Design and Implementation of Wide Band Quadrature Demodulators on Field Programmable Gate Arrays

by

Lieutenant(N) Joseph Mathieu Pierre Langlois, CD, rmc, BEng
Canadian Armed Forces

TRUTH • DUTY • VALOUR

A thesis
Presented to the School of Graduate Studies in the
Department of Electrical and Computer Engineering
Royal Military College of Canada
Kingston, Ontario

In partial fulfillment of the requirements for the degree
Master of Engineering
April 1999

0-612-44914-9

Abstract

Traditional digital implementations of high performance, wide band quadrature demodulators have targeted gate arrays and custom Application Specific Integrated Circuit (ASIC) technologies. These technologies involve significant non-recurring engineering costs. In this thesis, it is demonstrated that Field Programmable Gate Arrays (FPGAs) are a viable alternative, providing system performance in the same order of magnitude, with a significant reduction in non-recurring engineering costs.

Considerations relevant to the design and implementation of wide band quadrature demodulators are described. Fundamental principles for their digital realization are discussed and different theoretical approaches are presented. Specific attention is paid to the selection of digital filter architectures that map well to FPGA Configurable Logic Blocks (CLB), and to constant coefficient multiplier implementation.

The delayed-carry chain concept is proposed as an extension to traditional pipelining methods for multi-operand adders. The proposed concept was specifically applied to digital filter implementations following the so-called transposed form. The concept presents a significantly reduced overhead for a given performance criterion, especially for high filter orders. It is ideally suited to FPGA implementations and to other register-rich hardware technologies.

Four VHDL-based designs meeting different sets of specifications are described. One particular design implemented in a low speed grade FPGA is suitable for the processing of input signals on a 160 MHz Intermediate Frequency (IF) with a maximum theoretical bandwidth of 53.3 MHz. Implementation in a faster FPGA family would support a 100 MHz bandwidth signal centered on a 100 MHz IF.

Keywords: Quadrature Demodulation, Field Programmable Gate Arrays, Digital Filtering, VHDL Design Synthesis

Acknowledgements/ Remerciements

Many people deserve my sincere gratitude for their direct or indirect help and support related to this thesis. First, Dr. Dhamin Al-Khalili of RMC and Mr. Robert Inkol of DREO, for their time and patience with my endless streams of questions, for their countless useful suggestions and for their wise guidance.

Au CMR, le Dr. Jean Fugère; c'est grâce à sa flexibilité que j'ai pu poursuivre ces études; merci beaucoup. Dr. Y.T. Chan, for interceding in my favour.

At CRC, Dr. Valek Szwarc, for his cooperation and support. Mr. Luc Désormeaux for his insights into CRC's designs. Mr. Richard Wojcik, for coming in even though he had a cold. Ms. Huong Ho, for her useful suggestions.

At RMC, Mrs. Nancy Turkington for her outstanding service in the library. Mr. Pierre Adam, Mr. Scott Sararas, Ms. Kim Sararas, Mr. Wolf Rogal, and Mr. Chris Lai for their support, ideas, suggestions and great service.

Mr. Bob Stevenson of CMC for my invasion of his office.

My colleagues Capt. John Buitenga, Capt. Bryan Wright, Capt. Don Shaw, Maj. Stéfan Richard and Dr. Derrick Bouchard, for the free sharing of ideas. Maj. François Laboissonière pour les discussions, LCdr Mike Bourassa for getting me back on my board, Capt. Stéfan Fournier pour le squash, le support et le disque dur, Capt. Hughes Latour pour le support avec Unix, les diverses licences et le squash, et Lt(M) Claude Bernard, pour quand nous voulions prendre des ris mais que nous ne le pouvions pas.

Mon épouse, Caroline, pour tout le reste; sans elle, çà n'en aurait pas valu la peine.

Dédicace

À Caroline, mon épouse et meilleure amie,

et à Étienne, que j'attendais depuis si longtemps

Vita

Pierre Langlois was born in Sherbrooke, Québec, in 1967. After completing high school at Le Séminaire de Sherbrooke, he enrolled in the Canadian Forces and reported to the Collège Militaire Royal de St-Jean in 1985. He graduated from the Royal Military College, in Kingston, Ontario, in May 1990 with a degree in Electrical Engineering.

After completing Combat Systems Engineering training in various shore establishments and naval destroyers on Canada's both coasts, he served as project engineer with the Tribal Upgrade and Modernization Program (TRUMP), at the MIL Davie Shipyard in Lévis, Québec, from 1992 to 1994. He returned to sea in 1995 on board *HMCS IROQUOIS*, based in Halifax, Nova Scotia, to complete the Head of Department qualification. He then served as an instructor for radar and Electronic Warfare systems at the Canadian Forces Naval Engineering School, in Halifax, from January 1996 until August 1997, when he came back to RMC to undertake graduate studies.

Upon completion of his studies, he will remain at RMC as a lecturer in the Mathematics and Computer Science Department.

Contents

List of Figures

List of Tables

Acronyms

ADC	Analog-to-Digital Converter
ASIC	Application-Specific Integrated Circuit
BIST	Built-In Self Test
CLB	Configurable Logic Bloc
CMC	Canadian Microelectronics Corporation
CMOS	Complementary Metal Oxide Semiconductor
CMR	Collège Militaire Royal
CRC	Communications Research Center
CSD	Canonical Signed Digit
DC	Direct Current
DREO	Defence Research Establishment Ottawa
DUT	Device Under Test
ECL	Emitter Coupled Logic
EDA	Electronic Design Automation
EW	Electronic Warfare
FIR	Finite Impulse Response
FPGA	Field-Programmable Gate Array
HDL	Hardware Description Language
HMCS	Her Majesty's Canadian Ship
IF	Intermediate Frequency
IIR	Infinite Impulse Response
IMS	Integrated Measurement System
IOB	Input/Output Block
KCM	Constant-Coefficient Multiplier
LFSR	Linear Feedback Shift Register
LSB	Least Significant Bit
LUT	Look-Up Table
MHz	Megahertz

MIL	Marine Industries Limited
RF	Radio Frequency
RMC	Royal Military College
ROM	Read-Only Memory
TRUMP	Tribal Upgrade and Modernization Program
TTL	Transistor-Transistor Logic
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VLSI	Very Large Scale Integration

Chapter 1

Introduction

1.1 Overview

Quadrature demodulation is a process for obtaining a complex baseband representation of a real bandpass signal. It has a wide variety of applications in areas such as radar and sonar signal processing, digital communications, and biological signal analysis. The real signal obtained from a transducer such as an antenna, a hydrophone or a biological probe, is amplified, filtered, and possibly shifted to an appropriate Intermediate Frequency (IF) before quadrature demodulation. Once this process is done, the resulting complex signal representation contains the information present in the original signal, and its format facilitates subsequent processing, such as spectral analysis or the extraction of modulation information.

This thesis addresses the implementation of quadrature demodulators in Field Programmable Gate Array (FPGA) technology. FPGAs offer some attractive advantages over other implementations of Application Specific Integrated Circuits (ASICs). They are easily programmable, which means that a design can be implemented and tested in a very short time, reducing development time and cost dramatically. They are also re-programmable, so modifications can be made and tested in the field. A single chip may even have multiple purposes on one board, as it can be reprogrammed in-situ. Non-recurring engineering costs are also orders of magnitude lower than for other ASICs.

One disadvantage of FPGAs over full-custom design ASICs is the speed of operation. The achievable data rates have traditionally been well below the performance attainable with other technologies such as Gallium Arsenide. Therefore, a major challenge for the designs considered here will be to meet performance requirements normally associated with full custom designs on a FPGA. An additional disadvantage is the limitations in the acceptable input signal format. While other technologies allow level translation to be done on chip, most FPGAs require external hardware to do this.

The quadrature demodulators considered in the present research are primarily aimed at radar and Electronic Warfare (EW) applications. These fields pose particular design challenges when compared to sonar, communications or bio-medical applications. The most obvious differences are the high frequency and wide bandwidth requirements. The IF is typically in the MHz range, and the signal bandwidth may extend from DC to twice the IF. In many applications, there is a need for real-time operation. Consequently, techniques such as parallelism and pipelining may be necessary to achieve adequate processing speed. Another system requirement is the preservation of the input signal waveform and of any information contained in the signal modulation.

In an EW receiver, for which a conceptual block diagram is shown in Figure 1-1, intercepted radar pulses are first picked up by an antenna before being amplified and pre-filtered by a Radio-Frequency (RF) amplifier. They are then mixed down to a convenient Intermediate Frequency (IF), in one or more frequency conversion stages. The signal is then further amplified and sum frequency components are suppressed by an IF amplifier. After this step, quadrature demodulation is performed to convert the IF signal to a complex baseband representation with in-phase and quadrature components. This complex representation facilitates further processing to extract information contained in the radar pulses.

1.2 Motivation

Quadrature demodulation has been the subject of a collaborative research programme involving the Defense Research Establishment Ottawa (DREO), the Communications Research Center (CRC) and the Royal Military College (RMC). Significant contributions have been made to the field. A number of hardware implementations of wide band receivers with progressively improving performance have been developed using CMOS and GaAs ASIC technologies. The most recent designs have provided considerable additional functionality in addition to quadrature demodulation.

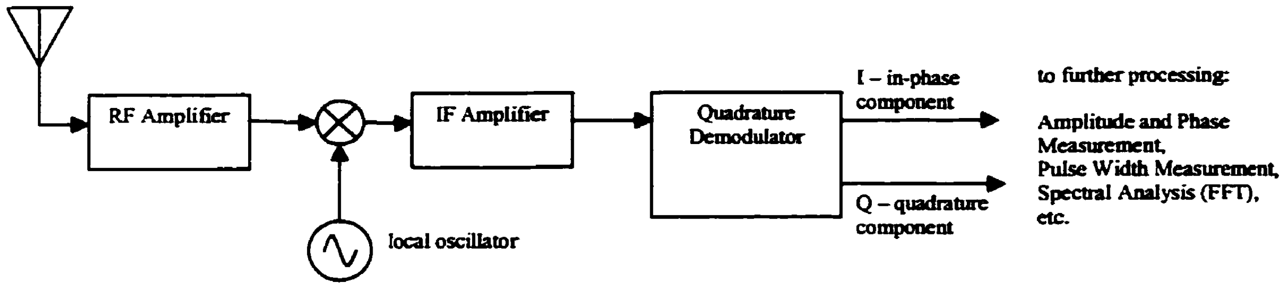

Figure 1-1 - Conceptual Block Diagram of RF Portion of EW Receiver

The advent of FPGAs, with their ability to implement complex systems with ever increasing performance [1] and low non-recurring engineering costs, opens up a wealth of new possibilities. In-system, on-board re-programmability means that designs can go from concept to field application in a very short time. Further, design changes can be quickly put in place and results observed.

There are many reasons to pursue further research in this specific field. Programmable logic poses special problems, but also presents appealing advantages over custom ASIC VLSI designs. A number of novel approaches to quadrature demodulation have surfaced in recent years, as have ingenious ways to implement digital filtering with increased efficiency. Ideas that have been previously discarded as being impractical with technologies available at the time deserve new consideration for programmable logic implementations. Finally, there has been on-going research at DREO into the selection of filter coefficients for quadrature demodulation specifically, and there is a need to validate results in hardware. The implementation of quadrature demodulation using FPGA technology is therefore a pertinent and promising area of research.

1.3 Objectives

Given the present subject, the research emphasis must be focused in a few specific directions. First, it is intended to assess the viability of using FPGAs for wide-band, high performance quadrature demodulators. This means that the effective data processing rate must be as high as possible so that the input signal bandwidth can be maximized. The designs should be optimized as well to minimize power consumption. For FPGA realizations, this means that special consideration will have to be given to architectural issues.

It is also desired to make a contribution to DREO's filter coefficient research, with the specific aim of producing filter designs whose coefficients minimize the hardware costs. This aim would also be in line with contributing to the general problem of fast digital filtering, with the specific considerations that apply to FPGA-based designs.

Finally, the design entry method selected for this research will be a Hardware Description Language (HDL). A synthesis tool will then be used to generate a design netlist. This will pose special challenges given that stringent performance requirements must be met. It is therefore hoped that a contribution can be made to the problem of optimizing the hardware realization of HDL-based designs.

1.4 Synopsis

This thesis is divided into 7 chapters. Chapter 2 will present fundamental principles for the digital realization of a quadrature demodulator. In Chapter 3, the specific problem of multiplier implementation will be presented. Chapter 4 will cover the very important topic of selection of a filter architecture. It is in Chapter 5 that a detailed description of quadrature demodulator designs will be made. Chapter 6 will deal with design verification and testing. In Chapter 7, conclusions will be drawn and recommendations for further work will be made.

Chapter 2

Digital Implementation of Quadrature Demodulation

2.1 Introduction

The input signal to a quadrature demodulator can be described by:

$$x(t) = A(t)\cos(\omega_c t + \phi(t)) \tag{2-1}$$

where $A(t)$ is the signal amplitude, ω_c its carrier frequency (in radians/second), and $\phi(t)$ its time varying phase angle. The signal $x(t)$ is assumed to be a real bandpass[1] signal.

The goal of quadrature demodulation is to express the signal $x(t)$ as a function of the in-phase and quadrature components $I(t)$ and $Q(t)$ as follows:

$$x(t) = I(t)\cos(\omega_c t) - Q(t)\sin(\omega_c t) \tag{2-2}$$

where $I(t)$ and $Q(t)$ are both functions of $A(t)$ and $\phi(t)$, and are equal to

[1] A bandpass signal is centered on a frequency other than 0 Hz and has a finite bandwidth.

$$I(t) = A(t)\cos(\phi(t)) \qquad\qquad Q(t) = A(t)\sin(\phi(t)) \tag{2-3}$$

Once the in-phase and quadrature components are available, the amplitude and phase information in the signal *x*(*t*) can be calculated as

$$A(t) = \sqrt{I(t)^2 + Q(t)^2} \qquad\qquad \phi(t) = \tan^{-1}(\frac{Q(t)}{I(t)}) \tag{2-4}$$

The traditional analog approach to quadrature demodulation is shown in Figure 2-1. The signal to be demodulated (in this case the output of the IF amplifier) is multiplied by two sinusoids with a 90 degree phase angle difference. This effectively creates quadrature versions of the signal nominally centered around zero frequency and at twice the carrier frequency. The signal component centered about $2\omega_c$ is then removed by low-pass filtering to leave a complex baseband signal. If a digital representation of the in-phase and quadrature components is desired, analog-to-digital conversion is performed.

From the block diagram, the in-phase component, *I*(*t*), is equal to:

$$\begin{aligned} I(t) &= LPF\{x(t) \times 2\cos\omega_c t\} \\ &= LPF\{2A(t)\cos(\omega_c t + \phi(t)) \times \cos(\omega_c t)\} \\ &= LPF\{A(t)[\cos(2\omega_c t + \phi(t)) + \cos(\phi(t))]\} \\ &= A(t)\cos(\phi(t)) \end{aligned} \tag{2-5}$$

Similarly, the quadrature component, *Q*(*t*), is equal to:

$$\begin{aligned} Q(t) &= LPF\{x(t) \times (-2\sin\omega_c t)\} \\ &= LPF\{-2A(t)\cos(\omega_c t + \phi(t)) \times \sin(\omega_c t)\} \\ &= LPF\{A(t)[-\sin(2\omega_c t + \phi(t)) + \sin(\phi(t))]\} \\ &= A(t)\sin(\phi(t)) \end{aligned} \tag{2-6}$$

By substituting equations (2-5) and (2-6) in equation (2-2), we get:

$$\begin{aligned} x(t) &= I(t)\cos(\omega_c t) - Q(t)\sin(\omega_c t) \\ &= A(t)\cos(\omega_c t)\cos(\phi(t)) - A(t)\sin(\omega_c t)\sin(\phi(t)) \\ &= \frac{1}{2}A(t)\left[\cos(\omega_c t + \phi(t)) + \cos(\omega_c t - \phi(t))\right] \\ &\quad - \frac{1}{2}A(t)\left[\cos(\omega_c t - \phi(t)) - \cos(\omega_c t + \phi(t))\right] \\ &= A(t)\cos(\omega_c t + \phi(t)) \end{aligned} \qquad \textit{(2-7)}$$

which is the same as the original equation (2-1).

