Institutionen för systemteknik
Department of Electrical Engineering

Examensarbete

VHDL Implementation of Flexible Frequency-Band Reallocation (FFBR) Network

Master thesis performed in Electronics Systems

by

Abrar Hussain Shahid

LiTH-ISY-EX--11/4466--SE

Linköping 2011
VHDL Implementation of Flexible Frequency-Band Reallocation (FFBR) Network

Master thesis in Electronics Systems at Linköping Institute of Technology

by

Abrar Hussain Shahid

LiTH-ISY-EX--11/4466--SE

Supervisor: Amir Eghbali

Examiner: Kent Palmkvist

Linköping 2011-05-25
**Språk**
- _____ Svenska
- X _____ Annat (Ange nedan)
- Engelska
- 60 _____ Antal sidor

**Typ av publikation**
- _____ Licentiatavhandling
- X _____ Examensarbete
- ____ C-uppsats
- ____ D-uppsats
- ____ Rapport
- ____ Annat (ange nedan)

**ISBN (licentiatavhandling)**

**ISRN:** LITH-ISY-EX--11/4466--SE

**Serietitel (licentiatavhandling)**

**Serienummer/ISSN (licentiatavhandling)**

**URL för elektronisk version**

http://www.ep.liu.se

**Title:** VHDL Implementation of Flexible Frequency-Band Reallocation (FFBR) Network

**Författare/Author:** Abrar Hussain Shahid

**Sammanfattning/Abstract**

In digital communication systems, satellites give us worldwide services. These satellites should effectively use the available bounded frequency spectrum and, therefore, to carry out flexible frequency-band reallocation (FFBR), on-board signal processing implementation on FFBR network is needed. In the future, to design desired dynamic communication systems, very flexible digital signal processing structures will be needed. The hardware, in the system, shall not be changed as simple changes in the software will be made.

The purpose of this thesis is to implement a $N$-channel FFBR network, where $N = 20$. A 20-channel FFBR network consists of different blocks, e.g., DFT, IDFT, complex multipliers, input/output commutators and polyphase components.

The whole 20-channel FFBR network will be implemented in VHDL. In a 20-channel FFBR network, it is a 20-point FFT/IFFT required. This 20-point FFT/IFFT is built by a combination of radix-4 and radix-5 butterflies. The Cooley-Tukey FFT algorithm is chosen to build the 20-point FFT/IFFT. The main aim is to build 20-point FFT/IFFT. There are 20 complex multipliers before the IFFT block and 20 complex multipliers after the IFFT block. In the same way, 20 complex multipliers are used before the FFT block and 20 complex multipliers are used after the FFT block. At the input/output to this FFBR network, 20 FIR filters (polyphase components) are used, respectively.

**Nyckelord/Keywords**

20-channel FFBR network, 20-point FFT/IFFT, complex multiplier, polyphase components, radix-2, radix-5
ABSTRACT

In digital communication systems, satellites give us worldwide services. These satellites should effectively use the available bounded frequency spectrum and, therefore, to carry out flexible frequency-band reallocation (FFBR), on-board signal processing implementation on FFBR network is needed. In the future, to design desired dynamic communication systems, very flexible digital signal processing structures will be needed. The hardware, in the system, shall not be changed as simple changes in the software will be made.

The purpose of this thesis is to implement an $N$-channel FFBR network, where $N = 20$. A 20-channel FFBR network consists of different blocks, e.g., DFT, IDFT, complex multipliers, input/output commutators and polyphase components.

The whole 20-channel FFBR network will be implemented in VHDL. In a 20-channel FFBR network, it is a 20-point FFT/IFFT required. This 20-point FFT/IFFT is built by a combination of radix-4 and radix-5 butterflies. The Cooley-Tukey FFT algorithm is chosen to build the 20-point FFT/IFFT. The main aim is to build 20-point FFT/IFFT. There are 20 complex multipliers before the IFFT block and 20 complex multipliers after the IFFT block. In the same way, 20 complex multipliers are used before the FFT block and 20 complex multipliers are used after the FFT block. At the input/output to this FFBR network, 20 FIR filters (polyphase components) are used, respectively.
ACKNOWLEDGMENT

I would like to thank my supervisor Amir Eghbali, who guided me so that I was able to solve problems in an easy way during my thesis. Many thanks for his availability when it was needed and for his effort to make this thesis work interesting.

I would like to thank my examiner Associate Professor Kent Palmkvist, for his glorious technical support and guidance. He was always available when I got any problem. Moreover, he also helped me to write a technical thesis report.

In addition, I would like to thank my friends Zaka Ullah and Kristian Stavåker, for proof reading my thesis report and for their moral support. Also, I am thankful to all my friends who gave me moral support during my thesis.

I express my marvellous gratitude to my parents for the moral support, prayers and love. Last but not the least, I express my tremendous gratitude to my wife and my children for their love and support. Without their support I would not have been able to complete my thesis.
I dedicate this thesis to my beloved mother
# ACRONYMS AND ABBREVIATIONS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>BER</td>
<td>Bit Error Rate</td>
</tr>
<tr>
<td>CFA</td>
<td>Common- Factor Algorithm</td>
</tr>
<tr>
<td>DFT</td>
<td>Discrete Fourier Transform</td>
</tr>
<tr>
<td>DIF</td>
<td>Decimation-In-Frequency</td>
</tr>
<tr>
<td>DIT</td>
<td>Decimation-In-Time</td>
</tr>
<tr>
<td>EVM</td>
<td>Error Vector Magnitude</td>
</tr>
<tr>
<td>FIR</td>
<td>Finite-length Impulse Response</td>
</tr>
<tr>
<td>FFBR</td>
<td>Flexible Frequency-Band Reallocation</td>
</tr>
<tr>
<td>FFT</td>
<td>Fast Fourier Transform</td>
</tr>
<tr>
<td>GB</td>
<td>Granularity Band</td>
</tr>
<tr>
<td>IDFT</td>
<td>Inverse Discrete Fourier Transform</td>
</tr>
<tr>
<td>IFFT</td>
<td>Inverse Fast Fourier Transform</td>
</tr>
<tr>
<td>MF/TDMA</td>
<td>Multiple Frequency/ Time Division Multiple Access</td>
</tr>
<tr>
<td>MIMO</td>
<td>Multi-Input Multi-Output</td>
</tr>
<tr>
<td>PFA</td>
<td>Prime- Factor Algorithm</td>
</tr>
<tr>
<td>PFBR</td>
<td>Perfect Frequency-Band Reallocation</td>
</tr>
<tr>
<td>RPFA</td>
<td>Rader Prime- Factor Algorithm</td>
</tr>
<tr>
<td>SISO</td>
<td>Single-Input Single-Output</td>
</tr>
</tbody>
</table>
# Contents

ABSTRACT ........................................................................................................................................... I  
ACKNOWLEDGMENT .................................................................................................................. II  
ACRONYMS AND ABBREVIATIONS ................................................................................................ IV  

## CHAPTER 1 FLEXIBLE FREQUENCY-BAND REALLOCATION (FFBR) NETWORK .................. 1  

1.1 Background .................................................................................................................................. 1  
  1.1.1 On-board signal processing architectures ............................................................................... 2  
  1.1.2 Configuration MIMO FFBR Network .................................................................................... 3  
1.2 Variable Oversampled Complex Modulated FBs for FFBR Network ........................................... 3  
  1.2.1 FFBR Network (An Efficient Realization) ............................................................................. 4  

## CHAPTER 2 INTRODUCTION TO DIGITAL FILTERS .................................................................. 7  

2.1 FIR FILTERS ............................................................................................................................... 7  
  2.1.1 Direct Forms ......................................................................................................................... 8  
  2.1.2 Cascade Form ....................................................................................................................... 9  
2.2 FIR Digital Filter Design ........................................................................................................... 9  
  2.2.1 The FIR Filter Order Estimation ......................................................................................... 9  
  2.3 Polyphase Decomposition Realization .................................................................................. 10  
  2.4 Linear-phase FIR Filters .......................................................................................................... 11  

## CHAPTER 3 FOURIER TRANSFORMS ......................................................................................... 13  

3.1 The DFT Algorithms .................................................................................................................... 14  
  3.1.1 The DFT’s Properties ............................................................................................................ 14  
  3.1.2 The Goertzel Algorithm ....................................................................................................... 17  
  3.1.3 The Rader Algorithm .......................................................................................................... 18  
  3.1.4 The Winograd DFT Algorithm ........................................................................................... 19  
3.2 The Fast Fourier Transform (FFT) Algorithms ...................................................................... 20  
  3.2.1 The Cooley-Tukey FFT Algorithm .................................................................................... 21  
  3.2.2 The Good-Thomas FFT Algorithm .................................................................................... 22  
  3.2.3 The Winograd Algorithm .................................................................................................. 24  
  3.2.4 Comparison of DFT and FFT Algorithms ........................................................................ 25
CHAPTER 4 IMPLEMENTATION OF FFBR NETWORK ......................................................... 27

4.1 FIR (Polyphase Decomposition) Filters ..................................................................... 28
4.2 Decimation-in-Time FFT Algorithms ......................................................................... 28
  4.2.1 Radix-2 Cooley-Tukey Algorithm Implementation .............................................. 30
4.3 Radix-5 Algorithm Implementation .......................................................................... 34
4.4 Implementation Of Twenty point FFT/IFFT ............................................................... 37

CHAPTER 5 CONCLUSION AND FUTURE WORK ................................................................. 41

REFERENCES .................................................................................................................. 43

APPENDIX A ..................................................................................................................... 45

APPENDIX B ..................................................................................................................... 47

APPENDIX C ..................................................................................................................... 49

APPENDIX D ..................................................................................................................... 50
1

FLEXIBLE FREQUENCY-BAND REALLOCATION (FFBR) NETWORK

1.1 Background
The Flexible Frequency-Band Reallocation (FFBR) network is also referred to as frequency multiplexing and demultiplexing [1, 2]. Three main network structures have been suggested for broadband satellite-based systems by the European space agency [3]. In the FFBR network, the communication between satellites and users occurs with multiple spot rays. Therefore, for effective reuse of the limited frequency spectrum is needed by the satellite on-board signal processing [3-7].

The satellite on-board signal processor has a digital part which is a Multi-Input Multi-Output (MIMO) system. Generally, the number of input signals could differ from the number of output signals and these input/output signals can have different bandwidths and bit rates, for instance, users from different telecommunication standards [8]. Above, the next generation’s satellite-based communication systems were mentioned which have to support different communication and connectivity scenarios. For example, multiple frequency/time division multiple access (MF/TDMA) scheme is such a scenario. The bandwidths of different users may vary with time and the bandwidths can be controlled, if the input ray is divided into several granularity bands (GBs). Any user can take up any number of GBs at any time. There are four main demands on FFBR networks:

i. **Flexibility**: To support several telecommunication scenarios and standards in digital signal processing, flexible digital signal processing structures are needed. No limiting conditions shall be imposed on the hardware.

ii. **Low complexity**: There is a need to minimize the implementation cost. It is predicted that it needs to improve in terms of system capacity and implementation complexity [3].
iii. Near Perfect Frequency-Band Reallocation (PFBR) which helps to fulfil the communication performance metric, that is, Bit Error Rate (BER) and Error Vector Magnitude (EVM), and so on [9, 10].

iv. Simplicity, for system design and system analysis.

On-board signal processor’s major role, in frequency spectrum, is to reallocate all frequency bands (subbands) to different output signals and positions. In Figure 1.1 it is shown that the number of different users which are using different bandwidths at the input of the FFBR networks have to reallocate to different positions at the output in the frequency spectrum. The bandwidth and position of every user could vary in time, if the system is dynamic [3].

![Figure 1.1: FFBR Network](image)

Figure 1.1. FFBR Network, input signal 1 and input signal 2 can be reallocated to any position in output signal 1, output signal 2, and output signal 3.

