

ASIC Implementation of a Filter Bank Analyzer

based on CSD Code

Christian Rosadini and Martin Hollreiser

September, 1999

TOS-ESM

Microelectronic Section

Control Data and Power Division

ESTEC

European Space Research

and Technology Center

Postbus 299 2200 AG Noordwijk

The Netherlands

ACKNOWLEDGEMENTS

I would like to thank Mr. Richard Creasey, Head of TOS-ES Division, for giving me the opportunity to work in the Division and for providing a peaceful working environment. I would also like to thank people working in TOS-ESM Section for their advice and help offered to me during this period.

It has been a great pleasure to work under Mr. Martin Hollreiser's supervision during six months of training at ESTEC. Head of the Microelectronic Section (TOS-ESM) at ESTEC, Mr. Hollreiser is a very professional person, competent and helpful in any situation; his enthusiasm and knowledgeable suggestions have made my Stage at ESTEC a pleasant and profitable one. I would like to thank him for his continuous cares and availability in every moment.

I would like to thank Mr. Pierangelo Terreni, Engineer and Professor at Electronic Engineering University of Pisa, supervisor of my degree thesis in Italy, for giving me the possibility to carry out my thesis work outside the University.

I would like to express my gratitude to Mr. Luca Fanucci, Engineer and Researcher at CNR-CSMDR, University of Pisa, for his constant availability and kindness, for his attentions and precious advice for the whole period of my Stage at ESTEC.

Thanks to my girlfriend, my brother and my parents for their enthusiasm, support and joy in every moment.

Finally, I would like to thank all friends who stood by me during this period for contributing to make my stay in Holland a pleasant one.

CONTENTS

1. Introduction	7
2. Polyphase Filter Bank Analyzer.....	8
2.1. General Form	8
2.2. Matlab Model	10
2.3. Two-stage Architecture	12
2.4. Parallel Implementation and Optimisation for real FDM signal	13
3. RTL VHDL Model of Filter Bank	15
3.1. DESIGN.....	19
3.1.1. Imaged HB Filter.....	19
3.1.2. Polyphase Filter.....	20
3.1.3. FFT.....	21
3.1.3.1. 56-point Good-Thomas	21
3.1.3.2. 64-point Radix-4.....	27
3.1.3.3. 64-point Radix-8.....	29
3.2. SIMULATION.....	31
3.2.1. NPR.....	32
3.2.2. Scattering Diagram and Eye Pattern Diagram.....	32
3.3. SYNTHESIS.....	34
4. Filter Bank Analyzer with CSD code	38
4.1. C++ Program for general DFT.....	39
4.2. C++ Program for Good-Thomas FFT.....	40
5. RTL VHDL Model of CSD Filter Bank	45
5.1. DESIGN	45
5.1.1. Imaged HB Filter	46
5.1.2. Polyphase Filter	46
5.1.3. Good-Thomas FFT	47
5.2. SIMULATION.....	48
5.2.1. NPR.....	48
5.2.2. Scattering Diagram and Eye Pattern Diagram.....	49
5.3. SYNTHESIS	50
References	52

Appendix A Matlab Models	Removed
Appendix B C++ Programs	Removed
Appendix C VHDL Code for RTL Implementations	Removed

Abbreviations

SSB: Single Sideband Modulation.

FDM: Frequency Division Multiplexing.

DFT: Discrete Fourier Transform.

FFT: Fast Fourier Transform.

$H(e^{j\omega})$ Frequency response of a linear digital system.

SNR: Signal to Noise Ratio.

1. INTRODUCTION.

Digital filter bank and spectrum analysis and synthesis concepts arise in many areas of science and engineering. They are extremely important, for example, in systems for speech analysis, bandwidth compression, radar and sonar processing and spectral parameterisation of signals. Nearly all of these types of systems have as their basis some form of filter bank decomposition or reconstruction of a signal in which the filter bank components occur in a decimated form.

The first part of this book describes how, starting from the general form of a filter bank analyzer, it is possible to obtain a parallel structure, which will be optimised for real FDM-signal. The filter required within the structure will be realised in a two-stage implementation, which allows simplification in terms of number of taps and complexity of the final FFT.

The second part deals with the design, the simulation and the synthesis of the filter bank analyzer by means of Synopsys and Matlab tools. The subject of this work is a 28-real channels filter bank analyzer, which, at the end, requires a 56-points FFT; however 64-points FFT with different algorithms have been developed.

The third part of the book points its attention on a CSD realisation of the filter bank, in order to avoid all the multiplications inside the structure. In this case, only the implementation of 28-real channels filter bank is carried out.

The fourth part reflects the second one, in which now the subject is the implementation of a CSD filter bank analyzer.

2. POLYPHASE FILTER BANK ANALYZER.

2.1.1. General Form.

One important characteristic that distinguishes different classes of filter banks is the manner in which the channel signals are modulated: basically it is possible to have *complex* (or *quadrature*) *modulation*, which is the one covered in this work, or *single-sideband (SSB) modulation*.

Another major difference in filter bank systems results from the manner in which the filters are designed for the channels: we can have *non-overlapping*, *slight overlapping* and *substantial overlapping* of the bands of the filter banks. The one that we deal with is the slight overlapping design.

A third major consideration in the design of filter banks is the rate at which the channel signals are sampled. From literature it is known that if the bandwidth of a channel is ω_{Δ} then, theoretically, the sampling rate can be reduced by a factor of M, where $M \leq \frac{2\pi}{\omega_{\Delta}}$ (for complex modulation). If the

quality applies, the bands are said to be *critically sampled*, while, if not, they are *oversampled*. For example if a FDM signal is made by K uniform and contiguously spaced channels, such that $\omega_{\Delta} = \frac{2\pi}{K}$, the filter bank is critically sampled if $M=K$. In this report we consider the oversampling by

two ($K=2M$) so that the channel signal is sampled at the double of the Nyquist rate.

The general idea of the filter bank analyzer is to shift, by means of a complex modulator, the k-th channel to baseband, to filter it from the near channel with a low pass filter, and then to reduce its sampling rate by a certain factor M:

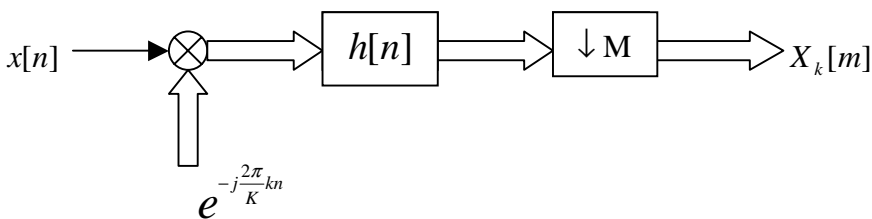


Figure 1 Single channel of a DFT filter bank analyzer.

where $\frac{2\pi k}{K} = \omega_k$ is the center frequency of the k-th channel and $k=0,1,\dots,K-1$.

The channel signal can be expressed in the following form:

$$X_k[m] = \sum_{n=-\infty}^{\infty} h[mM - n]x[n]W_k^{-kn} = \sum_{n=-\infty}^{\infty} h[n]x[mM - n]W_K^{-k(mM-n)}, \quad (2.1)$$

where $W_K = e^{j\frac{2\pi}{K}}$. Now, it is possible to rearrange the equation by naming:

$$n = rM - p, \quad p = 0, 1, \dots, M-1 \quad \text{and} \quad -\infty < r < \infty$$

$$X_k[m] = \sum_{r=-\infty}^{\infty} \sum_{p=0}^{M-1} h[rM - p]x[mM - rM + p]W_K^{-k(mM-rM+p)} \quad (2.2)$$

For a first analysis of the structure, we consider the critically oversampled filter bank, that is $M=K$. In this case, $W_K^{-kM(m-r)} = 1$.

Now, by using

$$\overline{h}_p[r] = h[rM - p] \quad (\text{Counterclockwise})$$

$$x_p[r] = x[rM + p] \quad (\text{Clockwise})$$

we obtain the final structure

$$X_k[m] = \sum_{p=0}^{M-1} \left(\sum_{r=-\infty}^{\infty} \overline{h}_p[r] x_p[m-r] \right) W_K^{-kp} = \text{DFT} \{ \overline{h}_p[m] \otimes x_p[m] \} \quad (2.3)$$

This form of the k-th channel leads to the overall structure

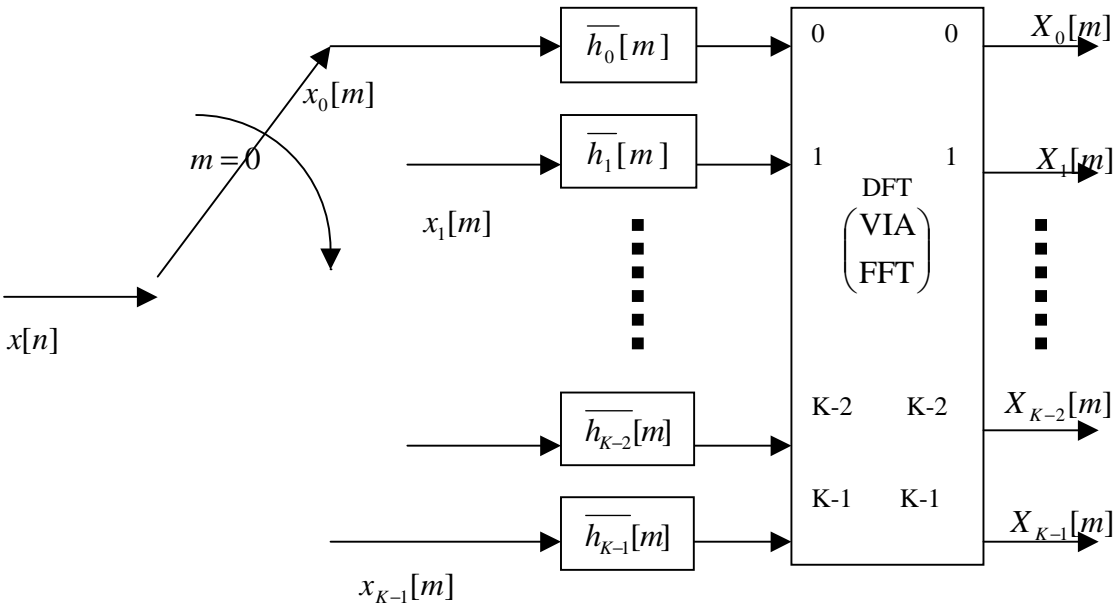


Figure 2 Polyphase structure for the k-th channel of a DFT filter bank analyzer.

2.2. Matlab Model.

In the previous paragraph the polyphase structure for a critically sampled filter bank has been introduced. As the subject of this work is an oversampled-by-two filter bank, the equations and a corresponding floating-point model in Matlab were developed.

By starting again from Eq. (2.1), we can split n in the following form:

$$n = rK + p, \quad p = 0, 1, \dots, K-1$$

$$X_k[m] = \sum_{p=0}^{K-1} \sum_{r=-\infty}^{\infty} h[mM - rK - p] x[rK + p] W_K^{-kp} \quad (2.4)$$

Now, let us consider an integer ratio between K and M , that is $K=M \cdot I$. With the same notation of the previous section, we have:

$$X_k[m] = \sum_{p=0}^{K-1} \left(\sum_{r=-\infty}^{\infty} \overline{h_p}[m - rI] x_p[r] \right) W_K^{-kp} \quad (2.5)$$

It is possible to recognise in this form the interpolator by a factor of I , followed by its filter:

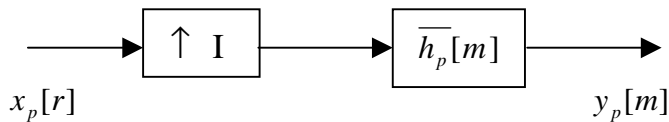


Figure 3 Integer Interpolator using the extended polyphase filter $\overline{h_p}[m]$ as interpolation filter.

According to [1], block structures (such as Polyphase FFTs) for filter bank are most efficient. If again the interpolator is implemented using Polyphase Filter, we will have $K \cdot I$ filters, whose length is the one of the reference low-pass filter divided by $K \cdot I$.

The Matlab model implements both the filter bank analyzer and the filter bank synthesizer: it generates an FDM signal, in which the user chooses the type of modulation, the number of symbols to transmit and the slots to fill. The program starts with the generation of random numbers; after that, they are modulated and then filtered by raised cosine filter, for which the users is prompt for the coefficient of *roll-off* (α). At this point, each channel is complex modulated in order to fill the required slot, all the channels are combined and then only the real part is extracted. This is provided as input to the filter bank analyzer-synthesizer. At the end, the Power Spectral Density of either the input or the output is displayed. This Matlab model of the filter bank analyzer/synthesizer has been realised with one stage architecture: the following figures show the input and the output of the system; in this case the number of the channel has been chosen to be 56.

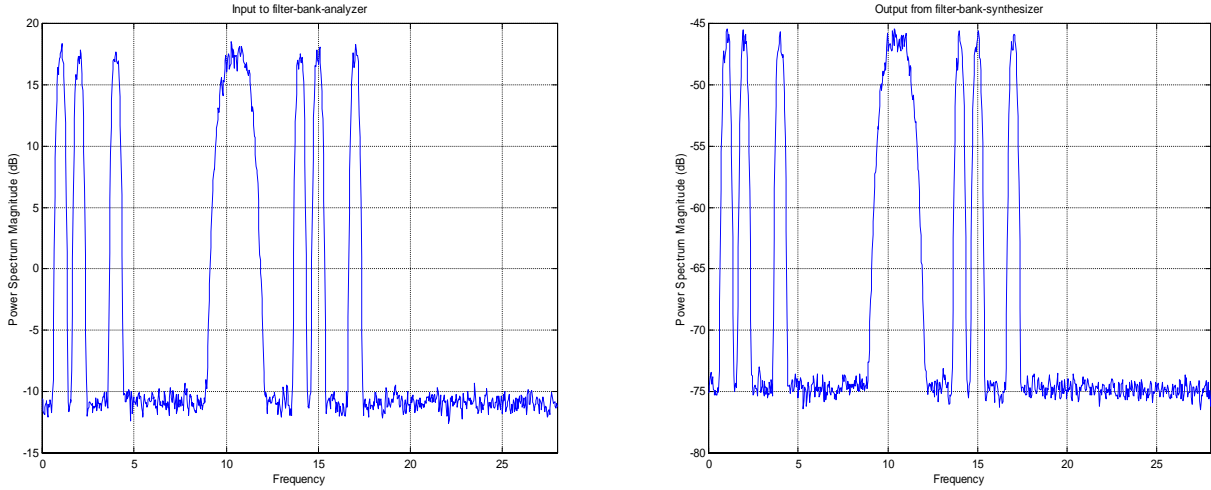


Figure 4 FDM signal to and from analyzer/synthesizer (Matlab version).

The low-pass filter, required for the filter bank, is designed with Matlab: the requirements of the filter were [1]:

$$H(e^{j\omega}) = \begin{cases} 1, & 0 \leq |\omega| \leq \min\left(\frac{\pi}{M}, \frac{\pi}{K}\right) \\ 0, & \text{otherwise} \end{cases} \quad (2.6)$$

both to separate each channel from the adjacent ones and then to prevent aliasing during decimation. The larger K, the more demanding are these requirements, and consequently the number of the taps of the filters can be too large. Concerning the previous example, to design an equiripple filter with K=56 we need at least (sampling frequency normalised at 1 Hz):

- $F_p = 7.6 * 10^{-3}$ (pass-band frequency);
- $F_s = 10.21 * 10^{-3}$ (stop-band frequency);
- $\delta_p = 0.02$ dB (ripple in pass-band);
- $\delta_s = 50$ dB (stop band rejection).

With these values, the required number of taps is very high: from the empirically derived formula [1]

$$N_{tap} = \frac{2}{3} \log_{10}(10\delta_s\delta_p) \frac{f_s}{\Delta_f}$$

where $\Delta_f = F_s - F_p$ and f_s is the sampling frequency (normalised at 1Hz), we obtain about 800 taps.

For this reason a two-stage architecture is chosen for the implementation, which results in a reduced number of taps.

2.3. Two-stage Architecture.

As seen from the previous example, the number of taps required by the filter is too large for an ASIC implementation. This problem can be solved by implementing a two-stage architecture, where the first stage is an *imaged half-band* filter and the second one a *polyphase filter*, with a relaxed transition band. The imaged half-band filter is based on zero filling: this causes imaging, which allows extraction of single channels from an FDM signal. One of the important aspects of a half-band filter is that, starting from a low-pass version, we obtain the high pass version simply by changing the sign of the center tap. With few more additions, we therefore obtain both the lp and the hp part of input signal. Zero filling in combination with a half-band filter allows splitting the FDM signal in two branches, the first one containing the even-stacked channels and the second one containing the odd-stacked channels. It is evident that the interpolation (zero filling) has to be made by $K/2$, if K is the maximum number of slots of the FDM signals.

