incorporated in r_g through a $2N_{ref}$-point frequency domain multiplication is $\frac{1}{2}N_{ref}$, since, according to the theory, we should form r*g*g, which then would have length $2N_{ref}$ (-2). So, for N_{ref}=1024, which is a typical number, a 512-point sequence g(n) could be used. So, a 512-point Hilbert filter impulse response could be completely incorporated. From the theory of Hilbert filters it is known that this is quite a long filter, able to cope with high relative

bandwidth, for reasonable ripple criteria. But, as it turns out, we do not need to really use the exact Hilbert filter spectrum: taking a simple rectangular spectral window provides high relative bandwidth, without introducing significant error, even if wrap-around error is counted as well.

3 AN EXAMPLE CASE

As an illustration of a practical case, and as a check on the validity of some of the approximations, an example will be presented, described by the following parameters:

- waveform: linear FM, 31 MHz sweep (symmetrical around offset), 12.6 μs pulse length
- offset frequency before digitization: 22 MHz
- sampling frequency: 80 MHz

The number of real samples in a pulse is 1024. The conventional implementation could be realized by, say, a 31-point Hilbert filter, and Fast Convolution pulse compression using 1024-point FFT's, and a 1024-point reference spectrum. The combined implementation can be realized by a Fast Convolution using either:

1. 2048-point forward FFT, 1024- or 2048-point reference spectrum, and 1024-point inverse FFT (impl. 1a/b)
2. only 1024-point FFT's, and two 1024-point reference spectra
3. 1024-point forward FFT, two 512-point reference spectra, and 512-point inverse FFT

The following plots show some examples of a compressed pulse, where various approximations have been incorporated.

The first plot represents the Hilbert filter approach, although it was obtained by frequency domain operations. The following steps are made:[2]

- A 31-point Hilbert filter impulse response (8 coefficients) is multiplied by 'j' and one sample with value '1' (delta function) is added as the real part of the middle sample of the 31-point sequence. Thus the G-filter is formed.
- g(n) is Fourier transformed (2048), squared, and multiplied by the Fourier transform of r(n), the real replica. The result is the Fourier transform of r*g*g.
- a single input pulse is convolved via the frequency domain with that sequence, with data segment length of 964, which is the FFT-length minus the length of r*g*g. The inverse FFT are also 2048-points so that the result still has the full sampling frequency of 80 MHz. Although the sampling frequency is reduced in a real implementation, this was not done here, because that would obscure the actual sidelobes.

Figure 3.1a shows the whole compressed pulse, 2048 samples, while figure 3.1b shows an enlargement of the peak. Note that even though the compressed signal still contains the offset frequency of 22 MHz, this is not visible since the absolute value of each sample is taken. As expected, the sidelobe level is about -14 dB.