1.1.1 On-board signal processing architectures

On-board signal processing architectures can be divided into four different types, that is, bentpipe, partial processing, hybrid, and full processing [11]. In Figure 1.1, it was shown how the bentpipe architecture (payload) operates. The different users with different bandwidths are reallocated by a bentpipe payload to different positions in the frequency spectrum.

A new approach of FFBR networks was introduced which was based on Finite-length Impulse Response (FIR) variable oversampled complex modulated FBs for bentpipe payloads. By this approach an efficient implementation structure was then obtained [12]. Moreover, it was shown that if the system had a good design then this system would approximate PFBR. In [12], the FFBR network has to process the analytic representation of the real uplink satellite signals. The results of the frequency multiplexing should be converted to real signals for transmission.
1.1.2 Configuration MIMO FFBR Network

An FFBR network is generally an $m$-input $n$-output system, where $m \leq n$. The MIMO structure has the same fixed structures as SISO (refer to Figures 17 and 23 of [12]) with some changes. The channel switch, in the case of MIMO, will operate between different SISO structures. Moreover, some branches will be set to zero at the output of the Discrete Fourier Transform (DFT) block in Figure 1.4, in case of $m < n$. More discussions on the MIMO system for both cases can be found that is, $m < n$ and $m = n$ in [12, 13].

1.2 Variable Oversampled Complex Modulated FBs for FFBR Network

![Flexible Frequency-Band Reallocation Network](image)

Figure 1.2. Fixed analysis filters and adjustable synthesis filters are part of the $N$-channel FFBR Network.

The fixed analysis filters $H_k(z), k = 0, 1, 2, \ldots, N - 1$ are used in the FFBR network in Figure 1.2. The input signal $x(n)$ is divided into $N$ sub channels followed by the downsample/upsampler. The adjustable synthesis filters $G_k(z)$ carry out the required frequency shifts of the sub channels and the channel combiner combines the sub channels to make a multiplexed output signal $y(n)$. An FFBR network can be realized using fixed analysis filters and a channel combiner, as adjustable synthesis filters have high implementation cost. If the same FFBR network is required to have the same functionality as the network shown in Figure 1.2, then suitable choice of system parameters and filter characteristics will be needed, which can help in reducing the arithmetic complexity sufficiently. In the next section, the discussion will be about the efficient realization of the FFBR network. This network uses adjustable synthesis filters, fixed analysis filters and a channel switch.
1.2.1 FFBR Network (An Efficient Realization)

Figure 1.3. Implementation of the N-channel FFBR network in Figure 1.2 with efficient DFT/IDFT.

The N-Channel FFBR network is built by the different blocks e.g., DFT, inverse DFT (IDFT), complex multipliers, polyphase components and input/output commutators. In Figure 1.3, the FFBR network structure is shown. The FFBR network has complex multipliers and therefore, it is a complex system. In Figure 1.3, if the FFBR network is real then the input/output signals will be real and if FFBR network is complex then the input/output signals will be complex. Both real and complex FFBR networks contain complex multipliers and, therefore, the FFBR network is a complex network by nature. In short, the system is supposed to have complex input signal which is divided into Q granularity bands and here

\[ Q = \frac{N}{A}, \quad A > 1, \quad A \in \mathbb{N} \quad (1.1) \]

The GBs are separated by a guardband which is equal to \( 2\Delta = \frac{2\pi \epsilon}{Q} \) and \( 0 \leq \epsilon \leq 1 \). The \( \epsilon \) specifies here the order of filters and also transition band. The frequency spectrum, covered by GBs, will become smaller if transition bands are large. Thus, there is a trade-off to be made. Any user can take up any rational number of GBs which means that users can have unspecified variable bandwidths and, through this, the bandwidth-on-demand is supported. The aliasing will be restrained and at the same time, the GBs will be shifted by all values of \( \frac{2\pi q}{Q} \), \( q = 0, 1, 2, \ldots, Q - 1 \) and the M should be a multiple of Q as

\[ M = BQ, \quad B \in \mathbb{N} \quad (1.2) \]

Moreover, if the stopband of the filters are reduced, it can lead to reduction of the aliasing and overlapping of passbands and also transition bands can be avoided. This can be attained if
a linear-phase prototype filter of length-$S$ with the transfer function

$$P(z) = \sum_{n=0}^{S-1} z^{-n} p(n) = \sum_{i=0}^{N-1} z^{-i} P_i$$  \hspace{1cm} (1.4)$$

and the frequency response is

$$P(e^{j\omega T}) = e^{-j\frac{\omega T(S-1)}{2}} P_R(\omega T)$$  \hspace{1cm} (1.5)$$

The analysis filters are

$$H_k(z) = \beta_k P(z W_N^{k+\alpha}) = \beta_k \sum_{i=0}^{N-1} z^{-i} \alpha_i P_i(z W_N^{N \alpha N}) W_N^{-ki}$$  \hspace{1cm} (1.6)$$

Here, $k = 0, 1, 2, \ldots, N-1$ and

$$W_N = e^{-j\frac{2\pi}{N}}$$  \hspace{1cm} (1.7)$$

$$\alpha_i = W_N^{-\alpha i}$$  \hspace{1cm} (1.8)$$

$$\beta_k = W_N^{(k+\alpha)D/2}$$  \hspace{1cm} (1.9)$$

Moreover, the real zero-phase frequency response is $P_R(\omega T)$ (with the magnitude response illustrated in Figure 1.4). Here, $\alpha$ is a real-valued constant. It sets the filters at the desired centre frequencies. If $P(z)$ is replaced with $P(z W_N^{k+\alpha})$, then the multipliers $\beta_k$ correct the phase rotations. Consequently, all analysis filters become linear-phase filters and these filters have the same delay as the prototype filter. In the synthesis filter bank (SFB), there are multipliers $\gamma_k$ which can be defined as follows:
\[ y_k = \beta_k W_N^{-k} \quad (1.10) \]

the synthesis filters will be defined as

\[ G_k(z) = \mu_{kr} H_{c_{kr}}(z) \quad (1.11) \]

where

\[ c_{kr} = k + A_{S_r} \quad (1.12) \]

\[ \mu_{kr} = W_N^{m_r N(S-1)} \quad (1.13) \]

\[ m_r = \begin{cases} B_{S_r} & s_r \geq 0 \\ M + B_{S_r} & s_r \geq 0 \end{cases} \quad (1.14) \]

Here, \( s_r \) tells the number of GBs by which subband \( r \) is shifted. This subband \( r \) is positive if it shifts to the right and this subband \( r \) is negative if it shifts to the left. This information will be required for programming the switch channel. Three things will be needed when there is need to programme the switch channel, that is, \( L, s_r \) and number of GBs for every user. One thing that should be kept in mind is that complex constants \( \mu_{kr} \) in equation (1.13) can be equal to unity with the help of proper choice of prototype filter order and at the cost of several additional delay. Since, here, it is supposed that \( \mu_{kr} = 1 \) and, consequently, as debated in Figure 15 of [12], these multipliers are not shown in Figure 1.4.
INTRODUCTION TO DIGITAL FILTERS

In this chapter, some basics of digital filters will be introduced. These filters are going to be used in the FFBR network. It starts with FIR filters and its classifications and then other special types of filters i.e., linear-phase FIR filters. The FFBR network can use these filters which will be described in the section (2.4). If an effective realization of the FFBR network is required, then the polyphase decomposition can be used. The discussion about the polyphase decomposition will be discussed in the section (2.3).

2.1 FIR FILTERS

An FIR filter (Finite-length Impulse Response) is a type of a discrete-time filter. The impulse response is finite because it settles to zero in a finite number of sample intervals. The impulse response of an Nth-order FIR filter gives N+1 sample, and then dies to zero. Its realization [14] is considered here. The transfer function $H(z)$ of an Nth order causal\(^1\) FIR filter is written as follows:

$$H(z) = \sum_{n=0}^{N} h[n]z^{-n}$$

(2.1)

In the time domain, the relationship between input sequence and output sequence of the above FIR filter is given by

$$y[n] = \sum_{k=0}^{N} h[k]x[n-k]$$

(2.2)

\(^1\) If $h[n] = 0$ for $n < 0$, then $h[n]$ is a causal filter. In addition, if you put a proper delay in any non-causal filter then you can get a causal filter.
In the equation (2.2), \( y[n] \) is the output sequence while \( x[n] \) is the input sequence. The FIR filters that can provide linear phase are frequently used in the many applications. Of course, there is a possibility to use non-recursive algorithms to realize FIR filters and, as a result of that, instability problems can be completely removed. The FIR filters which are used in the FFBR network are always stable. They will need \( N + 1 \) multipliers, \( N \) two-input adders and \( N \) delay elements in Figures 2.1 and 2.2. To reduce the implementation cost, you practically need more efficient structures. The polyphase realization and the multiplierless realization could be the examples [14, 16]. The polyphase realization will be described for FIR filters in the section (2.3).

### 2.1.1 Direct Forms

A \( N \)th order FIR filter has \( N + 1 \) coefficients. Therefore, it is needed \( N + 1 \) multipliers, \( N \) delay elements and \( N \) (two-input) adders to implement an FIR filter [14]. In structures where the coefficients of the multipliers are exactly the same coefficients of the transfer function, then these structures are called direct form structures. From equation (2.2), a direct form of an FIR filter can be realized. In the Figure 2.1, the structure of a direct form is shown.

![Figure 2.1. An Nth-order FIR filter and realization in direct form.](image1)

In the Figure 2.2, the transpose of the structure of Figure 2.1 is shown. This is the second direct form structure.

![Figure 2.2. A Nth-order FIR filter and realization in transposed direct form.](image2)
2.1.2 Cascade Form

There is a possibility to realize higher-order FIR transfer function as a cascade of FIR sections and each section is differentiated here by either a first-order or a second order transfer function. The transfer function $H(z)$ can be written in the form

$$H(z) = h[0] \prod_{k=1}^{K} (1 + \beta_{1k} z^{-1} + \beta_{2k}z^{-2})$$  \hfill (2.3)

Where $K = (N + 1)/2$ if $N$ is odd, with $\beta_{2k}$ is equal to zero, and $K = N/2$ if $N$ is even. Of course, each second-order transfer function can also be realized in the transposed direct form. For $N$th-order FIR filter, $N + 1$ multiplications and $N$ two-input additions in the cascade form will be needed.

2.2 FIR Digital Filter Design

Some basic approaches to the design of FIR digital filters will be discussed in this section. The filter order will be determined to meet the recommended specifications [14]. For an ideal digital filter, the frequency response $H(e^{j\omega T})$ is equal to one in the passband and zero in the stopband. Moreover, there does not exist any transition band in a brickwall characteristic. Thus, this kind of filter has an infinite length and this filter is not realizable. The example of such filter is an ideal lowpass sinc function. The sinc function can be defined as

$$\text{sinc}(x) = \begin{cases} 
1 & \text{if } x = 0 \\
\frac{\sin(x)}{x} & \text{if } x \neq 0
\end{cases}$$

In practice, the specification for a digital filter with frequency response $H(e^{j\omega T})$ is given by

$$1 - \delta_c \leq |H(e^{j\omega T})| \leq 1 + \delta_c, \quad \omega T \in \Omega_c$$

$$|H(e^{j\omega T})| \leq \delta_s, \quad \omega T \in \Omega_s$$  \hfill (2.4)

where, $\delta_c$ are the ripples in the passband and $\delta_s$ are the ripples in the stop band. Therefore, $\Omega_c$ and $\Omega_s$ are the passband and stop band regions, respectively. For example, for the lowpass filter, the passband region $\Omega_c$ is between 0 and $\omega_c T$ while the stopband region $\Omega_s$ is between $\omega_s T$ and $\pi$. Here, $\omega_c T$ and $\omega_s T$ are the passband edges and the stopband edges, respectively. Therefore, it has to determine the coefficients $h[n]$ such that equation (2.4) is satisfied for desired values of $\delta_c, \delta_s, \Omega_c$ and $\Omega_s$ after the estimation of the filter order.