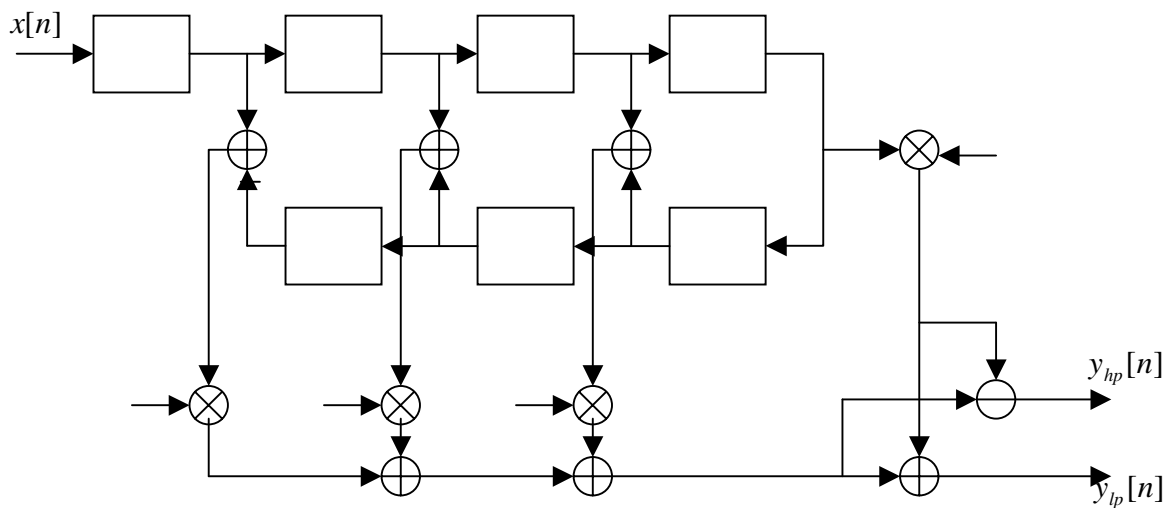


Figure 5 Direct structure for a half band filter

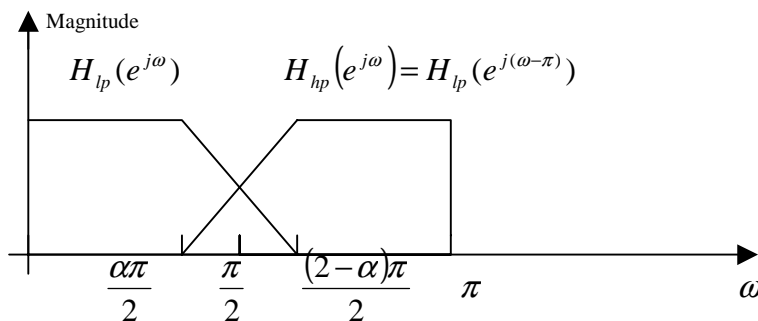


Figure 6 Frequency response of the half band filter (lowpass *lp*, highpass *hp*); α is the occupancy factor.

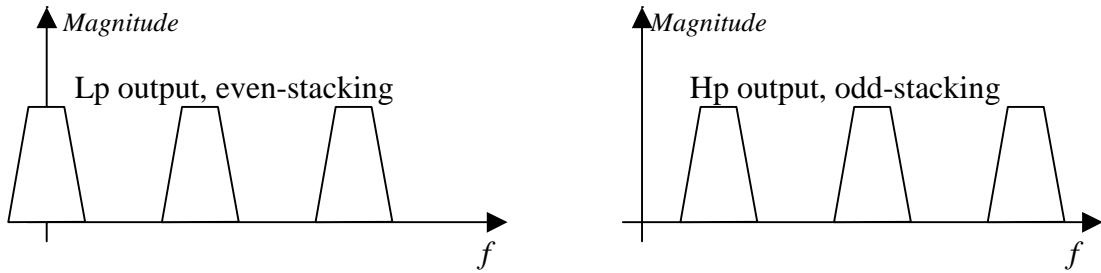


Figure 7 Frequency response of the imaged half-band filter

Now it is clear that the transition band of the reference filter can be wider than the previous version, being the channels more spaced. The general structure is the following:

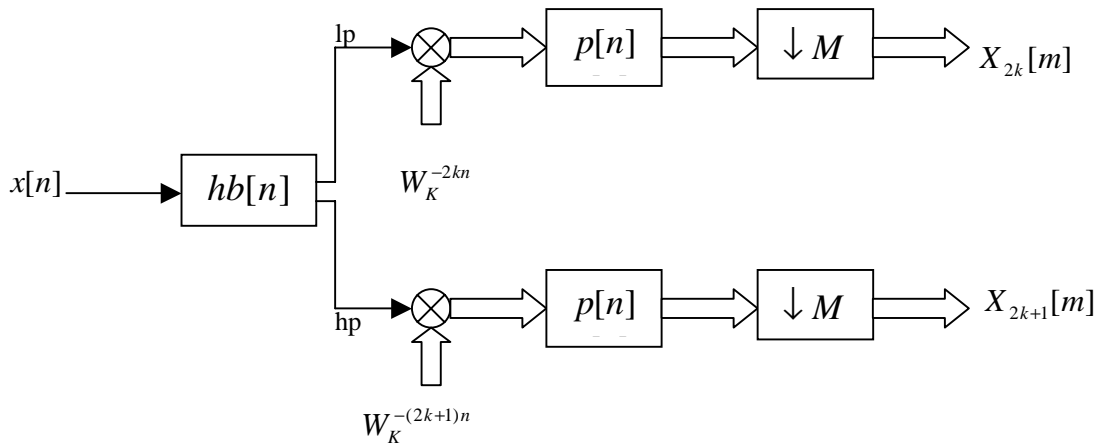


Figure 8 Two-stage filter bank analyzer

$$X_{2k}[m] = \left((x[n] \otimes hb_{lp}[n]) W_K^{-2kn} \right) \otimes p[n] \Big|_{n=mM} \tag{2.7}$$

$$X_{2k+1}[m] = \left((x[n] \otimes hb_{hp}[n]) W_K^{-(2k+1)n} \right) \otimes p[n] \Big|_{n=mM} \tag{2.8}$$

Eq.(2.7) and Eq.(2.8) explain how each channel can be extracted from an FDM signal: it can be seen that from the upper branch we obtain only even channels, while from the lower one only the odd ones. The problem of this structure is that it still works at high sampling frequency (we have decimation after the reference filter): this limits the maximum sampling frequency of the signal to be treated.

2.4. Parallel Implementation and Optimisation for Real FDM Signal.

A way to further improve the performance is to investigate a parallel solution for the complete system (not only the polyphase filter); this requires manipulation of Eq.(2.7) and Eq.(2.8). But there is another aspect that we have to consider in order to simplify the structure. The filter bank analyzer has

K outputs, each of which represents the baseband-modulated channel. If the FDM signal is real, its spectrum is symmetric; that means that half of the channels are a replica of the others. For this particular input, the parallel structure can be even further optimised. It is known [1] that in a normal filter bank analyzer, if the input is real, the required K -point real FFT can be reduced to a $K/2$ -point complex FFT. However, the two-stage structure, because of the presence of the odd-stacking channels, still requires the set of coefficients corresponding to a K -point FFT; we will show that the complexity of this K -point FFT is reduced compared to the general one.

Let's start with the calculation of the even channels.

$$\begin{aligned} X_{2k}[m] &= \left((x[n] \otimes hb_{lp}[n]) W_K^{-2kn} \right) \otimes p[n] \Big|_{n=mM} = \left(\sum_{r=-\infty}^{\infty} x[n-r] hb_{lp}[r] W_K^{-2kn} \right) \otimes p[n] \Big|_{n=mM} = \\ &= \sum_{r=-\infty}^{\infty} \sum_{q=-\infty}^{\infty} hb_{lp}[r] p[q] x[n-r-q] W_K^{-2k(n-q)} \Big|_{n=mM} = \sum_{r=-\infty}^{\infty} \sum_{q=-\infty}^{\infty} hb_{lp}[r] p[q] x[mM-r-q] W_K^{-2k(mM-q)} \end{aligned} \quad (2.9)$$

As in the upper part of the structure there are only $K/4$ useful channels, we can rewrite the indexes r and q in the following way:

$$\begin{cases} q = s \frac{K}{4} - l, \\ r = t \frac{K}{4} - i \end{cases} \quad l, i = 0, 1, \dots, K/4 - 1 \quad (2.10)$$

obtaining

$$X_{2k}[m] = \sum_{l=0}^{K/4-1} \sum_{s=-\infty}^{\infty} \sum_{i=0}^{K/4-1} \sum_{t=-\infty}^{\infty} p[s \frac{K}{4} - l] hb_{lp}[t \frac{K}{4} - i] x[mM - s \frac{K}{4} - t \frac{K}{4} + l + i] W_K^{-2k(mM - s \frac{K}{4} + l)} \quad (2.11)$$

Now, by observing that:

- $M = \frac{K}{2}$ (oversampling by two);
- imaged half-band filter is interpolated by $K/2 \Rightarrow hb_{lp}[t \frac{K}{4} - i] = \begin{cases} 0 & \text{if } i \neq 0, \\ \neq 0 & \text{otherwise} \end{cases} \Rightarrow$ the useful

branch of the filter is the one with $i = 0$ and it is interpolated by two (zero-filled by two);

- $hb_{lp}[t \frac{K}{4}] = \overline{hb_{lp}[t]}$; (counterclockwise)
- $p[s \frac{K}{4} - l] = \overline{p_l[s]}$ (counterclockwise)
- $x[mM - s \frac{K}{4} - t \frac{K}{4} + l] = x_l[2m - s - t]$ (clockwise).

$$X_{2k}[m] = \sum_{l=0}^{K/4-1} \sum_{s=-\infty}^{\infty} \sum_{t=-\infty}^{\infty} \overline{p_l[s]} \overline{hb_{lp}[t]} x_l[2m-s-t] W_K^{-2k(2m-s)} \frac{K}{4} W_K^{-2kl} = \quad (2.12)$$

$$= \sum_{l=0}^{K/4-1} \sum_{s=-\infty}^{\infty} \overline{p_l[s]} \left(\overline{hb_{lp}[2m-s]} \otimes x_l[2m-s] (-1)^{-k(2m-s)} \right) W_K^{-2kl} =$$

$$= \sum_{l=0}^{K/4-1} \overline{p_l[2m]} \otimes \left(\overline{hb_{lp}[2m]} \otimes x_l[2m] (-1)^{-k2m} \right) W_K^{-2kl} = \sum_{l=0}^{K/4-1} y_{k,l}[m] W_K^{-2kl} \quad (2.13)$$

$$\text{where } y_{k,l}[m] = \overline{p_l[2m]} \otimes \left(\overline{hb_{lp}[2m]} \otimes x_l[2m] (-1)^{-k2m} \right). \quad (2.14)$$

To reach the final structure, we have to understand the form of $y_{k,l}[m]$.

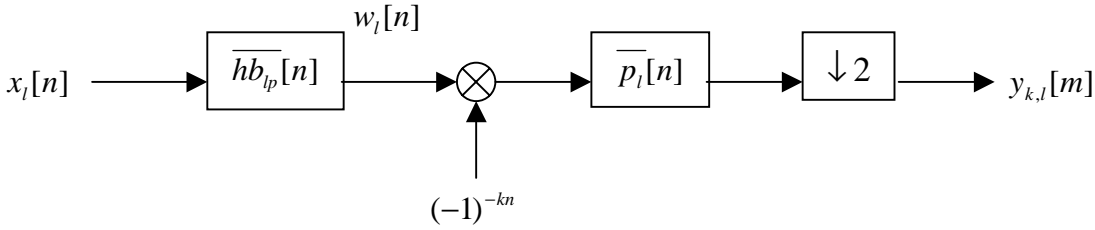


Figure 9 Branch preceding the FFT block.

As shown by Figure 9, it is possible to realise the final part (filter and decimator) using a polyphase structure. As we will look at a single branch now, the index l will be omitted in order to simplify the equations:

$$g[n] = \overline{p_l[n]}.$$

$$y_k[m] = \left(w[n] (-1)^{-kn} \right) \otimes g[n] \Big|_{n=2m} = \sum_{r=-\infty}^{\infty} w[n-r] (-1)^{-k(n-r)} g[r] \Big|_{n=2m} =$$

$$= \sum_{r=-\infty}^{\infty} w[2m-r] (-1)^{-k(2m-r)} g[r]. \quad (2.15)$$

By means of the known change of variable:

$$r = 2s - p, \quad p = 0, 1$$

we have:

$$y_k[m] = \sum_{p=0}^1 \sum_{s=-\infty}^{\infty} w[2m-2s+p] (-1)^{-k(2m-2s-p)} g[2s-p] = \sum_{p=0}^1 \sum_{s=-\infty}^{\infty} w_p[m-s] (-1)^{-kp} \overline{g_p[s]} =$$

$$= w_0[m] \otimes \overline{g_0}[m] + (-1)^{-k} w_1[m] \otimes \overline{g_1}[m]. \quad (2.16)$$

Eq. (2.16) results in the following scheme:

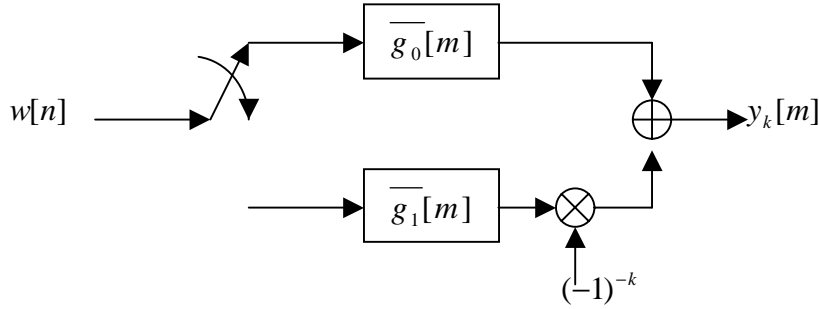


Figure 10 Polyphase version of the lowpass branch preceding the FFT block.

In a similar manner we extract the structure for the odd channels; we can start from Eq.(2.12) and then substitute lp with hp and $2k$ with $2k+1$, obtaining:

$$X_{2k+1}[m] = \sum_{l=0}^{K/4-1} \sum_{s=-\infty}^{\infty} \sum_{t=-\infty}^{\infty} p_l[s] \overline{h_{hp}}[t] x_l[2m-s-t] W_K^{-(2k+1)(2m-s)} \frac{K}{4} W_K^{-(2k+1)l} = \quad (2.17)$$

$$= \sum_{l=0}^{K/4-1} \sum_{s=-\infty}^{\infty} p_l[s] \left(\overline{h_{hp}}[2m-s] \otimes x_l[2m-s] (-j)^{-(2k+1)(2m-s)} \right) W_K^{-(2k+1)l} =$$

$$= \sum_{l=0}^{K/4-1} p_l[2m] \left(\overline{h_{hp}}[2m] \otimes x_l[2m] (-j)^{-(2k+1)2m} \right) W_K^{-(2k+1)l} = \sum_{l=0}^{K/4-1} z_{k,l}[m] W_K^{-(2k+1)l} \quad (2.18)$$

$$\text{where } z_{k,l}[m] = \overline{p_l}[2m] \otimes \left(\overline{h_{hp}}[2m] \otimes x_l[2m] (-j)^{-(2k+1)2m} \right) \quad (2.19)$$

Once again it is possible to obtain a polyphase structure for the branches for the odd channels. The slight difference from the previous one is due to the fact that the odd channels are shifted to higher frequency by one slot.

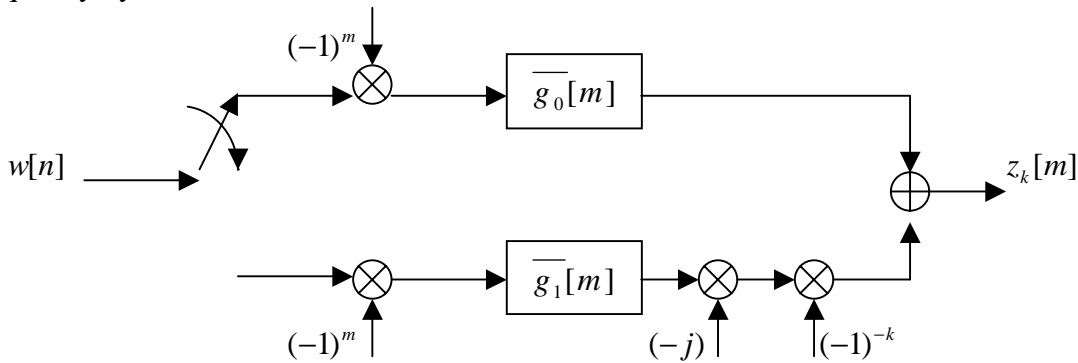


Figure 11 Polyphase branch version of the highpass branch preceding the FFT block.