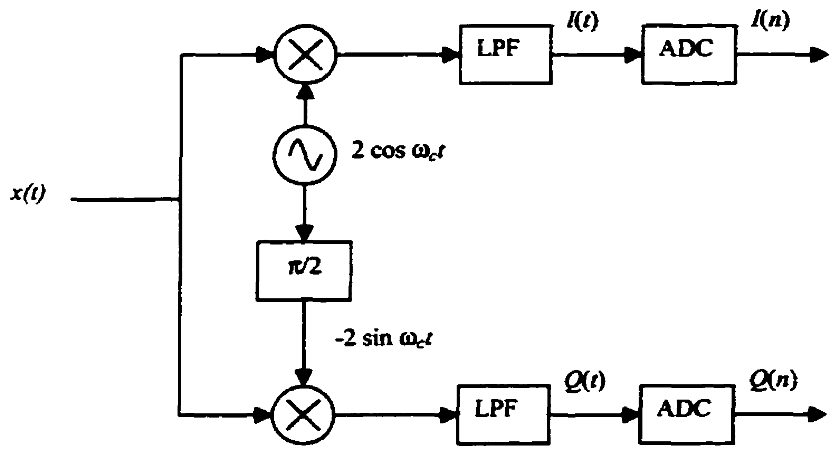

Figure 2-1 – Analog Quadrature Demodulator Block Diagram

In Figure 2-1, the modulating signals *cos* and *sin* are shown with an amplitude of 2 to make the input and output power levels equal. In practice, however, this factor is often neglected and it will not be considered in subsequent discussion.

The quadrature representation of signals can also be viewed from a complex number perspective. The received signal *x(t)* can be expressed as:

$$x(t) = A(t)e^{j(\omega_c t + \phi(t))} \qquad \textit{(2-8)}$$

and the two quadrature components *I(t)* and *Q(t)* are the baseband real and imaginary parts of *x(t)*, respectively.

Figure 2-2 shows two spectra relevant to the analog implementation of the quadrature demodulator. The top spectrum represents the bandpass signal $x(t)$, centered on a frequency f_c, with a bandwidth B. The lower spectrum is the magnitude of $x(t)$ shifted in frequency down to baseband, showing the high-frequency modulation products having been removed by low-pass filtering and the remaining bandwidth $B/2$.

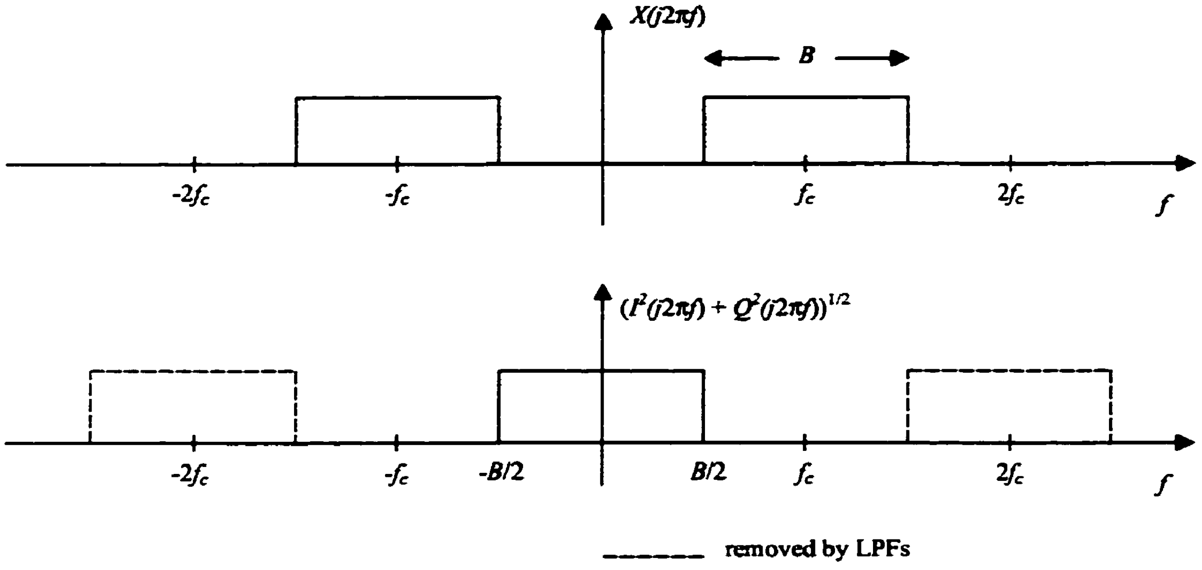

Figure 2-2 - Spectra for Analog Quadrature Demodulation

2.2 Basic Digital Approach to Quadrature Demodulation

The traditional analog implementation of quadrature demodulation, shown in Figure 2-1, suffers from many problems, especially to gain and phase mismatches between the I and Q channels and the presence of DC offsets [2]. In such an implementation, all processing, with the exception of the Analog-to-Digital Converters (ADC) used to digitize the I and Q signals, is carried out by analog circuits.

A more robust implementation is all digital, as shown in Figure 2-3. In this case, only one ADC is used, and the processes of down-conversion and filtering are done digitally. The last step is usually a decimation by an integer factor M, where only 1 out of M

samples are kept. The value of M depends on the initial sampling rate and on the bandwidth of the signal $x(t)$.

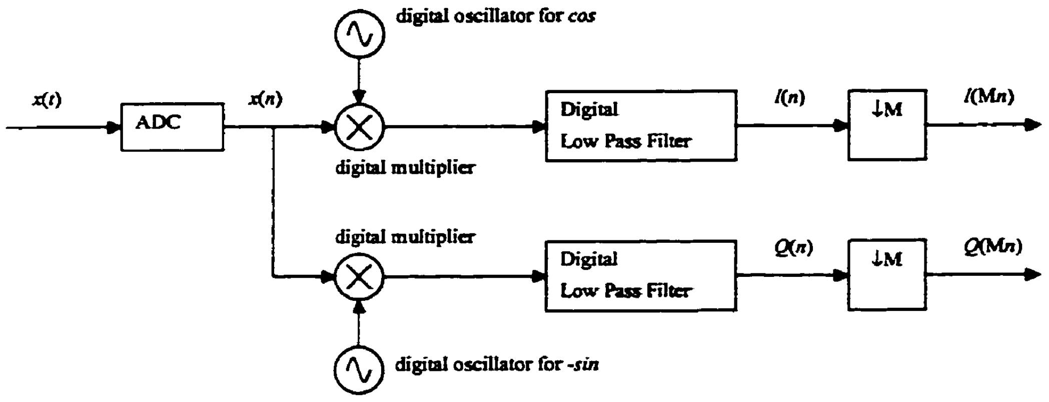

Figure 2-3 - Digital Quadrature Demodulator

2.2.1 Sampling Frequency Selection

The first processing step after conversion of the input analog signal to a digital format is frequency shifting to baseband. It involves multiplying the input data by cosine and sine sequences at the center frequency of the input signal, as shown previously. This step can be quite complex, first requiring the generation of the two sinusoids (possibly with a Look-Up Table approach), then their multiplication with the stream of input data. However, a careful selection of the sampling frequency can greatly simplify this problem. If it is selected such that $f_s = 4 \times f_c$, then the two sequences are represented by $cos(\pi n/2)$ and $-sin(\pi n/2)$, which reduce to:

cos: 1, 0, -1, 0, 1, 0, -1, 0, ...

-sin: 0, -1, 0, 1, 0, -1, 0, 1, ...

Obviously, multiplications by 0 or 1 are trivial. For multiplication by -1, the only processing required is sign inversion, an operation whose complexity depends on the number representation of the data. However, in the worst case (for 2's complement representation), the operation is simpler than the addition of two numbers, with each bit inverted and a carry added to the Least Significant Bit.

2.2.2 In-Phase and Quadrature Digital Filters

The second step after multiplication of the input data by quadrature sinusoids is low-pass filtering, where unwanted high-frequency mixing products are removed to obtain the results of equations (2-5) and (2-6). In a digital implementation of quadrature demodulation, digital filters are used.

The phase linearity of the filters used in the quadrature demodulator is an issue. For many applications such as those targeted for this research (radar and EW receivers), it is essential to preserve the information contained in the original signals. A non-linear phase filter is therefore unsuitable for the quadrature demodulator designs considered here.

While Infinite Impulse Response (IIR) filters usually have sharper transition bands than Finite Impulse Response (FIR) filters for a given filter order, they cannot have a linear phase characteristic [3], and thus they are not considered further in this document. FIR filters, however, can exhibit ideal linear phase under some conditions that will be described later. Therefore, the quadrature demodulator designs considered here will be restricted to linear-phase FIR filters.

The output $y(n)$ from a FIR filter with impulse response $h(n)$, filter length N, and input sequence $x(n)$ is given by the convolution of the input sequence and the filter impulse response:

$$y(n) = x(n) * h(n) = h(n) * x(n) = \sum_{m=0}^{N-1} h(m)x(n-m) \qquad (2\text{-}9)$$

In the basic quadrature demodulation approach, the two low-pass filters are identical, and the filter they reproduce is called the *prototype* filter. The impulse response of the low-pass prototype filter will hereafter be denoted by $h_{LP}(n)$. Its cutoff frequency, transition bandwidth and stopband attenuation are selected according to the characteristics of the signal to be demodulated, and especially the signal bandwidth. From the spectra shown in Figure 2-2, it should be obvious that the passband of the filter should be at least equal to $B/2$, where B is the bandwidth of the signal prior to demodulation.

From equation (2-9), given that the system sampling frequency is selected as $f_s = 4 \times f_c$, and that the prototype low-pass filter has an impulse response $h_{LP}(n)$, the output of the in-phase channel can be expressed as:

$$\begin{aligned} I(n) &= h_{LP}(n) * \left[x(n) \cdot \cos(\frac{\pi}{2} n) \right] \\ &= \sum_{m=0}^{N-1} h_{LP}(m) \cdot x(n-m) \cdot \cos(\frac{\pi}{2}(n-m)) \end{aligned} \qquad \textit{(2-10)}$$

and the output of the quadrature channel can be expressed as:

$$\begin{aligned} Q(n) &= h_{LP}(n) * \left[x(n) \cdot -\sin(\frac{\pi}{2} n) \right] \\ &= \sum_{m=0}^{N-1} h_{LP}(m) \cdot x(n-m) \cdot \sin(-\frac{\pi}{2}(n-m)) \end{aligned} \qquad \textit{(2-11)}$$

2.2.3 Decimation of Filter Outputs

The input to the quadrature demodulator is a bandpass signal centered on a frequency of f_c. Hence, its maximum bandwidth B is $2 \times f_c$. After demodulation, all the signal's information content is stored in the in-phase and quadrature channel outputs, and each has a maximum bandwidth $B/2 = f_c$.

Ideally, the prototype low pass filters in the in-phase and quadrature channels should remove all frequency components outside of the bandwidth of interest. Therefore, according to the Nyquist criterion, the minimum sampling rate that could be used to process the in-phase and quadrature signals, without aliasing, is $2 \times f_c$. However, since the sampling frequency was selected as $f_s = 4 \times f_c$ for the advantages already mentioned, this implies that the filter outputs are oversampled by a factor equal or greater to two, and thus that every other sample, at least, can be discarded. The process of retaining only a fixed proportion of data samples is known as *decimation*. It is rarely advantageous to carry redundant information about a signal, and decimation should generally be done to maintain the lowest possible processing and communications rates in a system.

Figure 2-4 shows the signal spectra at various stages in the demodulation process for the case of decimation by a factor of two.

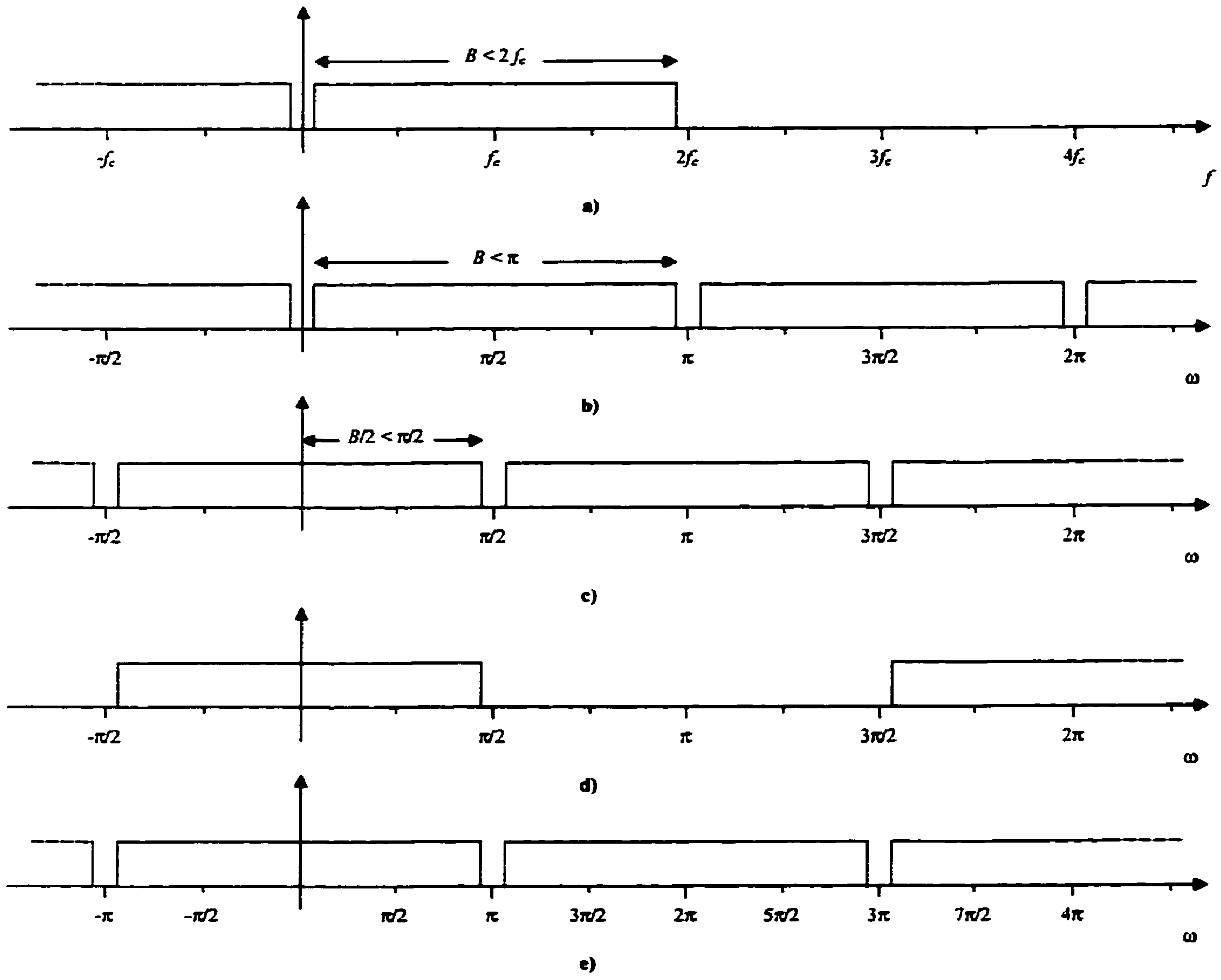

Figure 2-4 - Spectra for Decimation-by-Two Case
a) analog input signal. b) digitized signal, with the sampling frequency equal to four times the carrier frequency. c) digitized signal shifted to baseband. d) result after ideal low-pass filtering. e) result after decimation by two.