[2] All simulations used overlapp-add to segment the input

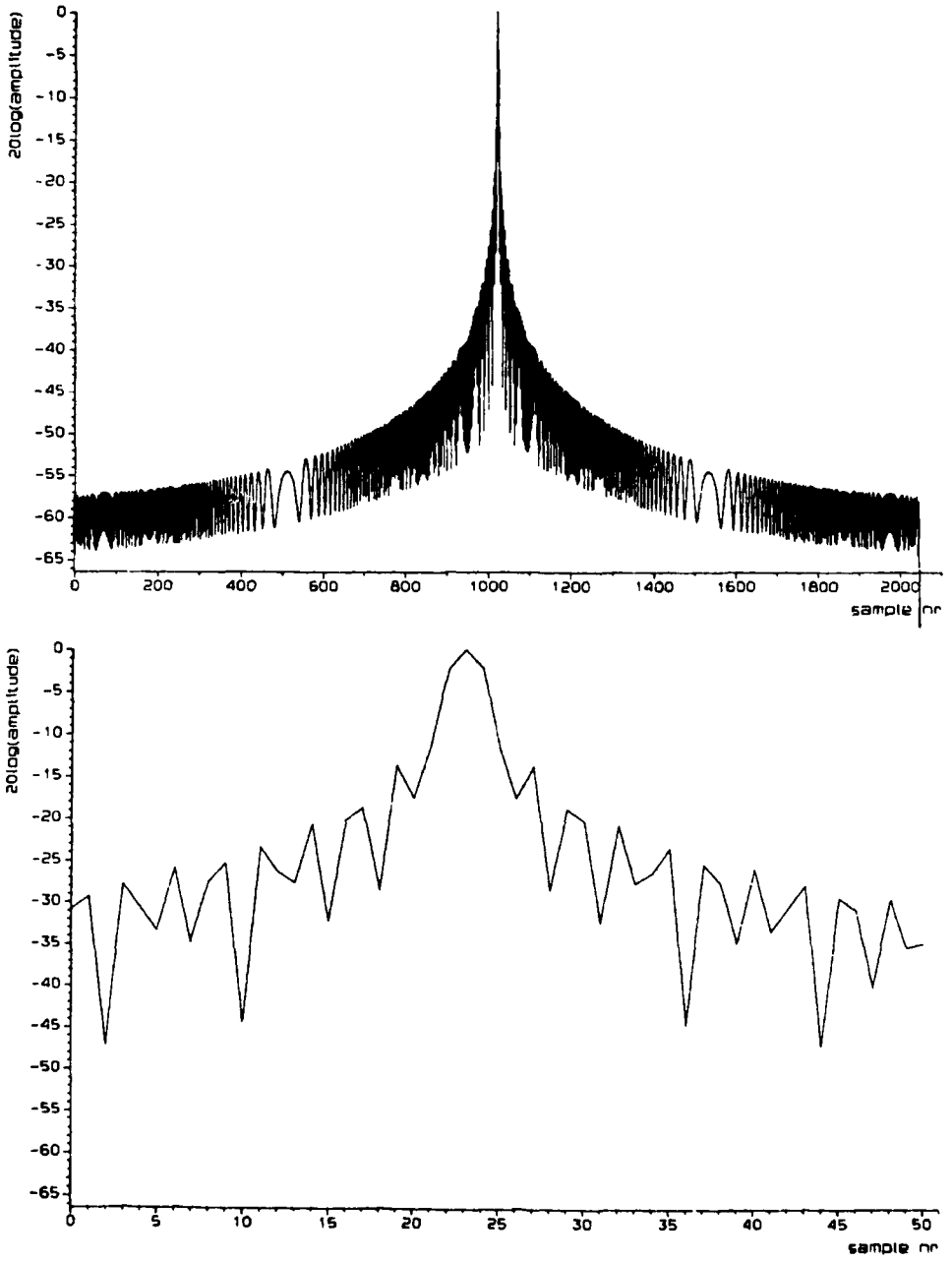


Figure 3.1 Pulse compression, including Hilbert filter characteristics

Top: unweighted, full compressed pulse
Bottom: enlargement

Figure 3.2 shows the result, when a hanning weighting is applied to the reference. The dotted lines indicate what the signal would look like after downsampling, so as to aid comparison with some of the following plots.

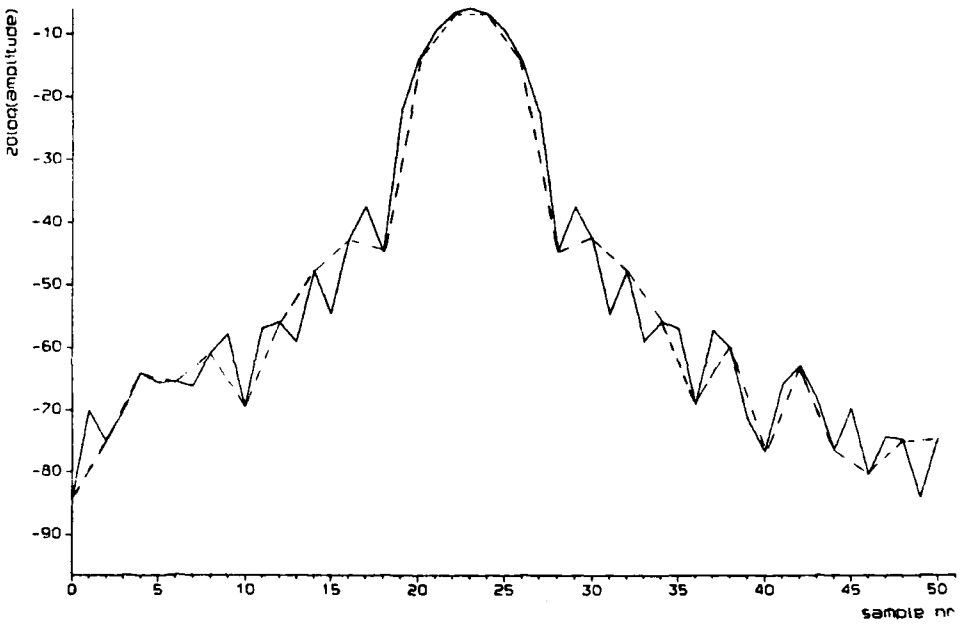


Figure 3.2 Pulse compression, including Hilbert filter characteristics and hanning weighting