From the scheme of Figure 10 and Figure 11 we can see that both $X_{2k}[m]$ and $X_{2k+1}[m]$ come from branches which have a dependency on k : the term $(-1)^{-k}$ appears. This means that if k is even ($k=2j$),

$v_{k,l}[m]$
 for the channels $X_{2k}[m] = X_{4j}[m]$ and $X_{2k+1}[m] = X_{4j+1}[m]$ ($X_0[m], X_4[m], \dots$ and $X_1[m], X_5[m], \dots$) we obtain (+), while if it is odd ($k=2j+1$), for the channels $X_{2k}[m] = X_{4j+2}[m]$ and $X_{2k+1}[m] = X_{4j+3}[m]$ ($X_2[m], X_6[m], \dots$ and $X_3[m], X_7[m], \dots$) we obtain (-). For this reason, each branch has two outputs, one resulting from the addition of two sub-branches, the other one resulting from the subtraction. Figure 13 shows the overall structure. Now let's turn our attention to the FFT. As shown by Eq.(2.13) and Eq.(2.18), the output block is reduced FFT. The restricted range of index l is due to the focus on the useful channels of both even-stacked and odd-stacked branches (we have $K/4$ slots for each branch), while the difference in the index k ($2k$ and $2k+1$) is depending on the channels. It is possible to have only one equation for the reduced FFT, by putting the following conditions on $v_{k,l}[m]$:

$$X_k[m] = \sum_{l=0}^{K/4-1} v_{k,l}[m] W_K^{-kl} \tag{2.20}$$

$$\text{where } v_{k,l}[m] = \begin{cases} y_{k,l}[m] & \text{for } k \text{ - even} \\ z_{k,l}[m] & \text{for } k \text{ - odd} \end{cases} \tag{2.21}$$

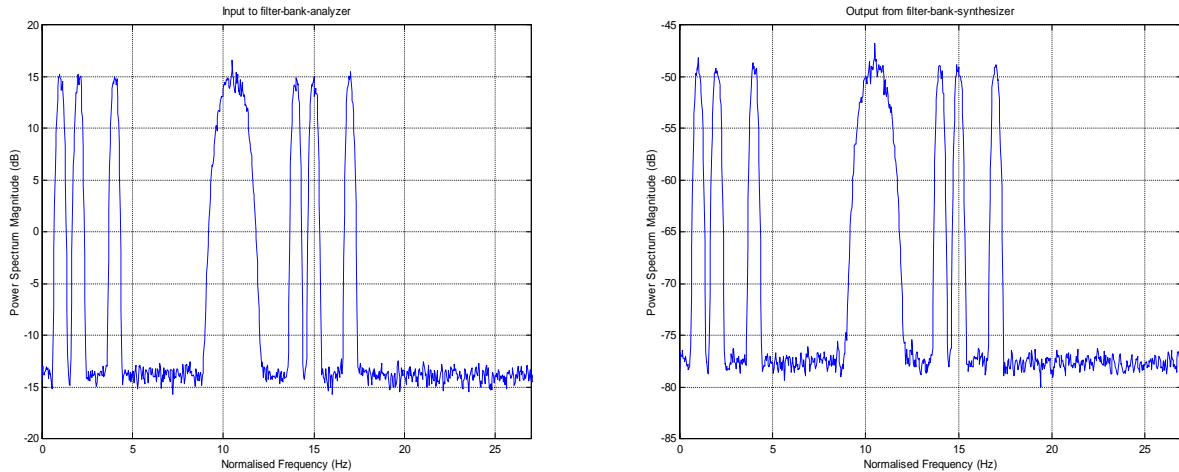


Figure 12 FDM signal as input to analyzer/synthesizer with two-stage architecture.

The requirement to know in advance what kind of input to use to generate a desired channel (for number 0,4,8,... and 1,5,9,... we keep the outputs coming from the addition branches, while for number 2,6,10,... and 3,7,11,... the ones coming from the subtraction branches) implies that the first stage of

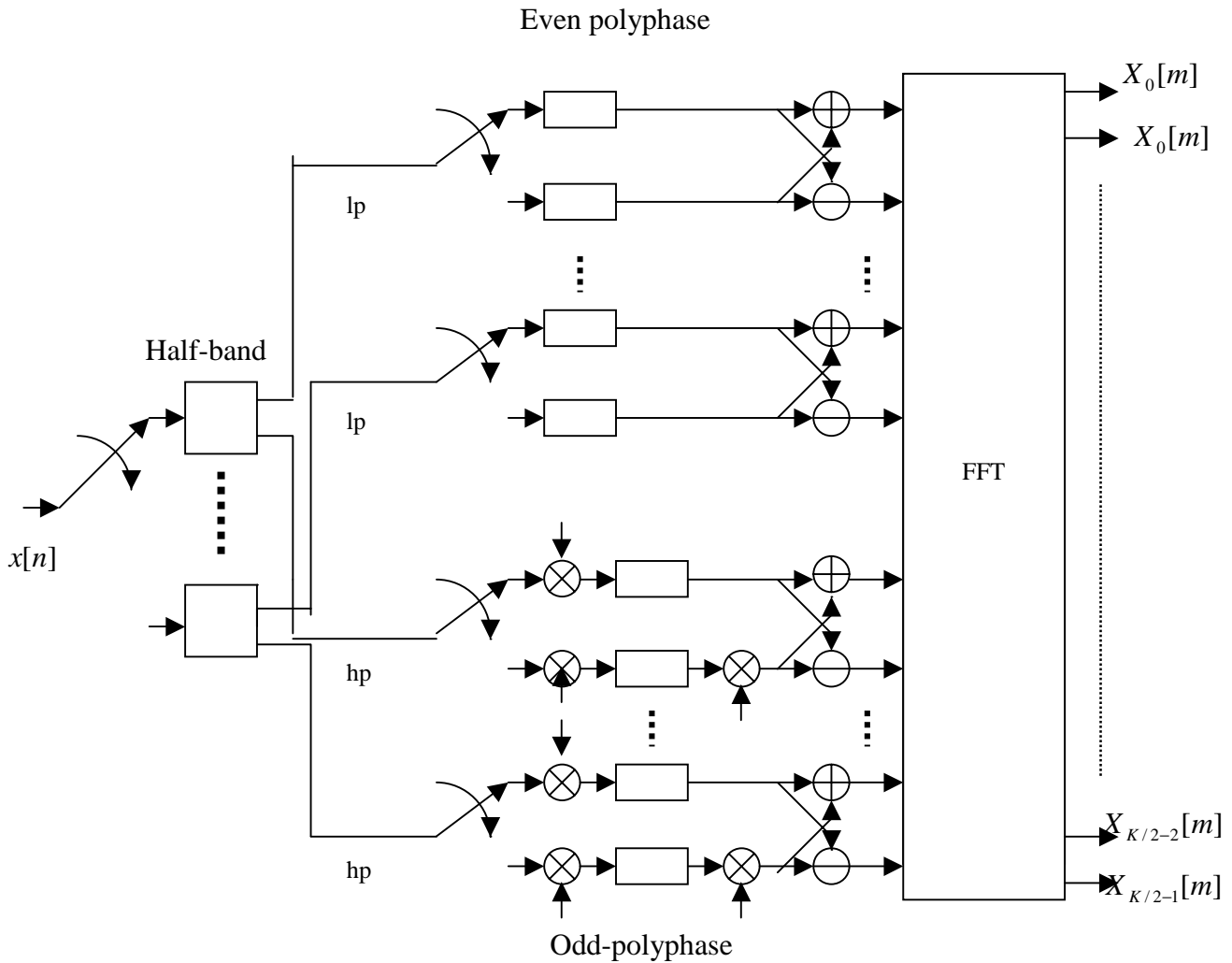


Figure 13 Overall structure of the two-stage parallel filter bank analyzer for real input. Globally the structure has $K/4$ half band filters (interpolated by two), $K/2$ even-polyphase filters, $K/2$ odd-polyphase filters and one FFT block.

this reduced FFT has to be different from the same stage of a normal FFT with the same algorithm. We will show that it will be also of lower complexity (in terms of number of operations).

A Matlab model of the two-stage architecture filter bank analyzer with real input has been realised (Figure 12 shows the plots related to it): compared to the previous model, this one has a fixed number of channels (56, which means 28 useful channels). This is due to the fact that the reduced FFT in the last stage specifically optimised for 56 channel. For the filter bank synthesizer we keep the same structure as in the previous model: but now we have at the output of the analyzer only half of the channels, therefore we fill the other ones with zero.

3. RTL LEVEL VHDL MODEL OF THE FILTER BANK.

3.1. DESIGN

The RTL modelling involves the generation of the VHDL code for all the blocks in the previous chapter. The chapter will show the architectural implementation of each block, with comments where needed. The code is attached in the Appendix of this report.

All the files are contained in the folder “*Filter_bank/*”. Files of general use are:

- “*general.vhd*” containing types and constants used within the structure;
- “*trunc.vhd*” containing mathematic functions for rounding-method.

All the other files are specific for each block, so they will be mentioned with the proper structure.

3.1.1. Imaged HB Filter.

Related files:

- “*hb_coeff.vhd*”: package containing coefficients for the related filter;
- “*hb.vhd*”: entity representing the filter.

The architecture used to build this filter is a direct one, in which the symmetry in the impulse response allows reduction of the operations needed. Since the direct structure has the negative aspect of several cascade adders [1], levels of pipeline are required in order to improve the speed. With pipeline, with need also a sort of compensation of the delays to make the filter to work correctly. Input and outputs are all real.

5

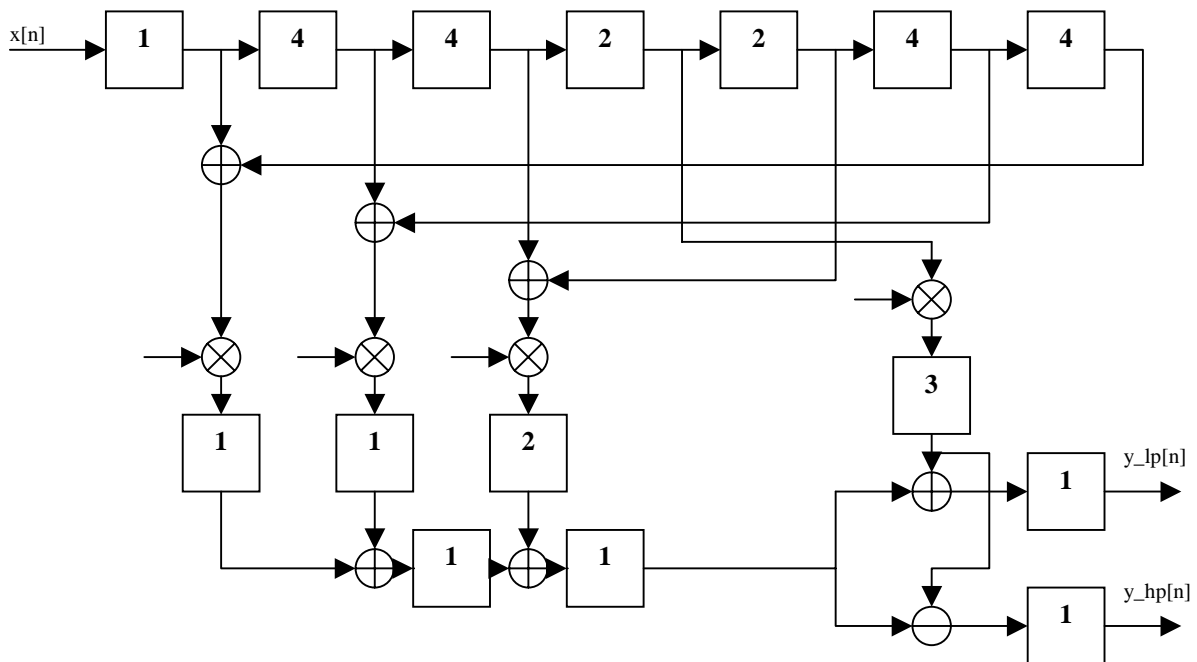


Figure 14 First direct structure of the imaged half-band filter with delays compensation.

The number within the blocks of Figure 13 indicates the length of the delay. It is possible to eliminate some added delays by grouping them in the main shift-register, without growing its dimension, as Figure 15 shows.

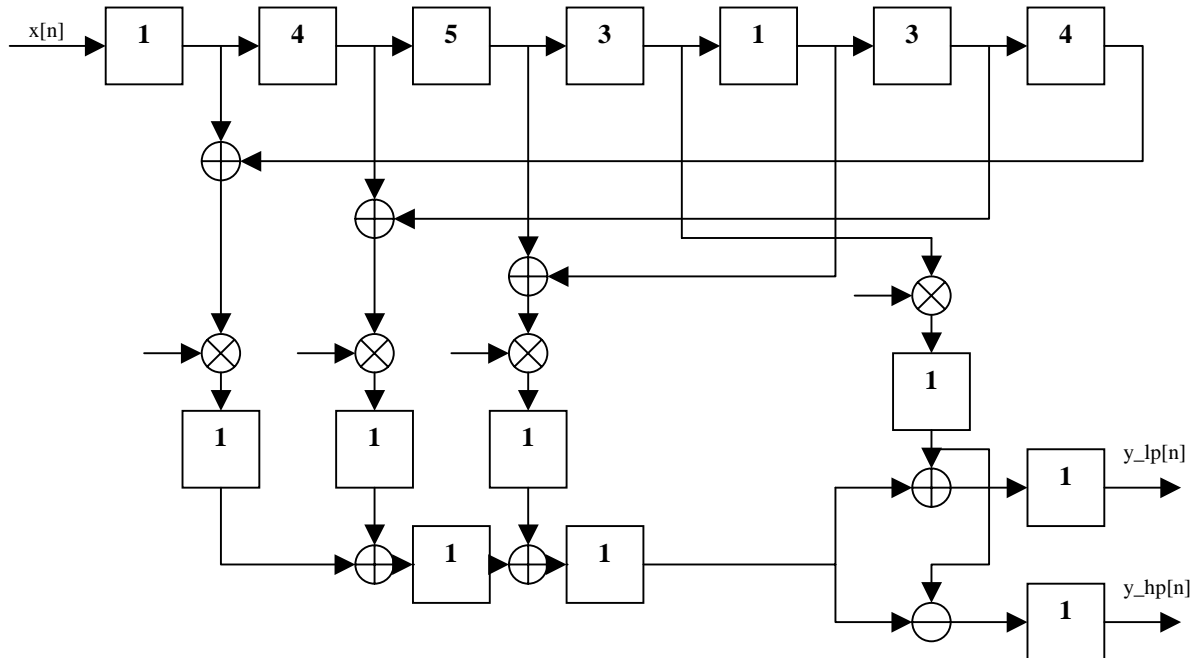


Figure 15 Optimised direct structure of the imaged half-band filter with delays compensation.

3.1.2. Polyphase Filter.

Related files:

- “*polyphase_coeff.vhd*” : package containing the coefficients of the low pass filter, before its splitting into a polyphase structure;
- “*even_polyphase.vhd*” , “*odd_polyphase.vhd*” : files for the respective entities.

These blocks realise the structure of Figure 10 and Figure 11. Even_polyphase filter has one real input and two real outputs, while the odd_polyphase filter has one real input and two complex outputs. This is due to the fact that within the second filter there is a multiplication by $(-j)$. It has to be noted that all the multiplications involved in the odd-polyphase filter, externally to the two sub-filters, are trivial: they consist only in changes of the sign or in exchanging the real and the imaginary part. Finally the addition/subtraction at the output of the odd_polyphase filter is just forming the complex number, there is no operation. The architecture is a transposed one, without any symmetry among each subset of coefficients.

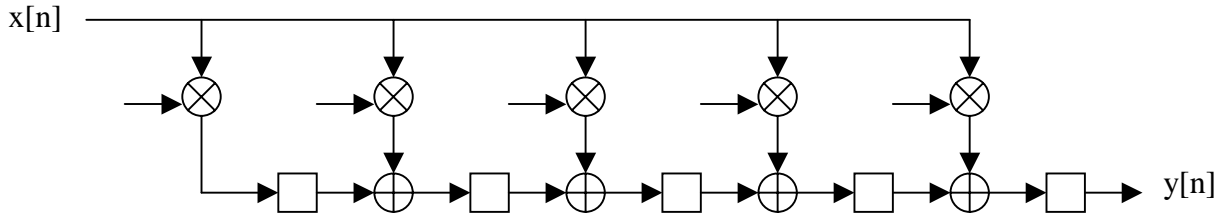


Figure 16 Transposed structure of a sub-filter contained in both even_polyphase and odd_polyphase.

3.1.3. FFT.

Concerning the FFT, all the general models are based on complex data, so both the inputs and the outputs are complex: this makes the FFT suitable for different purposes. The FFT for this filter bank, being optimised for it itself, has the first stage working with a subset of the inputs which is always real (the ones coming from the even_polyphase filters). The general form for an N-point DFT is the following:

$$X_k = \sum_{n=0}^{N-1} x[n] W_N^{-kn}, \quad \text{where } W_N = e^{j\frac{2\pi}{N}}, k = 0, 1, \dots, K-1. \quad (3.1)$$

3.1.3.1. 56-point Good-Thomas Algorithm.