The signal bandwidth of $2 \times f_c$ discussed above represents a maximum for a signal centered on a frequency f_c. If the signal bandwidth is sufficiently smaller, then decimation by a factor greater than two is possible. For example, if the input signal to the quadrature demodulator has a maximum bandwidth of f_c, then after demodulation to baseband all that will remain will be one sideband of bandwidth $f_c/2$. According to the Nyquist criterion, the corresponding minimum sampling rate without aliasing is thus f_c. Following the approach already described, if the sampling frequency is selected as $f_s = 4 \times f_c$, then the un-decimated filter outputs are oversampled by a factor of four. Figure 2-5 shows signal spectra at different points of the processing for this case.

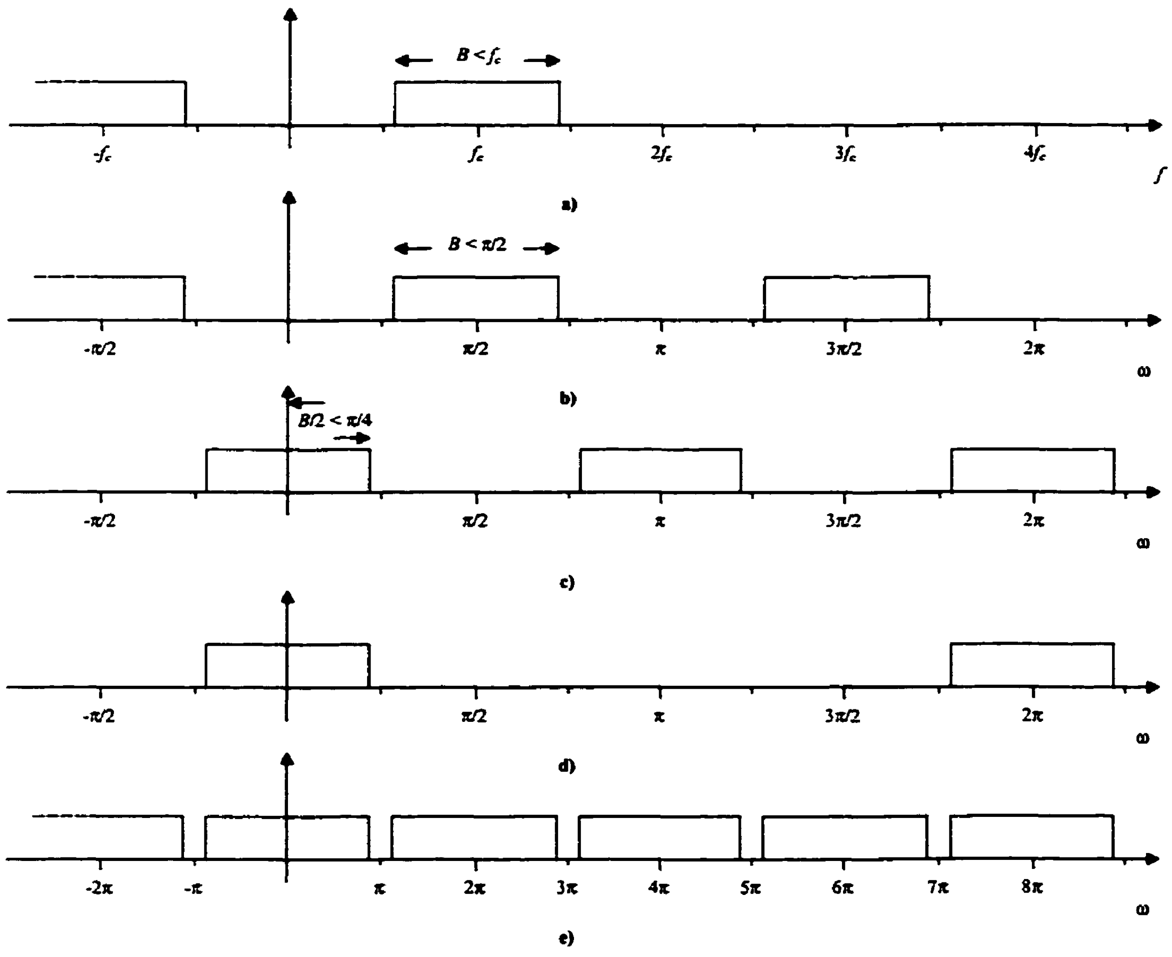

Figure 2-5 - Spectra for Decimation-by-Four Case
a) analog input signal. b) digitized signal, with the sampling frequency equal to four times the carrier frequency. c) digitized signal shifted to baseband. d) result after ideal low-pass filtering. e) result after decimation by four.

2.3 Improved Digital Approaches

In the basic digital approach to quadrature demodulation, presented in the previous section, data is processed throughout the system at the sampling rate of the ADC, then decimated at the output of the filters. This situation is a case of multirate signal processing [4]. Since it is a waste to expend resources to calculate signal data only to discard it later, a much more efficient approach is to decimate *before* the multiplication and filtering are done. This section discusses designs based on this concept.

2.3.1 Low-Pass Filter Approach

If the bandwidth of the input signal to the quadrature demodulator is such that the filter outputs are oversampled by a factor of two ($B < 2 \times f_c$), then the filter processing rate can be reduced by half [5].

Referring back to equation (2-10), decimating the output of the in-phase filter by 2 gives:

$$I(2n) = \sum_{m=0}^{N-1} h_{LP}(m) \cdot x(2n-m) \cdot \cos(\frac{\pi}{2}(2n-m)) \qquad \textbf{\textit{(2-12)}}$$

Noting that the summation terms will be null for odd values of the summation index m, a change of variable is made such that $m = 2k$ to give:

$$\begin{aligned} I(2n) &= \sum_{k=0}^{K_I} h_{LP}(2k) \cdot x(2n-2k) \cdot \cos(\frac{\pi}{2}(2n-2k)) \\ &= \sum_{k=0}^{K_I} h_{LP}(2k) \cdot x(2(n-k)) \cdot \cos(\pi(n-k)) \\ &= \sum_{k=0}^{K_I} h_{LP}(2k) \cdot x(2(n-k)) \cdot (-1)^{(n-k)} \\ &= h_{LPI}(n) * \left[(-1)^n x(2n)\right] \end{aligned} \qquad \textbf{\textit{(2-13)}}$$

where K_I is equal to $(N - 1)/2$ for N odd, and to $(N/2) - 1$ for N even. Effectively, the input data is decimated by two and sign changes are applied to alternate remaining samples. The resulting data stream is filtered by a new low-pass filter. The impulse response of the new in-phase low-pass filter $h_{LPI}(n)$ is given by:

$$h_{LPI}(n) = h_{LP}(2n) \tag{2-14}$$

for $n = 0, 1, 2, \ldots, K_I$. A block diagram of the resulting realization of the in-phase channel is given in Figure 2-6. It must be noted that the new filter processing speed is half of the input data rate.

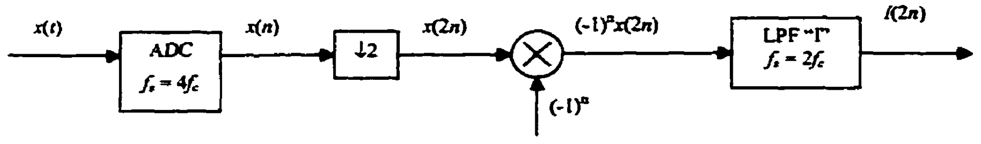

Figure 2-6 – In-Phase Channel for Decimation by Two

Similarly, starting from equation (2-11), decimating the output of the quadrature channel by 2 gives:

$$Q(2n) = \sum_{m=0}^{N-1} h_{LP}(m) \cdot x(2n-m) \cdot \sin(-\frac{\pi}{2}(2n-m)) \tag{2-15}$$

Noting that the summation terms will be null for even values of the summation index *m*, a change of variable is made such that $m = 2k+1$ to give:

$$\begin{aligned} Q(2n) &= \sum_{k=0}^{K_Q} h_{LP}(2k+1) \cdot x(2n-(2k+1)) \cdot \sin(-\frac{\pi}{2}(2n-(2k+1))) \\ &= \sum_{k=0}^{K_Q} h_{LP}(2k+1) \cdot x(2(n-k)-1) \cdot \sin(\frac{\pi}{2} - \pi(n-k)) \\ &= \sum_{k=0}^{K_Q} h_{LP}(2k+1) \cdot x(2(n-k)-1) \cdot (-1)^{(n-k)} \\ &= h_{LPQ}(n) * \left[(-1)^n x(2n-1)\right] \end{aligned} \tag{2-16}$$

where K_Q is equal to $(N-3)/2$ for N odd, and to $N/2 - 1$ for N even. Effectively, the input data is decimated by two and sign changes are applied to alternate remaining samples. It must be noted that there is one sample relative delay between the in-phase and quadrature channels. The resulting data stream is filtered by a new low-pass filter. The impulse response of the new quadrature low-pass filter $h_{LPQ}(n)$ is given by:

$$h_{LPQ}(n) = h_{LP}(2n+1) \tag{2-17}$$

for $n = 0, 1, 2, \ldots, K_Q$. A block diagram of the resulting realization of the quadrature channel is given in Figure 2-7. Again, the processing rate is half of the input data rate.

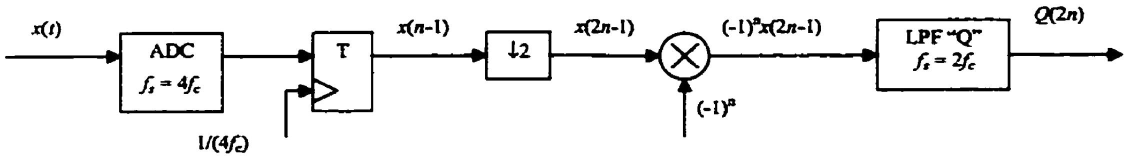

Figure 2-7 - Quadrature Channel for Decimation by Two

Combining Figure 2-6 and Figure 2-7 gives an overall digital quadrature demodulator block diagram already presented in [6]. It is shown in Figure 2-8.

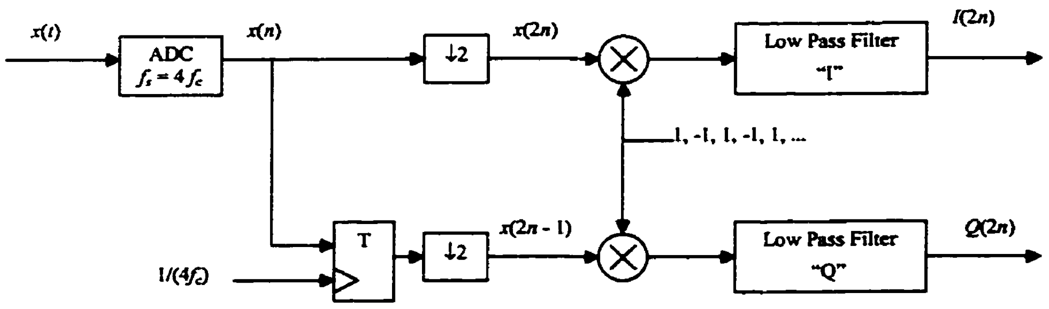

Figure 2-8 - Digital Quadrature Demodulator with $f_s = 4 \times f_c$ and Decimation by Two

From this block diagram, it is important to note that the input signal is still sampled at four times its carrier frequency. However, all samples are not passed to both digital filters. The "even" samples are passed to the in-phase filter, with every other one undergoing a sign change. Similarly, the "odd" samples are passed to the quadrature filter, and every other one also undergoes a sign change.

The advantage of this quadrature demodulator configuration is that it accommodates the widest possible input signal bandwidth ($B < 2 \times f_c$) while keeping the processing rate at half of the input data rate in the two low-pass filters.

2.3.2 High-Pass Filter Approach

This approach, proposed in [5], is applicable where the input signal bandwidth B is less than f_c, and when it is appropriate to decimate the filter outputs by a factor of 4. A slightly modified version of the proposed block diagram is shown in Figure 2-9. The approach has the advantage that the filters can operate at a quarter of the input data rate, since the last decimation step should be done inside the filters. It is also interesting to note that the multiplications by +1 and -1 are imbedded in the filter coefficients.

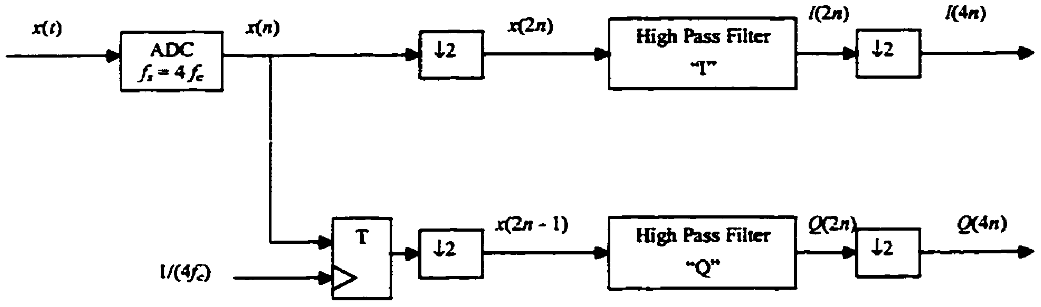

Figure 2-9 - Digital Quadrature Demodulator with High-Pass Filter Approach

It can be shown that the impulse response of the in-phase high-pass filter $h_{HPI}(n)$ is given by:

$$h_{HPI}(n) = (-1)^n h_{LP}(2n) \qquad \textbf{(2-18)}$$

for $n = 0, 1, 2, \ldots, K_I$, where K_I is equal to $(N-1)/2$ for N odd, and to $N/2 - 1$ for N even.

Similarly, the impulse response of the quadrature high-pass filter $h_{HPQ}(n)$ can be shown to be equal to:

$$h_{HPQ}(n) = (-1)^n h_{LP}(2n+1) \qquad \textit{(2-19)}$$

for $n = 0, 1, 2, \ldots, K_Q$, where K_Q is equal to $(N\text{-}3)/2$ for N odd, and to $(N/2) - 1$ for N even.

2.3.3 Polyphase Filter Approach

It has been suggested in [6] that a polyphase filter architecture ([7] [8] [4]) be used for quadrature demodulation. This approach has the advantage of keeping the filter processing rate at its lowest for all cases. The derivation included here will assume a decimation factor of four. For this case, the approach is equivalent to the High Pass Filter approach. However, it leads to a system description that is more readily translatable to a hardware realization.

Starting from equation (2-10), decimating the output of the in-phase channel by 4 gives:

$$\begin{aligned} I(4n) &= \sum_{m=0}^{N-1} h_{LP}(m) \cdot x(4n-m) \cdot \cos(\frac{\pi}{2}(4n-m)) \\ &= \sum_{m=0}^{N-1} h_{LP}(m) \cdot x(4n-m) \cdot \cos(\frac{m\pi}{2}) \end{aligned} \qquad \textit{(2-20)}$$

This summation can be expanded to find underlying symmetry:

$$\begin{aligned} I(4n) = & h_{LP}(0)x(4n) - h_{LP}(2)x(4n-2) \\ & + h_{LP}(4)x(4n-4) - h_{LP}(6)x(4n-6) \\ & + h_{LP}(8)x(4n-8) - h_{LP}(10)x(4n-10) \\ & + h_{LP}(12)x(4n-12) - h_{LP}(14)x(4n-14) \\ & + \ldots \end{aligned} \qquad \textit{(2-21)}$$

and it can then be re-written as the difference between two distinct summations:

$$\begin{aligned} I(4n) &= \sum_{r=0}^{R_{I0}} h_{LP}(4r) \cdot x(4(n-r)) - \sum_{r=0}^{R_{I1}} h_{LP}(4r+2) \cdot x(4(n-r)-2) \\ &= h_{LPI0} * x(4n) - h_{LPI1} * x(4n-2) \end{aligned} \qquad \textit{(2-22)}$$

which is the difference between the outputs of two new low-pass polyphase filters of impulse responses $h_{LPI0}(n)$ and $h_{LPI1}(n)$ given by:

$$h_{LPI0}(n) = h_{LP}(4n) \qquad \textit{(2-23)}$$

for $n = 0, 1, 2, \ldots, R_{I0}$, and:

$$h_{LPI1}(n) = h_{LP}(4n+2) \qquad \textit{(2-24)}$$

for $n = 0, 1, 2, \ldots, R_{I1}$. The values of the constants R_{I0} and R_{I1} are given in Table 2-1.