2.2.1 The FIR Filter Order Estimation

There are three different formulas for estimating the minimum value of the filter order $N$ directly from the digital filter specifications and these three formulas are:
- Kaiser’s formula
- Bellanger’s formula
- Hermann’s formula

For estimating a linear-phase FIR filter of order $N$, a very common formula is used. This formula is called Bellanger’s formula and is given by [14]

$$N \approx -\frac{2}{3} \log(10\delta_c\delta_s) \frac{2\pi}{\omega_s T - \omega_c T}$$

The equation (2.5) gives us a good approximation if the filter has reasonable orders. For nonlinear-phase FIR filters, there are no such formulas. Consequently, a manual search is the only method to find the filter order.

There is always a goal when designing a filter. This goal is to find a set of coefficients which are satisfying a specific criterion. This criterion could be for example, maximum ripple, energy, or combination of them that leads to LS, minimax, or constrained LS approaches.

### 2.3 Polyphase Decomposition Realization

One of the very interesting realizations of an FIR filter is the polyphase decomposition. Here, it is illustrated how the transfer function in equation (2.1) can be decomposed into its $L$-branch polyphase components.

The polyphase decomposition of the transfer function of order $N$ is of form

$$H(z) = \sum_{n=-\infty}^{\infty} h(nL)z^{-L} + z^{-1} \sum_{n=-\infty}^{\infty} h(nL + 1)z^{-nL} \quad (2.6)$$

$$\ldots$$

$$+z^{-(L-1)} \sum_{n=-\infty}^{\infty} h(nL + L - 1)z^{-nL}$$

The above mentioned equation can be written in a compact form as
\[ H(z) = \sum_{i=0}^{L-1} z^{-i}H_i(z^L) \]  

(2.7)

So that \( H_i(z) \) are the polyphase components, and

\[ h_i(n) = h(nL + i), \quad i = 0, 1, 2, \ldots, L - 1 \]  

(2.8)

This type of decomposition is often called Type I polyphase decomposition. There is another type of decomposition as well which is called Type II polyphase decomposition.

The Type II polyphase decomposition of equation (2.1) is

\[ H(z) = \sum_{i=0}^{L-1} z^{-(L-1-i)}R_i(z^L) \]  

(2.9)

where \( R_i(z) = H_{L-1-i}(z) \) [15].

It will require \( L \) subfilters of length \( \frac{N+1}{L} \), if an \( N \)-th order FIR filter is to be realized which is using \( L \)-polyphase decomposition. Due to polyphase decomposition, it is a system where the filters will operate at the lowest frequency. The polyphase decomposition reduces the cost of implementation but the total number of additions and multiplications remain the same. However, if multipliers and adders operate at a lower rate then they reduce their implementation cost.

### 2.4 Linear-phase FIR Filters

An impulse response \( h[n] \) of a linear-phase FIR filter of order \( N \) can be either symmetric or antisymmetric as [14]

Symmetric: \( h[n] = h[N - n], \quad n = 0, 1, 2, \ldots, N \)

Antisymmetric: \( h[n] = -h[N - n], \quad n = 0, 1, 2, \ldots, N \)  

(2.10)

The total number of multipliers can be reduced by using the symmetric (antisymmetric) property of a linear-phase FIR filter. However, the number of adders will be remaining the same. A linear-phase FIR filter’s frequency response can be written as follows:

\[ H(e^{j\omega T}) = e^{-j\left(\frac{N\omega T}{2} + c\right)}H_R(\omega T) = e^{j\Theta(\omega T)}H_R(\omega T) \]  

(2.11)

Here, \( H_R(\omega T) \) is the frequency response, also called the real-zero phase frequency response. If \( c = 0 \) in equation (2.11), then it will get a symmetric impulse response and, for
\( c = \frac{\pi}{2} \) in equation (2.11), it will get an antisymmetric impulse response. Moreover, the group delay \( \tau_g(\omega T) \) and the phase response \( \Phi(\omega T) \) is related to \( \Theta(\omega T) \) as

\[
\Phi(\omega T) = \begin{cases} 
\Theta(\omega T) & \text{if } H_R(\omega T) \geq 0 \\
\Theta(\omega T) \pm \pi & \text{if } H_R(\omega T) < 0
\end{cases}
\]

\[
\tau_g(\omega T) = -\frac{d\Phi(\omega T)}{d(\omega T)}
\] (2.12)

The group delay can be reduced to constant equal to half \( \left( \frac{N}{2} \right) \) of the multipliers in the case of the linear-phase FIR filters. There are four different types of linear-phase FIR filters. These four types depend upon the impulse response \( h[n] \) and \( N \) being symmetric or antisymmetric and being even or odd, respectively. These four types of linear-phase FIR filters have four different expressions for \( H_R(\omega T) \) [14]. The definitions of four types are

- **Type I**: \( h[n] = h(N - n), \quad N \text{ even} \)
- **Type II**: \( h[n] = h(N - n), \quad N \text{ odd} \)
- **Type III**: \( h[n] = h(N - n), \quad N \text{ even} \)
- **Type IV**: \( h[n] = h(N - n), \quad N \text{ odd} \) (2.13)
In digital signal processing, a major role is played by the discrete Fourier transform (DFT) and its fast implementation, the fast Fourier transform (FFT). Some of the most important algorithms will be discussed in this chapter, which are shown in Figure 3.1.

Figure 3.1. A Classification of DFT and FFT algorithms.

A terminology [18] called multidimensional index maps will be followed. This terminology classifies FFT algorithms in terms of their input and output sequences. Combinations between DFT and FFT algorithms often used in the most efficient implementations. The combination, for instance, between Rader prime algorithm and the Good-Thomas FFT gives a very good result in VLSI Implementations.
3.1 The DFT Algorithms

First, the important properties of DFT will be reviewed. Afterwards, a review of the basic DFT algorithms will be introduced. The Fourier transform is defined as follows

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft}dt \leftrightarrow x(t) = \int_{-\infty}^{\infty} X(f)e^{j2\pi ft}df$$  \hspace{7cm} (3.1)

In the above equation (3.1), an assumption of a continuous signal is made. The signal is infinite and its bandwidth is infinite as well. In practice, sampling in frequency and time for this signal will be needed. Of course the amplitudes will be quantized as well. Moreover, for the implementation purpose, it should have finite number of samples in time and frequency. To have this, discrete Fourier transform (DFT) [18] will be needed, where $N$ samples will be used in time and frequency. Below, a DFT is defined as

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{j2\pi kn/N} = \sum_{n=0}^{N-1} x[n]W_N^{kn}$$ \hspace{7cm} (3.2)

While an inverse DFT (IDFT) is defined as

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k]e^{j2\pi kn/N} = \frac{1}{N} \sum_{k=0}^{N-1} X[k]W_N^{-kn}$$ \hspace{7cm} (3.3)

These expressions mentioned in equations (3.2) and (3.3), can be written in vector/matrix form

$$X = Wx \leftrightarrow x = \frac{1}{N}W^*X$$ \hspace{7cm} (3.4)

3.1.1 The DFT’s Properties

A summary of the most important properties of the DFT is shown in Table 3.1. The properties are mostly identical with Fourier transform, e.g., real and imaginary parts are related through the Hilbert transform, the superposition applies and the transform is unique. Since there is an alternative inversion algorithm due to similarity in forward and inverse transform. The expression used in equation (3.4) of the DFT is following:

$$X = Wx \leftrightarrow x = \frac{1}{N}W^*X$$
From this expression a conclusion can be drawn that

$$x^* = \frac{1}{N} (W^*X)^*$$

(3.5)

It means that computing the IDFT, the DFT of $X^*$ can be used which is scaled by $\frac{1}{N}$.

<table>
<thead>
<tr>
<th>Theorem</th>
<th>$x[n]$</th>
<th>$X[k]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transform</td>
<td>$x[n]$</td>
<td>$\sum_{n=0}^{N-1} x[n]e^{-j\frac{2\pi nk}{N}}$</td>
</tr>
<tr>
<td>Inverse Transform</td>
<td>$\frac{1}{N} \sum_{k=0}^{N-1} X[k]e^{j\frac{2\pi nk}{N}}$</td>
<td>$X[k]$</td>
</tr>
<tr>
<td>Superposition</td>
<td>$s_1x_1[n] + s_2x_2[n]$</td>
<td>$s_1X_1[k] + s_2X_2[k]$</td>
</tr>
<tr>
<td>Time reversal</td>
<td>$x[-n]$</td>
<td>$X[-k]$</td>
</tr>
<tr>
<td>Conjugate complex</td>
<td>$x^*[n]$</td>
<td>$X^*[k]$</td>
</tr>
<tr>
<td>Real part</td>
<td>$\Re(x[n])$</td>
<td>$(X[k] + X^*[-k])/2$</td>
</tr>
<tr>
<td>Imaginary part</td>
<td>$\Im(x[n])$</td>
<td>$(X[k] + X^*[-k]j)/(2j)$</td>
</tr>
<tr>
<td>Real even part</td>
<td>$x_e[n] = (x[n] + x[-n])/2$</td>
<td>$\Re(X[k])$</td>
</tr>
<tr>
<td>Real odd part</td>
<td>$x_o[n] = (x[n] - x[-n])/2$</td>
<td>$j\Im(X[k])$</td>
</tr>
<tr>
<td>Symmetric</td>
<td>$X[n]$</td>
<td>$N\Re x[-k]$</td>
</tr>
<tr>
<td>Cyclic convolution</td>
<td>$x[n] \otimes f[n]$</td>
<td>$X[k] F[k]$</td>
</tr>
<tr>
<td>Multiplication</td>
<td>$x[n] \times f[n]$</td>
<td>$\frac{1}{N} X[k] \otimes F[k]$</td>
</tr>
<tr>
<td>Periodic shift</td>
<td>$x[n - d \mod N]$</td>
<td>$X[k] e^{-j\frac{2\pi nk}{N}}$</td>
</tr>
<tr>
<td>Parseval theorem</td>
<td>$\sum_{n=0}^{N-1}</td>
<td>x[n]</td>
</tr>
</tbody>
</table>

Table 3.1. DFT Theorems [18]

When the input sequence is real then it can get some savings in computations of DFT (FFT). If there is a real sequence at the input then there are two options:

- Compute with one $N$-point DFT, the DFT of two $N$-point sequences.
- Compute with an $N$-point DFT, a length $2N$ DFT of a real sequence.

A real sequence has an even-symmetric real spectrum and an odd imaginary spectrum, see in Table 3.1., and therefore, the following algorithms [18] can be synthesized.

**Length 2$N$ Transform with $N$-point DFT**

To compute the $2N$-point DFT $X[k] = X_r[k] + jX_i[k]$ from $x[n]$(time sequence) is following:

- $N$-point sequence $y[n] = x[2n] + jx[2n + 1]$ will be built, and
$n = 0, 1, 2, \ldots, N - 1.$

- Compute $\mathcal{F}(y[n]) = Y[k]$ and $Y[k] = Y_r[k] + jY_i[k]$, where $Y_r[k]$ is the real part and $Y_i[k]$ is the imaginary part.

- Let us compute

\[
X_r[k] = \frac{Y_r[k] + Y_r[-k]}{2} + \cos\left(\frac{\pi k}{N}\right)\frac{Y_i[k] + Y_i[-k]}{2} \\
- \sin\left(\frac{\pi k}{N}\right)\frac{Y_r[k] - Y_r[-k]}{2}
\]

\[
X_i[k] = \frac{Y_i[k] + Y_i[-k]}{2} - \sin\left(\frac{\pi k}{N}\right)\frac{Y_i[k] + Y_i[-k]}{2} \\
- \cos\left(\frac{\pi k}{N}\right)\frac{Y_r[k] - Y_r[-k]}{2}
\]

and $k = 0, 1, 2, \ldots, N - 1$.