The algorithm described first by Good in 1958 [2] was not generally competitive with the radix-2 FFT algorithm prior to the advent of efficient small N-DFT algorithms. This method is applicable when N of Eq.(3.1) can be expressed in the form $N=N_1*N_2$, with N_1 and N_2 relative prime factors. In this case, the N-point DFT reduces itself to a computation of N_2 different N_1 -point DFT and N_1 different N_2 -point DFT, each of which is structured with the mentioned small N-DFT algorithm. The general idea of the Good-Thomas technique is to split both n and k of Eq.(3.1) in terms of their prime factors, using the SIR method(Second Integer Representation) for n and the CRT method(Chinese Remainder Theorem) for k (or vice-versa) [2,3,4]:

$$\begin{cases} n = \text{mod}(n_1 * N_2 + n_2 * N_1, N), \\ k = \text{mod}(t_1 * k_1 * N_2 + t_2 * k_2 * N_1, N), \end{cases} \begin{cases} t_1 \text{ such as } \text{mod}(t_1 * N_2, N_1) = 1, \\ t_2 \text{ such as } \text{mod}(t_2 * N_1, N_2) = 1 \end{cases} \begin{cases} n_1, k_1 = 0, 1, \dots, N_1 - 1 \\ n_2, k_2 = 0, 1, \dots, N_2 - 1 \end{cases} \quad (3.2)$$

With this substitution we reach:

$$\begin{aligned} X_k &= X(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x[n_1, n_2] W_N^{-(n_1 N_2 + n_2 N_1)(t_1 k_1 N_2 + t_2 k_2 N_1)} = \\ &= \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x[n_1, n_2] W_N^{-(n_1 t_1 k_1 N_2^2)} W_N^{-(n_2 t_2 k_2 N_1^2)} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x[n_1, n_2] W_{N_1}^{-(n_1 t_1 k_1 N_2)} W_{N_2}^{-(n_2 t_2 k_2 N_1)} = \end{aligned}$$

$$\begin{aligned}
 &= \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x[n_1, n_2] W_{N_1}^{-n_1 k_1 (1-N_1)} W_{N_2}^{-n_2 k_2 (1-N_2)} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x[n_1, n_2] W_{N_1}^{-n_1 k_1} W_{N_2}^{-n_2 k_2} = \\
 &= \sum_{n_1=0}^{N_1-1} W_{N_1}^{-n_1 k_1} \sum_{n_2=0}^{N_2-1} x[n_1, n_2] W_{N_2}^{-n_2 k_2} \tag{3.3}
 \end{aligned}$$

As shown by Eq.(3.3) we are left with the computation of two smaller DFTs, the first one of N_2 - point, the second one of N_1 -point. For our purpose $N_1=8$ and $N_2=7 \Rightarrow$

$$\begin{cases} n = \text{mod}(7n_1 + 8n_2, 56) & \text{SIR} \\ k = \text{mod}(7k_1 t_1 + 8k_2 t_2, 56) & \text{CRT} \end{cases} \Rightarrow \begin{cases} t_1 \text{ such as } \text{mod}(7t_1, 8) = 1 \Rightarrow t_1 = 7, \\ t_2 \text{ such as } \text{mod}(8t_2, 7) = 1 \Rightarrow t_2 = 1 \end{cases}$$

$$\begin{cases} n = \text{mod}(7n_1 + 8n_2, 56) & \text{SIR} \\ k = \text{mod}(49k_1 + 8k_2, 56) & \text{CRT} \end{cases} \tag{3.4}$$

$$= \sum_{n_1=0}^7 W_8^{-n_1 k_1} \sum_{n_2=0}^6 x[n_1, n_2] W_7^{-n_2 k_2} \tag{3.5}$$

The resulting structure appears in Figure 17. To build this FFT, we need now the algorithms for the small N-DFT. Some slight modification has been implemented for the small 8-DFT algorithm, but it doesn't affect the result (it is a way of compacting the equations when writing VHDL code).

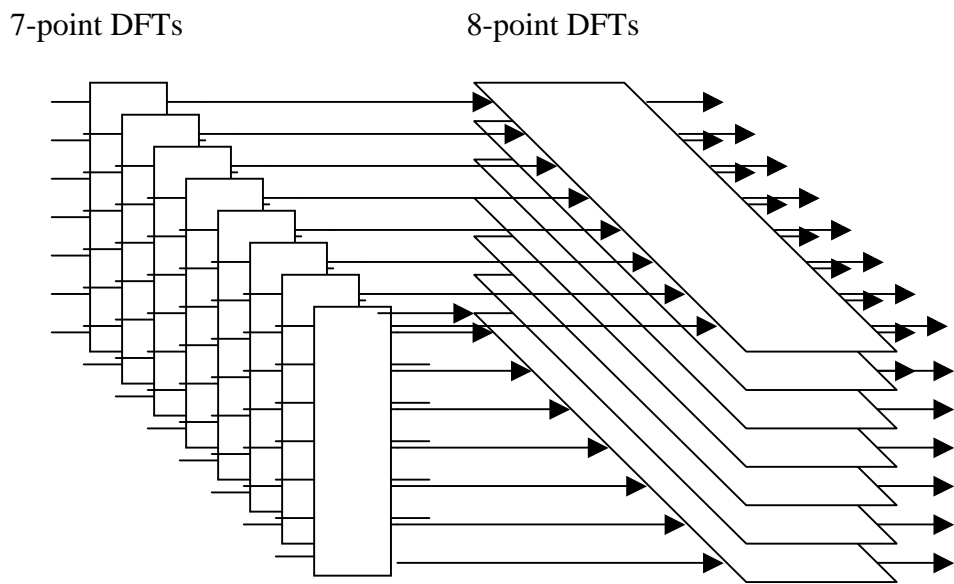


Figure 17 Structure of the 56-point Good-Thomas FFT.

$$N = 7, u = \frac{2}{7}\pi$$

$$t_1 = x[1] + x[6], \quad t_t = x[2] + x[5], \quad t_3 = x[3] + x[4],$$

$$t_4 = t_1 + t_t + t_3,$$

$$t_5 = x[1] - x[6], \quad t_6 = x[2] - x[5], \quad t_7 = x[2] - x[5]$$

$$m_0 = 1 * (x(0) + t_4)$$

$$m_1 = \left[\frac{1}{3} (\cos(u) + \cos(2u) + \cos(3u)) - 1 \right] * t_4, \quad m_2 = \frac{1}{3} (2 \cos(u) - \cos(2u) - \cos(3u)) * (t_1 - t_3)$$

$$m_3 = \frac{1}{3} (\cos(u) - 2 \cos(2u) + \cos(3u)) * (t_3 - t_2), \quad m_4 = \frac{1}{3} (\cos(u) + \cos(2u) - 2 \cos(3u)) * (t_2 - t_1)$$

$$m_5 = -j \frac{1}{3} (\sin(u) + \sin(2u) - \sin(3u)) * (t_5 + t_6 + t_7), \quad m_6 = j \frac{1}{3} (2 \sin(u) - \sin(2u) + \sin(3u)) * (t_7 - t_5)$$

$$m_7 = j \frac{1}{3} (\sin(u) - 2 \sin(2u) - \sin(3u)) * (t_6 - t_7), \quad m_8 = j \frac{1}{3} (\sin(u) + \sin(2u) + 2 \sin(3u)) * (t_5 - t_6)$$

$$s_1 = m_0 + m_1, \quad s_2 = s_1 + m_2 + m_3, \quad s_3 = s_1 - m_2 - m_4, \quad s_4 = s_1 - m_3 + m_4,$$

$$s_5 = m_5 + m_6 + m_7, \quad s_6 = m_5 - m_6 - m_8, \quad s_7 = m_5 - m_7 + m_8$$

$$X_0 = m_0, \quad X_1 = s_2 + s_5, \quad X_2 = s_3 + s_6, \quad X_3 = s_4 - s_7, \quad X_4 = s_4 + s_7, \quad X_5 = s_3 - s_6, \quad X_6 = s_2 - s_5$$

Total (complex data): 18 multiplications (2 trivial if no scaling is performed), 72 additions.

$$N = 8, \quad u = \frac{2}{8}\pi$$

$$t_0 = x[0] + x[4], \quad t_1 = x[1] + x[5], \quad t_2 = x[2] + x[6], \quad t_3 = x[3] + x[7],$$

$$t_4 = x[0] - x[4], \quad t_5 = x[1] - x[5], \quad t_6 = x[2] - x[6], \quad t_7 = x[3] - x[7]$$

$$m_0 = 1 * (t_0 + t_2), \quad m_1 = 1 * (t_1 + t_3), \quad m_2 = 1 * (t_0 - t_2), \quad m_3 = 1 * (t_1 - t_3), \quad m_4 = 1 * (t_4 + jt_6),$$

$$m_5 = \cos(u) * [(1 - j)(t_4 - jt_6)], \quad m_6 = 1 * (t_5 + jt_7), \quad m_7 = \cos(u) * [(1 - j)(t_5 - jt_7)]$$

$$X_0 = m_0 + m_1, \quad X_1 = m_5 + m_7, \quad X_2 = m_2 - jm_3, \quad X_3 = m_4 - jm_6$$

$$X_4 = m_0 - m_1, \quad X_5 = m_5 - m_7, \quad X_6 = m_2 + jm_3, \quad X_7 = m_4 + jm_6$$

Total (complex data): 16 multiplications (12 trivial if no scaling is performed), 52 additions.

Eq (3.4) shows the sequence in which input data have to be presented to the following blocks; also the outputs have to be reordered. Within the structure the data go from the 7-point DFT to the 8-point DFT in the following way: the first outputs of each 7-DFT go to the first 8-DFT, all the second outputs to the second 8-DFT and so on. From Figure 18 and Figure 19 it is possible to see the input reordering by SIR method and the output reordering by CRT method.

0	7	14	21	28	35	42	49
8	15	22	29	36	43	50	1
16	23	30	37	44	51	2	9
24	31	38	45	52	3	10	17
32	39	46	53	4	11	18	25
40	47	54	5	12	19	26	33
48	55	6	13	20	27	34	41

Figure 18 SIR method for ordering the inputs. Each column is a 7-point DFT.

0	8	16	24	32	40	48
49	1	9	17	25	33	41
42	50	2	10	18	26	34
35	43	51	3	11	19	27
28	36	44	52	4	12	20
21	29	37	45	53	5	13
14	22	30	38	46	54	6
7	15	23	31	39	47	55

Figure 19 CRT method for reordering the outputs. Each column is an 8-point DFT.

Now let's turn our attention on the Good-Thomas FFT required for the 28 channels filter bank analyzer with real input. According to Eq.(2.20) index l runs between 0 and 13 ($K/4-1$) while index k runs between 0 and 27 ($K/2-1$). This requires a particular way of writing both the indexes with SIR and CRT method. Because of the restricted range of l , it is possible to have [2,3]:

$$\left[\begin{array}{l} l = \text{mod}(7n_1 + 8n_2, 14) = 7n_1 + 8n_2 - \left\lfloor \frac{7n_1 + 8n_2}{14} \right\rfloor 14 = 7n_1 + 8n_2 - \left\lfloor \frac{n_1}{2} + \frac{4n_2}{7} \right\rfloor 14 \\ (n_1 = 0,1 ; n_2 = 0,1,\dots,6) \end{array} \right. \quad \text{SIR} \quad (3.6)$$

where $\lfloor n \rfloor$ denotes the largest integer less than or equal to n .

No simplification of the structure is achieved applying the similar procedure to k . This means at the output all channels are available and the required ones have to be selected.

$$\left[\begin{array}{l} k = \text{mod}(49k_1 + 8k_2, 56) \\ (k_1 = 0,1,\dots,7 ; k_2 = 0,1,\dots,6) \end{array} \right. \quad \text{CRT.} \quad (3.7)$$

0	8	2	10	4	12	6
7	1	9	3	11	5	13

Figure 20 SIR method for the inputs.

0					21	14	7
8	1					22	15
16	9	2					23
24	17	10	3				
	25	18	11	4			
		26	19	12	5		
			27	20	13	6	

Figure 21 CRT method for the outputs.

With these substitutions Eq.(2.20) becomes:

$$\begin{aligned}
 X(k_1, k_2) &= \sum_{n_2=0}^6 \sum_{n_1=0}^1 v_{k_1, k_2}(n_1, n_2) W_{56}^{-(7n_1+8n_2) \left\lfloor \frac{n_1+4n_2}{2+7} \right\rfloor 14(49k_1+8k_2)} = \\
 &= \sum_{n_2=0}^6 \sum_{n_1=0}^1 v_{k_1, k_2}(n_1, n_2) W_8^{-k_1 n_1} W_7^{-k_2 n_2} W_4^{\left\lfloor \frac{n_1+4n_2}{2+7} \right\rfloor k_1} = \sum_{n_2=0}^6 W_7^{-k_2 n_2} \sum_{n_1=0}^1 v_{k_1, k_2}(n_1, n_2) W_8^{-k_1 n_1} W_4^{\left\lfloor \frac{n_1+4n_2}{2+7} \right\rfloor k_1} \quad (3.8)
 \end{aligned}$$

Eq.(3.8) gives a reduced form of the Good-Thomas FFT. The 7-point DFT is unchanged, while the 8-point DFT is reduced compared to the normal one. To better understand its structure, we can define

$$V_{k_2}(k_1, n_2) = \sum_{n_1=0}^1 v_{k_1, k_2}(n_1, n_2) W_8^{-k_1 n_1} W_4^{\left\lfloor \frac{n_1+4n_2}{2+7} \right\rfloor k_1} \quad (3.9)$$

Let's have a look of the terms appearing in Eq.(3.9):

- a) $v_{k_1, k_2}(n_1, n_2)$ are the inputs. As stated in paragraph 2.4. they are depending on k . For example if k is even, $v_{k_1, k_2}(n_1, n_2)$ is corresponding to the output of the even-polyphase filters ($y_{k,l}[m]$); if it is odd, $v_{k_1, k_2}(n_1, n_2)$ is corresponding to the output of the odd-polyphase filters ($z_{k,l}[m]$). From Figure 21 and Eq.(3.7) we can see that k even or odd is related to k_1 even or odd. From paragraph 2.4. we know that for channels number 0,4,8,... and 1,5,9,... the correct outputs to feed to the FFT are the ones resulting from the addition-branch of each polyphase filters; for channels number 2,6,10,... and 3,7,11,... the correct outputs are the ones resulting from the subtraction-

branch of each polyphase filter. Once again this order is maintained by k_1 , as shown by Figure 21. This means that in Eq.(3.9) k_2 dependency disappears.

- b) $W_8^{-k_1 n_1}$ is the weight for each input, related to the 8-point DFT. However Eq.(3.6) and Eq.(3.9) show that two input per time are involved compared to eight for a normal 8-point DFT: this implies that the matrix of the coefficients can be reduced considerably which leads to a reduction of complexity. As k_1 runs between 0 and 7, 8 outputs are obtained. However only four outputs of the 8-point DFT are required. We need to multiplex the coefficients to cover all the possible outputs, before selecting the useful ones. This operation is performed inside the 8-point DFT as shown by Eq.(3.10) and Eq.(3.11): when k_1 runs between 0 and 3 we have additions of the two terms; in the other half, we have subtractions. This solution can be easily implemented within the structure, but it requires a clock two times faster than the normal one.
- c) $W_4^{\lfloor \frac{n_1 + 4n_2}{2} \rfloor k_1}$ is a factor, which doesn't appear in a normal 8-point DFT. Its complexity in term of multiplication is zero, because it only implies a change in the sign or exchange between real and imaginary part. It is a coefficient depending on input and output (n_1, n_2, k_1) , like $W_8^{-k_1 n_1}$, but including the $\lfloor \rfloor$ operator. However its processing within what we will call by now “*partial_8_fft*” is simple.

$$\begin{cases} V(0, n_2) = y_+[0, n_2] & + & y_+[1, n_2] \\ V(1, n_2) = z_+[0, n_2](j)^{\lfloor \frac{4n_2}{7} \rfloor} & + & z_+[1, n_2]W_8^{-1}(j)^{\lfloor \frac{1+4n_2}{2} \rfloor} \\ V(2, n_2) = y_-[0, n_2](-1)^{\lfloor \frac{4n_2}{7} \rfloor} & + & y_-[1, n_2](-j)(-1)^{\lfloor \frac{1+4n_2}{2} \rfloor} \\ V(3, n_2) = z_-[0, n_2](-j)^{\lfloor \frac{4n_2}{7} \rfloor} & + & z_-[1, n_2](-j)W_8^{-1}(-j)^{\lfloor \frac{1+4n_2}{2} \rfloor} \end{cases} \quad (3.10)$$

$$\begin{cases} V(4, n_2) = y_+[0, n_2] & - & y_+[1, n_2] \\ V(5, n_2) = z_+[0, n_2](j)^{\lfloor \frac{4n_2}{7} \rfloor} & - & z_+[1, n_2]W_8^{-1}(j)^{\lfloor \frac{1+4n_2}{2} \rfloor} \\ V(6, n_2) = y_-[0, n_2](-1)^{\lfloor \frac{4n_2}{7} \rfloor} & - & y_-[1, n_2](-j)(-1)^{\lfloor \frac{1+4n_2}{2} \rfloor} \\ V(7, n_2) = z_-[0, n_2](-j)^{\lfloor \frac{4n_2}{7} \rfloor} & - & z_-[1, n_2](-j)W_8^{-1}(-j)^{\lfloor \frac{1+4n_2}{2} \rfloor} \end{cases} \quad (3.11)$$

Notation:

- y and z are the outputs, respectively, of the even-polyphase and odd-polyphase filters;
- $+$ and $-$ indicates the addition and the subtraction branch;

- $W_8^{-1} = \frac{(1-j)}{\sqrt{2}}$ is the coefficient involved in the computation for the odd-channels. Although it is complex, it requires only two (one) multiplications by $\frac{1}{\sqrt{2}}$ if the other operand is complex (real).