For the quadrature channel, starting from equation (2-11) and decimating by 4 gives:

$$\begin{aligned} Q(4n) &= \sum_{m=0}^{N-1} h_{LP}(m) \cdot x(4n-m) \cdot \sin(-\frac{\pi}{2}(4n-m)) \\ &= \sum_{m=0}^{N-1} h_{LP}(m) \cdot x(4n-m) \cdot \sin(\frac{m\pi}{2}) \end{aligned} \qquad \textit{(2-25)}$$

This summation can be expanded to find underlying symmetry:

$$\begin{aligned} Q(4n) = h_{LP}(1)x(4n-1) &- h_{LP}(3)x(4n-3) \\ + h_{LP}(5)x(4n-5) &- h_{LP}(7)x(4n-7) \\ + h_{LP}(9)x(4n-9) &- h_{LP}(11)x(4n-11) \\ + h_{LP}(13)x(4n-13) &- h_{LP}(15)x(4n-15) \\ + \ldots & \end{aligned} \qquad \textit{(2-26)}$$

and it can then be re-written as the difference between two distinct summations

$$\begin{aligned} Q(4n) &= \sum_{r=0}^{R_{Q0}} h_{LP}(4r+1) \cdot x(4(n-r)-1) - \sum_{r=0}^{R_{Q1}} h_{LP}(4r+3) \cdot x(4(n-r)-3) \\ &= h_{LPQ0} * x(4n-1) - h_{LPQ1} * x(4n-3) \end{aligned} \qquad \textit{(2-27)}$$

which is the difference between the outputs of two new low-pass polyphase filters of impulse responses $h_{LPQ0}(n)$ and $h_{LPQ1}(n)$ given by:

$$h_{LPQ0}(n) = h_{LP}(4n+1) \tag{2-28}$$

for $n = 0, 1, 2, \ldots, R_{Q0}$, and:

$$h_{LPQ1}(n) = h_{LP}(4n+3) \tag{2-29}$$

for $n = 0, 1, 2, \ldots, R_{Q1}$.

Table 2-1 below gives the index upper bounds for the sub-sampling of the prototype low-pass filter into the new I and Q polyphase filters, depending on the filter order.

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
N mod 4 = 0	$N/4 - 1$	$N/4 - 1$	$N/4 - 1$	$N/4 - 1$
N mod 4 = 1	$(N-1)/4$	$(N-1)/4 - 1$	$(N-1)/4 - 1$	$(N-1)/4 - 1$
N mod 4 = 2	$(N-2)/4$	$(N-2)/4$	$(N-2)/4 - 1$	$(N-2)/4 - 1$
N mod 4 = 3	$(N-3)/4$	$(N-3)/4$	$(N-3)/4$	$(N-3)/4 - 1$

Table 2-1 - Upper Bounds for Prototype Filter Subsampling into Polyphase Filters ($M = 4$)

The resulting quadrature demodulator block diagram is shown in Figure 2-10. It can be seen that all processing is done at a quarter of the input sampling rate. Each branch and sub-branch processes data that is de-interleaved from the output of the ADC, with two successive steps of decimation by 2. An interesting consequence of the polyphase filter approach, when compared to the low-pass filter approach with decimation by two, is that the multiplication step for frequency conversion has been eliminated. For the present case with a decimation by four, this approach to quadrature demodulation is functionally equivalent to the High-Pass approach suggested in [5].

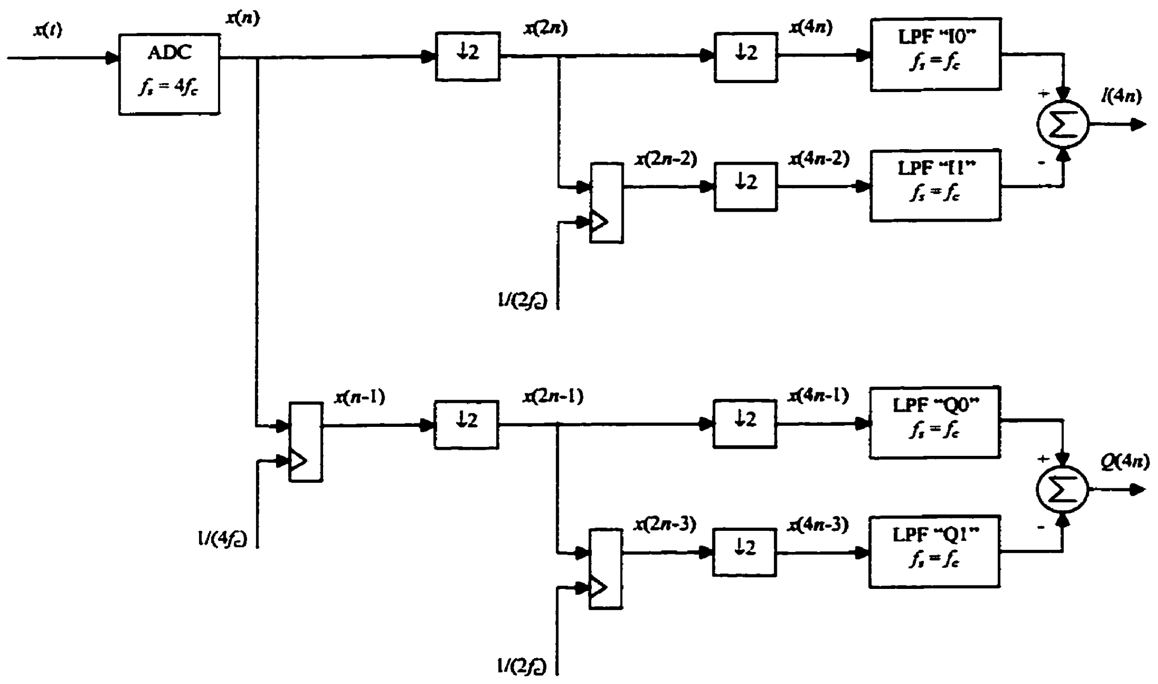

Figure 2-10 - Quadrature Demodulator Polyphase Filter Approach ($M = 4$)

2.3.4 Duplicated Polyphase Filters Approach

In the previous section, an appealing use of polyphase filters was made to keep the data processing rate as low as possible when decimating by four. Here, a method proposed in [9] will be described. It has the advantage of doubling the maximum theoretical input signal bandwidth while maintaining the processing rate at a quarter of the sampling rate. This case would therefore be equivalent to the Low-Pass Filter approach in terms of input signal bandwidth, but to the Polyphase Filter approach in terms of processing rate.

Equations (2-22) and (2-27) give expressions for $I(4n)$ and $Q(4n)$, the decimated-by-four filter outputs. If an overall decimation by two is desired instead, then the number of calculated output samples must be doubled. Since $I(4n)$ and $Q(4n)$ are available, then

I(4*n* - 2) and *Q*(4*n* - 2) must be calculated. Following the same approach as before, the in-phase component is calculated to be:

$$\begin{aligned} I(4n-2) &= \sum_{m=0}^{N-1} h_{LP}(m)\cdot x(4n-2-m)\cos(\frac{\pi}{2}(4n-2-m)) \\ &= -\sum_{m=0}^{N-1} h_{LP}(m)\cdot x(4n-2-m)\cos(\frac{m\pi}{2}) \\ &= -\sum_{r=0}^{R_{I0}} h_{LP}(4r)\cdot x(4(n-r)-2) \\ &+ \sum_{r=0}^{R_{I1}} h_{LP}(4r+2)\cdot x(4(n-r)-4) \\ &= -h_{LPI0} * x(4n-2) + h_{LPI1} * x(4n-4) \end{aligned} \qquad \textit{(2-30)}$$

Similarly, the quadrature component is expressed as:

$$\begin{aligned} Q(4n-2) &= \sum_{m=0}^{N-1} h_{LP}(m)\cdot x(4n-2-m)\sin(\frac{\pi}{2}(4n-2-m)) \\ &= -\sum_{m=0}^{N-1} h_{LP}(m)\cdot x(4n-2-m)\sin(\frac{m\pi}{2}) \\ &= -\sum_{r=0}^{R_{Q0}} h_{LP}(4r+1)\cdot x(4(n-r)-3) \\ &+ \sum_{r=0}^{R_{Q1}} h_{LP}(4r+3)\cdot x(4(n-r)-5) \\ &= -h_{LPQ0} * x(4n-3) + h_{LPQ1} * x(4n-5) \end{aligned} \qquad \textit{(2-31)}$$

This means that the approach of Figure 2-10 can be expanded whereby hardware area is traded for speed of operation. The four polyphase filters are duplicated. The input to the second set of filters is a delayed version of the input to the first set by exactly two clock periods, and the signs of the adder branches are reversed. A resulting block diagram, derived from [9], is given in Figure 2-11. In such a configuration, the processing rate in the filters can be halved or the achievable bandwidth doubled for a given technology, and the hardware implementation problem now becomes one of data de-interleaving and re-interleaving.

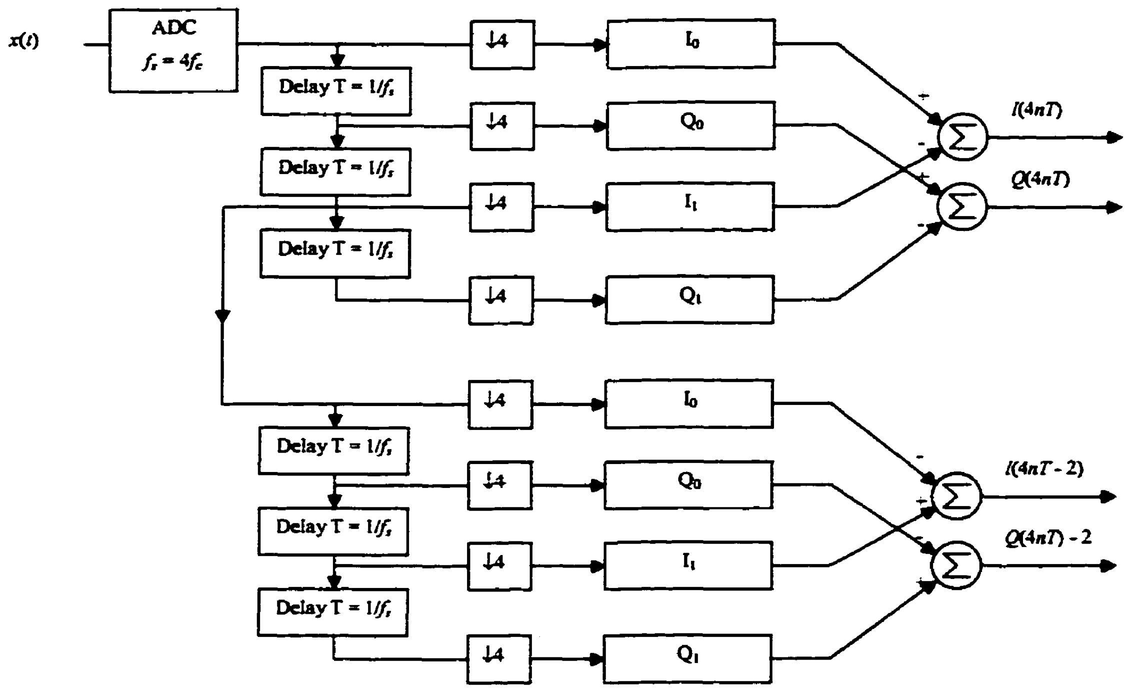

Figure 2-11 - Duplicated Polyphase Filter Architecture

Alternatively, trading area for speed can be advantageous from a power consumption perspective. Lowering the processing rate may allow a reduction in the supply voltage. Since power is directly proportional to clock frequency, but to the *square* of the supply voltage, halving the frequency while doubling chip area can still lead to a power reduction. For an FPGA implementation, however, this would not be the case since the supply voltage cannot be reduced.

2.3.5 Other Cases

In the previous sections, only two specific cases were considered for input signal bandwidth and corresponding permissible decimation factor. These two cases, where M is equal to 2 and 4, are the most common. However, it would certainly be possible to extend the discussion to cover cases where M is another power of 2, or an odd value.

2.4 Odd-Length, N^{th}-Band Prototype Low-Pass Filter

2.4.1 Half-Band Filter

A low-pass filter can be designed such that its frequency response is symmetric around the digital frequency $\pi/2$, and that the normalized magnitude response gain at this frequency is 0.5:

$$\begin{aligned} &H(e^{j\omega}) + H(e^{j(\pi-\omega)}) = 1 \\ &H(e^{j\pi/2}) = 0.5 \end{aligned} \qquad \textit{(2-32)}$$

If a FIR filter is designed with these specification and with equal pass and attenuation band ripples, the filter is called a *half-band filter* [7]. An odd-length, half-band filter has the additional properties that nearly half its coefficients are zero, and its center coefficient is equal to one half:

$$h_{1/2}(n) = \begin{Bmatrix} 1/2, & n = 0 \\ 0, & n = \pm 2, \pm 4, \ldots \end{Bmatrix} \qquad \textit{(2-33)}$$

These properties make the odd-length half-band filter economical to implement; there is a reduction of almost 50% in computational cost. The symmetry of the impulse response about the origin allows a further halving of the number of necessary multiplications.

In the case of the low-pass approach of section 2.3.1, an interesting consequence arises if the low-pass prototype filter is designed to be an odd length, half-band filter. Using equations (2-14) and (2-17) to obtain the impulse response of the in-phase and quadrature filters, it is found that the in-phase filter is an odd-length filter with only one non-zero coefficient, and the quadrature filter is an even-length filter:

$$\begin{aligned} h_{LPI}(n) &= [\ldots\ 0\ \ 0\ \ 0\ \ h_{LP}(0)\ \ 0\ \ 0\ \ 0\ \ \ldots] \\ h_{LPQ}(n) &= [\ldots h_{LP}(-5)\ h_{LP}(-3)\ h_{LP}(-1)\ h_{LP}(1)\ h_{LP}(3)\ h_{LP}(5) \ldots] \end{aligned} \qquad \textit{(2-34)}$$

Implementing the in-phase filter can then be done at very low cost. Its all pass characteristic, however, is a disadvantage in the presence of DC offsets from the ADC.