The computational complexity, thus, in addition to an $N$-point DFT are $4N$ real additions and multiplications due to the twiddle factors $\pm \exp\left(\frac{j\pi k}{N}\right)$.

Furthermore, if it is required to transform two length-$N$ sequences with a length-$N$ DFT, the fact is used from Table 3.1, that a real sequence has an even spectrum while an imaginary sequence has an odd spectrum. The following algorithm has this basis.

- **Two Length $N$ Transforms with one $N$-point DFT**

  If $\mathcal{F}(g[n]) = G[k]$ and $\mathcal{F}(h[n]) = H[k]$ so the algorithm to compute the $N$-point DFT will be as follows:

  - $N$-point sequence will be built, i.e., $y[n] = h[n] + jg[n]$ where $n = 0, 1, \ldots, N - 1$.

  - Afterwards, $\mathcal{F}(y[n]) = Y[k]$, and $Y[k] = Y_r[k] + jY_i[k]$ will be computed where $Y_r[k]$ is the real part and $Y_i[k]$ is the imaginary part.

  - Now compute

\[
H[k] = \frac{Y_r[k] + Y_r[-k]}{2} + j\frac{Y_i[k] - Y_i[-k]}{2}
\]
To form the two \( N \)-point DFTs, in addition to an \( N \)-point DFT are \( 2N \) real additions required.

To compute the convolutions by the DFT (or FFT), is a very common application.

The convolution is defined as follows:

Let \( \mathcal{F} \) denote the Fourier transform, so that \( \mathcal{F}\{f\} \) and \( \mathcal{F}\{g\} \) are the Fourier transforms of \( f \) and \( g \), respectively. Then

\[
\mathcal{F}\{f \ast g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\}
\]

where \( \cdot \) denotes point-wise multiplication. By applying the inverse Fourier transform \( \mathcal{F}^{-1} \), the expression will be

\[
f \ast g = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\}
\]

The DFT now computes a periodic convolution and not computing a linear convolution, compared with the Fourier transform. In the fast convolution, the input sequences are often real. Therefore, an efficient convolution can be achieved with a real transform.

### 3.1.2 The Goertzel Algorithm

When the DFT computes a single spectral component \( X[k] \), the result will be written as

\[
\]

\[
\]

From this expression, it can be seen that, this is a recursive computation of \( X[k] \) and this expression is called the Goertzel algorithm. This algorithm is illustrated in Figure 3.3

![Figure 3.2. The length-4 Goertzel algorithm [18]](image_url)
In Figure 3.3, it is observed that the algorithm starts with the last value of the input sequence $x[N - 1]$. The spectrum value of $X[k]$ is available at the output $y[n]$ after step 3, see Figure 3.2. Furthermore, the complexity can be reduced with the combination of the factors type $e^{\pm 2\pi n/N}$, when it is required to compute various spectral components. The result will be a second-order system containing a denominator as follows:

$$x^2 - 2z\cos\left(\frac{2\pi n}{N}\right) + 1$$

This leads to, that the complex multiplications will then be reduced to real multiplications.

The Goertzel algorithm usually is appealing when it is needed to compute only a few spectral components. The effort is of order $N^2$, for the whole DFT, it does not give any advantage when it is tried compared with direct DFT computation.

### 3.1.3 The Rader Algorithm

To compute the DFT [18] by the use of the Rader algorithm,

$$X[k] = \sum_{n=0}^{N-1} x[n]W_N^{nk}, \quad k, n \in \mathbb{Z}_N; \quad \text{ord}(W_N) = N \tag{3.6}$$

The Rader algorithm is defined only for the prime numbers $N$. It starts first with computing the DC components with the help of the following equation:

$$X[0] = \sum_{n=0}^{N-1} x[n] \tag{3.7}$$

Since, $N = p$ is a prime number, so that, there is a primitive element and this primitive element is defined as a generator $g$. This generator $g$ generates all elements of $n$ and $k$. Now if $n$ replaces by $g^n \mod N$ and $k$ by $g^k \mod N$ then the equation (3.7) can be written as follows:
\[ X[g^k \mod N] - x[0] = \sum_{n=0}^{N-1} x[g^n \mod N] W_N^{n+k \mod N} \] (3.8)

for \( k \in \{1, 2, 3, \ldots, N - 1\} \). An important thing is observed that the right side of the equation (3.9) is a cyclic convolution, i.e.,

\[ [x[g^0 \mod N], x[g^1 \mod N], \ldots, x[g^{N-2} \mod N]] \odot [W_N, W_N^{g}, \ldots, W_N^{g^{N-2} \mod N}] \] (3.9)

Moreover, to realize more efficient FIR filters, the symmetries of the complex pairs \( e^{\pm j2k\pi/N}, k \in \left[0, \frac{N}{2}\right] \) can be used by the using the Rader algorithm. It is very important to note that when a Rader prime-factor DFT is implemented, it means that an FIR filter is being implemented.

### 3.1.4 The Winograd DFT Algorithm

The combination of the Rader algorithm and Winograde’s [18] short convolution algorithm leads to the Winograd algorithm. The Rader algorithm translates a DFT into the periodic convolution and implements the fast-running FIR filters. To have this, the short convolution algorithm will be required. Therefore the length is limited to prime numbers or power of prime numbers. Hence, Table 3.2 in the following shows the information about how many arithmetic operations will be needed.

<table>
<thead>
<tr>
<th>Block length</th>
<th>Total number of real multiplications</th>
<th>Total number of nontrivial multiplications</th>
<th>Total number of real additions</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>5</td>
<td>17</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>8</td>
<td>36</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>2</td>
<td>26</td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td>10</td>
<td>44</td>
</tr>
<tr>
<td>11</td>
<td>21</td>
<td>20</td>
<td>84</td>
</tr>
<tr>
<td>13</td>
<td>21</td>
<td>20</td>
<td>94</td>
</tr>
<tr>
<td>16</td>
<td>18</td>
<td>10</td>
<td>74</td>
</tr>
<tr>
<td>17</td>
<td>36</td>
<td>35</td>
<td>157</td>
</tr>
<tr>
<td>19</td>
<td>39</td>
<td>38</td>
<td>186</td>
</tr>
</tbody>
</table>

Table 3.2. Complexity for the Winograd DFT algorithm with real inputs. For complex inputs, the number of operations is twice as large [18].
It was mentioned earlier that the combination of the Rader algorithm and short convolution algorithm leads to the Winograd DFT algorithm. Later on, in this chapter, the Winograd FFT algorithm will be discussed. The Winogard FFT algorithm uses the least number of multiplications compared to other FFT algorithms.

3.2 The Fast Fourier Transform (FFT) Algorithms

In the beginning of this chapter, the terminology called multidimensional index maps was used. All FFT algorithms are simply classified by different multidimensional index maps of the input and output sequences. Therefore, the transform of the length $N$ DFT will be as follows:

$$X[k] = \sum_{n=0}^{N-1} x[n] W_n^{nk}$$

(3.10)

where $W_n^{nk} = e^{-j2\pi nk/N}$.

The equation (3.11) can be written in multidimensional representation as

$$N = \prod_i N_i$$

Generally, it is enough to discuss only the two-factor case, since higher dimensions can be built simply by iteratively replacing again one of these factors. In this section, three FFT algorithms will be discussed. All these three FFT algorithms are presented in the terms of two-dimensional index transform.

Let us transform the (time) index $n$ with

$$n = An_1 + Bn_2 \bmod N \quad \{0 \leq n_1 \leq N_1 - 1, \quad 0 \leq n_2 \leq N_2 - 1\}$$

(3.11)

where $N = N_1 N_2$ and $A, B \in \mathbb{Z}$ are constants. Another index mapping $k$ will be applied to the output (frequency) and this can be written as

$$k = Ck_1 + Dk_2 \bmod N \quad \{0 \leq k_1 \leq N_1 - 1, \quad 0 \leq k_2 \leq N_2 - 1\}$$

(3.12)

and $C, D \in \mathbb{Z}$ are the constants.

There are two types of algorithms which are used in the different FFT algorithms. These two types are common-factor algorithms (CFAs) and prime-factor algorithms (PFAs). In common-factor algorithms the $\gcd(N_1, N_2) > 1$, while in prime-factor algorithms $\gcd(N_1, N_2) = 1$. In the next section, the CFA algorithm will be described and, the Cooley-Tukey algorithm is the example of that.
3.2.1 The Cooley-Tukey FFT Algorithm

The Cooley-Tukey FFT algorithm is the most commonly used FFT algorithms. In this algorithm, it is possible to have any factorization of \( N \). If the transform length \( N \) is the power of a basis \( r \), for example \( N = r^v \), then the Cooley-Tukey FFTs are very popular. These algorithms are usually to be said radix-\( r \) algorithms.

In equation (3.12), \( A = N_2 \) and \( B = 1 \), then the mapping will be

\[
 n = N_2n_1 + n_2 \mod N \begin{cases} 0 \leq n_1 \leq N_1 - 1 \\ 0 \leq n_2 \leq N_2 - 1 \end{cases} \tag{3.13}
\]

If an inverse mapping from equation (3.13) has \( C = 1 \) and \( D = N_1 \), then the mapping results are

\[
k = k_1 + N_1k_2 \mod N \begin{cases} 0 \leq k_1 \leq N_1 - 1 \\ 0 \leq k_2 \leq N_2 - 1 \end{cases} \tag{3.14}
\]

Now if both \( n \) and \( k \) are replaced in \( W_N^{nk} \) according to equation (3.13) and equation (3.14), respectively, the expression will be

\[
 W_N^{nk} = W_N^{N_2n_1k_1+N_1N_2n_1k_2+n_2k_1+N_1n_2k_2} \tag{3.15}
\]

Since, \( W \) is of order \( N = N_1N_2 \), therefore, \( W_N^{N_1} = WN_2 \) and \( W_N^{N_2} = WN_1 \). The equation (3.15) will be written

\[
 W_N^{nk} = W_N^{n_1k_1}W_N^{n_2k_1}W_N^{n_2k_2} \tag{3.16}
\]

Now if the equation (3.16) replaces in the DFT equation (3.10). The new equation will be

\[
 X[k_1, k_2] = \sum_{n_2=0}^{N_2-1} W_N^{n_2k_2} \left( W_N^{n_2k_1} \sum_{n_1=0}^{N_1-1} x[n_1, n_2] W_N^{n_1k_1} \right) \tag{3.17}
\]

\[
 = \sum_{n_2=0}^{N_2-1} W_N^{n_2k_2} \bar{x}[n_2, k_1] \tag{3.18}
\]
The whole Cooley-Tukey FFT algorithm can be now defined as follows:

There are several steps to compute the $N = N_1 N_2$ - point DFT

- Follow the equation (3.13) for computing an index transform of the input sequence.
- Now $N_2$ DFTs of length $N_1$ will be computed.
- Twiddle factors $W_{N_2}^{n_2 k_1}$ will be applied at this stage to the output of the first transform.
- Now $N_1$ DFTs of length $N_2$ will be computed.
- Follow the equation (3.14) for computing an index transform of the output sequence.

### 3.2.2 The Good-Thomas FFT Algorithm

The two types of algorithms used in different FFT algorithms such as CFAs and PFAs were earlier mentioned. The example of the CFAs, is the Cooley-Tukey FFT algorithm. Now those FFT algorithms will be described which are using PFAs algorithms. The examples are the Good-Thomas FFT algorithm and the Winograd FFT algorithm.