As the reduced FFT is the only one applied to the filter bank, its files are in the same directory; the others ones, instead, have their own directory. However the packages, except for the one for the coefficients, are always the same.

Related files to the general Good-Thomas FFT:

- working directory “/Good_fft/”;
- “coeff_7.vhd”: package containing the coefficients for the small 7-point DFT algorithm;
- “seven_fft.vhd”, “eight_fft.vhd”: they contain entity and architecture for the 7-point DFT and the 8-point DFT. For the latter one, its coefficient is in the package *general.vhd*.
- “splitter.vhd”: block which orders the inputs with the SIR method;
- “Good_fft.vhd”: top hierarchy file. The final reordering of the outputs is performed inside this block;
- “Good_fft_tb.vhd”: test-bench for the whole structure, holding also the configuration for the current architecture.

Related files to the reduced Good-Thomas FFT and to the final whole filter bank:

- “coeff_7.vhd”, “seven_fft.vhd”: the same of the general one;
- “partial_8_fft.vhd”: it contains the structure whose equations are described by Eq.(3.10) and Eq.(3.11);
- “filter_bank.vhd”: it is the top hierarchy file for the global structure. The SIR method for the inputs and the CRT method for the outputs are implemented internally.
- “filter_bank_tb.vhd”: test-bench for the whole structure, including the related configuration.

3.1.3.2. 64-point Radix-4.

Radix-4 FFTs [3,4,6] are defined by an FFT whose number of input (output) points is a power of 4. They offer a reduced number of multiplications with respect to radix-2 transforms. By starting from Eq.(3.1) with $N=64$ and with the so called *decimation in time* [3] algorithm for the inputs, we have:

$$n = 16r + 4p + l, \quad r, p, l \in [0,3] \quad (3.12)$$

$$X(k) = \sum_{l=0}^3 \sum_{p=0}^3 \sum_{r=0}^3 x[16r + 4p + l] W_{64}^{-k(16r+4p+l)} = \sum_{l=0}^3 W_{64}^{-kl} \sum_{p=0}^3 W_{64}^{-4kp} \sum_{r=0}^3 x[r, p, l] W_4^{-kr} \quad (3.13)$$

We can see that it is possible to reduce the calculation of a 64-point FFT to that one of several 4-point DFTs, in combination with multiplication by a twiddle factor. The computation of a 4-point DFT is very simple, because there is no multiplication involved, only additions. The matrix representing a 4-point DFT shows this.

$$W_4^{-kr} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \quad (3.14)$$

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & -j \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & j \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ x[3] \end{bmatrix} \quad (3.15)$$

In order to reduce the complexity of the structure (the number of multiplications is still high), we can split index k in the following way:

$$k = 16t + 4j + i, \quad t, j, i \in [0,3] \quad (3.16)$$

Eq.(3.13) now becomes:

$$X(t, j, i) = \sum_{l=0}^3 W_4^{-tl} W_{64}^{-(4j+i)l} \sum_{p=0}^3 W_4^{-jp} W_{64}^{-4ip} \sum_{r=0}^3 W_4^{-ir} x[r, p, l] \quad (3.17)$$

Eq.(3.17) shows the whole structure. With 64-point radix-4 FFT we have three stages of 4-point DFT (*butterfly*) and two stages of twiddle factors. As explained before, each butterfly requires only additions. Eq.(3.15) shows that 8 complex additions are involved, therefore we have:

$A_b = N_{add-but} * N_{but-stage} * N_{stage} = 8 * 16 * 3 = 144$ (A_b is the number of complex additions in the butterfly). Each twiddle stage contains 3 non-trivial multiplications. In the first stage there is 1 trivial block of twiddle factors (all the coefficients are 1), while in the second one there are 4. Globally we have M_t (multiplications due to the twiddle):

$M_t = N_{mul-twi} * (N_{twi-stage} * N_{stage} - N_{trivial-twi}) = 3 * (16 * 2 - 5) = 81$ complex multiplications. A complex multiplication requires 4 real multiplications and 2 real additions.

Related files:

- `"/Radix_4_fft/":` working directory;
- `"coeff_64.vhd":` package for the coefficients. There are the coefficients for the general 64-point DFT, so most of them are never used;
- `"splitter.vhd":` proper reordering of the input before to send them to the several butterflies;

- “*first_btfly_4.vhd*”, “*second_btfly_4.vhd*”, “*third_btfly_4.vhd*”: they realise the 4-point DFT. We

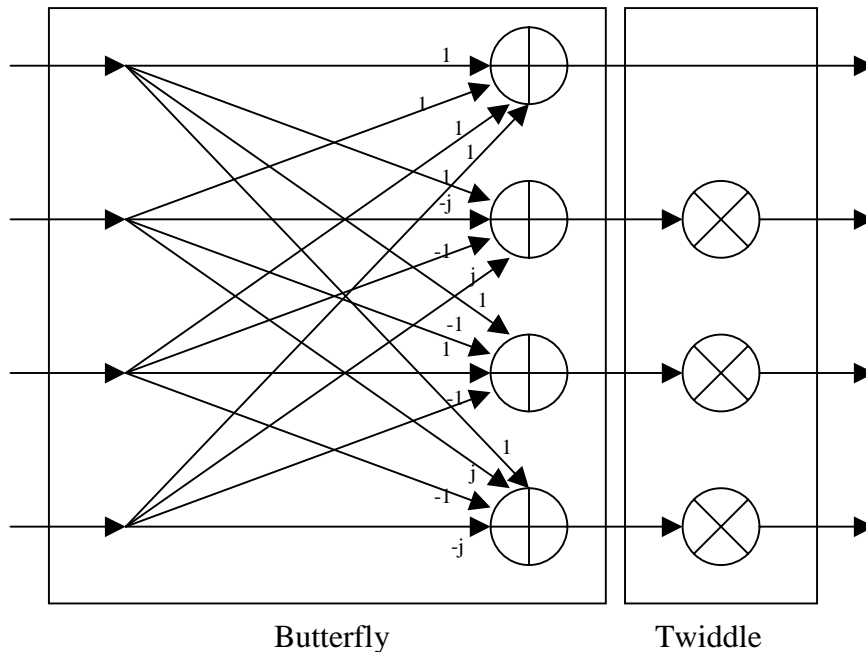


Figure 22 Link between butterfly and twiddle (except for the last stage).

have three structures because of the difference in the number of the bits used to represent input and outputs, which implies different internal operations. The second stage of butterflies is the only one that can be repeated in a higher order radix-4 FFT;

- “*first_twiddle.vhd*”, “*second_twiddle.vhd*”: they are the twiddle stages. The reason of two is the same as for the butterflies. The second twiddle can be repeated in a higher order of radix-4 FFT;
- “*radix_4_fft.vhd*”: top hierarchy file which allocates all the needed structures;
- “*radix_4_fft_tb.vhd*”: related test-bench holding also the configuration for the current architecture.

3.1.3.3. 64-point Radix-8.

The last algorithm developed for the 64-point DFT is based on the radix-8 algorithm. This implementation requires a more complex structure for the butterfly (which has 8 inputs/outputs), but a reduced number of multiplications involved for its computation. The method to obtain a cascade structure is the same used for the Radix-4:

$$n = 8p + l \quad k = 8j + i \quad p, l, j, i \in [0, 7]$$

With these substitutions Eq.(3.1) becomes:

$$\begin{aligned}
X(j, i) &= \sum_{p=0}^7 \sum_{l=0}^7 x[8p+l] W_{64}^{-(8j+i)(8p+l)} = \sum_{p=0}^7 \sum_{l=0}^7 x[p, l] W_8^{-ip} W_8^{-jl} W_{64}^{-il} = \\
&= \sum_{l=0}^7 W_8^{-jl} W_{64}^{-il} \sum_{p=0}^7 x[p, l] W_8^{-ip}
\end{aligned} \tag{3.18}$$

Two stages of butterfly (the ones with W_8) and one stage of twiddle factor can be seen. In order to reduce the number of multiplications, the so-called *butterfly_8* is realised with the small 8-point DFT algorithm. Let's calculate now the number of multiplications:

$$M_b = N_{mul-but} * N_{but-stage} * N_{stage} = 4 * 8 * 2 = 64 \text{ real multiplications due to the butterflies,}$$

$$M_t = N_{mul-twi} * (N_{twi-stage} * N_{stage} - N_{trivial-twi}) = 7 * (8 * 1 - 1) = 49 \text{ complex multiplications due to the twiddle.}$$

For the number of additions in the butterflies, we have:

$$A_b = N_{add-but} * N_{but-stage} * N_{stage} = 26 * 8 * 2 = 416 \text{ complex additions.}$$

Related files:

- “/Radix_8_fft/”: working directory;
- “coeff_64.vhd”: package for the coefficients for the 64-point DFT. Because of the Radix-8 algorithm, most of them are not used;
- “splitter.vhd”: it reorders the inputs;
- “first_btfly_8.vhd”, “btfly_8”: structures for the first and the second stage of butterflies, realised with the small 8-point DFT algorithm;
- “twiddle.vhd”: stage of twiddle factors.
- “radix_8_fft.vhd”: top hierarchy file;
- “radix_8_fft_tb.vhd”: test bench for the related FFT holding also the configuration for the current architecture.

Now that all the algorithms used have been described, a comparison between the different structures is understandable, even if the number of point (56 and 64) for the FFT is different. However some explanation is required:

- the 8-point DFT in the Good-Thomas FFT has been realised with no scaling of the coefficients, so that we have only 4 real multiplications; the same thing has been done for the 7-point, so that we have 18 real multiplications;
- the coefficients for 64-point Radix-4 and Radix-8 FFT have been scaled in order to have a range between -127 and $+127$ (8 bit for quantization). Each multiplication for coefficient 1 ($+127$) has been realised with one addition ($127\alpha = 128\alpha - \alpha$);

- the 8-point DFT in the Radix-8 FFT has been realised with no scaling of the coefficients, so that we have only 4 real multiplications.

	Real additions	Real multiplications
56-point Good-Thomas	940	172
64-point Radix-4	544	324
64-point Radix-8	960	260

Table 1 Comparison between the several structures in terms of required numeric operations

3.2. SIMULATION.

As explained in the previous chapter, a test bench for the whole filter bank has been written: it reads input data from the file “*input.txt*” (it must be in the same directory as the test bench) and writes output results in the file “*output.txt*”. The input is serial and the filter bank, as shown in Figure 12, requires a commutator: as it works at high sampling frequency (nearly 600 MHz), in any case GaAs is used for its realisation. For our purposes the commutator is implemented within the test bench architecture. The input is generated using Matlab tool. The output coming from the filter bank is written in the following way: it starts with the first sample (first real and then imaginary part) of the channel number 0 and so on until the last channel is reached; at this point the second sample of the channel number 0 follows and so on. The output file is read again by calling Matlab synthesizer: it recombines the channels and plots the resulting signal which allows performance evaluation of the filter bank analyzer. It has to be noted that, being the Matlab synthesizer a floating point model, all the degrading observed is due only to the fixed point VHDL filter bank analyzer (except for eventual phase rotation due to the filters).

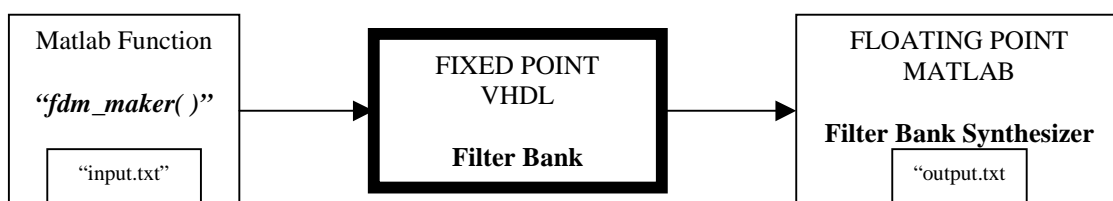


Figure 23 Simulation flow.

3.2.1. NPR.

For this kind of evaluation we generated an FDM signal in the following way: we fill six contiguous channels, then we leave one empty and then again six channels. The SNR at the input, after the ADC converter (in our case performed within the Matlab function), is chosen to be about 40 dB.

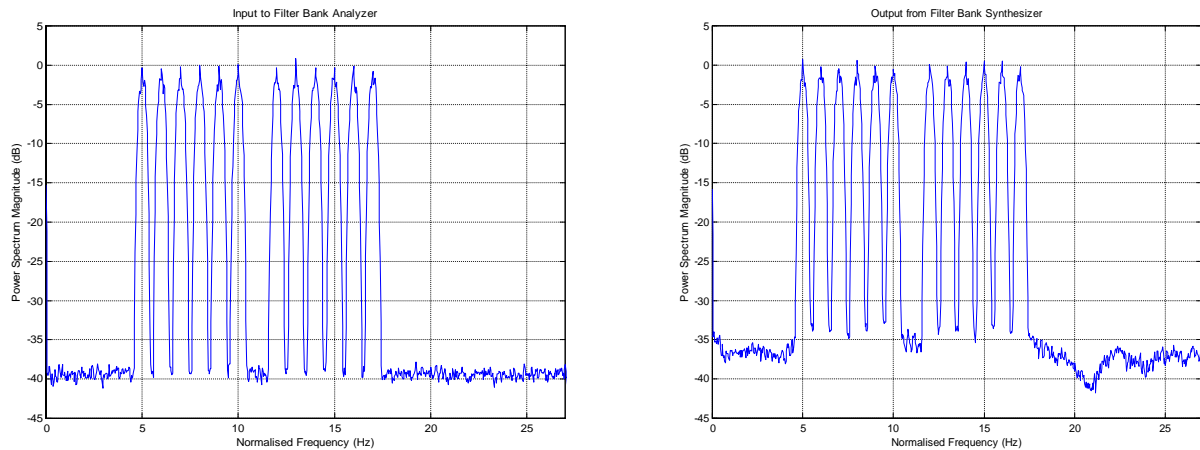


Figure 24 FDM signal as input and output to filter bank analyzer/synthesizer for NPR evaluation.

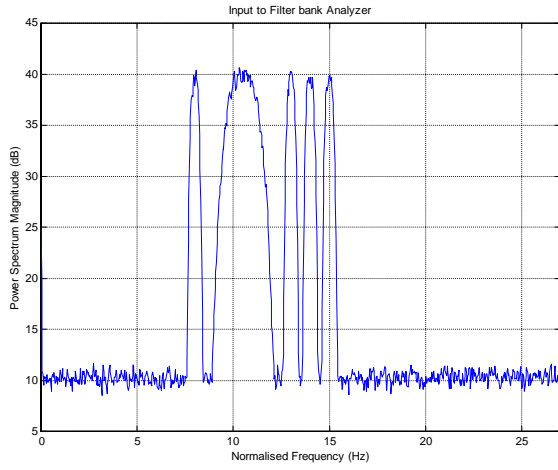
An estimation of the performance degradation due to the quantization, aliasing, adjacent channel interference can be obtained by comparing the noise level in the empty slot: 5dB of difference between input SNR and output SNR.

3.2.2. Scattering and Eye Pattern Diagram.

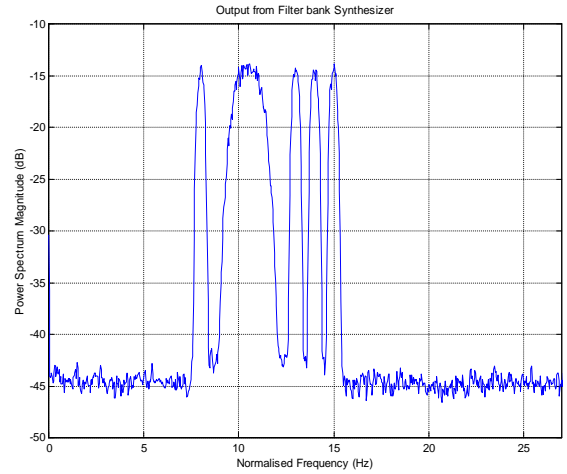
For this kind of evaluation we extract a desired slot from the signal coming from the Matlab synthesizer. This solution is preferable to the one which keeps directly the output from the filter bank analyzer because in this way we have the possibility to examine channels with an occupancy of more than one slot: the single output of the analyzer for this kind of signal has no meaning. The input FDM signal is made by a channel which has four times the bandwidth of a normal one (occupancy=4 slots), surrounded by other channels (occupancy=1 slot); then we look at this wide signal and at a narrow one. Each channel is a QPSK (roll off of 0.7) modulated signal, with baud rate of 10.5 Mhz (42 Mhz for the wider one). The operations over the output FDM signal are:

- complex modulation in order to shift the desired channel in baseband;
- lowpass filtering in order to take away the adjacent channels;
- decimation of the sample rate.