2.4.2 Third-Band Filter

An odd-length, *third band* FIR filter has similar properties to the half-band filter. Its pass band and stop band ripples are again the same, but the cut-off frequency is π/3. Nearly a third of the filter coefficients are equal to zero:

$$h_{1/3}(n) = \begin{Bmatrix} 1/3, & n = 0 \\ 0, & n = \pm 3, \pm 6, \ldots \end{Bmatrix} \qquad \textit{(2-35)}$$

There is an interesting consequence from designing the prototype filter to be an odd-length, third-band filter. If the low-pass approach of section 2.3.1 is again followed, equations (2-14) and (2-17) show that the zero coefficients will be distributed evenly between the in-phase and quadrature filters:

$$\begin{aligned} h_{LPI}(n) &= [\ \ldots\ h(-8)\ \ 0\ \ h_{LP}(-4)\ \ h_{LP}(-2)\ \ h_{LP}(0)\ \ h_{LP}(2)\ \ h_{LP}(4)\ \ 0\ \ h_{LP}(8)\ \ \ldots\ ] \\ h_{LPQ}(n) &= [\ \ldots\ h_{LP}(-7)\ \ h_{LP}(-5)\ \ 0\ \ h_{LP}(-1)\ \ h_{LP}(1)\ \ 0\ \ h_{LP}(5)\ \ h_{LP}(7)\ \ \ldots\ ] \end{aligned} \qquad \textit{(2-36)}$$

Compared with the half-band prototype filter approach, this would increase the total number of non-zero coefficients, and hence chip area requirements and power consumption, but it would even out the amount of computation between the two filters. If the order *N* of the prototype filter is large, then there are approximately *N*/2 and 2*N*/3 non-zero coefficients for the half- and third-band filters, respectively. For the half-band prototype filter case, the in-phase filter has one non-zero coefficient and the quadrature filter has *N*/2. For the third-band filter, both the in-phase and quadrature filters have approximately *N*/3 non-zero coefficients. Hence, the total area and dissipated power are increased by a factor of 4/3 for the third-band filter compared to the half-band case. However, the largest sub-filter length is multiplied by a factor of 2/3.

Thus, using the third-band prototype filter would reduce the maximum filter size, at the expense of total chip area and power dissipation. Alternatively, if the two sub-filters are distributed on two different chips, it may be better from a system perspective if the overall computation effort is spread out evenly, and again the third-band filter approach may be a better choice.

From a hardware implementation point of view, there is an important disadvantage to following the Low-Pass Filter approach of section 2.3.1 when the prototype filter is a third-band filter. This choice implies that the output of the quadrature demodulator would be oversampled by a factor of 3/2, and thus that computing resources inside the quadrature demodulator would be wasted. Communications with the next device in the system would also have to take place at a data rate potentially greater than is necessary.

However, there are important advantages to the third-band case. Compared to the half-band case, the filters provide a bandpass filtering function. In quadrature demodulator system design, this function is often useful to remove unwanted signals such as DC offsets from the ADC, and to compensate for non-ideal performance of the IF amplifier [10]. The requirements for a separate digital bandpass filter can be relaxed or eliminated. The allowable input signal bandwidth and number of zero coefficients is also greater than for the quarter-band case, discussed in the following section.

2.4.3 Quarter-Band Filter

Following the half- and third-band cases, designing the prototype low-pass filter as an odd-length, fourth-band filter would imply that almost one out of four coefficients would be zero. This approach would be alternative for a case where the input signal bandwidth is such that decimation by four is appropriate.

If the polyphase approach of section 2.3.3 is followed, it can be shown that all zero coefficients would be mapped to only one of the four polyphase sub-filters. This filter would only inherit of the center coefficient of the prototype filter, and would therefore be reduced to a weighed delay line. All other polyphase sub-filters would become even-length filters.

Therefore, the quarter-band prototype filter presents interesting advantages. As for the third band case, a quarter band prototype filter provides useful stop bands when compared to the half-band case. The number of zero coefficients is reduced when compared to the third-band case. Additionally, the computational efficiency is greater, since a reduction in processing rate is possible because the reduced pass band allows an additional decimation by two of the output data.

2.4.4 General Case

In general, the cutoff frequency of an odd-length N^{th}-band low-pass filter is equal to π/N, and its impulse response is symmetric about the origin and is given [11] by

$$h_{1/N}(n) = \left\{ \begin{array}{ll} \frac{1}{N} & ,n=0 \\ \frac{1}{N} \times \frac{\sin(n\pi/N)}{(n\pi/N)} & ,\text{otherwise} \end{array} \right\} \qquad (2\text{-}37)$$

By inspection, the term $sin(n\pi/N)$ will be null for all values of n that are integer multiplies of N, and therefore an odd-length N^{th}-band filter will have almost 1 out of N zero coefficients (the center coefficient is never null). Equation (2-37) can be used to design an N^{th}-band filter, in conjunction with an appropriate window function which is selected to trade-off transition bandwidth for reduced pass and stop band ripples.

2.5 Frequency Translation by Undersampling

The discussion of quadrature demodulation so far has assumed a sampling frequency equal to four times the carrier frequency. The obvious difficulty with this choice is the high sampling rate that is thus required. For example, if the target center frequency is f_c = 160 MHz, this would imply a sampling frequency f_s = 640 MHz. The system architectures described in the previous chapters have shown that if the signal bandwidth is small enough, then the processing rate in the filters can be reduced to their minimum. However, effort must be expended to de-interleave the ADC output data into multiple streams, not to mention the potentially high cost and complexity of the ADC itself since it must work at a high sampling rate. An alternative approach, suggested in [6], is that undersampling be used to translate a signal from a high frequency to a lower one, when allowable by the signal bandwidth.

If the sampling frequency f_s is chosen such that

$$f_s = \frac{4f_c}{2m+1} \qquad \textbf{\textit{(2-38)}}$$

where m is a positive integer, then the center frequency of the signal, f_c, will be aliased to $f_s/4$. The advantages of satisfying $f_s = 4 \times f_c$ are therefore effectively maintained, without the costs of a correspondingly high sample rate. All system architectures presented so far are therefore suitable to input signal undersampling.

Table 2-2 lists possible sampling frequencies for a carrier frequency of 160 MHz, for the Low-Pass Filter approach. The corresponding maximum filter processing rates and resulting signal bandwidth after demodulation are also given. For example, with f_c = 160 MHz and m = 1, we have f_s = 4/3×160 MHz = 213.3 MHz, which is one third of the 640 MHz calculated previously. Figure 2-12 illustrates this example.

f_c (MHz)	[illegible]	[illegible]	[illegible]	[illegible]
160	0	640.0	320.0	160.0
160	1	213.3	106.7	53.3
160	2	128.0	64.0	32.0
160	3	91.4	45.7	22.9
160	4	71.1	35.6	17.8

Table 2-2 - Alternative Undersampling Frequencies, Low-Pass Filter Approach

A necessary condition for this scheme to work, obviously, is that the Nyquist criterion be respected such that f_s remain at least twice the input signal's bandwidth. In our present case, a sampling frequency of 213.3 MHz means that the quadrature demodulator would work with signals on a center frequency of 160 MHz with a maximum possible input bandwidth of 106.7 MHz. After decimation by two, the processing rate in each digital filter would also be 106.7 MHz, which lends itself well to implementation in some of the faster FPGAs available.

The results of Table 2-2 can also be used to specify a sampling frequency given an expected signal bandwidth. As a general rule, the lowest acceptable processing rate should always be chosen in order to minimize power dissipation.

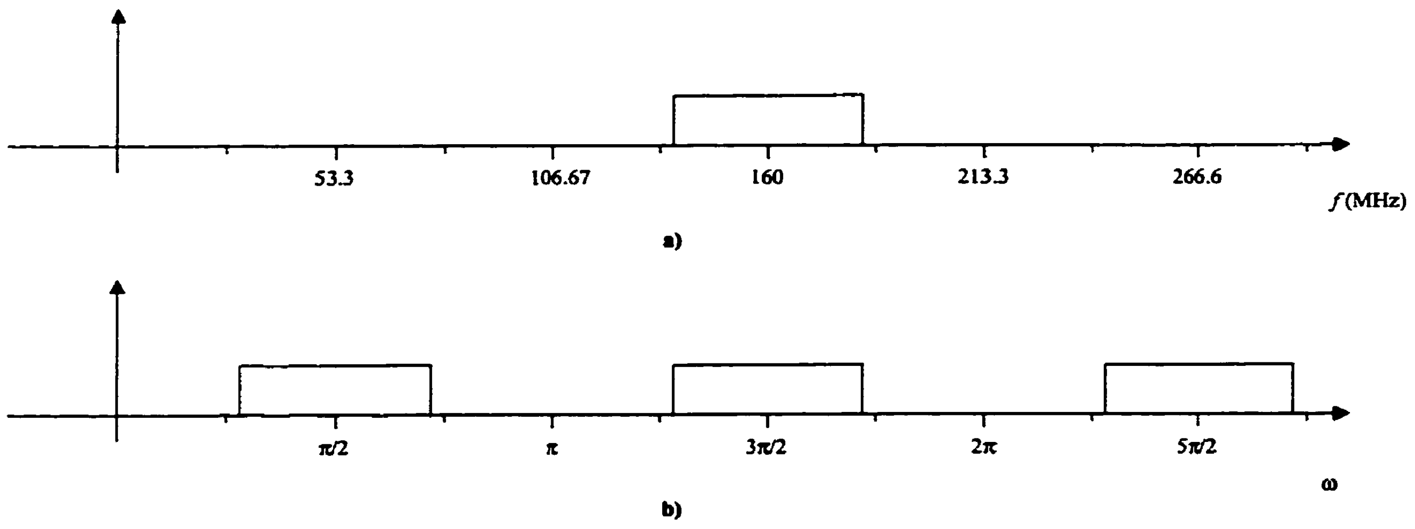

Figure 2-12 - Example of Frequency Translation by Undersampling
a) positive spectrum of an analog bandpass signal centered on 160 MHz. b) spectrum of the same signal, sampled at a frequency of 213.3 MHz, with an effective frequency shift to a quarter of the sampling rate.

2.6 Decision

In this chapter, fundamental principles for the digital realization of quadrature demodulators were presented. This included a review of basic and improved digital approaches, a discussion on prototype filter design to minimize hardware impact, and the description of a strategy to minimize signal sampling rate by undersampling.

As discussed in the previous sections, the selection of an implementation approach depends on a number of factors, but principally on the passband width of the prototype filter and on the desired output decimation factor. The chosen set of filter coefficients for the designs described in this thesis is discussed in Chapter 5. The prototype filter is a quarter-band filter, and a decimation by four is desired. Therefore, either of the High-Pass or Polyphase Filter approaches would be suitable since they are equivalent. The Polyphase Filter approach is selected since it leads to a system description that is more readily translatable to a hardware realization. It can also serve as a basis for the Duplicated Polyphase Filters approach.

Undersampling will be used to reduce the signal sampling rate. The 160 MHz IF signal will be sampled at 213.3 MHz. After decimation by four, the filter processing rate will be 53.3 MHz and the maximum theoretical input signal bandwidth will also be 53.3 MHz.

Chapter 3

Multiplier Implementation for Digital Filtering

3.1 Introduction

The equation for the output $y(n)$ of a FIR filter of order N, with impulse response $h(n)$ and input data sequence $x(n)$ was given (equation (2-9)) as:

$$y(n) = x(n) * h(n) = h(n) * x(n) = \sum_{m=0}^{N-1} h(m)x(n-m)$$

For a filter of order N, there are N multiplications performed for every filter output data sample. Zero-valued coefficients reduce this number, and so does the exploitation of the symmetry of the impulse response. However, for large N, many multiplication operations still have to be performed.

A n-bit, two operand unsigned multiplier generally requires a silicon area proportional to n^2. Since multiplication is a computation intensive process, a major portion of the quadrature demodulator design effort should be expended on selecting the best possible approach to implement the multipliers. In this chapter, different approaches will be studied with the goal of reducing the number of adders and shifters required to implement multiplication.

When discussing multipliers, it is customary to use the terms ***multiplicand*** and ***multiplier*** to identify the two numbers being multiplied together, with ***multiplier*** meaning the multiplication factor applied to the ***multiplicand.*** However, in this discussion, the following terminology will be adopted. ***Multiplier*** will refer to the device, architecture or circuit performing the multiplication operation. Since the action of the ***multipliers*** will be to multiply input data by fixed filter coefficients, the term ***coefficient*** will be used to represent the multiplication factor. The term ***multiplicand*** will be used with its usual meaning.

3.2 Power-of-Two Coefficients

Since multiplication can be thought of as a shift-and-add process, a multiplier can be built from a two-dimensional array of shifted adders. As an example, consider the multiplication of the two 4-bit unsigned numbers 1001_2 and 1101_2. It is accomplished as follows:

$$\begin{array}{r} 1001 \\ \underline{\times 1101} \\ 1001 \\ 00000 \\ 100100 \\ \underline{+1001000} \\ 1110101 \end{array}$$

From this example, it is seen that the multiplication requires that 4 rows be added because the coefficient, 1101_2, has four bits. Each row is either a shifted version of the multiplicand, 1001_2, or made up of all zeroes. The choice is made based on the value of the corresponding bit in the coefficient. In fact, each bit of the adder rows is the result of the AND operation between a coefficient bit and a multiplicand bit. In the present example, three two-operand adders would be necessary to perform the multiplication. Many texts describe such general-purpose multipliers in detail [12].

The multipliers of the quadrature demodulator filters do not need to be array multipliers, because the filter coefficients are fixed for a given design. Each multiplier is therefore a so-called *constant coefficient multiplier*, or *KCM*. In the previous example, if the coefficient 1101_2 is fixed, there is no need to consider the shifted row of zeros and the multiplication process can be reduced as follows.

$$\begin{array}{r} 1001 \\ \underline{\times 1101} \\ 1001 \\ 100100 \\ \underline{+1001000} \\ 1110101 \end{array}$$

In this case, only three rows are added together, which means that only two adders are required. The three rows correspond to the three non-zero bits of the multiplicand. In general, if the coefficient has d non-zero digits, then d-1 two-operand adders are required in the multiplier.

A logical extension of this last example is that a multiplier with a coefficient that is represented by a single power of two is "free" since the coefficient has only one non-zero bit. The multiplication result is a simple shifted version of the multiplicand, and no addition is required. Therefore, from an implementation point of view, ideal filter coefficients would be selected such that they are equal to a power of two. The multiplications in the quadrature demodulator would then require no additions. Alternatively, coefficients should be equal to the sum of a few (say 2 or 3) powers of two. For every additional non-zero digit in a coefficient, one extra adder will be required in its corresponding multiplier.

3.3 Signed Digit Representation of Coefficients

In the previous section, the point was made that a reduction in the number of non-zero digits used to represent the filter coefficients would lead to smaller multipliers. Representing the coefficients with signed digits can reduce the number of non-zero digits necessary to represent a given number. In such a case, each digit is allowed to take a value from the set {-1, 0, 1}, which means that numbers can now be represented by a sum and/or a *difference* of powers of two. As an example, the number 127_{10} is represented by 01111111_2 in the standard 8-bit binary format. Multiplication by a filter coefficient equal to 127_{10} would therefore require 6 two-operand adders. Alternatively, allowing signed digits means that 127_{10} can be represented by $1000000\underline{1}_2$ (where $\underline{1}$ means -1), or 128_{10} - 1. A multiplier using this representation for the coefficient 127_{10} would thus require only one two-operand subtracter.

Intuitively, there may be many possible valid signed digit representations for a given coefficient, and some of them may require even more non-zero digits than the standard binary approach[1]. It may be shown, however, that the signed digit approach allows the representation of any number with the least amount of non-zero binary digits. Further, if it is chosen such that no two non-zero digits are adjacent, the representation is called the *Canonical Signed Digit* (CSD) representation for that number. The proof of minimal representation and a corresponding algorithm to calculate the CSD representation of numbers can be found in [13].

Another advantage of using CSD is that a broader range of numbers can be represented with a given number of bits, as compared to sign & magnitude or two's complement approaches. With 8 bits, any number between +255 and -255 can be represented with

[1] As an example, $5_{10} = 4 + 1 = 00000101_2 = 128_{10} - 123_{10} = 1\underline{1}\underline{1}\underline{1}\underline{1}0\underline{1}\underline{1}$.

CSD. With sign & magnitude, the range is between +127 and -127. For two's complement, it is between +127 and -128. Table 3-1 illustrates the range of values for different number representations, given a fixed number of bits *n*.