In the Good-Thomas FFT algorithm [18], the twiddle factors are not involved as it has already been seen in the Cooley-Tukey FFT algorithm. The price will be paid here for twiddle factor free flow, is that the factors must be in form, $\gcd(N_k, N_l) = 1$ for $k \neq l$, i.e., coprime. The index mapping will become somehow more complicated. In the elimination of the twiddle factors presented through the index mapping of $n$ and $k$ according to equation (3.11) and equation (3.12), respectively, the expression is

$$ W_{N}^{nk} = W_{N}^{(An_1+Bn_2)(Ck_1+Dk_2)} = W_{N}^{(ACn_1k_1+ADn_1k_2+BCn_2k_1+BDn_2k_2)} = W_{N}^{N_2n_1k_1}W_{N_1}^{n_2k_2} = W_{N_2}^{n_2k_1}W_{N_1}^{n_1k_2} \tag{3.19} $$

therefore, the following necessary conditions must be satisfied at the same time:

$$ \langle AD \rangle_N = \langle BC \rangle_N = 0 \tag{3.20} $$

$$ \langle AC \rangle_N = N_2 \tag{3.21} $$

$$ \langle BD \rangle_N = N_1 \tag{3.22} $$

The index mapping that has been suggested by the Good-Thomas [18] satisfies this condition and will be written as follow:

$$ A = N_2 \quad B = N_1 \quad C = N_2 \langle N_2^{-1} \rangle_{N_1} \quad D = N_1 \langle N_1^{-1} \rangle_{N_2} \tag{3.23} $$
The condition (3.21) can be expressed as

\[
\langle AC \rangle_N = \langle N_2 N_2 \langle N_2^{\phi(N_2)} \rangle_{N_2} \rangle_{N_1} \rangle_N
\]  

(3.24)

where \( \phi \) is an Euler totient function. The inner modulo reduction is solved which is following with \( \nu \in \mathbb{Z} \) and, \( \nu N_1 N_2 \mod N = 0 \). The following expression becomes

\[
\langle AC \rangle_N = \langle N_2 N_2 (N_2^{\phi(N_2)} + \nu N_1) \rangle_N = N_2
\]

(3.25)

The same argument will be applied for the condition (3.22). Thus, it has now shown that all three conditions (3.20), (3.21), (3.22) are fulfilled, if the Good-Thomas mapping (3.23) is used. This concludes a following theorem

\* Good-Thomas index Mapping

For \( n \), the Good-Thomas index mapping gives

\[
n = N_2 n_1 + N_1 n_2 \mod N \begin{cases} 0 \leq n_1 \leq N_1 - 1 \\ 0 \leq n_2 \leq N_2 - 1 \end{cases}
\]

(3.26)

for \( k \), index mapping gives us

\[
k = N_2 \langle N_2^{-1} \rangle_{N_1} k_1 + N_1 \langle N_1^{-1} \rangle_{N_2} k_2 \mod N \begin{cases} 0 \leq k_1 \leq N_1 - 1 \\ 0 \leq k_2 \leq N_2 - 1 \end{cases}
\]

(3.27)

If the Good-Thomas index map is replaced in the equation (3.10), the equation will be

\[
X[k_1, k_2] = \sum_{n_2=0}^{N_2-1} W_{N_2}^{n_2 k_2} \left( \sum_{n_1=0}^{N_1-1} x[n_1, n_2] W_{N_1}^{n_1 k_1} \right)
\]

\[\begin{array}{c}
\text{N}_1\text{-point transform} \\
\tilde{x}[n_2, k_1]
\end{array}\]

(3.28)

\[
= \sum_{n_2=0}^{N_2-1} W_{N_2}^{n_2 k_2} \tilde{x}[n_2, k_1]
\]

(3.29)

N_2\text{-point transform}
It can be concluded that the Good-Thomas algorithm anyhow is similar to Cooley-Tukey algorithm. But the Good-Thomas algorithm has different index mapping and no twiddle factors.

\* Good-Thomas FFT Algorithm

These are the different steps to follow in order to compute the $N = N_1 N_2$-point DFT

- Index transform of the input sequence, according to equation (3.26).
- Compute the $N_2$ DFTs of length $N_1$.
- Compute the $N_1$ DFT of length $N_2$.
- Index transform of the output sequence, according to equation (3.27).

\subsection*{3.2.3 The Winograd Algorithm}

The Winograd FFT algorithm \cite{18} shows that the IDFT matrix in equation (3.4), of dimension $N_1 \times N_2$, and having $\gcd(N_1, N_2) = 1$ leads to the following expression

\begin{equation}
    x[n] = \sum_{k=0}^{N-1} X[k] W_N^{-kn} \tag{3.30}
\end{equation}

Note! no $\frac{1}{N}$ factor.

\begin{equation}
    x = W^*X \tag{3.31}
\end{equation}

Note! no $\frac{1}{N}$ factor.

The equation (3.30) and equation (3.31) can be written, if the Kronecker product\footnote{We can define this product as follows:
\[ A \otimes B = [a_{i,j}]B = \begin{bmatrix} a[0,0]B & \cdots & a[0,L-1]B \\
\vdots & \ddots & \vdots \\
\end{bmatrix} = a[k-1,0]B \cdots a[k-1,L-1]B \]
where $A$ is a $K \times L$ matrix.} is used, as two quadratic IDFT matrices each, with dimension $N_1$ and $N_2$, respectively. With the help of the Good-Thomas algorithm, the indices of $X[k]$ and $x[n]$ can be written in 2-dimensions. Afterwards, the indices can be read row by row. This can be written in matrix/vector form as follows:

\begin{equation}
    \tilde{x} = W_{N_1} \otimes W_{N_2} \tilde{X} \tag{3.32}
\end{equation}

The Winograd DFT algorithm will be applied here for short DFTs, for example.
\[ W_{N_i} = A_i \times B_i \times C_i \] (3.33)

where \( A_i \) includes the output additions and \( B_i \) is a diagonal matrix with the Fourier coefficients, while \( C_i \) inserts the input additions. If now equation (3.32) is replaced into equation (3.33), then the expression becomes

\[ W_{N_1} \otimes W_{N_2} = (A_1 \times B_1 \times C_1) \otimes (A_2 \times B_2 \times C_2) \]

\[ = (A_1 \otimes A_2)(B_1 \otimes B_2)(C_1 \otimes C_2) \] (3.34)

Note that the total number of required multiplications is therefore identical to the number of the diagonal elements i.e., \( B = B_1 \otimes B_2 \). The combination of the different steps will build a Winograd FFT.

- **To design a Winograd FFT**
  - Index transform of the input sequence according to the Good-Thomas index mapping (3.26). Afterwards read the row of the indices.
  - Use the Kronecker to factorize the DFT matrix.
  - Use the Winograd DFT algorithm to replace the length \( N_1 \) and \( N_2 \) DFT matrices.
  - All multiplications will be centralized.

There are three steps to follow to compute the Winograd FFT algorithm:

- **Winograd FFT Algorithm**
  - Preadditions such as \( C_1 \) and \( C_2 \) will be computed.
  - Multiplications will be computed according to matrix \( B_1 \otimes B_2 \).
  - Postadditions such as \( A_1 \) and \( A_2 \) will be computed.

### 3.2.4 Comparison of DFT and FFT Algorithms

It is very obvious now that there are numerous ways to implement a DFT. A short DFT algorithm can be selected from among those shown in Figure 3.1. To build long DFTs, the short DFT can be used which is using the different index mapping scheme. In implementation, minimum multiplication complexity is needed. This is a feasible criterion when the implementation cost of multiplication is very higher compared with other operations, such as index computation, additions or data access.

In Table 3.3, the number of multiplications needed for different FFT lengths is shown. Thus, the Winograd FFT algorithm certainly is most desirable because it is based on multiply complexity criterion. So far, the discussion has taken place round multiplications but of course there are other restrictions as well, such as index computation, coefficient or data size in the memory, run-time code length, possible transform lengths and number of additions. The Cooley-Tukey algorithm gives the best overall solutions, see the Table 3.3.
<table>
<thead>
<tr>
<th>Property</th>
<th>Cooley-Tukey</th>
<th>Good-Thomas</th>
<th>Winograd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any transform</td>
<td>yes</td>
<td>no</td>
<td>gcd($N_k, N_l) = 1$</td>
</tr>
<tr>
<td>Length</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum order of $W$</td>
<td>$N$</td>
<td>max($N_k$)</td>
<td></td>
</tr>
<tr>
<td>Twiddle factors needed</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>#Multiplications</td>
<td>bad</td>
<td>fair</td>
<td>best</td>
</tr>
<tr>
<td>#Additions</td>
<td>fair</td>
<td>fair</td>
<td>fair</td>
</tr>
<tr>
<td>#Index computation</td>
<td>best</td>
<td>fair</td>
<td>bad</td>
</tr>
<tr>
<td>Data in-place</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Implementation advantages</td>
<td>small butterfly can use RPFA, full parallel, med</td>
<td>small size for</td>
<td></td>
</tr>
</tbody>
</table>
IMPLEMENTATION OF FFBR NETWORK

In chapter 1, the FFBR (Flexible Frequency-Band Reallocation) network was described. An \( N \)-channel FFBR network is built using different blocks and these blocks are e.g., DFT, IDFT, polyphase components, input/output commutators and complex multipliers. In this thesis project, a 20-channel FFBR network has been used.

In this chapter, an analysis will be presented of a comparison between the MATLAB results and VHDL results. The analysis shows the deviation in VHDL results i.e., quantization error. In this project, the word length(\( \text{wl} = \text{word length} \)) of the data at input/output is 16-bit. During the simulations in mentor environment, an important and interesting thing has been observed that the more bits at the input the more efficient result at the output, that is, quantization error is smaller.

In 20-channel FFBR network, 20-point FFT/IFFT and FIR filters are the key blocks. Therefore, it is required to have as small as possible quantization error in VHDL results for these blocks. To have as small as possible quantization error in VHDL results, different word lengths were used, e.g., 4-bit, 8-bit, and 16-bit. The 16-bit data word length gives satisfactory results i.e., smaller quantization error compared to other word lengths. How much this quantization error will affect the output has not been taken in consideration. Moreover, in this 20-channel FFBR network, there are commutators at input/output. After input commutators, twenty FIR filters (polyphase decomposition) will follow and at output, twenty FIR filters (polyphase decomposition) will be located before the output commutators. Furthermore, there are twenty complex multipliers before/after the FFT/IFFT blocks, respectively.
### 4.1 FIR (Polyphase Decomposition) Filters

The order of FIR filters which are used in this FFBR network is $L = 68$ and the coefficients are $L + 1$. It was mentioned earlier that there are twenty FIR filters at input after the commutators and twenty FIR filters at the output before the commutators.

It should first be checked if the FIR filter’s VHDL code is working or not. An impulse at the input of the FIR filter will be sent and at the output, a set of the FIR filter’s coefficients should be received. If these coefficients, in the VHDL result, are exactly the same, as in the MATLAB result, then the FIR filter code is working well. The quantization error will not occur because the VHDL result is not rounded. The word length data at the FIR filter’s input is 16 bit and the output data at FIR filter’s output is 32 bit. Since, in the FFBR network 16 bit word length data is used, a reduction of 32 bit to 16 bit will be needed at FIR filter’s output.

### 4.2 Decimation-in-Time FFT Algorithms

The Cooley-Tukey FFT algorithm can be implemented in two different ways. Either it can be implemented as a Decimation-in-Time (DIT) algorithm, or it can be implemented as a Decimation-in-Frequency (DIF) algorithm. In this thesis project, Decimation-in-Time (DIT) algorithm has been chosen. Algorithms in which the sequence $x[n]$ is decomposed into successively smaller sub sequences are called Decimation-in-Time (DIT) algorithms. The method of Decimation-in-Time is shown by looking at special case of $N$ being an integer power of 2, i.e.,

$$N = 2^u$$

A DFT is introduced and the focus is on the direct transform

$$X[k] = \sum_{n=0}^{N-1} x[n]W_N^{nk}$$

Where, $W_N^{nk} = e^{-j2\pi nk/N}$, and $k = 0,1,2,...,N-1$.