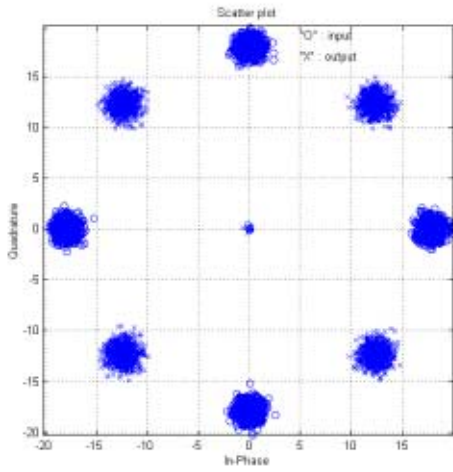
Output channels have been given a rotation of 45 degrees in order to show input and output in the same plot.



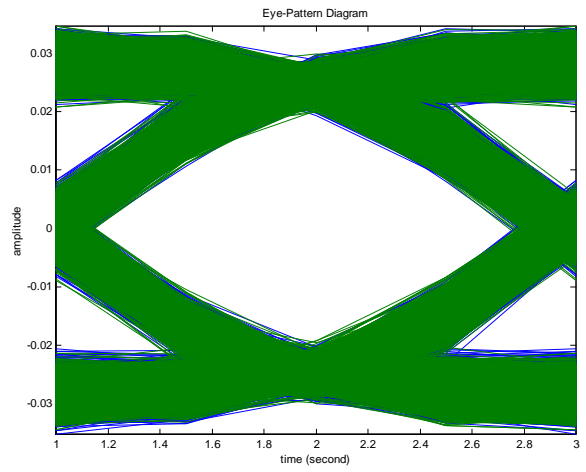
(a)



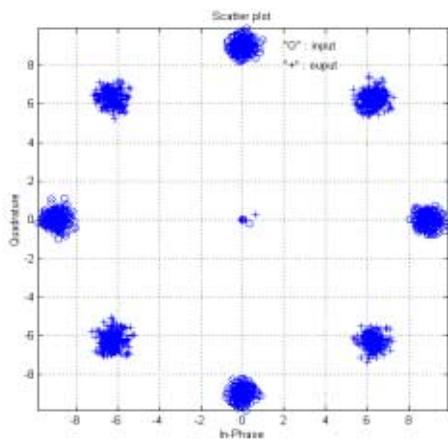
(b)



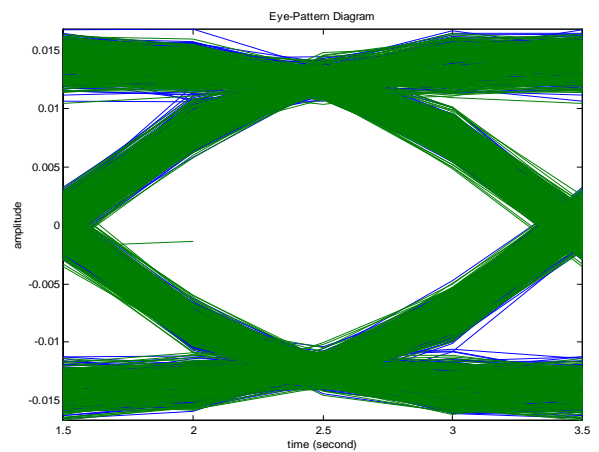
(c)



(d)



(e)



(f)

Figure 25 Scatter diagrams and eye pattern diagrams for the channel centred in 10.5Hz (c,d) and in 13Hz (e,f). Input signal (a) with SNR of 30dB.

3.3. SYNTHESIS.

All structures have been synthesised using Synopsys™ Design Analyzer™ Version 1999.05-2. Compared to the previous version 1998.02-2, this new version offers specific features for large hierarchically structured designs. The old version provided several methodologies and strategies for design synthesis, but for large designs there are essentially two:

- *top-down hierarchical compile* strategy;
- *compile-characterize-write_script-recompile* strategy.

With the *top-down hierarchical compile* strategy Design Compiler automatically compiles hierarchical circuits without collapsing the hierarchy. After each module in the design is compiled, Design Compiler continues to optimise the circuit until the constraints are met. When the performance goals are achieved or when no further improvement can be made, the compile process stops. This kind of approach requires three steps:

- 1) read in the entire design;
- 2) resolve multiple references using the *set_dont_touch* or *uniquify* command. Apply constraints and attributes to the top level;
- 3) compile.

The top-down hierarchical compile strategy is an easy, push-button approach. However for design over 100K gates, like the ones, which have been implemented in this work, it might require much longer than usual runtime.

The *compile-characterize-write_script-recompile* strategy is an alternative to hierarchical compilation. This method allows first to optimise nonunique design using context information or time budgets, then to optimise higher-level blocks with the lower blocks marked as *dont_touch*. It consists in seven steps:

- 1) compile subblocks independently using estimates for drive and load (the best thing is to use a default script to estimate drive and load);
- 2) read in the entire compiled design;
- 3) characterise one subblock;
- 4) use *write_script* to save the information from characterise;
- 5) clear memory, read in the previously characterised subblock and recompile the subblock using the saved script;
- 6) read in the entire compiled design again without the old subblock; use the recompiled subblock;
- 7) choose another subblock and repeat steps 3 through 7 until all subblocks are recompiled using their actual environments.

This method is without doubt better than the previous one in terms of required computer resources when the design are very large. Although it requires much more steps than the top-down hierarchical compile strategy, especially when there are many subblocks within the design, the use of a good script file with loop commands allows the user to save a lot of manual work.

The new version of Synopsys™ has introduced a command called *set_simple_compile_mode*, which will map a design with many repeated blocks much quicker than a top down design. The following example script (a very simple one) shows how it is done:

```
read -format vhdl {my_top_design.vhd}  
reset_design  
include my_top_constraints.dc  
set_simple_compile_mode true  
compile
```

With this command Design Compiler will automatically perform a bottom up compile without the need for manual uniquifying or characterising of subblocks. Multiple instantiated designs are mapped only once. This can be much faster (but possibly less optimal) than a top down compile that will require “uniquifying” and compile each block independently.

A strategy based on the use of this command is fast (for the user) like a top-down hierarchical compile strategy, requires less computer resources like the compile-characterize-write_script-recompile strategy and gives good results for a quick estimation of the performances of the design. These are the reasons for which we have used this kind of approach.

All the synthesis have been performed using MIETEC 0.35µm CMOS Library, selecting WCIND (Worst Case INDustrial) as operating conditions. A part of the implemented structures has been synthesised in two ways:

- using a full library;
- using a reduced library, in order to see the influence of complexity.

Design Compiler requires as a minimum a 2 inputs NAND port, a 2 inputs NOR port, an INVERTER and a 1 bit FLIP-FLOP D-EDGE TRIGGERED. As we wanted to perform a synthesis using only NAND and FLIP-FLOP, we these components as *preferred*, as shown by the following script:

```

set_dont_use MTC45000/* # remove all the components
remove_attribute MTC45000/IV dont_use # use INVERTER
remove_attribute MTC45000/ND2 dont_use # use NAND
remove_attribute MTC45000/NR2 dont_use # use NOR
remove_attribute MTC45000/FD1QM dont_use # use FLIP-FLOP
set_prefer MTC45000/FD1QM # FLIP-FLOP is preferred
set_prefer MTC45000/ND2 # NAND is preferred
    
```

The following tables show the results obtained by synthesizing with both libraries. An improvement of positive (negative) x% in area means that the biggest (smallest) ASIC structure is x% bigger (smaller) than the one it is compared to. When referring to the time, positive (negative) x% means x% slower (faster).

64-point RADIX-4-FFT		
Library	Full	Reduced
Area(gates)	186498	300225
Time(ns)	23.66	22.38
Improv. Area (%)		-37.88
Improv. Time (%)		5.7
Improv. Area*Time (%)		-34.33

Table 2 Result of the synthesis for 64-point Radix-4 FFT.

64-point RADIX-8-FFT		
Library	Full	Reduced
Area(gates)	174495	278314
Time(ns)	33.32	32.93
Improv. Area (%)		-37.3
Improv. Time (%)		1.2
Improv. Area*Time (%)		-36.6

Table 3 Result of the synthesis for 64-point Radix-8 FFT.

56-point Good-Thomas-FFT		
Library	Full	Reduced
Area(gates)	126986	
Time(ns)	24.39	
Improv. Area (%)		
Improv. Time (%)		
Improv. Area*Time (%)		

Table 4 Result of the synthesis for the 56-point Good-Thomas FFT.

FILTER-BANK Analyzer		
Library	Full	Reduced
Area(gates)	240506	Not performed
Time(ns)	23.68	Not performed
Improv. Area (%)		Not performed
Improv. Time (%)		Not performed
Improv. Area*Time (%)		Not performed

Table 5 Result of the synthesis for the Filter-Bank analyzer for real FDM signals.

4. FILTER BANK ANALYZER WITH CSD CODE.

The representation of a number by additions and subtractions of power-of-two is formally known as a radix-2 signed digit code. The radix-2 signed digit (SD) representation of a number x has the general form:

$$x = \sum_{k=1}^L s_k 2^{-p_k} \quad (4.1)$$

where $s_k \in \{-1,0,+1\}$ and $p_k \in [0, M]$. The representation given by Eq.(4.1) has $M+1$ total ternary digit and L nonzero digits.

The main advantage of radix-2 SD code over a conventional radix-2 binary code, such as two's complement, is that the added flexibility of negative digits allows most numbers to be represented with much fewer nonzero digits.

In general there are several signed-digit representations for a given number. the minimal representation refers to a code requiring the minimum number of nonzero digits. There may also be more than one minimal representation. The Canonic Signed Digit representation (CSD), however, is unique. It is defined as the minimal representation for which no two nonzero digits s_k are adjacent. Therefore, the product of an input data by a CSD coefficient allows us to use less or equal number of adders/subtracters than the rest of the signed digit representations of this coefficients, with the consequent saving in terms of hardware. The number of adders/subtracters required to implement a CSD coefficient is one less than the number of nonzero digits in the code (L).

Being a quantization algorithm, CSD code has its own distribution of the rounded coefficients. A uniform distribution of the rounded coefficients is the one that minimises the maximum quantization error. Two's complement is the most common example of a uniformly distributed number representation. The distribution of CSD code is, however, very non-uniform. Furthermore, if the number of nonzero digits in the CSD code is kept fixed, the size of the gaps between two consecutive CSD codes will not be reduced, even by approaching the wordlength of the CSD code to infinity.

The aim of this part of the work is to use CSD code for the complete filter bank, in order to avoid multiplication within the structure; the code will be used for both the imaged half-band and the polyphase filters and for the FFT. Concerning the FIR filters, a work has been carried out at ESTEC, in the TOS-ESM Section before. It dealt with generation of all possible SD/CSD coefficients belonging to interval $\{-1,+1\}$ and with choosing a subset representing the coefficients for the FIR filter, based on different optimisation criteria. The most popular methods for SD/CSD coefficient optimisation are the MILP (Mixed Integer Linear Programming) and local search. The latter one,

involving less computational resources than the MILP and giving similar results, has been chosen in this and in the previous study. The strategy is made in two steps: the *scale-factor search* and the *vary-strategy*.

In the *scale factor strategy* the set of the coefficient representing the impulse response of the filter is multiplied by a factor in the range [0.5,1]; considerable improvement are achieved during this optimisation process. This is due to the fact that the set of CSD coefficients is very non-uniformly distributed, and each time we multiply the impulse response by a new scale-factor we obtain a different set of SD/CSD coefficients with different quantization errors.

The *vary-strategy* performs a search in the neighbourhood of the rounded coefficients in order to improve the performance criteria. Two different kinds of algorithm have been carried out: the *single-variate-strategy* (changing one coefficient per iteration) and the *bivariate-strategy* (changing two coefficients per iteration). The optimisation criteria has to be chosen, depending on the value of the SD/CSD coefficient: FIR filters (frequency selective, matched filters), FFT and so on.

4.1. C++ Program for general DFT.

The function of the program called "*csdffft.exe*" is to optimise the coefficients involved in the computation of an N-point DFT. Two different *optimisation criteria* can be selected:

- maximisation of the SNR (Signal to Noise Ratio), optimising *rms* error;
- maximisation of the SFDR (Spurious Frequency Dynamic Range), optimising *max* error.

An N-sample sine wave, sampled in a way to avoid spectral leakage, represents the input to the FFT.

The principal steps of the program are:

- a) the user is prompt for several parameters to be introduced (number of the point of the FFT, number of digits for CSD code,...);
- b) generation of the table of all the possible values of SD/CSD numbers belonging to [-1,+1];
- c) scaling of the coefficients resulting in the best scale factor;
- d) variation of the coefficients of +/- one quantization step by means of a chosen *variation-strategy*.

This is performed as long as the value of the optimisation criteria of two consecutive trials differs by more than a programmable convergence parameter;

- e) writing of the file containing the coefficients (code and value).

Concerning the variation strategy, two different algorithms have been realised:

- *single-variation strategy*: only one coefficient at a time is varied. As the number of coefficients is N (for an N-point DFT), each time more than one of them varies in the DFT-matrix. At the end we have N different sets; we select from them the one which maximises the optimisation criteria.

At this point the program starts again the variation, without touching the coefficients already varied, until no further improvement is obtained.

- *Bivariate strategy*: two coefficients at a time are varied. More computational effort is required compared to the previous one.

It has to be noted that the coefficients involved in the computation of the DFT are complex, based on a pair of real number; therefore each time one of them is varied, 4 different pairs have to be managed. Other possible strategies could be the same already examined, with the difference that each coefficient of the matrix representing the DFT is considered different from all the other ones. As computational time and memory required increase exponentially with N, they have not been taken in account.

4.2. C++ Program for Good-Thomas FFT.

In this case the DFT is no longer representable by a matrix because of the presence of a multistage structure and of the small N-point DFT algorithm. What we deal with are two sets of coefficients: the first one for the small 8-point DFT and the second one for the small 7-point DFT. With same slight modifications of the previous program, we obtain the “*csd56gt.exe*” for this particular FFT. However there is a little difference due to the presence of the small 8-point DFT. As seen from its algorithm, if no scaling is performed there is the possibility to reduce the number of multiplications to 4 (replaced with additions by coding them in SD/CSD). For this reason, the user can choose whether or not to scale the coefficients of the 8-point DFT. The variation-strategies work independently on both the FFT coefficients: with the single variation, one coefficient of each set per time is varied and so on; unlike the general DFT, the sets of number are smaller and only real, so this computation requires less time than the previous one. At the end, the program generates also the VHDL files containing the packages for the respective small DFT. The following flow charts correspond to the program for general N-point DFT; the one for the 56-point Good-Thomas is very similar.

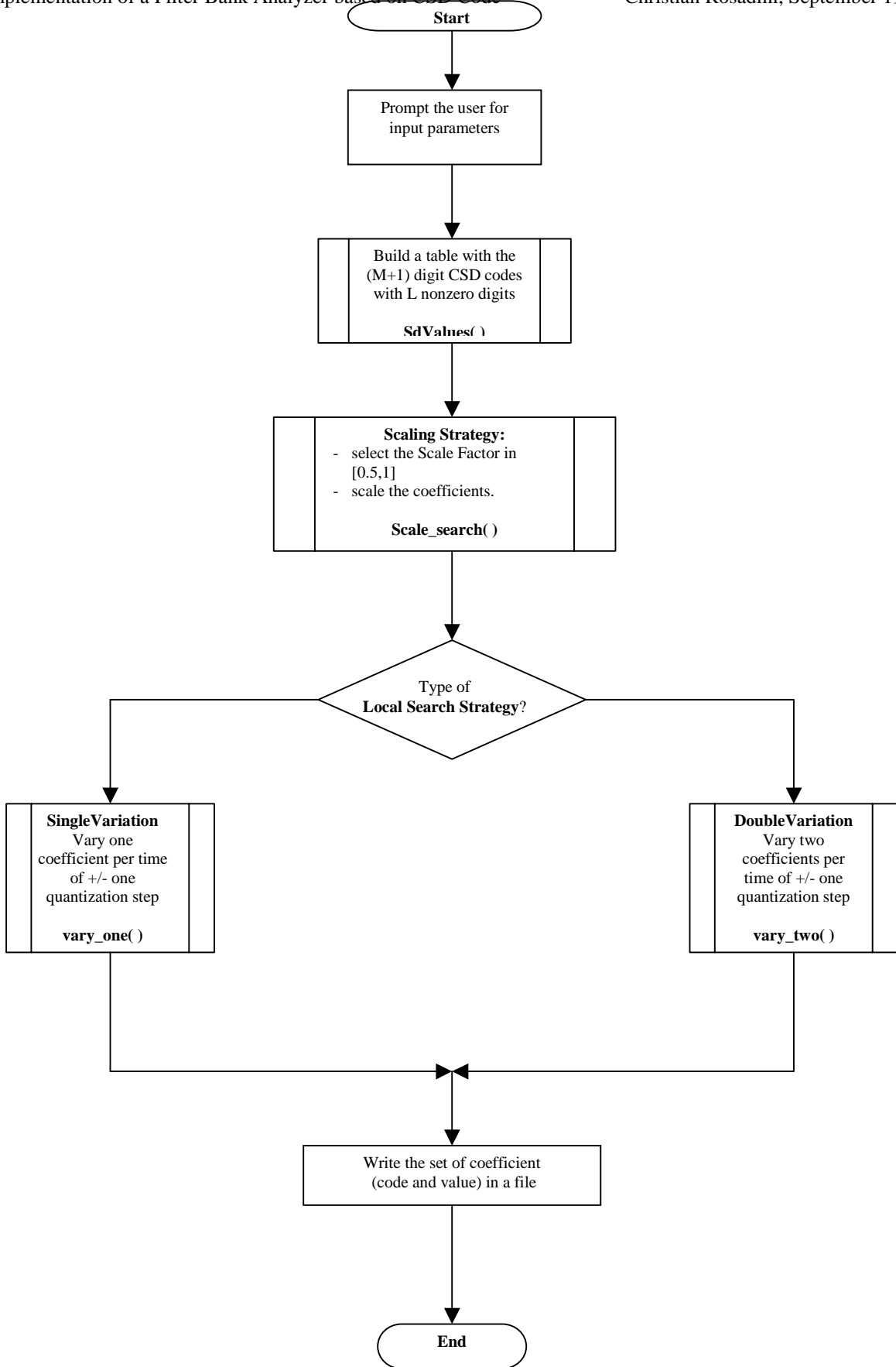


Figure 26 Flow chart of the “csdffft.exe”.