Format [illegible]	[illegible]	[illegible]
Sign & Magnitude	$-(2^{n-1}-1)$	$2^{n-1}-1$
One's Complement	$-(2^{n-1}-1)$	$2^{n-1}-1$
Two's Complement	-2^{n-1}	$2^{n-1}-1$
CSD	$-(2^{n}-1)$	$2^{n}-1$

Table 3-1 - Range of Values for Different Number Representations with* n *bits

Since coefficients encoded in CSD require the fewest nonzero digits, they lead to multipliers with a reduced number of necessary shift and add operations. However, the inclusion of negative coefficient digits adds complexity, because shifted versions of the multiplicand may now need to be subtracted as well as added. The binary number format chosen to represent the multiplicand is therefore critical. The one leading to the simplest multiplier implementation is 2's complement. When a negative signed digit is encountered in the coefficient, the bits of the multiplicand must be inverted and a carry must be added, then the result must be shifted as before by an appropriate number of bits. Sign extension of the number is also necessary, which means that the Most Significant Bit is repeated as necessary to fill the row to the left such that each row has the same number of digits. Once all these operations are done, simple addition will produce the correct result.

The following example illustrates multiplication by a constant CSD coefficient. The multiplicand is a two's complement number equal to -113_{10} (10001111_2) and the multiplier coefficient is $+159_{10}$ ($1010000\underline{1}$ in CSD). The expected result is -17967_{10}.

```
        10001111
       ×10100001
----------------
0000000001110001
1111000111100000
+1100011110000000
----------------
1011100111010001
```

Four important aspects of multiplication implementation arise from this example. First, when one of the coefficient's digits is negative, the two's complement of the multiplicand must be taken. In the example, the result of the sign inversion is shown directly in the first adder row. In a digital implementation, however, each digit would be first inverted, then a carry would be added. Second, sign extension is required when adding two's complement numbers together. The Most Significant Bit must be repeated left as many times as necessary to make the length of the number equal to the widest possible expected sum. Third, using two's complement representation for the multiplicand means that regular addition can be done on the shifted rows. Fourth, as expected, only two additions would be required since the multiplier has three non-zero digits.

3.4 Minimum Number of Adders for Multiplier Implementation

In the approach to multiplier implementation considered so far, the effort expended depends on the number of non-zero digits required to represent the multiplier coefficient. If there are d non-zero digits, then d-1 adders are necessary. Since the CSD representation requires the least number of digits to represent a given number, one would think that CSD encoding of multipliers should yield the lowest cost multipliers (where cost is defined as the number of two-operand adders in the multiplier). However, a method proposed in [14] and [15] achieves an average improvement in the number of two-operand adders of 26.6% and 16% for 32 and 12-bit word multipliers, respectively, over a CSD approach. The following example illustrates the principle of the method.

Consider multiplication by the coefficient 45_{10}. The normal binary representation of this coefficient, using 8 bits, is 00101101_2 for $45 = 2^5 + 2^3 + 2^2 + 2^0$. The CSD representation is $010\underline{1}0\underline{1}01$ (where "$\underline{1}$" represents a -1), for $45 = 2^6 - 2^4 - 2^2 + 2^0$. Thus, in both cases there are 4 non-zero digits and consequently 3 adders should be required[1]. However, 45_{10} can also be expressed as $45 = 9 \times 5 = (2^3 + 1)(2^2 + 1)$. Thus, the multiplicand can first be multiplied by 9, which requires only one addition. The intermediate result is then multiplied by 5, which also requires only one addition. Therefore, only 2 adders are required for the complete operation. Figure 3-1 illustrates this example.

Algorithms for decomposing multiplier coefficients such that the multiplication process is minimum are given in [14], [15], [16] and [17].

Obviously, a reduction in the required number of adders between this method and the standard one with CSD representation will depend on the value of the filter coefficients. Coefficients equal to a power of two, and those represented by a sum or a difference of two power of two numbers cannot be reduced any further. Intuitively, the greatest reduction should be attained for coefficients which can be decomposed into many factors.

In Chapter 4, a filter architecture that exploits redundancy in the factors that repeat in a set of coefficients will be presented. For example, the coefficients 45 and 18 each share the factor 9, which is implemented with only one adder. If the partial product corresponding to the factor 9 could be reused somehow, a further reduction in the total number of adders necessary to implement a digital filter could be gained.

[1] For this specific example, there is no advantage gained from using a CSD representation for the multiplier coefficient instead of the standard binary format.

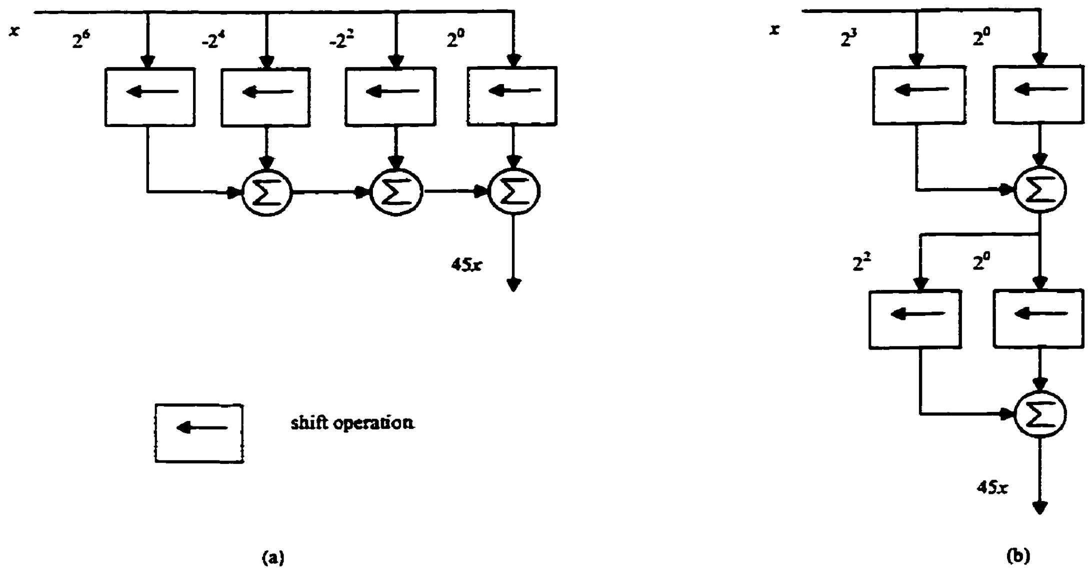

Figure 3-1 - Multiplication by 45
(a) Standard approach, with CSD. (b) Minimum number of adders approach.

3.5 Look-Up Table Approach for FPGA Multiplication

FPGA Configurable Logic Blocks (CLB) can be programmed to behave as Read-Only Memory (ROM). This feature offers an interesting alternative technique to multiplier implementation with constant coefficients.

For the Look-Up Table (LUT) approach, the order of the multiplication process is reversed such that the coefficient is multiplied by the multiplicand. For example, say that the 12-bit multiplicand $4A9_{Hex}$ must be multiplied by the 8-bit coefficient $B5_{Hex}$. Instead, we multiply $B5_{Hex}$ by $4A9_{Hex}$. The process is as follows:

$$\begin{array}{r} B5 \\ \underline{\times 49A} \\ A \times B5 \\ 16_{10} \times 9 \times B5 \\ + \underline{256_{10} \times 4 \times B5} \\ \ldots \end{array}$$

It is obvious from this example that a stored table with the values of the 16 hexadecimal digits multiplied by the constant $B5_{Hex}$ would be very useful. If such a table existed, then the multiplication process could be reduced to two additions of pre-calculated values with the appropriate shifts.

Each CLB in the Xilinx 4000 series can be programmed as a 16 × 2-bit memory, by using the F and G function generators as 16 × 1-bit memories. This is the basic building block for the LUT multiplier. In general, for a coefficient expressed with c bits, then $c + 4$ bits are required to express all possible results from the multiplications of that coefficient with a 4-bit number. Since each CLB can store 2 bits, then $(c + 4)/2$ CLBs are required for each LUT.

The multiplicand is decomposed into slices of 4 bits. These four bits select one of the 16 possible pre-calculated products of a LUT. If the multiplicand is expressed with m bits, then $\lceil m/4 \rceil$ LUTs will be required, where the brackets signify rounding up to the nearest integer.

Therefore, in the previous example ($4A9_{Hex} \times B5_{Hex}$), three LUTs (because the multiplicand is expressed with three slices of four bits) each composed of 6 CLBs (storing 12 bits = 4 + 8 for the coefficient) would be required. The three LUTs would be identical, and would store the 16 possible products of x times B5, for $x = 0, 1, 2, \ldots$ E, F. Two adders would complete the design of the LUT multiplier.

Figure 3-2 below gives a block diagram for a LUT multiplier where both the multiplicand M and the coefficient C are expressed with 8 bits, with a product P of 16 bits [18].

3.5.1 Advantages of LUT Approach to Multiplication in FPGAs

The LUT approach to multiplication by a constant coefficient presents many advantages in FPGAs.

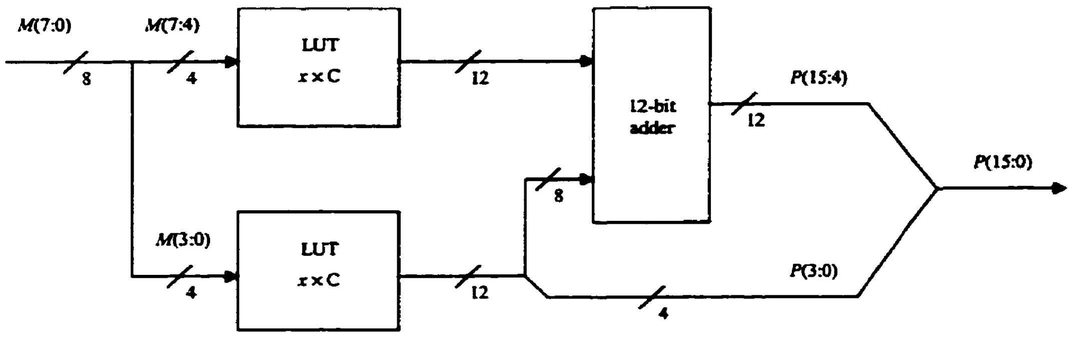

Figure 3-2 - LUT Multiplication Block Diagram

When compared to a general-purpose multiplier, there is obviously a great area utilization advantage to the LUT approach, as there was for the CSD method for constant coefficients. The LUT approach is very compact, and greatly reduces the number of arithmetic operations that must be performed to calculate a product.

When creating a system with many multipliers, or when designing many systems that will utilize multipliers, the LUT approach can greatly simplify the design process. For a given set of multiplicand and coefficient size, the multiplier only needs to be designed once. The placement of CLBs and routing of signals internal to the multiplier can be carefully optimized for speed, area and/or power consumption, then the multiplier can be considered as a building block. Modifying the value of the coefficient doesn't involve any structural changes, only the stored values in the CLBs need to be replaced. The multiplier building block can then be reused as necessary. While this would be true for a general-purpose multiplier building block, the advantage of the LUT approach is again the great reduction in necessary resources. Alternatively, CSD encoding of the coefficient requires the designer to optimize the multiplier for every coefficient.

For FIR filter designs, the LUT approach has a few interesting advantages over the CSD method discussed previously. First, the number of non-zero digits used to represent a coefficient is irrelevant, and this simplifies the filter design greatly since no time need be spent on optimizing the quantization of floating point coefficients[1]. Second, since all multipliers in the filter are structurally identical (same placement and routing), the LUT approach favors filter architectures that exploit the repetition of a regular structure. This will be discussed further in section 4.5.3. This approach not only increases design density on the chip, it also makes the design process much more simple. Finally, for a given filter order, changing the filter coefficients simply requires that the LUTs be reprogrammed: there is no need for mapping, placing and routing the design again. In fact, an optimized version of the filter can itself be considered a building block from that point on.

3.5.2 Comparison of the Area Used by LUT and CSD Approaches

The LUT approach to multiplier implementation has one significant disadvantage. For trivial coefficients, such as zero, one, and any power of two, it produces a multiplier that is grossly inefficient. The multiplier is also far from optimum for coefficients equal to the sum of a few powers of two. The analysis presented in this section will assume that the multiplicand is represented in two's complement using 8 bits, and that the coefficient is restricted to values between -128_{10} and $+127_{10}$.

In the CSD approach, the number of CLBs used to implement a multiplier depends directly on the number of non-zero digits used for the coefficient. It was mentioned already that if a CSD coefficient has d non-zero digits, then $d - 1$ two-operand additions will have to be performed by the multiplier. The number of bits that need to be added at every step of the shift-and-add process is equal to the number of bits in the multiplicand plus one, for two's complement addition. At every step, a portion of the least significant bits do not require an adder, since they would be added to zero. If the multiplier is fully pipelined, however, then these least-significant bits will need to be pipelined.

[1] However, the choice of a suitable scale factor prior to rounding may still be helpful.

Implementing the addition of two 2's-complement, 8-bit numbers requires 5 CLBs for a 9-bit result [19]. Therefore, following our assumptions, the number of CLBs in a non-pipelined multiplier is equal to $(d-1) \times 5$, where d is the number of non-zero digits in the coefficient.

If the multiplier is pipelined, the required number of CLBs varies depending on the coefficient. In the worst case, the coefficient has non-zero digits in extreme positions. The coefficients 01010101 and 10000001 are examples of this. For the coefficient 10000001, the seven least significant bits of the multiplicand need to be registered, which requires 3.5 CLBs. Table 3-2 gives worst-case quantities of CLBs for non-pipelined and fully pipelined CSD multipliers with a coefficient between -128 and +127. Column 2 of the table lists the distribution of these 256 possible coefficients according to the number of non-zero digits they have, and column 3 gives the amount as a percentage of the total. For example, zero is the only coefficient with no non-zero digits.

# of non-zero digits in coefficient	[illegible]	[illegible]	[illegible]	[illegible]
4	48	18.8%	15	25.5
3	120	46.9%	10	20.0
2	72	28.1%	5	8.5
1	15	5.9%	0	4.0
0	1	0.4%	0	0

Table 3-2 - 8-bit CSD Coefficient (-128 to +127) Multiplier Statistics

For the LUT approach, the calculation of the number of required CLBs is much more straightforward. For an 8-bit multiplicand, and a coefficient expressed with 8 bits (i.e. between –128 and +127), we have the situation of Figure 3-2. Each 12-bit LUT occupies 6 CLBs. The 12-bit, two-operand adder requires 7 CLBs. The total is therefore 19 CLBs for a non-pipelined case. If one level of pipelining is added between the LUTs and the adder, then the four least-significant bits must be registered, necessitating two more CLBs for a total of 21.

Therefore, from the strict point of view of area utilization, the LUT approach to multiplier implementation is preferable for only 19% of the coefficients between -128 and +127. As long as a CSD coefficient has less than 4 non-zero digits, the CSD approach is better.

3.5.3 Comparison of the Speed Between the LUT and CSD Approaches

In both the LUT and CSD approaches to multiplier implementation, the maximum processing rate ultimately depends on the width of the adders within the multiplier. As before, it will be assumed that both the multiplicand and the coefficient are 8-bit numbers.

For the LUT approach shown in Figure 3-2, there is one 12-bit wide adder, and typically only one level of pipelining. For the CSD approach, the addition of any two shifted replicas of the multiplicand requires a 9-bit adder. Therefore, the CSD approach is generally always faster than the LUT approach by a small margin. It would be possible to improve the speed of the LUT approach by pipelining its adder, but its implementation cost would then increase further.

3.6 Decision

The selection of a multiplier approach depends on a number of factors, and especially the level of design optimization that is required in terms of chip area used, power consumption and target processing rate. If those are unimportant, then the Look-Up Table approach should be followed because it simplifies the design process greatly and enhances design reusability. However, for the quadrature demodulators related to the present research, design optimization is of prime consideration. The chosen set of filter coefficients for the design was carefully selected to minimize the total number of non-zero digits. A more detailed description of these coefficients and the design constraints followed to obtain them will be given in Chapter 5. It was also essential to maximize processing speed.