If the size of the input is even [17], the $X[k]$ can be computed by dividing $x[n]$ into two $\frac{N}{2}$-point sequences. These two $\frac{N}{2}$-point sequences consist of even-numbered points in $x[n]$

---

3 In FFT algorithms, the words sample and point are usually used interchangeably to mean sequence value. Therefore, when it is said a sequence of length $N$, it means an $N$-point sequence. Thus the DFT of a sequence of length $N$ will be called an $N$-point DFT.
and odd-numbered points in $x[n]$. The even- and odd-numbered points obtained are shown as follows:

$$X[k] = \sum_{n\text{ even}} x[n] W_N^{nk} + \sum_{n\text{ odd}} x[n] W_N^{nk}$$

or if variable $n$ is replaced by $2m$ for $n$ even and $2m + 1$ for $n$ odd, then the expression can be written as follows

$$X[k] = \sum_{m=0}^{(N/2)-1} x[2m] W_N^{2mk} + \sum_{m=0}^{(N/2)-1} x[2m + 1] W_N^{(2m+1)k}$$

$$= \sum_{m=0}^{(N/2)-1} x[2m] [W_N^2]^{mk} + W_N^k \sum_{m=0}^{(N/2)-1} x[2m + 1] [W_N^2]^{mk} \quad (4.2)$$

where $W_N^2 = e^{-2j(2\pi/N)} = e^{-2j\pi/(N/2)} = W_{N/2}$.

Then, the equation (4.2) will be rewritten as follows:

$$X[k] = \sum_{m=0}^{(N/2)-1} x[2m] [W_{N/2}]^{mk} + W_N^k \sum_{m=0}^{(N/2)-1} x[2m + 1] [W_{N/2}]^{mk}$$

$$= G(k) + W_N^k H(k) \quad (4.3)$$

In equation (4.3), the first sum is the $\frac{N}{2}$-point DFT of the even-numbered points of the original sequence and the second sum is the $\frac{N}{2}$-point DFT of the odd-numbered points of the original sequence. However, the index $k$ ranges over $N$ values and, $k = 0,1,2,\ldots,N-1$. The sums are needed in equation (4.3) to compute for $k$ between 0 and $\frac{N}{2}$-1. When both sums in equation (4.3) are computed, they are then combined to generate the $N$-point DFT, $X[k]$. 

29
Figure 4.1. Flow graph of the decimation-in-time decomposition of an $N$-point DFT computation into two $N/2$-point DFT computation where $N = 8$.

Figure 4.1 shows how to compute $X[k]$ according to equation (4.3) for an 8-point sequence. Thus, one thing to be noted in the Figure 4.1 is that two 4-point DFTs are computed with even-numbered and odd-numbered, respectively. The output $X[0]$ is obtained as the result from $H[0]$ is multiplied with $W_N^0$ and then adding the product to the result from $G[0]$. The outputs $X[1]$, $X[2]$ and $X[3]$ are obtained in the same way but $W_N^k$ will be different. For obtaining $X[4]$, the result from $H[0]$ is multiplied with $W_N^4$ and then the product is added to the result from $G[0]$. $X[5]$, $X[6]$ and $X[7]$ are computed in the same way but $W_N^k$ will be different. A $W_N^k$ is usually called twiddle factor multiplier. Moreover, Figure 4.1 shows as well an example of radix-4 computation as the sequence is $N = 8$.

4.2.1 Radix-2 Cooley-Tukey Algorithm Implementation

The basic computation in the flow graph is shown in Figure 4.2 [17].
The equations from this flow graph are of the form

\[ X_{m+1}[p] = X_m[p] + W_N^T X_m[q] \]
\[ X_{m+1}[q] = X_m[p] + W_N^{(r+N/2)} X_m[q] \]  \hspace{1cm} (4.4)

Above, the computation of the flow graph is referred to as a butterfly computation because the flow graph looks like a butterfly. Equations (4.4) suggest reducing the number of complex multiplications by a factor of 2. In the following, it shows that

\[ W_N^{N/2} = e^{-j(2\pi/N)\times N/2} = e^{-j\pi} = -1 \]

Consequently, equations (4.4) can be written as follows:

\[ X_{m+1}[p] = X_m[p] + W_N^T X_m[q] \]
\[ X_{m+1}[q] = X_m[p] - W_N^T X_m[q] \]  \hspace{1cm} (4.5)

The flow graph of Figure 4.3 describes equations (4.5). Hence, since there are \(N/2\) “butterflies” of the form of Figure 4.3 per stage and \(\log_2 N\) stages, the total number of complex multiplications required is \((N/2)\log_2 N\) and the total number of complex additions required is \(N\log_2 N\). A basic flow graph of Figure 4.3 is shown as a replacement of the form of Figure 4.2.
A butterfly processor contains [18] the butterfly itself and an additional complex multiplier for the twiddle factors. A radix-2 FFT can be efficiently implemented by using a butterfly processor.

A radix-2 butterfly processor consists of a complex adder, a complex subtraction, and a complex multiplier for the twiddle factors. Four real multiplications and two add/subtract operations will be needed to implement a complex multiplication with twiddle factor. However, there is a possibility to make the complex multiplier with only three real multiplications and three add/subtract operations, since one operand is precomputed. An efficient complex multiplier algorithm is as follows:

- **Efficient Complex Multiplier Algorithm**
  - \( R + jI = (X + jY) \times (C + jS) \) is the complex twiddle factor multiplication. This multiplication can be simplified, since \( C \) and \( S \) are precomputed and stored in a table. Consequently, it is possible to store these three coefficients, that is, \( C, C + S, C - S \).
  - With the help of these three precomputed coefficients, \( E = X - Y \) will be computed first and afterwards \( Z = C \times E = C \times (X - Y) \) will be computed.
  - The final product is computed using \( R = (C - S) \times Y + Z \) and \( I = (C + S) \times X - Z \).

The algorithm has used three multiplications, two subtractions and one addition at the cost of an additional, third table. The implementation of the twiddle factor complex multiplier is illustrated in the following example.
Example: Twiddle Factor Multiplier

Some specific parameters are needed for the twiddle factor complex multiplier [18]. Suppose that 8-bit input data is available, the coefficients should have 8 bits i.e., 7 bits plus sign bit. It is multiplied by \( e^{j\pi/9} \). The twiddle factor will become \( C + jS = 128 \cdot e^{j\pi/9} = 121 + j39 \) if the twiddle factor is quantized to 8 bits. An input value of \( 70 + j50 \) is used. The expected output result will be

\[
(70 + j50) \cdot e^{j\pi/9} = (70 + j50)(121 + j39)/128
\]

\[= (6520 + j8780)/128 = 50 + j68.\]

To compute the complex multiplication with the help of the efficient complex multiplier algorithm, mentioned above, the three factors will become:

\[C = 121, \ C + S = 160, \ C - S = 82.\]

In general, the tables \( C + S \) and \( C - S \) have one more bit of precision than the \( C \) and \( S \) tables. The twiddle factor multiplier uses component instantiations of three \texttt{lpm\_mult} and three \texttt{lpm\_add\_sub} modules. The output data is scaled so that it has the same data format as the input. This is very sensible, since multiply by the complex exponential \( e^{j\pi/9} \) does not change the magnitude of the complex input. To make certain that short latency (for an in-place FFT), the complex multiplier only has output registers, with no internal pipeline registers.

With only one data memory, i.e., in-place implementation is possible, because the butterfly processor is designed without pipeline stages. For example, [18] if additional pipeline stages (one for the butterfly and three for multiplier) are introduced the size of the design will insignificantly be increased. On the other hand, it is observed that the speed increases significantly. If this pipeline design is used then this will pay cost for that, namely, extra data memory for the whole FFT. In this case it has to separate read and write memories, i.e., no more in-place implementation can be done. The VHDL code for the twiddle factor multiplier is shown in the Appendix A.

The twiddle factor complex multiplier has now been introduced. If this twiddle factor is used, it is possible to design a butterfly processor for a radix-2 Cooley-Tukey FFT.
Example: Butterfly Processor

A butterfly processor for a radix-2 Cooley-Tukey FFT [18] will be designed in this example. The butterfly processor computes the two (scaled) butterfly equations since overflow will be avoided in the arithmetic. The two (scaled) butterfly equations are

\[ D_{re} + j * D_{im} = ((A_{re} + j * A_{im}) + (B_{re} + j * B_{im}))/2 \]
\[ E_{re} + j * E_{im} = ((A_{re} + j * A_{im}) - (B_{re} + j * B_{im}))/2 \]

The temporary result \( E_{re} + j * E_{im} \) is then multiplied by the twiddle factor. The VHDL code of the whole butterfly processor is illustrated in the Appendix B.

When a butterfly processor is implemented, it uses one adder, one subtraction and the twiddle factor multiplier instantiated as a component. Flip-flops have been implemented for input \( A, B \), the three table values and the output port \( D \), in order to have single input/output registered design. To design the radix-2 butterfly processor any pipeline stage is not used.

4.3 Radix-5 Algorithm Implementation

Let us consider [17] the application of the decimation-in-time method in the case where \( N \) is a product of factors that are not all necessarily equal to 2.

Let us define

\[ q_1 = p_2 p_3 \ldots p_v \]

and

\[ N = p_1 \cdot q_1 \]

The input sequence can be divided, when \( N = p_1 \cdot q_1 \), into \( p_1 \) sequences of \( q_1 \) samples.

Example:

If \( p_1 = 5 \) and \( q_1 = 4 \), then \( N = 20 \). The input sequence \( x[n] \) divides into 5 sequences and the length \( q_1 \) of these sequences is 4.

The first sequence contains \( x[0], x[5], x[10], x[15] \); the second sequence contains \( x[1], x[6], x[11], x[16] \); the third sequence will contain \( x[2], x[7], x[12], x[17] \); the forth sequence consists of \( x[3], x[8], x[13], x[18] \); and the fifth sequence consists of \( x[4], x[9], x[14], x[19] \). In general \( X[k] \) can be written as follows:
\[ X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn} \]

\[ = \sum_{r=0}^{q_1-1} x[p_1 r] W_N^{p_1rk} + \sum_{r=0}^{q_1-1} x[p_1 r + 1] W_N^{rk} W_N^{p_1rk} + \ldots \]

\[ + \sum_{r=0}^{q_1-1} x[p_1 r + p_1 - 1] W_N^{(p_1-1)k} W_N^{p_1rk} \]

or

\[ X[k] = \sum_{l=0}^{p_1-1} W_N^{lk} \sum_{r=0}^{q_1-1} x[p_1 r + l] W_N^{p_1rk} \quad (4.6) \]

The inner sums can be expressed as the \( q_1 \)-point DFTs

\[ G_t[k] = \sum_{r=0}^{q_1-1} x[p_1 r + l] W_N^{r k} \quad (4.7) \]

When the original sequence is divided into five subsequences, then \( X[k] \) can be written as

\[ X[k] = \sum_{l=0}^{4} W_N^{lk} \sum_{r=0}^{3} x[5r + l] W_N^{5rk} \quad (4.8) \]

\[ = G_0[k] + W_N^{k} G_1[k] + W_N^{2k} G_2[k] + W_N^{3k} G_3[k] + W_N^{4k} G_4[k] \]

The basic 5-point DFT operation is slightly more complicated but still an in-place computation. In the case of factors of 2 the number of multiplications can be reduced by a factor of 2 by exploiting symmetry. The flow graph of the basic computation of factors of 5 is shown in Figure 4.4.
Figure 4.4. Flow graph of basic computation for factor of 5.

\[ X[k] = G_0[k] + W_N^{(k+q_1)} G_1[k] + W_N^{2k} G_2[k] + W_N^{3k} G_3[k] + W_N^{4k} G_4[k] \]

\[ X[k+q_1] = G_0[k] + W_N^{(k+q_1)} G_1[k] + W_N^{2(k+q_1)} G_2[k] + W_N^{3(k+q_1)} G_3[k] + W_N^{4(k+q_1)} G_4[k] \]

\[ X[k+2q_1] = G_0[k] + W_N^{(k+2q_1)} G_1[k] + W_N^{2(k+2q_1)} G_2[k] + W_N^{3(k+2q_1)} G_3[k] + W_N^{4(k+2q_1)} G_4[k] \]

\[ X[k+3q_1] = G_0[k] + W_N^{(k+3q_1)} G_1[k] + W_N^{2(k+3q_1)} G_2[k] + W_N^{3(k+3q_1)} G_3[k] + W_N^{4(k+3q_1)} G_4[k] \]

\[ X[k+4q_1] = G_0[k] + W_N^{(k+4q_1)} G_1[k] + W_N^{2(k+4q_1)} G_2[k] + W_N^{3(k+4q_1)} G_3[k] + W_N^{4(k+4q_1)} G_4[k] \]

\[ k = 0, 1, \ldots, p_1 - 1 \]
Since $N=5q_1$, therefore basic complex multiplier $W_N^{q_1}$ will be as follows

$$W_N^{q_1} = e^{-j(2\pi/5)}$$

Moreover, some of the advantages and disadvantages can be indicated of using values of $N$ with factor other than 2. The basic advantages are, for instance, increased flexibility and speed in some cases. If the complexity of the computational algorithm is greatly increased then it is pointed out as the basic disadvantage.