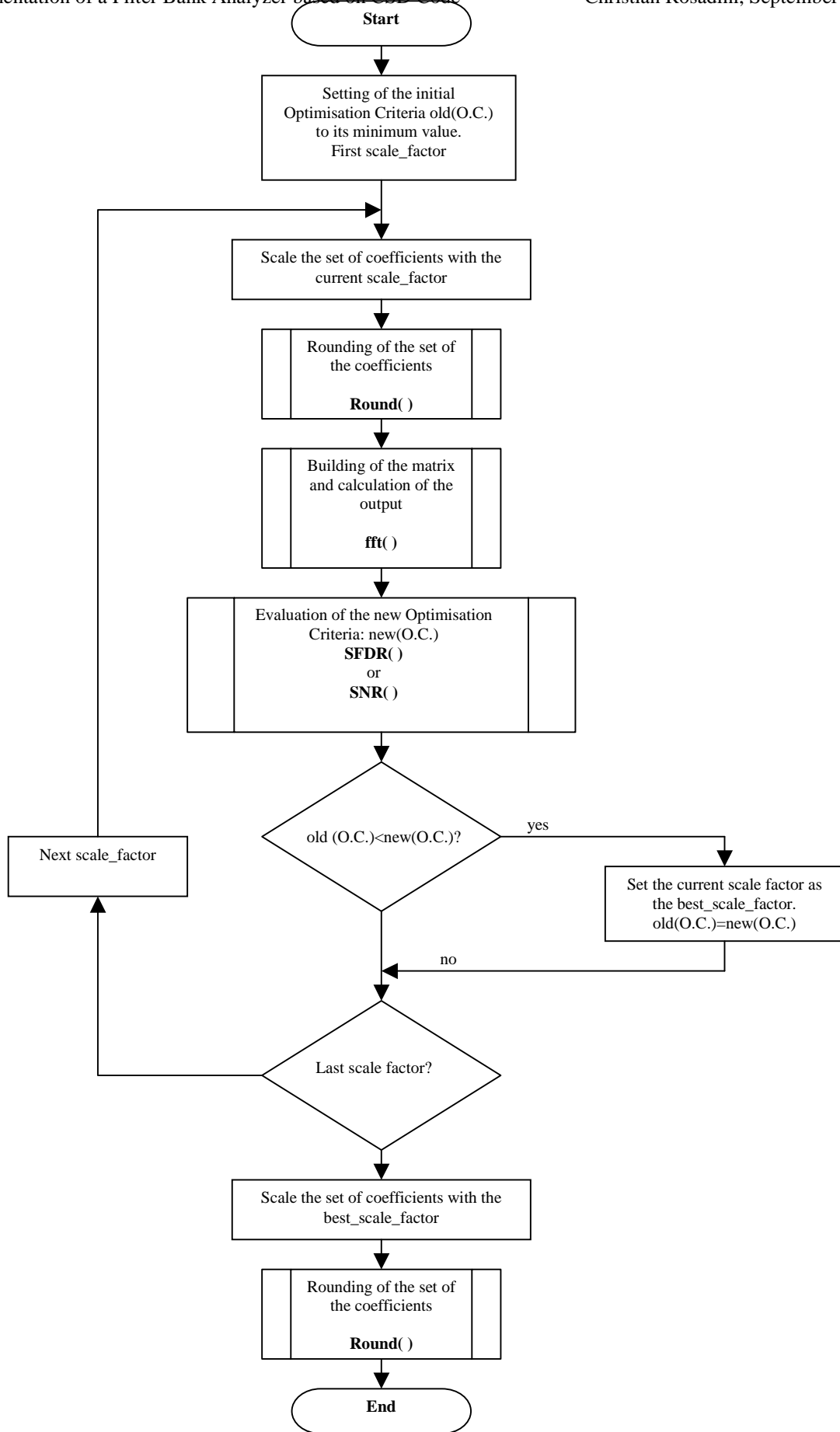


Figure 27 Flow chart of the Scale Factor strategy; function "scale_search".

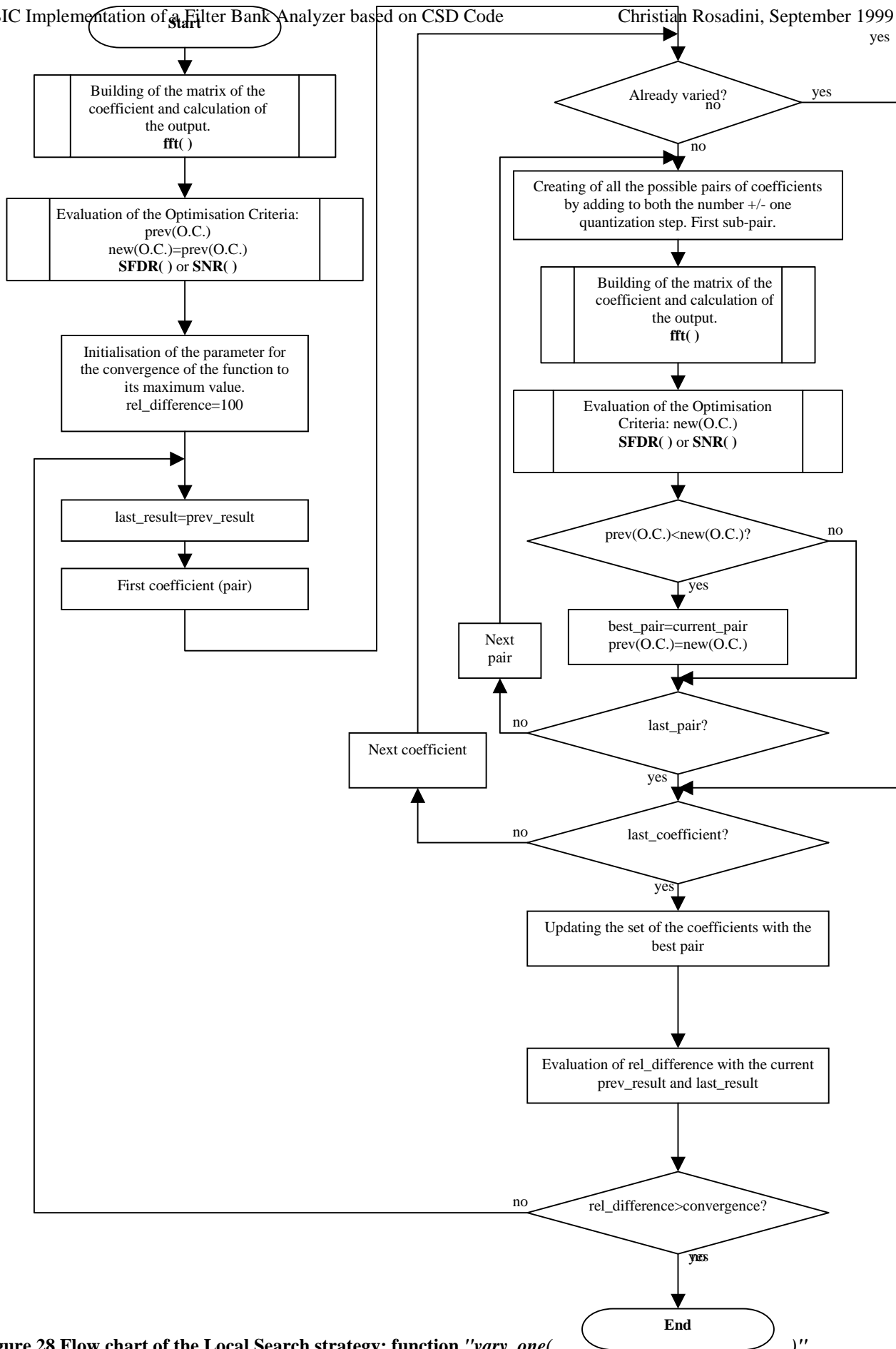


Figure 28 Flow chart of the Local Search strategy: function "vary_one()".

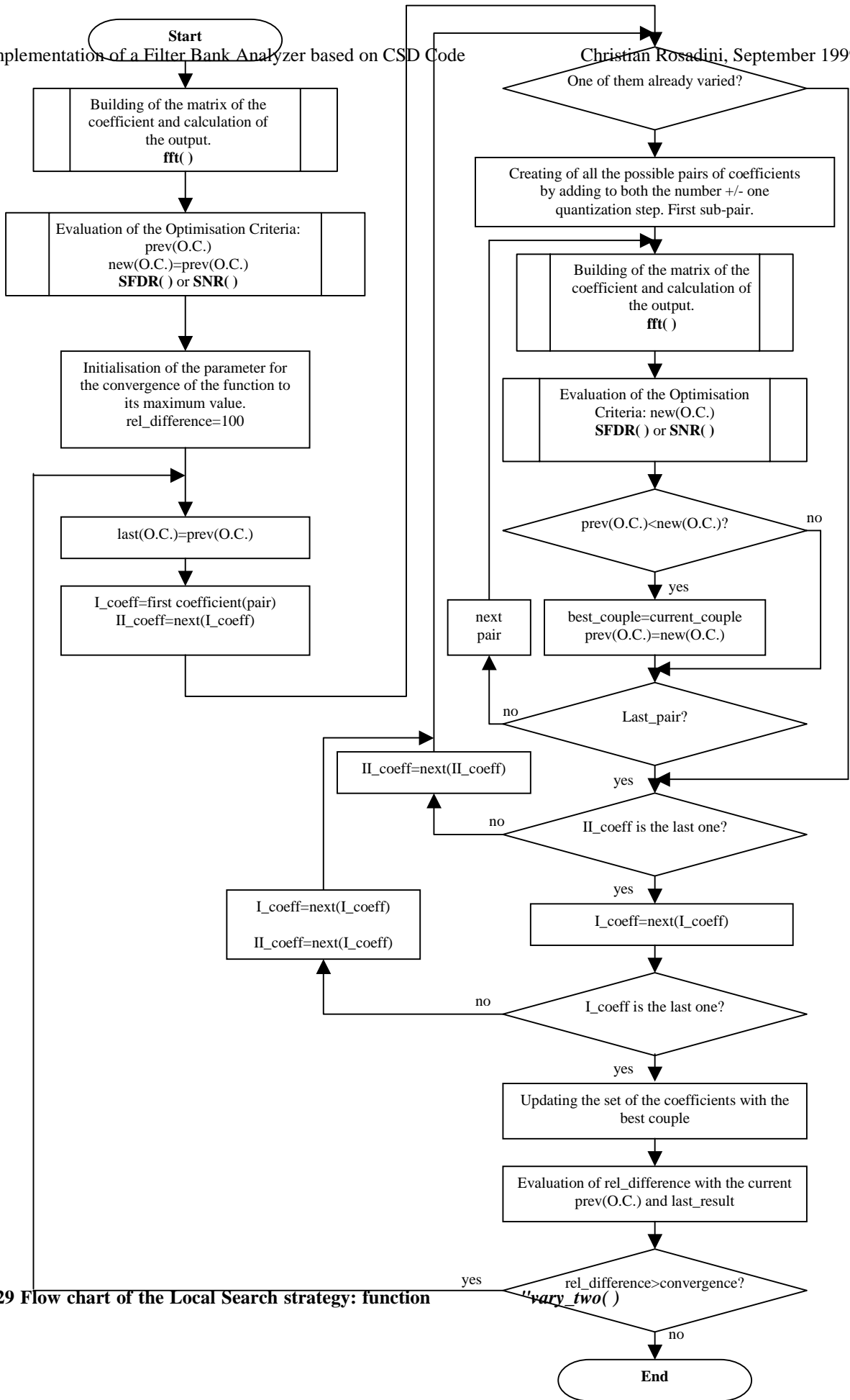


Figure 29 Flow chart of the Local Search strategy: function

5. RTL VHDL MODEL OF SD/CSD FILTER BANK.

5.1. DESIGN.

By means of the coefficients coded in SD/CSD format it is now possible to replace each multiplier used within each structure (both filter and DFT) with adders; the number of adders required for every single multiplier depends on how many non-zero digits we use to code the operand while the precision depends on the total number of digits. Based on previous work [4] carried out in the TOS-ESM Section on the development of the SD/CSD FIR filters, information, like the number of digits (9) and the number of non-zero digits (2; 3 for those coefficients with magnitude larger than 0.5 after normalisation of the maximum coefficient to the unity) was obtained. From the same work VHDL models of the "CSD-multiplier" have been obtained: it is based on adders and shifters depending on the position of the non-zero digits. Logic synthesis removes the "Shifter".

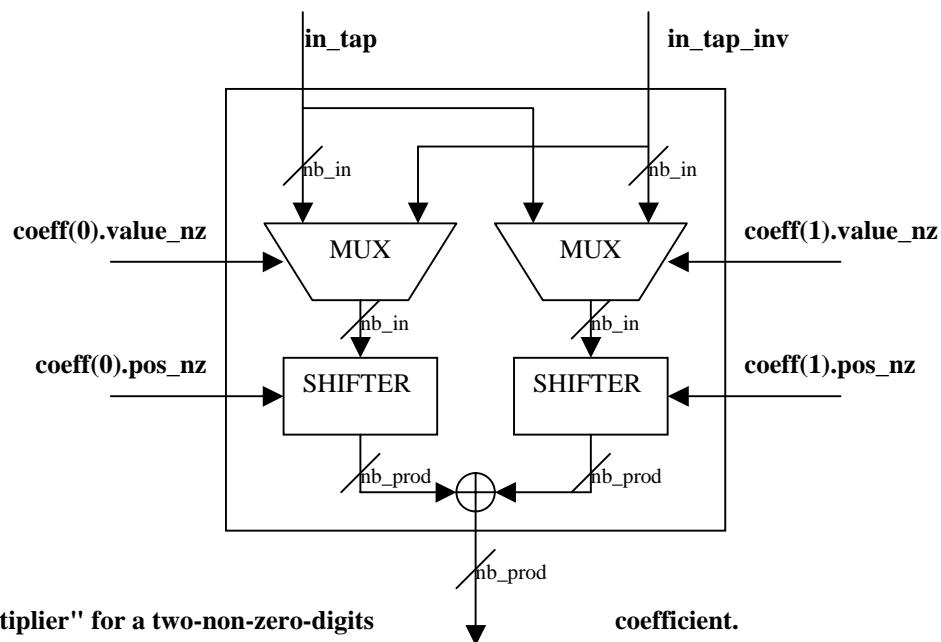


Figure 30 "CSD-multiplier" for a two-non-zero-digits

The coefficients for the half-band filter and for the polyphase filter have been optimised with the Matlab program "*csdfir.m*" (a result from the the previous work), while the ones for the DFT come from the C++ program "*csd56gt.exe*". Two architectures have been implemented for the different blocks (filters, FFT): one based on 2's complement and the other one based on SD/CSD code for the coefficients.

The configuration for all the following blocks is held in the configuration file related to the top entity “filter_bank”.