Therefore, the CSD approach to multiplication is selected in order to minimize the total area occupied by the design and to maximize processing speed.

Chapter 4

Filter Architecture Selection

4.1 Introduction

Since the in-phase and quadrature digital filters perform virtually all of the computations in the quadrature demodulator, they are therefore the subject of most of the design effort. An appropriate filter architecture must be selected with great care, as it will have a major impact on many hardware realization performance metrics, such as speed of operation, power consumption, ease of placement and routing of mapped blocks, register requirements, overall CLB count, and ease of pipelining of the data processing paths.

4.2 Basic FIR Filter Architectures

The output $y(n)$ from a FIR filter with impulse response $h(n)$, filter length N, and input sequence $x(n)$ was given in equation (2-9) and is repeated here. It is equal to the convolution of the input sequence and the filter impulse response:

$$y(n) = x(n) * h(n) = h(n) * x(n) = \sum_{m=0}^{N-1} h(m)x(n-m)$$

4.2.1 Direct Form Realization

From equation (2-9), it can be seen that the filter output is a weighted sum of a finite number of present and past filter inputs. The direct hardware realization of this equation is presented in many texts [20] and consists of a chain of data registers, with taps between each register leading to a constant-coefficient multiplier. The multiplier coefficients are equal to the filter's discrete impulse response values. The outputs of all multipliers are added together to form the filter output. This filter realization is appropriately named the *direct* or *canonic* form, and a block diagram is given in Figure 4-1.

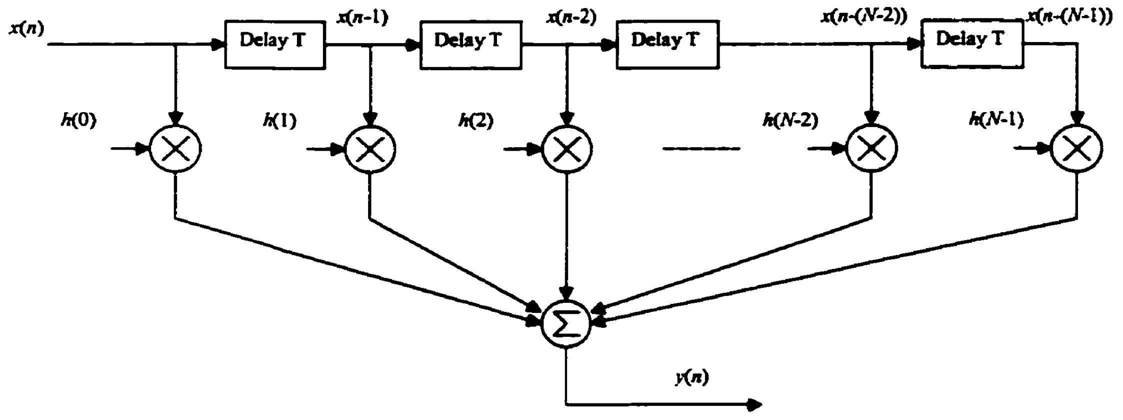

Figure 4-1 - Direct Form Realization

The multiple operand adder in this architecture will pose a problem when the processing rate is a critical factor. There are a number of ways to solve this problem, including pipelining, and these techniques will be discussed in the following sections.

The multi-operand adder can also present an interesting design alternative. Instead of adding individual products, all shifted versions of the multiplicands can be added together in a larger multi-operand adder. This approach is the one adopted in the existing version of a quadrature demodulator by CRC [21]. The resulting adder therefore grows in complexity, and is ideally implemented with pipelined blocks of Carry-Save Adders. Since the coefficients are fixed, the number of sign changes is known a priori, and the carries generated from the sign inversions can also be added a priori. Data multiplied by a negative coefficient therefore only needs to have all its bits inverted prior to addition.

4.2.2 Transposed Form Realization

Alternatively, the Transposition Theorem [3] can be used to realize equation (2-9) differently. The resulting architecture is known as the *transposed* or *inverted* form. In this case, only the present filter input is processed. Other previous inputs are not kept in memory. Instead, partial results are computed and registered for every filter input. Each partial result is a sum of a previous partial result and of the multiplication of a filter coefficient with the present input. The architecture is shown in Figure 4-2.

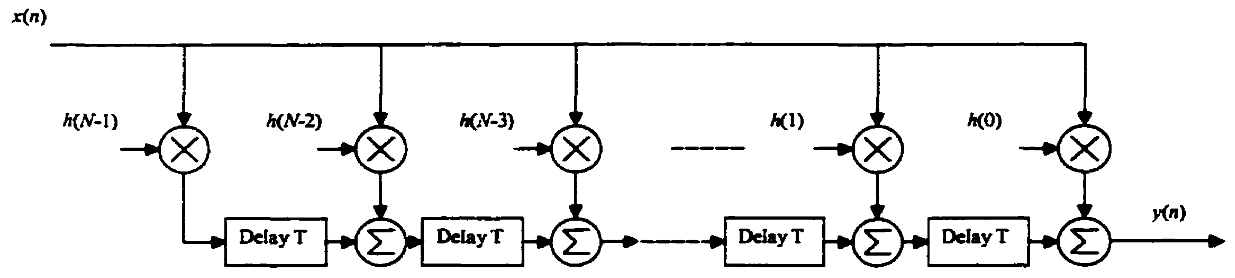

Figure 4-2 - Transposed Form Realization

A major advantage of the transposed form is the inherent pipelining built within the filter structure. Consequently, there are no multi-operand adders in the filter, except possibly inside the multipliers. This is ideal for an FPGA implementation targeting the Xilinx 4000 series, which favors the implementation of two-operand ripple carry adders.

In the same vein, the registers embedded in the adder chain do not require extra silicon area in the FPGA, since each CLB's output can be of a registered type. The adders are therefore converted into registered-output adders. This is a much more efficient utilization of FPGA resources than for the direct form realization, where the delayed input chain requires that a large number of CLBs be set aside solely for their output flip-flops. Other resources within these CLBs, such as function generators and carry chains, are not used and are not available for other purposes.

4.2.3 Cascade Form Realization

A third filter architecture is the *cascade* form [3]. It is characterized by a chain of independent filters, with the output of a given filter being fed as input to the next one in the chain. For such a realization, the overall filter transfer function, $H(Z)$, must be broken down into a product of other transfer functions of lesser order:

$$H(Z)=\prod_{k=1}^{K} H_k(Z) \tag{4-1}$$

The most simple breakdown would be one where each sub-filter transfer function is a quadratic expression, but there is no restriction to the transfer function order. In fact, there is also no restriction on the particular architecture (direct or transposed) selected for each one of the sub-filters. In that respect, the selection of the cascade form realization would imply that an independent architecture selection can be done for each of the sub-filters. A block diagram of the cascade realization is given in Figure 4-3.

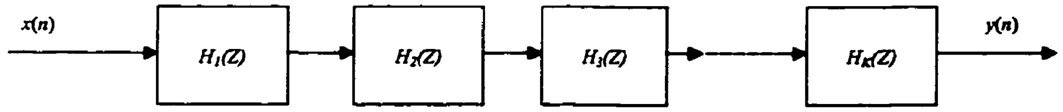

Figure 4-3 - Cascade Form Realization

Given an overall filter impulse response, the task of calculating coefficients for each of the sub-filters is non-trivial. The overall transfer function must be calculated, then a separation among the sub-filters must be done, and finally filter coefficients can be calculated. For the case where the sub-filters are low-pass, one advantage of this approach is that decimation operations can be distributed among the various stages to obtain the optimal trade-off between accuracy and computational cost.

The selection of the cascade form must be made in conjunction with the design of the desired filter transfer function and corresponding coefficient search. From a hardware point of view, each sub-filter should then be designed based on either of the direct or transposed forms, as appropriate.

4.3 Linear-Phase FIR Filter Architectures

As discussed previously, a linear phase characteristic is very desirable for wide band quadrature demodulator filters, to preserve information contained in the input signal. A linear phase characteristic also presents a potential advantage from a hardware realization point of view, as will be seen shortly.

It can be shown that FIR filters with linear phase characteristics can be obtained by constraining the filter coefficients to be symmetrical about the center coefficient. Specifically, for a filter of length *N*, if the filter impulse response satisfies

$$h(n) = h(N-1-n) \tag{4-2}$$

then the filter has linear phase [3].

This condition can be applied to equation (2-9) to take advantage of the impulse response symmetry. For a linear phase, even length filter (*N* even), the output of the filter is given by:

$$y(n) = \sum_{k=0}^{N/2-1} h(k)\left[x(n-k) + x(n-(N-1-k))\right] \tag{4-3}$$

From this equation, it is obvious that there is a reduction by a factor of 2 in the number of multiplications when compared to the non symmetrical case. This is likely to be a clear advantage for a hardware realization. A similar equation can be derived for an odd-length filter.

Two alternate filter architectures can be derived from the basic direct and transposed forms for linear phase FIR filters, each exploiting the symmetry property of the impulse response. Since two samples are multiplied by any one coefficient (except the center coefficient of an odd-length filter), it can be much more effective to add the two samples together before multiplication. The equivalent realizations, for direct and transposed forms, are shown in Figure 4-4 and Figure 4-5, respectively.

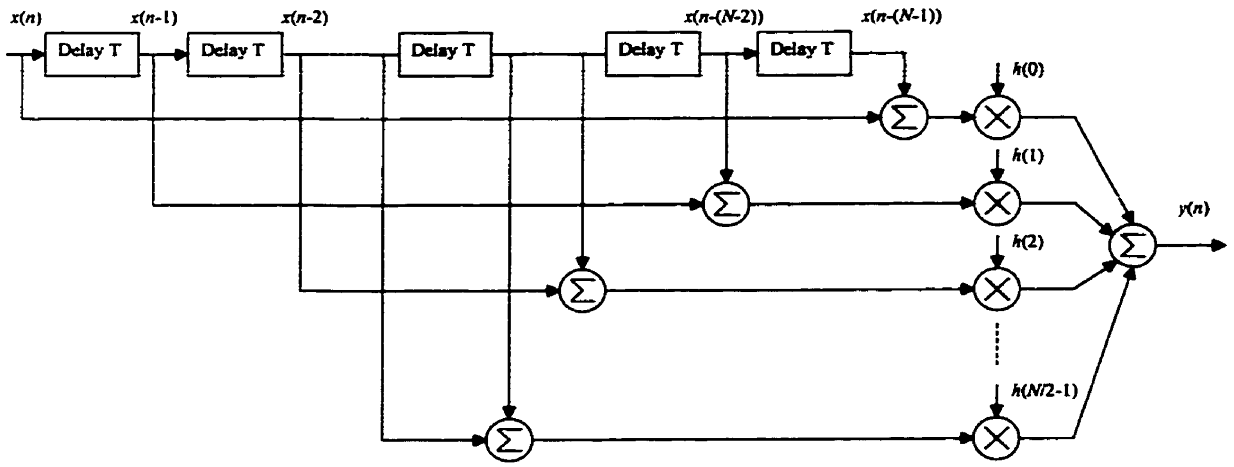

Figure 4-4 - Direct Form Architecture, FIR linear phase filter, N even

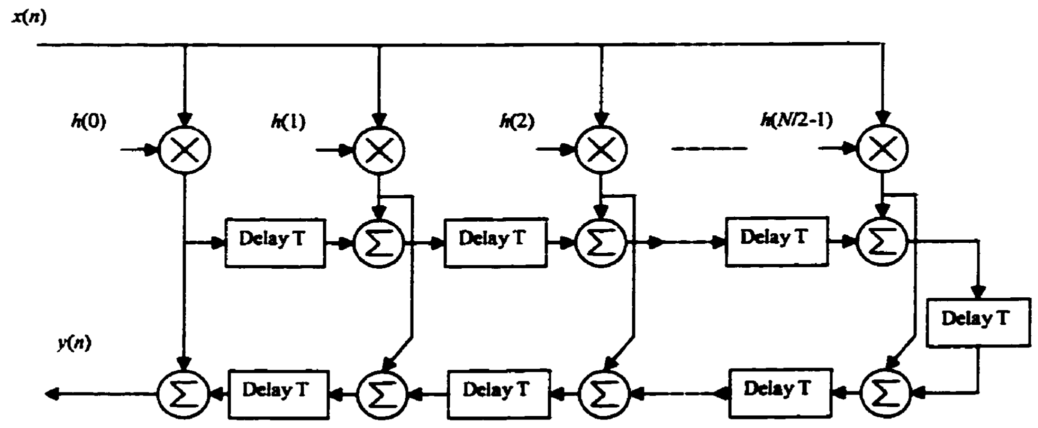

Figure 4-5 - Transposed Form Architecture, FIR linear phase filter, N even

For the transposed architecture, there is an additional benefit in that the fanout of the input data is reduced by a factor of two. For the direct form architecture, a small disadvantage may arise for high filter orders. Long interconnects may be required to add signals that are at opposite ends of the register chain (for example, $x(n)$ and $x(n-(N-1))$). The long interconnects will complicate routing, increase power consumption, and possibly increase overall delay. Placement of the register chain components would therefore be critical.

4.4 Transposed Forms with Multiplier Block: Exploiting Coefficient Redundancy

In section 3.4, the principle of decomposing filter coefficients in their factors to reduce the number of adders for multiplication was presented. It was also suggested that redundancy of factors in a set of coefficients could be exploited, as reported in [14]. This approach could then ensure that the total multiplication effort to implement a filter would be minimized.

If the transposed form realization shown in Figure 4-2 is used, then all multiplications of the input data occur at the same time. Using the minimum representation of multipliers, it may be possible to reuse partial products between coefficients. For example, say that coefficient $h(i)$ is 45, and coefficient $h(j)$ is 18. The total multiplication effort for the two coefficients requires only 2 adders. The input data x is first multiplied by 9, as in Figure 3-1(b), to give the interim result $9x$. It is then multiplied by 5 to yield $45x$. The interim result is also shifted one bit to the left to yield $18x$. Thus, both multiplications are accomplished using only 2 adders.

The individual multipliers of the transposed form can be combined in one major multiplication block, as shown in Figure 4-6 and Figure 4-7. The direct form realization of Figure 4-1 does not lend itself well to exploiting coefficient redundancy, because each multiplier in the filter is operating on a different data sample. The reuse of partial results would require a complex registering mechanism that would make the approach inefficient.