### 4.4 Implementation Of Twenty point FFT/IFFT

In chapter 3 the Cooley-Tukey FFT Algorithm was already introduced. Here, this algorithm will compute the 20-point FFT [18]. Suppose that $N_1 = 4$ and $N_2 = 5$. Then it follows that $n = 5n_1 + n_2$ and $k = k_1 + 4k_2$. The following tables compute the index mappings:

<table>
<thead>
<tr>
<th>$n_2$</th>
<th>$n_1$</th>
<th>$k_2$</th>
<th>$k_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$x[0]$</td>
<td></td>
<td>$X[0]$</td>
</tr>
<tr>
<td>0</td>
<td>$x[5]$</td>
<td></td>
<td>$X[1]$</td>
</tr>
<tr>
<td>1</td>
<td>$x[10]$</td>
<td></td>
<td>$X[2]$</td>
</tr>
<tr>
<td>6</td>
<td>$x[16]$</td>
<td></td>
<td>$X[7]$</td>
</tr>
<tr>
<td>7</td>
<td>$x[2]$</td>
<td></td>
<td>$X[8]$</td>
</tr>
<tr>
<td>8</td>
<td>$x[7]$</td>
<td></td>
<td>$X[9]$</td>
</tr>
<tr>
<td>9</td>
<td>$x[12]$</td>
<td></td>
<td>$X[10]$</td>
</tr>
<tr>
<td>12</td>
<td>$x[8]$</td>
<td></td>
<td>$X[13]$</td>
</tr>
<tr>
<td>14</td>
<td>$x[18]$</td>
<td></td>
<td>$X[15]$</td>
</tr>
<tr>
<td>15</td>
<td>$x[4]$</td>
<td></td>
<td>$X[16]$</td>
</tr>
<tr>
<td>16</td>
<td>$x[9]$</td>
<td></td>
<td>$X[17]$</td>
</tr>
<tr>
<td>17</td>
<td>$x[14]$</td>
<td></td>
<td>$X[18]$</td>
</tr>
<tr>
<td>18</td>
<td>$x[19]$</td>
<td></td>
<td>$X[19]$</td>
</tr>
</tbody>
</table>

Table 4.1. Index mapping computation

The signal flow graph can be drawn with the help of this transform. The signal flow graph is shown in Figure 4.5. The Figure 4.5 shows that five DFTs of radix-4 first has to be computed, followed by the multiplication with twiddle factors. Two radices of 2-point DFTs are used to build the radix-4 butterfly processor. The flow graph of radix-4 is shown in Appendix C. Finally, four DFTs of radix-5 is computed. To prevent overflow in the arithmetic, the results of the radix-4 butterfly are divided by 4 and to prevent overflow in the arithmetic for radix-5, the results of radix-5 will be divided by 5.
Figure 4.5. Flow graph of computation for $N = 20$ with Cooley-Tukey FFT Algorithm.
A direct computation of the 20-point DFT shows that it will \(20^2 = 400\) complex multiplications and \(19^2 = 361\) complex additions be needed. While computing the Cooley-Tukey FFT with the same length, it needs a total of 20 complex multiplications for the twiddle factors, including 8 trivial (i.e., ±1 or ±j) multiplications. The 4-point DFTs can be computed using 8 real additions and no multiplications, according to Table 3.2 [18]. It needs 16 multiplications and 20 additions for 5-point DFTs. The (fixed coefficient) complex multiplications use 3 multiplications and 3 additions (see Efficient Complex Multiplier Algorithm) and that \(W^0 = 1\) is trivial. According to Figure 4.5, the total complexity for the 20-point Cooley-Tukey FFT is given by

\[
5 \times 30 + 4 \times 16 = 214 \text{ real multiplications and}
\]

\[
5 \times 30 + 12 \times 3 + 4 \times 20 = 266 \text{ real additions}
\]

The direct implementation needs \(20^2 \times 3 = 1200\) real multiplications and \(2 \times 19^2 + 20^2 \times 3 = 1442\) real additions. Now it is very obvious after the comparison that why the Cooley-Tukey algorithm is called the “Fast Fourier Transform” FFT. Moreover, Cooley-Tukey algorithm requires only \(O(N \log_2 (N))\) operations.

The IFFT (Inverse FFT) can be defined as follows:

\[
x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k]W_{N}^{-kn}
\]

If the input read process modifies, the IFFT can be computed [19]. By computing FFT of the following sequence, the IFFT can be computed. The example of such sequence is shown in below

\[
X'[n] = \begin{cases} 
X[0]/N & \text{for } n = 0 \\
X[N-n-0]/N & \text{for } n = 1, 2, ..., N - 1 
\end{cases} \quad (4.9)
\]

Hence, at address 0 the first value is stored and the other values are stored in reverse order. By changing the address lines to the memory, this operation implements very easily in hardware. There is also another method to compute the IFFT. In this method real and imaginary parts are interchanged first. The FFT will perform afterwards. Finally, real and imaginary parts are interchanged again. The results can be compared with MATLAB built-in IFFT. This method can be used if a FFT processor is to be implemented. Some results are shown in Appendix D, which are taken from MATLAB- and MODELSIM simulations. The 20-channel FFBB network is quantized to 16 bits. The values that are taken from MODELSIM differ from MATLAB values. This difference shows that there is quantization error. However, one observation that has been made is that the more bits at the input, the smaller
quantization error at the output. In this project, three different word lengths are tested, that is, 4-bit data, 8-bit data and 16-bit data. Thus finally, it was decided to use 16-bit data in the FFBR network due to smaller quantization error compared to 4-bit data and 8-bit data. In Appendix A, the VHDL code [18] for complex multiplier is shown and in Appendix B, the VHDL code for radix-2 is shown. With the help of the VHDL code for radix-2 butterfly, the VHDL code for radix-4 will be written. In other words, radix-2 is instantiated to build a radix-4. In Appendix C, the method is illustrated to build a radix-4 by radix-2.

In Appendix D the FFT computation shows, of certain values for radix-4, that there is difference between the MATLAB result and the MODELSIM result. This means that the quantization error exists. The FFT result of radix-4 is divided by 4 in MATLAB to match the result in MODELSIM. In VHDL code for radix-4, a division by 4 is needed to avoid the overflow in the arithmetic. The comparison shows the maximum error and also the minimum error. The maximum error is given by \( y_1 \text{re}_\text{out} \) and \( y_1 \text{im}_\text{out} \) in MODELSIM. The difference is thus 0.0002 in both real and imaginary values. The minimum error is given by the rest of outputs and the difference is 0.0001 in both real and imaginary values.

The MATLAB FFT result of radix-5 is divided by 5 to match the MODELSIM result. A division by 5 is needed to avoid the overflow in the arithmetic in VHDL code for radix-5. The comparison between the two results shows the maximum error at \( y_{im}^0 \text{out} \). The maximum error is there 0.0003. The minimum error is, at \( y_{re}^4 \text{out} \), 0.0001. These two radices i.e., radix-4 and radix-5 are used to build the 20-point FFT.

The comparison shows quantization errors at different outputs in 20-point FFT result. The output \( y_{im}^1 \text{out}10 \) shows the maximum error, that is 0.0015. The outputs \( y_{re}^0 \text{out}0 \), \( y_{im}^0 \text{out}6 \), \( y_{im}^0 \text{out}12 \) show the error where the difference is 0.0005 compared to MATLAB result. The outputs where the difference is 0.0004 are \( y_{re}^2 \text{out}2 \), \( y_{re}^5 \text{out}5 \), \( y_{re}^9 \text{out}9 \), \( y_{re}^13 \text{out}13 \), \( y_{re}^18 \text{out}18 \) and \( y_{im}^1 \text{out}19 \). The minimum error is 0.0001 at \( y_{re}^8 \text{out}8 \). An observation is made that maximum quantization error has not increased as it was expected. However, the minimum quantization error is constant that is, 0.00001.
CONCLUSION AND FUTURE WORK

In this thesis, a 20-channel FFBR network has been introduced. The FFBR network consists of different blocks, for instance, polyphase components, DFT, IDFT, complex multipliers, and input/output commutators.

20-point DFT/IDFT is used in the FFBR network. The radix-4 and radix-5 are used to build the 20-point DFT/IDFT. This combination is very rare because the common way to use a DFT/IDFT of size \(N = 2^n\), where \(N\) is an even integer. The number of bits used in the FFBR for input/output is 16. There is quantization error as shown when the MATLAB results are compared to VHDL results. One important thing is observed that if the word length increases, the quantization error will be decreased. For example with 4-bit word length the quantization error was larger compared to 8-bit and 16-bit word length. The quantization error is then least as 16-bit word length compares to 8-bit word length. The Figure 5.1 shows the relation between quantization error and word length.

![Figure 5.1. Relation between quantization error and word length](image.png)
Twenty FIR filters at the input and twenty FIR filters at the output are used in the FFBR network. The order of the FIR filter is 68. The polyphase decomposition has been used in the FFBR network because it reduces the implementation cost. However, the total number of multiplications and additions are not changed. One more important thing to observe that polyphase decomposition makes it possible to have a system where the filters operate at the lowest possible frequency.

There are some areas where it is possible to do some work in the future. These areas are for instance:

- A consideration is to implement the whole project on FPGA.
- There are different FFT architectures to build FFT, therefore it is possible to use any FFT architecture to build such FFT which has low implementation cost, uses low power, shows better performance etc.
- A 20-channel FFBR network is used in this project. Suppose, if the number of channels increases, that is $N = 21, 22, \ldots$, then the complexity of FFBR network will be increased too, but at the same time the GBs will be decreased.
- Polyphase decomposition and a lot of multipliers in the FFBR network is used. To reduce the number of multipliers in the network, some other realization methods can be used with the combination of polyphase decomposition.
REFERENCES


APPENDIX A

VHDL code: complex multiplier

LIBRARY lpm;
USE lpm.lpm_components.ALL;

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY cmult IS
    GENERIC (W2 : INTEGER := 33; -- Multiplier bit width
              W1 : INTEGER := 17; -- Bit width c+s sum
              W   : INTEGER := 16); -- Input bit width
    PORT (clk  : STD_LOGIC; -- Clock for the output register
          x_in, y_in, c_in -- Inputs
            : IN  STD_LOGIC_VECTOR(W - 1 DOWNTO 0);
          cps_in, cms_in -- Inputs
            : IN  STD_LOGIC_VECTOR(W1 - 1 DOWNTO 0);
          r_out, i_out -- Results
            : OUT STD_LOGIC_VECTOR(W - 1 DOWNTO 0));
END cmult;

ARCHITECTURE fpga OF cmult IS
    SIGNAL x, y, c : STD_LOGIC_VECTOR(W - 1 DOWNTO 0); -- Inputs and outputs
    SIGNAL r, i, cmsy, cpsx, xmyc : STD_LOGIC_VECTOR(W2 - 1 DOWNTO 0); -- Products
    SIGNAL xmy, cps, cms, sxtx, sxty : STD_LOGIC_VECTOR(W1 - 1 DOWNTO 0); -- x-y etc.