Figure 3.3 shows the outcome of implementation 1a (see previous paragraph), without and with hanning weighting respectively. Some of the modulation-like variations in Figure 3.3a are due to the reduced sampling frequency. Also, an enlargement of the unweighted case would not show the sidelobes: to reveal these, one run was done with 2N-point inverse FFT, including all the zero spectrum samples, see figure 3.4.

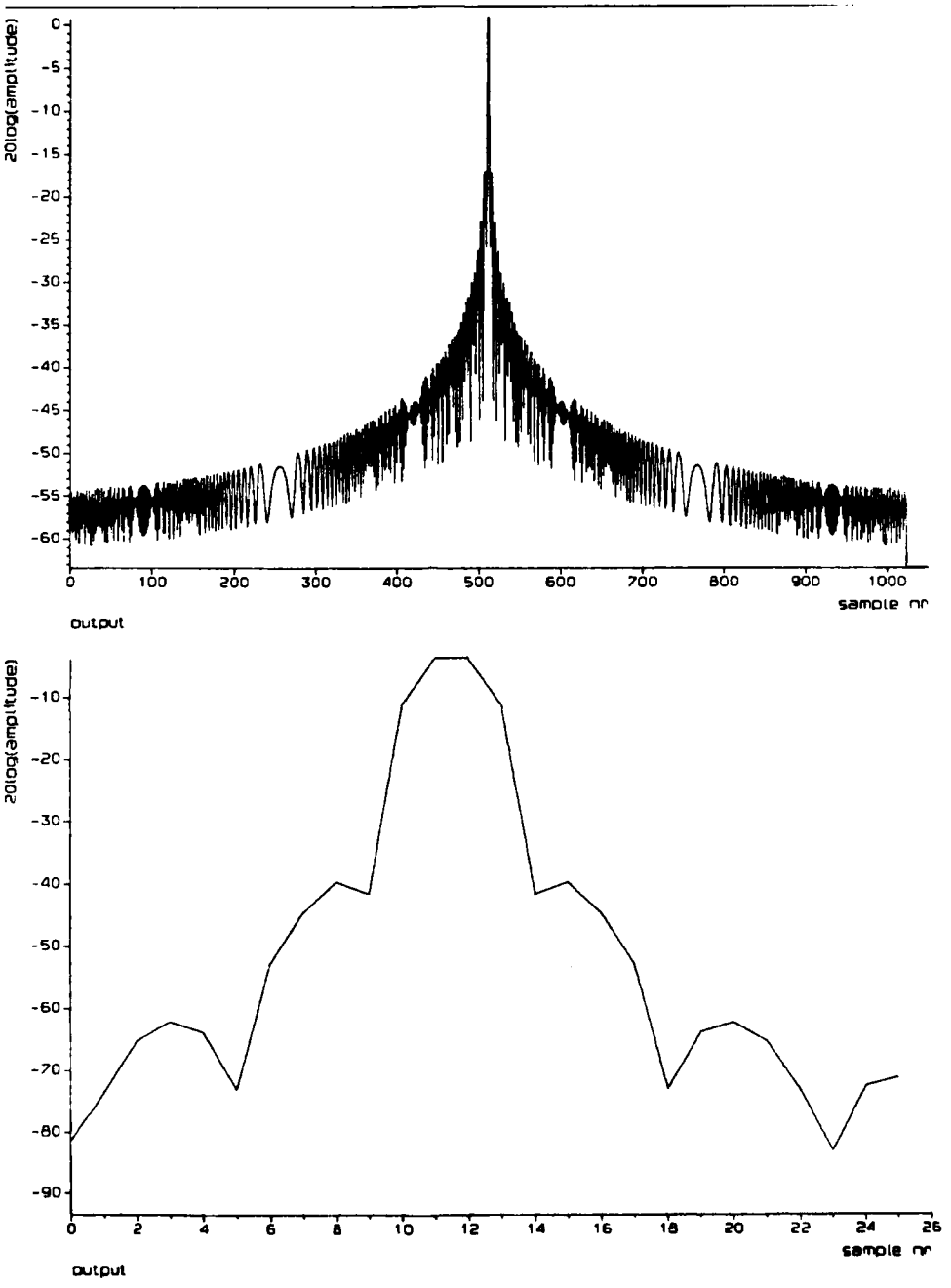


Figure 3.3 Result of implementation 1a

Top: unweighted, full compressed pulse
Bottom: hanning weighting: enlargement

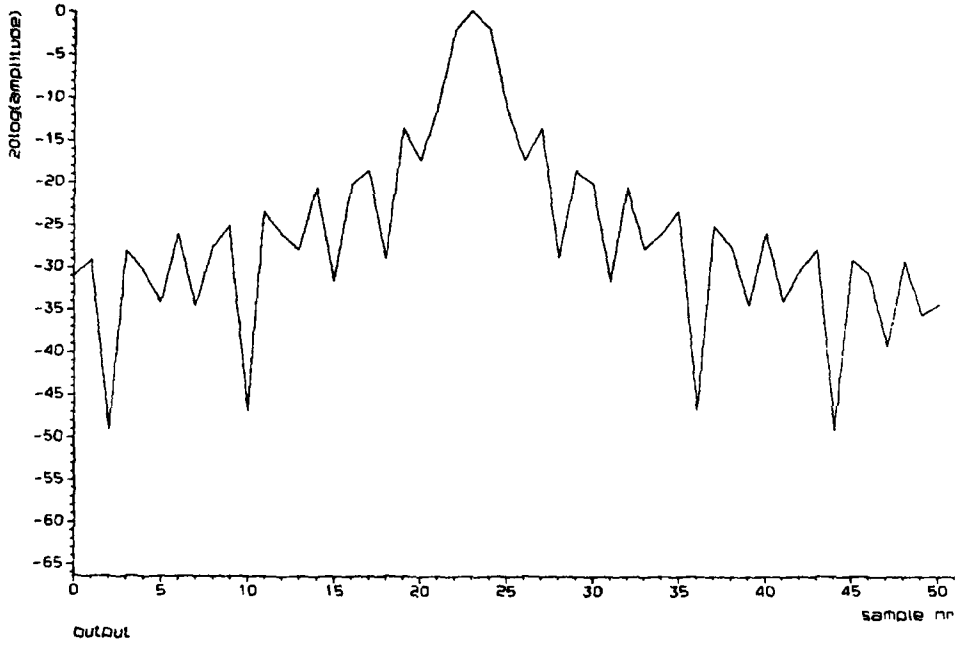


Figure 3.4 Result of an implementation based on 1a,
without downsampling
unweighted: enlargement

The figures 3.1 to 3.4 demonstrate that taking a rectangular spectral window in stead of a Hilbert filter-derived spectrum, and accepting some wrap-around error do not degrade the result.
Not all implementations will be demonstrated here, since all the plots turn out the same. One additional example, to be more convincing, is shown in figure 3.5. It is (part of) the result of implementation 2, the one which separates even and odd input samples. Again hanning weighting was applied: the result is indistinguishable from that of other implementations.