5.1.1. Imaged HB Filter

Related files:

- “*hb_csd_coeff.vhd*”: respective package for the coefficients;
- “*hb_csd_mul.vhd*”: 9 different taps with non-zero coefficients are required for the half band filter;
- “*csd_hb.vhd*”: file containing the CSD architecture;

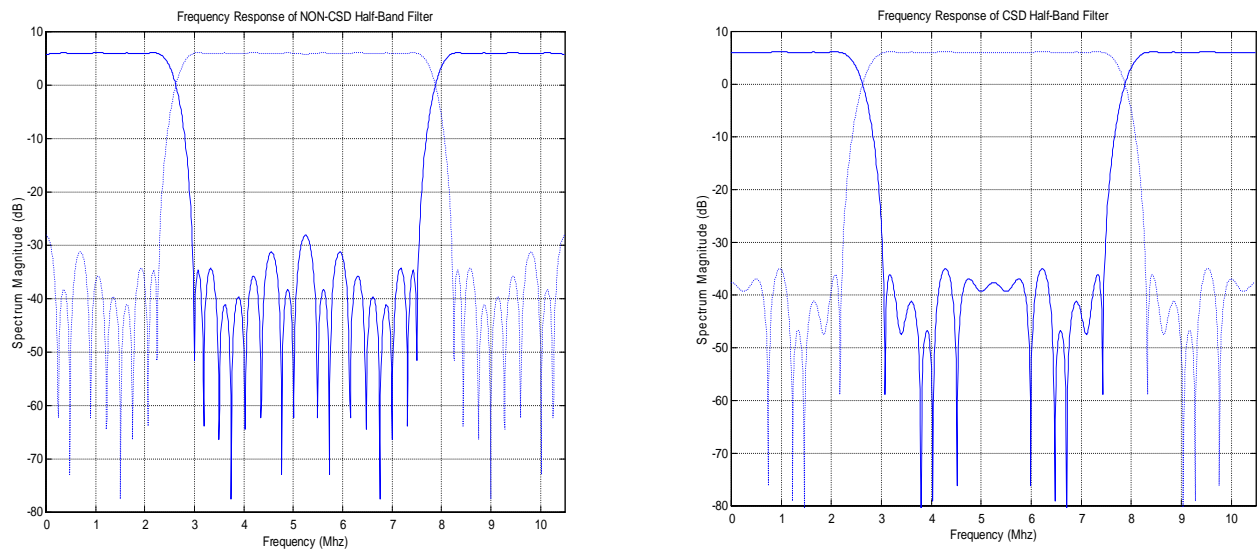


Figure 31 Half band filters: low-pass and high-pass. Quantization using 8-bit for NON-CSD and 9-bit (with 2+1 non-zero digits) for the CSD half band filter.

5.1.2. Polyphase Filter.

Related files:

- “*polyphase_csd_coeff.vhd*”: package holding the coefficients;
- “*polyphase_csd_mul.vhd*”: “multiplier” made by 6 taps (168 taps of the reference filter divided by 28 polyphase filters);
- “*csd_even_polyphase.vhd*”, “*csd_odd_polyphase.vhd*”: files containing the respective SD/CSD architecture.

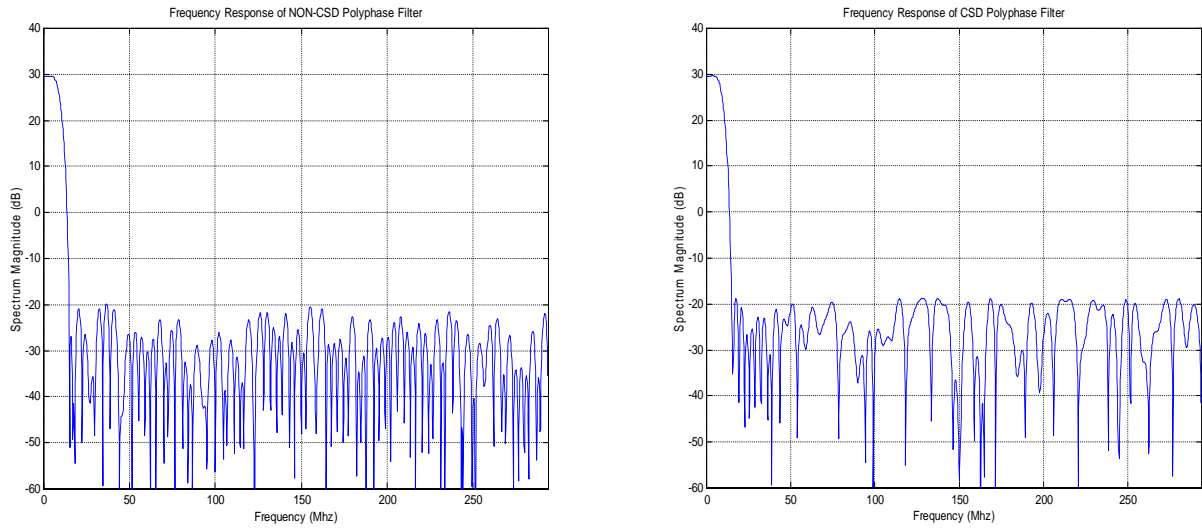


Figure 32 Polyphase filters. Quantization using 8-bit for NON-CSD and 9-bit (with 2+1 non-zero digits) for the CSD half band filter.

5.1.3. Good-Thomas FFT.

Related files to the general form:

- “*csd_coeff_7.vhd*”, “*csd_coeff_8.vhd*”: packages;
- “*eight_csd_mul.vhd*”, “*seven_csd_mul.vhd*”;
- “*csd_eight_fft.vhd*”, “*csd_seven_fft.vhd*”: files holding the respective CSD architecture;
- “*csd_Good_fft_cfg.vhd*”, “*non_csd_Good_fft_cfg.vhd*”: configuration files;
- “*Good_fft_tb.vhd*”: test-bench. Its configurations are held in the same file.

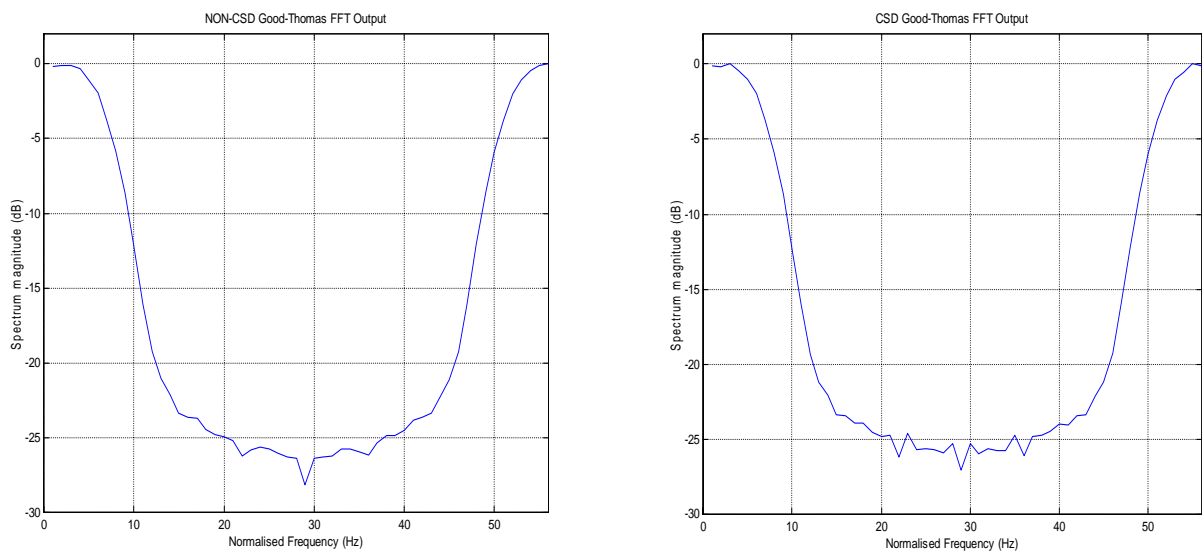


Figure 33 Spectrum of a QPSK modulated signal obtained by both the FFTs; the signal has 4 samples per symbol.

Related files to the reduced Good-Thomas FFT and to the complete filter bank:

- “*csd_coeff_7.vhd*”, “*csd_coeff_8*”: packages;
- “*eight_csd_mul.vhd*”: this set of “multipliers” is different from the previous one, because it is based on 4 multiplications, the ones with the coefficient different from 1;
- “*seven_csd_mul.vhd*”;
- “*csd_partial_8_fft.vhd*”, “*csd_seven_fft.vhd*”: files holding the respective CSD architecture;
- “*csd_filter_bank_cfg.vhd*”, “*non_csd_filter_bank_cfg.vhd*”: configuration files;
- “*filter_bank_tb.vhd*”: test-bench. Its configurations are hold in the same file.

5.2. SIMULATION.

The simulation for the implemented SD/CSD-structure has been carried out in the same way as the previous non-CSD one, and with the same inputs (see Chapter 3.2).

5.2.1 NPR.

By a graphical interpretation of the results in Figure 33 we obtain a gap of about 7dB between input SNR and output SNR.

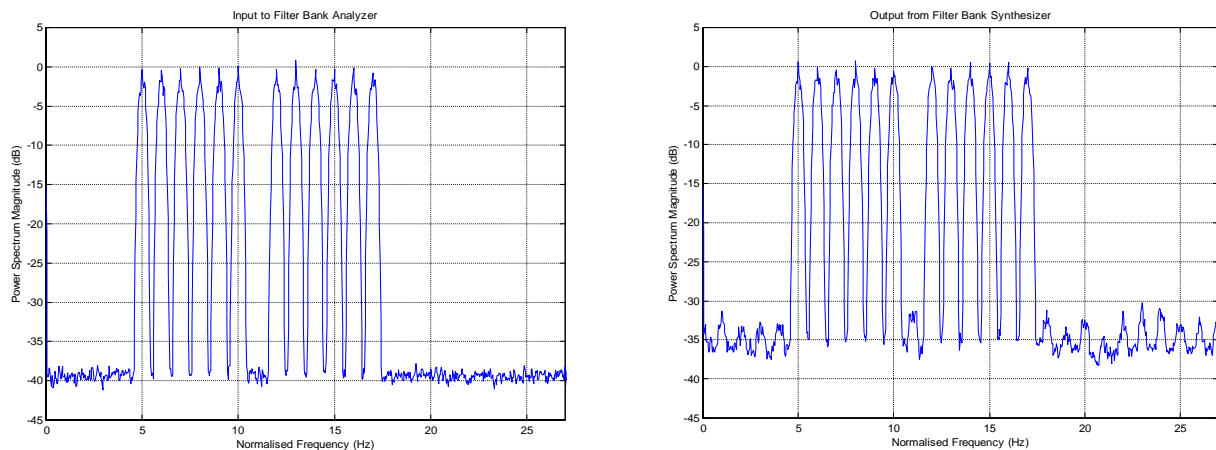
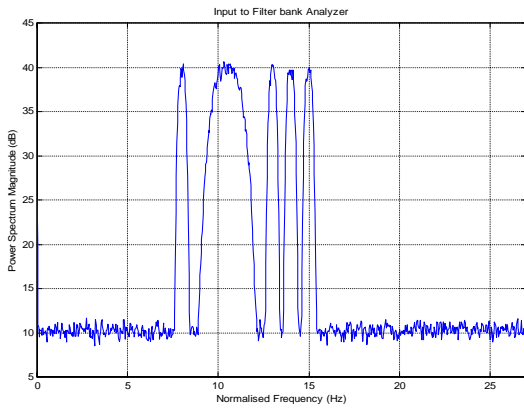


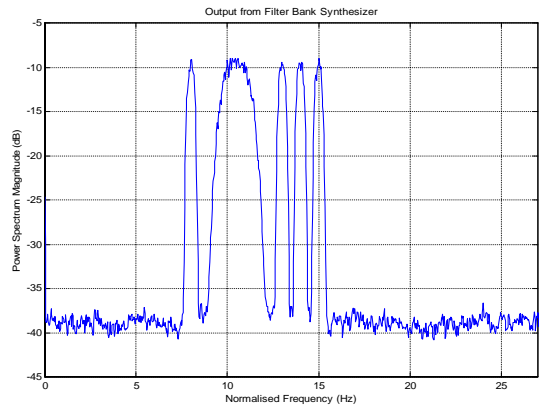
Figure 34 FDM signal as input and output for NPR evaluation with CSD-Filter-Bank.

5.2.2. Scattering Diagram and Eye Pattern Diagram.

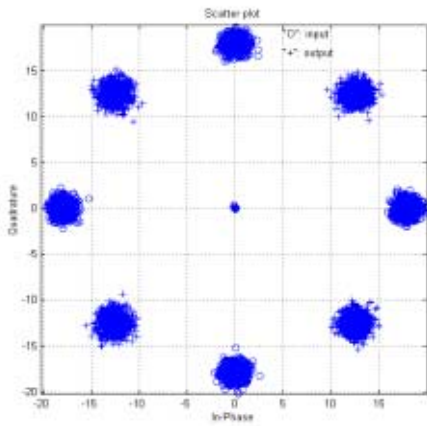
In order to show both input and output channels in the same plot, we rotated the output by 45 degrees.



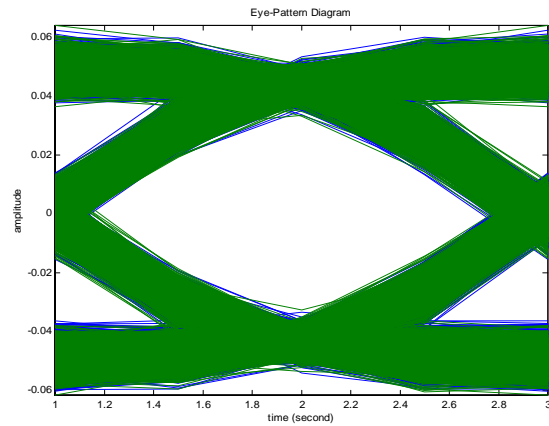
(a)



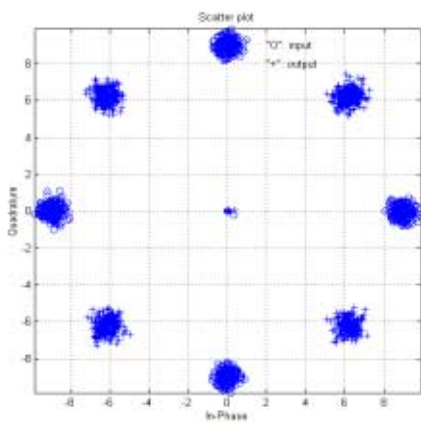
(b)



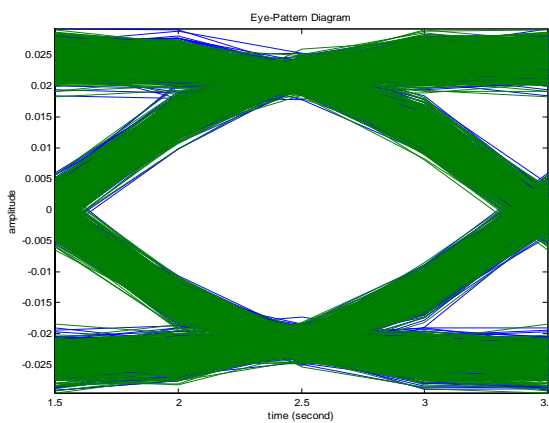
(c)



(d)



(e)



(f)

Figure 35 Scatter diagrams and eye pattern diagrams for the channel centred in 10.5Hz (c,d) and in 13Hz (e,f). Input signal (a) with SNR of 30dB.

5.3. SYNTHESIS.

The synthesis has been carried out in a similar manner as we have done for the non-CSD filter bank analyzer. The two architectures are compared in the same table in order to see what kind of improvement we can reach by using either.

56-point Good-Thomas FFT		
Architecture	NON-CSD	CSD
Area(gates)	126986	120906
Time(ns)	24.39	23.77
Improv. Area (%)		5
Improv. Time (%)		2.6
Improv. Area*Time (%)		7.8

Table 6 Result of the synthesis for the Good-Thomas FFT.

FILTER-BANK Analyzer		
Architecture	NON-CSD	CSD
Area(gates)	240506	216723
Time(ns)	23.68	23.4
Improv. Area (%)		11
Improv. Time (%)		1.2
Improv. Area*Time (%)		12.3

Table 7 Result of the synthesis for the Filter-Bank analyzer for real FDM signals.

Conclusion

Due to the lack of the period, very little effort was spent on synthesys analysis. More investigation is required to analyze the results obtained and to compare to [5]. Moreover the optimum scaling of the FFT should be verified.

REFERENCE:

- [1] R. E. Crochiere, L. R. Rabiner, "Multirate Digital Signal Processing", Prentice-Hall Signal Processing Series, 1983.
- [2] D. F. Elliott, K. R. Rao, "Fast Transforms: Algorithms Analyses Applications", Academic Press, 1982.
- [3] H. J. Nussbaumer, "Fast Fourier Transform and Convolution Algorithms", Springer Series in Information Sciences, 1982.
- [4] R. E. Blahut, "Fast Algorithms for Digital Signal Processing", Addison Wesley, 1985.
- [5] M. F. Tarrega, "ASIC Implementation of Canonic Signed Digit FIR Filters", Final Report, TOS-ESM Division, 1998.
- [6] P. Trinadh, "RTL Level VHDL Model for Radix-4 Pipeline FFT Processor", Final Report, XRM Division, 1996.
- [7] P. S. Cornfield, "Wideband Processor Architecture", Engineering Specification, MMS, 1997.