    BEGIN
        x <= x_in; -- x
        y <= y_in; -- y
        c <= c_in; -- cos
        cps <= cps_in; -- cos + sin
        cms <= cms_in; -- cos * sin

        PROCESS
            BEGIN
                WAIT UNTIL clk='1';
                r_out <= r(W2-3 DOWNTO W-1); -- Scaling and FF
                i_out <= i(W2-3 DOWNTO W-1); -- for output
            END PROCESS;

        sxtx <= x(x'high) & x; -- Possible growth for
        sxty <= y(y'high) & y; -- sub_1 -> sign extension

        csb_1: lpm_add_sub
            SUB: x • y;
        GENERIC MAP ( LPM_WIDTH => W1, LPM_DIRECTION => "SUB",
                      LPM_REPRESENTATION => "SIGNED")
        PORT MAP (dataa => sxtx, datab => sxty, result => xmy);

        cmul_1: lpm_mult
            -- Multiply (x•y)•c = xmyc
        GENERIC MAP ( LPM_WIDTHA => W1, LPM_WIDTHB => W,
                      LPM_WIDTHP => W2, LPM_WIDTHS => W2,
                      LPM_REPRESENTATION => "SIGNED")

    END fpga;
PORT MAP (dataa => xmy, datab => c, result => xmyc);

cmul_2: lpm_mult
   -- Multiply (c-s)*y = cmsy
   GENERIC MAP (LPM_WIDTHA => W1, LPM_WIDTHB => W, LPM_WIDTHP => W2, LPM_WIDTHS => W2, LPM_REPRESENTATION => "SIGNED")
   PORT MAP (dataa => cms, datab => y, result => cmsy);

cmul_3: lpm_mult
   -- Multiply (c+s)*x = cpsx
   GENERIC MAP (LPM_WIDTHA => W1, LPM_WIDTHB => W, LPM_WIDTHP => W2, LPM_WIDTHS => W2, LPM_REPRESENTATION => "SIGNED")
   PORT MAP (dataa => cps, datab => x, result => cpsx);

csub_2: lpm_add_sub
   -- Sub: i <= (c-s)*x - (x-y)*c;
   GENERIC MAP (LPM_WIDTH => W2, LPM_DIRECTION => "SUB", LPM_REPRESENTATION => "SIGNED")
   PORT MAP (dataa => cpsx, datab => xmyc, result => i);

cadd_1: lpm_add_sub
   -- Add: r <= (x-y)*c + (c+s)*y;
   GENERIC MAP (LPM_WIDTH => W2, LPM_DIRECTION => "ADD", LPM_REPRESENTATION => "SIGNED")
   PORT MAP (dataa => cmsy, datab => xmyc, result => r);

END fpga;
APPENDIX B

VHDL code: radix-2 butterfly

LIBRARY lpm;
USE lpm.lpm_components.ALL;

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;

-- PACKAGE mul_package IS -- User-defined components
-- COMPONENT cmult
-- GENERIC (W2 : INTEGER := 33; -- Multiplier bit width
-- W1 : INTEGER := 17; -- Bit width c+s sum
-- W : INTEGER := 16); -- Input bit width
-- PORT
-- (clk : IN STD_LOGIC; -- Clock for the output register
-- x_in, y_in, c_in : IN STD_LOGIC_VECTOR(W-1 DOWNTO 0);
-- -- Inputs
-- cps_in, cms_in : IN STD_LOGIC_VECTOR(W1-1 DOWNTO 0);
-- -- Inputs
-- r_out, i_out : OUT STD_LOGIC_VECTOR(W-1 DOWNTO 0);
-- -- Results
-- END COMPONENT;
--END mul_package;

LIBRARY work;
USE work.mul_package.ALL;

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
LIBRARY lpm;
USE lpm.lpm_components.ALL;

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_arith.ALL;
--USE ieee.std_logic_unsigned.ALL;
USE ieee.std_logic_signed.ALL;
ENTITY newbutterfly IS
GENERIC (W2 : INTEGER := 33; -- Multiplier bit width
W1 : INTEGER := 17; -- Bit width c+s sum
W : INTEGER := 16); -- Input bit width
PORT
(clk : STD_LOGIC;
Are_in, Aim_in, c_in, Bre_in, Bim_in : IN STD_LOGIC_VECTOR(W-1 DOWNTO 0); --16 bits input
cps_in, cms_in : IN STD_LOGIC_VECTOR(W1-1 DOWNTO 0); -- 17 bit coefficients
Dre_out, Dim_out, Ere_out, Eim_out : OUT STD_LOGIC_VECTOR(W-1 DOWNTO 0) -- 16 bit results
:= (OTHERS => '0'));
END newbutterfly;

ARCHITECTURE fpga OF newbutterfly IS
SIGNAL dif_re, dif_im : STD_LOGIC_VECTOR(W-1 DOWNTO 0); -- BF out

47
SIGNAL Are, Aim, Bre, Bim : INTEGER RANGE -32768 TO 32767:=0;

.-- Inputs as integers
SIGNAL c : STD_LOGIC_VECTOR(W-1 DOWNTO 0)
:= (OTHERS => '0'); -- Input
SIGNAL cps, cms : STD_LOGIC_VECTOR(W-1 DOWNTO 0)
:= (OTHERS => '0'); -- Coeff in
SIGNAL EEre_out, EEim_out : STD_LOGIC_VECTOR(W-1 DOWNTO 0);

BEGIN

PROCESS -- Compute the additions of the butterfly using
--variable temp: integer :=0;
BEGIN -- integers and store inputs in flip-flops

WAIT UNTIL clk = '1';
Are <= CONV_INTEGER(Are_in);
Aim <= CONV_INTEGER(Aim_in);
Bre <= CONV_INTEGER(Bre_in);
Bim <= CONV_INTEGER(Bim_in);
c <= c_in;
-- Load from memory cos
cps <= cps_in;
-- Load from memory cos+sin
cms <= cms_in;
-- Load from memory cos-sin
Dre_out <= CONV_STD_LOGIC_VECTOR((Are + Bre )/2, W);
Dim_out <= CONV_STD_LOGIC_VECTOR((Aim + Bim)/2, W);

--temp := CONV_INTEGER(EEre_out);
--temp := -temp;
--Ere_out <= CONV_STD_LOGIC_VECTOR(temp,W);
--temp := CONV_INTEGER(EEim_out);
--temp := -temp;
--Eim_out <= CONV_STD_LOGIC_VECTOR(temp,W);

END PROCESS;

-- No FF because butterfly difference "diff" is not an
PROCESS (Are, Bre, Aim, Bim) -- output port
BEGIN
dif_re <= CONV_STD_LOGIC_VECTOR(Are/2 - Bre/2, 16);
dif_im <= CONV_STD_LOGIC_VECTOR(Aim/2 - Bim/2, 16);

--variable temp : EEre_out ;
END PROCESS;

---- Instantiate the complex twiddle factor multiplier ----
cmul_1: cmult
GENERIC MAP ( W2 => W2, W1 => W1, W => W)
PORT MAP ( clk => clk, x_in => dif_re, y_in => dif_im,
c_in => c, cps_in => cps, cms_in => cms,
r_out => Ere_out, i_out => Eim_out);

END fpga;
APPENDIX C

radix-2 Decimation-in-Time FFT algorithm for a length-4 signal
### APPENDIX D

#### MATLAB result

<table>
<thead>
<tr>
<th>Input $\rightarrow$ FFT</th>
<th>MODELSIM result</th>
<th>Difference (Quantization error)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real part</td>
<td>Imaginary part</td>
<td>Real part</td>
</tr>
<tr>
<td>-9074</td>
<td>-22768i</td>
<td>4895</td>
</tr>
<tr>
<td>-1512</td>
<td>-10390i</td>
<td>-5944</td>
</tr>
<tr>
<td>3182</td>
<td>-31136i</td>
<td>-7841</td>
</tr>
<tr>
<td>26983</td>
<td>+1128i</td>
<td>-185</td>
</tr>
</tbody>
</table>

Radix-4 FFT results (Comparison between MATLAB and MODELSIM)

#### MATLAB result

<table>
<thead>
<tr>
<th>Input $\rightarrow$ FFT</th>
<th>MODELSIM result</th>
<th>Difference (Quantization error)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real part</td>
<td>Imaginary part</td>
<td>Real part</td>
</tr>
<tr>
<td>-26696</td>
<td>-3196i</td>
<td>-413</td>
</tr>
<tr>
<td>29680</td>
<td>-9125i</td>
<td>-7386</td>
</tr>
<tr>
<td>4161</td>
<td>+17920i</td>
<td>-17317</td>
</tr>
<tr>
<td>-29929</td>
<td>+31375i</td>
<td>-12857</td>
</tr>
<tr>
<td>20721</td>
<td>+31617i</td>
<td>11276</td>
</tr>
</tbody>
</table>

Radix-5 FFT results (Comparison between MATLAB and MODELSIM)
<table>
<thead>
<tr>
<th>MATLAB result</th>
<th>MODELSIM result</th>
<th>Difference (Quantization error)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Real part</strong></td>
<td><strong>Imaginary part</strong></td>
<td><strong>Real part</strong></td>
</tr>
<tr>
<td>-26696</td>
<td>+21487i</td>
<td>757</td>
</tr>
<tr>
<td>-29680</td>
<td>-1170i</td>
<td>-2070</td>
</tr>
<tr>
<td>4161</td>
<td>+27824i</td>
<td>6624</td>
</tr>
<tr>
<td>29929</td>
<td>+30605i</td>
<td>-11158</td>
</tr>
<tr>
<td>-20721</td>
<td>+22240i</td>
<td>-8093</td>
</tr>
<tr>
<td>-3196</td>
<td>+24829i</td>
<td>2854</td>
</tr>
<tr>
<td>-9125</td>
<td>+24350i</td>
<td>-6557</td>
</tr>
<tr>
<td>17920</td>
<td>-12852i</td>
<td>3074</td>
</tr>
<tr>
<td>31375</td>
<td>-21478i</td>
<td>1721</td>
</tr>
<tr>
<td>31617</td>
<td>-5609i</td>
<td>5595</td>
</tr>
<tr>
<td>-5164</td>
<td>+23135i</td>
<td>1118</td>
</tr>
<tr>
<td>31804</td>
<td>-1043i</td>
<td>2460</td>
</tr>
<tr>
<td>31364</td>
<td>+9074i</td>
<td>470</td>
</tr>
<tr>
<td>-15904</td>
<td>-1512i</td>
<td>601</td>
</tr>
<tr>
<td>-26223</td>
<td>+3182i</td>
<td>-4577</td>
</tr>
<tr>
<td>-4649</td>
<td>-26983i</td>
<td>1099</td>
</tr>
<tr>
<td>13820</td>
<td>-22768i</td>
<td>-4781</td>
</tr>
<tr>
<td>-30006</td>
<td>-10390i</td>
<td>590</td>
</tr>
<tr>
<td>25959</td>
<td>+31136i</td>
<td>-2611</td>
</tr>
<tr>
<td>-31440</td>
<td>-1128i</td>
<td>-13811</td>
</tr>
</tbody>
</table>

20-point FFT results (Comparison between MATLAB and MODELSIM)