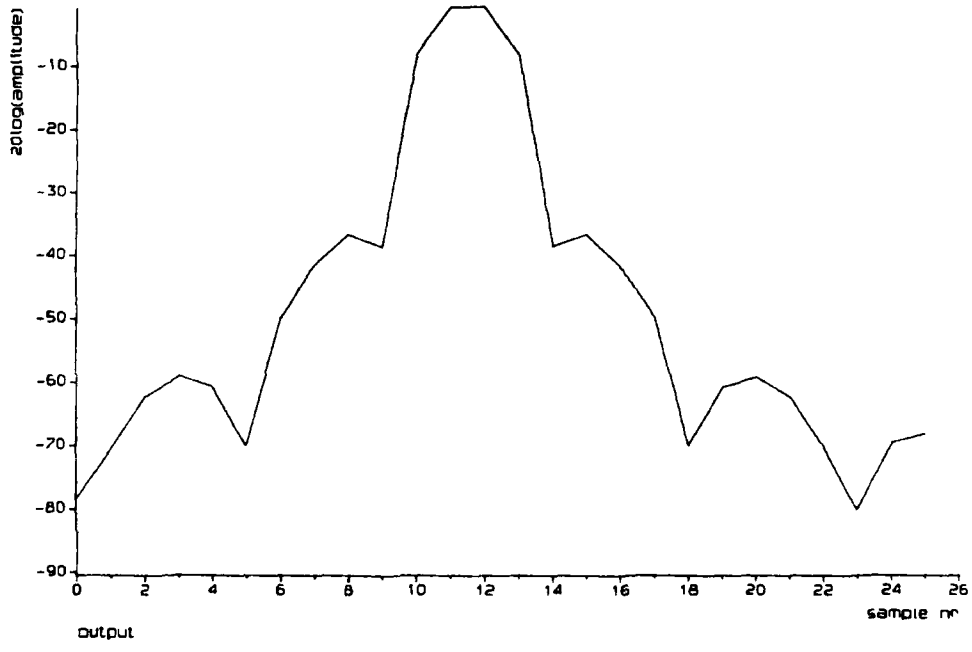


Figure 3.5 Result of implementation 2
hanning weighting: enlargement

4 CONCLUSIONS

Hilbert filtering (actually G-filtering) and pulse compression can be combined into one module, by modifying the pulse compressor. Whether it is advantageous to do so, depends on the type of implementation (hardware/software) and the application (e.g. adaptive/non-adaptive pulse compression).

Three implementations have been devised and tested by simulation. All of these yield good results, even for very high relative bandwidths (well over 95%). The price to be paid for the fact that Hilbert filtering is eliminated is different for each of these implementations. If the original pulse compressor is taken as a reference, the following can be said:

- Implementation 1 needs a double length forward FFT (not inverse), nevertheless, the arithmetical efficiency is the same.
- Implementation 2 is very good for applications with one (or a set of) fixed reference(s), but not very well suited for adaptive applications.
- Implementation 3 needs some extra buffering of data in the frequency domain, but is well suited for adaptive schemes.

In software, the second implementation is to be preferred, since it is exact (no wrap-around error), and therefore guarantees good performance for any kind of signal. For hardware, all three are candidates. The choice then depends on available components (FFT-lengths), required adaptivity, and in some (but almost pathological) cases on signal parameters.

Ir. G.A. van der Spek
(Groupleader)

Ir. M.P.G. Otten
(Author)

ERROR ANALYSIS

The effect of wrap-around error is considered. This will be done on the basis of the example parameters given in chapter 3.
By using a simple rectangular window in the frequency domain, the reference sequence has acquired leading and trailing edges, which give rise to wrap-around error during the Fast Convolution process.[1]
Moreover, the leading edge appears at the end of the data block representing the reference, since the spectral window has a non-causal counterpart in the time domain.
So, one question is whether this reference should be shifted circularly, so as to at least have the leading edge 'were it belongs'. This appears to be unwise: in the first place, the wrap around error is very small anyway. By shifting the reference, a bit of compressed signal, may shift circularly from the end of a batch to the beginning, thus giving rise to a small false peak. This causes larger errors than the wrap-around error itself.
A rough idea of the wrap-around error may be obtained as follows:
For the given signal bandwidth, a 31-point sequence g(n) would be a good approximation to the impulse response that would effectively cut half the spectrum off. If this filter were actually used (twice to form r*g*g), the replica would become longer by 60 samples. By reversing the argument, we can say that, if R(k) is simply cut in the frequency domain, the part of the corresponding $r_g(n)$ which is <u>significant for our purposes</u>, is about 60 samples longer that the original r(n). If Fast Convolution (unweighted) is now performed with the original data segment length of 1024, there will be an error contribution, in each output sample, due to wrap around, formed by some 60 integrated samples. At the peak of the compressed signal, 1024 samples are integrated, all in-phase, so the relative error contribution should not be more than 20log(60/1024) = -24.6 dB. Remember that the excess sample are in fact

[1] It is assumed that the input data segment length is <u>not</u> reduced by an amount equal to the length of the edges

the leading and trailing edges of r_g: let's assume, for the sake of simplicity, that these samples exhibit a linear rise and fall off. Then halving the contribution (in amplitude) seems more realistic. This way we arrive at an expected maximum error which is at least 30 dB down, with respect to the peak of the compressed pulse.

The sum of the absolute values of all samples outside the original reference length was determined numerically, using the above parameters (unweighted reference). A value of -35 dB was obtained. In reality, there is never a perfect match between these erroneous samples and the input signal, so that actual error must always be less than this value. An actual error plot was obtained by performing a Fast Convolution, with only the incorrect part of the reference. Note that, in spite of wrap-around, superposition still applies, so that subtracting the 'correct' part of the reference will indeed reveal only the error in the output. Figure A.1 shows that the actual errors are in fact below -50 dB, with respect to the peak of the compressed pulse. The overall shape of this sequence remains the same when the input signal is shifted in time: The largest errors occur near the pulse peak location, since it is here were the match between error samples and the input signal is 'best'. So, not only are the errors very small, they are the largest where the output signal itself is the largest, so that the errors will go completely unnoticed.

In case of a weighted reference, the error will be even less, since then the edges of r(n) have already been reduced.

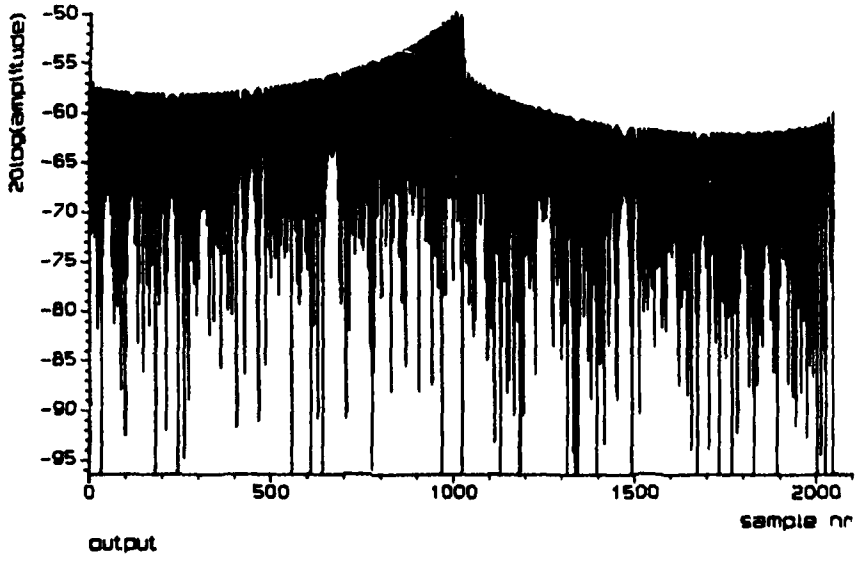


Figure A.1 Wrap-around error plot

FORMATION OF THE REFERENCE

In most pulse compression applications, wrap-around error is negligible. However, to avoid wrap-around effects under all circumstances, i.e. with any conceivable code, weighting etc., the sequence $r_{w,g}$ should be cut in the time domain to its original length. However, since this causes spectrum broadening, we can no longer discard half the spectrum, and so the idea of using half length FFT's on the inverse transforms seems to be lost. Fortunately, we can still do this, by adjusting the reference function accordingly. Assume we have a spectrum Y(k) of length 2N. Now, of the inverse transform, we only want half the samples, e.g. the even ones. The IFFT is given by:

$$y(n) = \sum_{k=0}^{2N-1} Y(k) \cdot \exp[j(2\pi/2N)n \cdot k] = \qquad (B.1)$$

$$0 \leq n \leq 2N-1$$

and so the even samples of y(n) by:

$$y(2m) = \sum_{k=0}^{2N-1} S(k) \cdot \exp[j(2\pi/N)m \cdot k] =$$

$$\sum_{k=0}^{N} \{S(k)+S(N+k)\} \cdot \exp[j(2\pi/N)m \cdot k] \qquad (B.2)$$

$$0 \leq m \leq N-1$$

So, *N-point FFT's can be used, providing the upper half of the spectrum is added to the lower half on beforehand*. In fact one deliberately introduces aliasing this way.

Now, two cases must be distinguished:

1. The Fast Convolution is carried out using 2N-point forward FFT's. Then a 2N reference spectrum must be stored. During the convolution process, this spectrum is multiplied by 2N signal spectra, and the upper and lower halves can be added prior to an N-point inverse FFT (this corresponds to implementation 1b).
2. The Fast Convolution is carried out using N-point FFT's by splitting the input signal in even and odd samples (s_e and s_o). The 'original' multiplied spectrum is:

$$S_{2N}(k) = R(k) \cdot S(k)$$
$$= R(k) \cdot \{S_e(k) + \exp(-j\pi k/N) \cdot S_o(k)\} \qquad (B.3)$$

The spectra S_e and S_o are N-point spectra so that:

$$S_N(k) = S_{2N}(k) + S_{2N}(N+k) = \{R(k)+R(N+k)\} \cdot S_e(k) +$$
$$\{R(k)-R(N+k)\} \cdot \exp(-j\pi k/N) \cdot S_o(k)$$
$$0 \le k \le N-1$$

This makes it possible to use N-point FFT's all round, by using two N-point reference spectra: one is formed by adding the lower and upper halves of the 2N-point reference spectrum, the other by subtracting them, and multiplying by $\exp(-j\pi k/N)$.

REPORT DOCUMENTATION PAGE (MOD-NL)

1. DEFENSE REPORT NUMBER (MOD-NL)	2. RECIPIENT'S ACCESSION NUMBER	3. PERFORMING ORGANIZATION REPORT NUMBER
TD89-3564		FEL-89-C226
4. PROJECT/TASK/WORK UNIT NO.	**5. CONTRACT NUMBER**	**6. REPORT DATE**
21877-2		AUGUST 1989
7. NUMBER OF PAGES	**8. NUMBER OF REFERENCES**	**9. TYPE OF REPORT AND DATES COVERED**
36		INTERIM REPORT

10. TITLE AND SUBTITLE
ALTERNATIVE SOLUTIONS FOR HILBERT FILTERING IN A PULSE COMPRESSION RADAR.

11. AUTHOR(S)
M.P.G. OTTEN

12. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
PHYSICS AND ELECTRONICS LABORATORY, TNO
P.O. BOX 96864, 2509 JG, THE HAGUE,, THE NETHERLANDS

13. SPONSORING/MONITORING AGENCY NAME(S)
BCRS, P.O. BOX 5023, 2600 GA DELFT, THE NETHERLANDS

14. SUPPLEMENTARY NOTES
TECHNICAL REPORT

15. ABSTRACT (MAXIMUM 200 WORDS, 1044 POSITIONS)
DIGITAL RADAR SIGNAL PROCESSING IS USUALLY DONE ON SAMPLED DATA IN A COMPLEX FORMAT. THIS COMPLEX FORM CAN BE OBTAINED BY USING A HILBERT TRANSFORMER ON REAL DATA. VERY OFTEN, THE PROCESSING STAGE FOLLOWING THE CONVERSION TO COMPLEX DATA IS PULSE COMPRESSION. IN THIS CASE, THE OPERATIONS NORMALLY PERFORMED BY THE HILBERT FILTER CAN BE INCORPORATED IN THE PULSE COMPRESSION. THREE METHODS FOR DOING THIS HAVE BEEN DEVISED, AND THEIR RELATIVE MERITS INVESTIGATED. THE FIRST OF THESE RESULTS IN THE MOST STRAIGHTFORWARD IMPLEMENTATION, THE SECOND IS MORE EFFICIENT IN TERMS OF REQUIRED HARDWARE, AND THE THIRD IS MOST SUITABLE FOR ADAPTIVE APPLICATIONS (I.E. APPLICATIONS IN WHICH A NEW REFERENCE SPECTRUM HAS TO BE DETERMINED RELATIVELY OFTEN). THE ARITHMETICAL EFFICIENCY OF ALL THREE SOLUTIONS IS ABOUT THE SAME, AND PRACTICALLY EQUAL TO THAT OF PULSE COMPRESSION ALONE. MOREOVER, ALL THREE SOLUTIONS CAN COPE WITH HIGH RELATIVE SIGNAL BANDWIDTHS, CLOSE TO 100%.

16. DESCRIPTORS
RADAR
SIGNAL PROCESSING
DIGITAL FILTERS
PULSE COMPRESSION

IDENTIFIERS
RADAR ECHOES

17a. SECURITY CLASSIFICATION (OF REPORT)	17b. SECURITY CLASSIFICATION (OF PAGE)	17C. SECURITY CLASSIFICATION (OF ABSTRACT)
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED

18. DISTRIBUTION/AVAILABILITY STATEMENT	17d. SECURITY CLASSIFICATION (OF TITLES)
UNLIMITED	UNCLASSIFIED