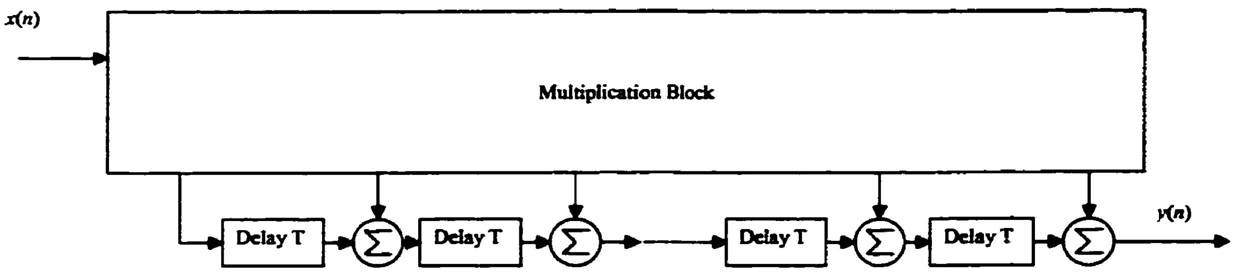

Figure 4-6 - Transposed Form FIR Filter with Multiplication Block

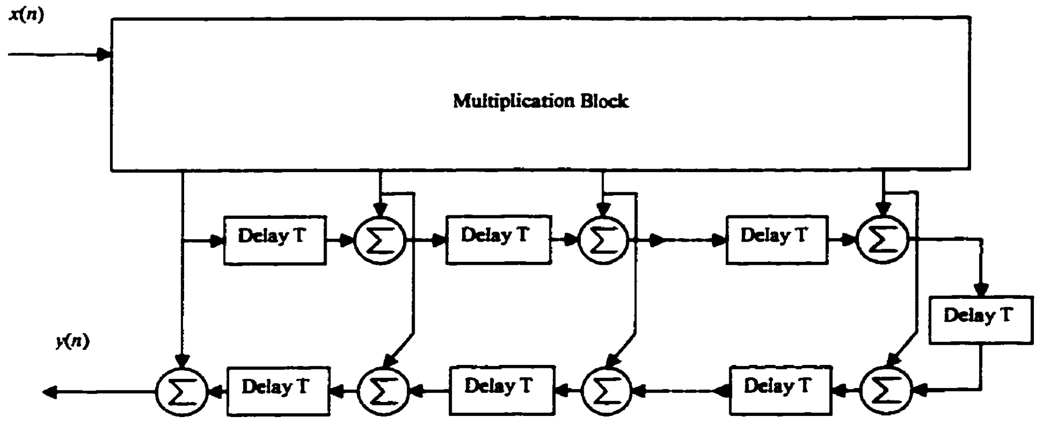

Figure 4-7 - Transposed Form Symmetric FIR Filter, N Even, with Multiplication Block

When realizing a filter in hardware, a legitimate question is therefore: Is the effort required to extract coefficient redundancy worth it? Dempster and Macleod [17] have studied this question by applying their redundancy-finding algorithm to a large number of random sets of coefficients of different sizes. Their first result is intuitive: the larger the set of coefficients, the more likely that some redundancy can be found and exploited. The second result is also intuitive: a smaller word length for the coefficients is likely to yield more redundancy as well, since the coefficients are more likely to have similar values. Thirdly, the improvement over the standard filter design will depend on how many adders are required for the multipliers *before* redundancy is exploited. As the authors point out, most FIR filters have many small coefficients that can often be represented by only one signed digit with acceptable accuracy. In these situations, no adders are required and little redundancy can be exploited. The final conclusion from the paper is that the multiplier block technique can lead to filter realizations where the contribution of the multipliers in the overall complexity is far less significant than the contribution from the structural adders and delay elements.

4.5 Pipelined Architectures

In order to process data at the highest possible rates on a FPGA, pipelining of the data paths is essential in the quadrature demodulator. This technique consists of breaking up data paths into smaller blocks, with processing done in parallel among the blocks. The latency of the overall operation is increased, but the output data rate can be increased significantly. Latency is defined as the number of clock cycles required between the time a given data appears at the input, and the time its effect is first seen at the output. In some applications, an increase in latency is not acceptable, and other design and optimization techniques must be used to increase processing speed. This is generally not the case for quadrature demodulators.

FPGAs are also prime candidates for extensive pipelining because they are register-rich. As mentioned already, each CLB output has an attached flip-flop that only needs to be activated at chip programming time, and these flip-flops can therefore be considered "free" from a chip real-estate point of view.

It is true that adding flip-flops in a data path requires that setup and hold times be considered, and that the extra circuitry increases power consumption. However, it may be the only way to meet a stringent timing requirement. In many ASIC technologies, the problem of clock distribution is increased with register count. For FPGAs, however, dedicated clock distribution networks are available on the chip at no additional routing cost. Therefore, pipelining techniques are the method of choice to increase processing rate, especially in FPGAs, and they can be used extensively throughout fast quadrature demodulator designs.

4.5.1 Pipelined Adder Tree for the Direct Form (Version I)

The main disadvantage of the direct form realization of Figure 4-1 and Figure 4-4 is the multi-operand adder. For high-speed applications such as the quadrature demodulator, this adder must be broken down into a number of pipelined stages. As discussed previously, an existing quadrature demodulator design at CRC breaks the large adder into a series of pipelined 3-to-2 compression stages [21]. For an implementation on a Xilinx 4000-series FPGA, however, the dedicated carry logic favors two-operand adder configurations based on ripple carry. For the direct form, an adder tree configuration, shown in Figure 4-8, is therefore the most appealing approach.

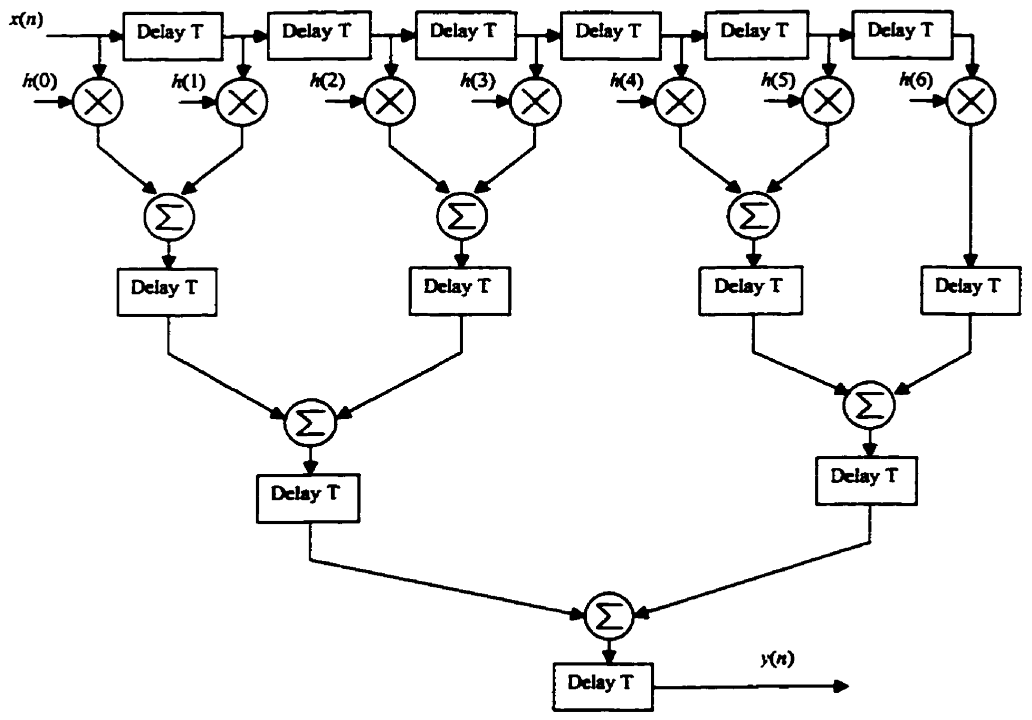

Figure 4-8 - Pipelined Direct Form, Version I: Adder Tree

There are two main disadvantages to the adder tree. The first one is that the tree only scales optimally for filter orders that are a power of 2. For example, in Figure 4-8 the right-most multiplier output must be registered prior to addition to an intermediate sum. The second disadvantage is that the tree interconnects get progressively longer with higher filter orders, and the structure is not easily compacted in a regular array of CLBs on an FPGA.

4.5.2 Alternate Pipelined Direct Form (Version II)

An alternate pipelined direct form architecture is shown in Figure 4-9. It doesn't suffer from interconnects that get progressively longer as in the tree structure described above. It is very similar to the basic transposed form of Figure 4-2, but requires a second, half-rate clock for the input delay chain (or, alternatively, twice as many registers). In either case, an increase in the number of registers over the basic transposed form is necessary. There are no significant advantages to using this architecture, and it will not be considered further.

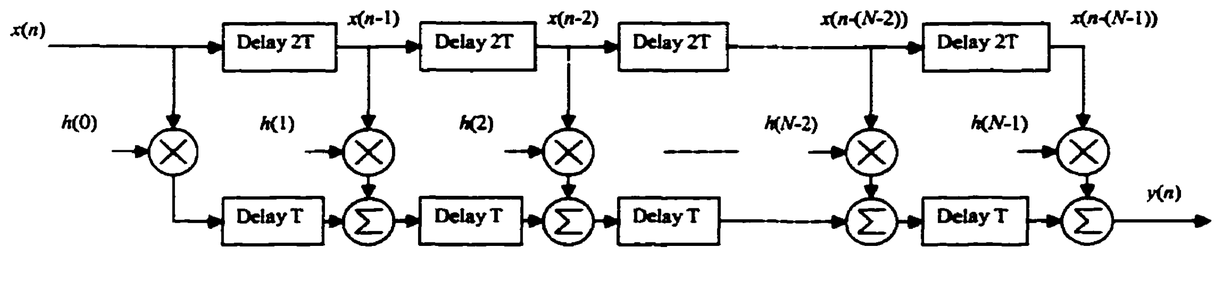

Figure 4-9 – Pipelined Direct Form, Version II

4.5.3 Transposed Form with Pipelined Input for Facilitated Placing and Routing

One of the drawbacks of the transposed form realizations of Figure 4-2 and Figure 4-5 is the high fanout of the input data stream for high filter orders. The consequences of such a high fanout is a longer propagation delay, and possible difficulty in routing interconnects on the chip. The input data is not only routed to many multipliers, it must also drive a number of adders within each multiplier. Pipelining can be used to reduce this problem by adding a data register before each of the multipliers, as shown in Figure 4-10. The fanout of the input data is then reduced to the order of the filter, or to half of the filter order for the linear-phase case. Each added register now acts as a data buffer to drive its adjoining multiplier. If the input data fanout is still too high, a further improvement would involve a "tree" distribution, a technique often used to distribute a clock signal in a chip. An alternative approach would be to break the input data and addition paths at the same level and insert additional pipeline registers.

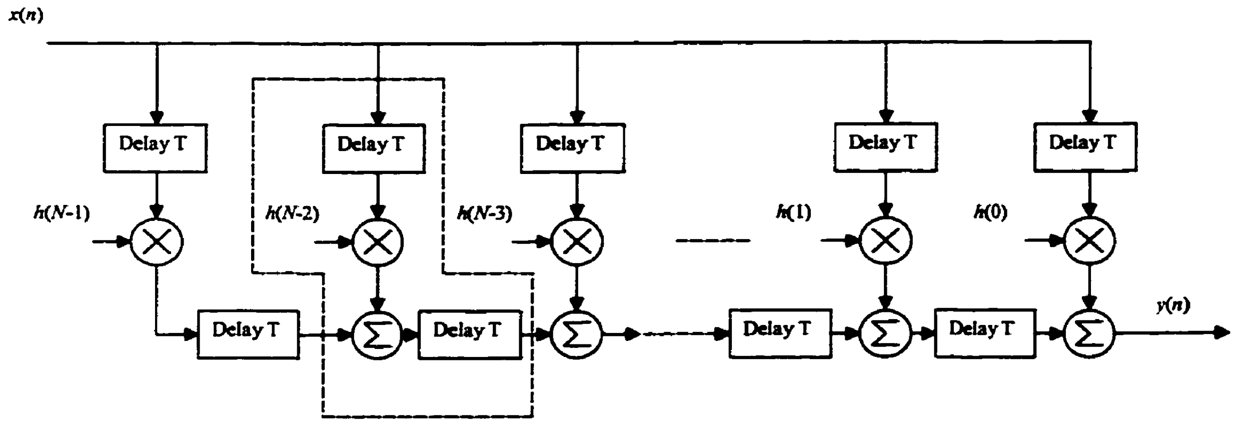

Figure 4-10 - Transposed Form with Pipelined Input

This architecture also presents the advantage of a regular block structure, highlighted in the figure, which is a most desirable feature for FPGA realization and for other VLSI implementation approaches[1]. Once a block has been optimized, a high order filter is built by simply connecting together many blocks, which involves simple placement and routing. It also increases the density of the design on the FPGA since the blocks can be neatly stacked one beside the other. For optimal input data distribution, all sub-blocks

[1] This assumes that all multipliers in the filter have the same physical structure on the FPGA (i.e. number and relative position of CLBs, and routing). Since interesting filters have coefficients that are not all identical to each other, this implies a Look-Up Table approach for the realization of the multipliers.

must be aligned horizontally so that one of the long distribution lines can be used for routing. The technique is easily extendable to implementation across multiple chips, and is equally applicable to the realization of linear-phase filters.

4.5.4 Pipelined Multiplier Block for the Transposed Form

The transposed architecture with multiplier block was presented in section 4.4 and the advantages of exploiting coefficient redundancy were explained in section 3.4. There is another advantage to the multiplier block architecture.

In all cases, if pipelining of the multipliers is introduced, then each multiplier's latency must be identical. Since some multipliers may be reduced to a simple shift of the input data, they would not normally require pipelining. However, if some coefficients require multipliers with many stages of pipelining, then for the standard approach even the simple shift-multiplier will require a large number of pipelining stages. For a high order filter, there would be a considerable overhead in extra registers.

With the multiplier block approach, the input data can be registered to a depth corresponding to the number of pipelining stages required by the most complex coefficient. Each multiplier then "extracts" the input data at the required pipeline depth for its own processing. For the most simple multipliers, whose coefficients are expressed by one or two non-zero digits, the input data is taken from the last pipeline stage. For the multipliers with coefficients expressed by three non-zero digits (i.e. requiring two additions), data is taken from the two last pipeline stages. Multipliers with more complex coefficients progressively take data sample from more pipeline stages.

An example of a pipelined multiplier block is shown in Figure 4-11.

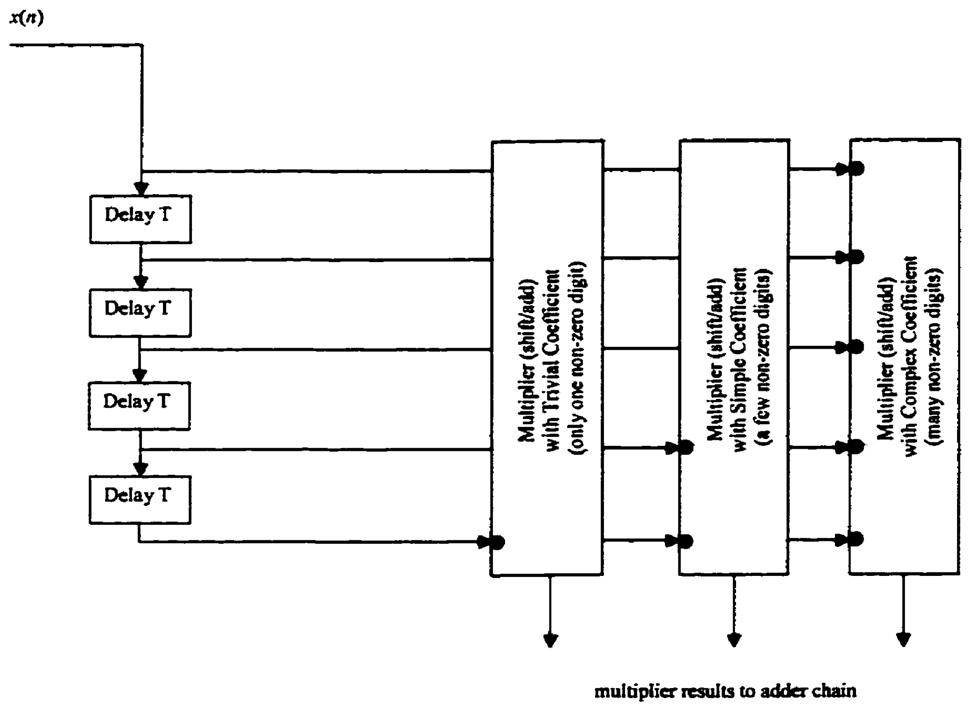
x(n)
Delay T
Delay T
Delay T
Delay T
Multiplier (shift/add) with Trivial Coefficient (only one non-zero digit)
Multiplier (shift/add) with Simple Coefficient (a few non-zero digits)
Multiplier (shift/add) with Complex Coefficient (many non-zero digits)
multiplier results to adder chain

Figure 4-11 - Pipelined Multiplier Block Example

4.6 Fast Addition for Digital Filter Architectures

In this section, the specific problem of increasing the speed of the additions in the filter architecture will be considered.

The Dedicated Carry Logic in the Xilinx 4000 series FPGAs leads to very fast two-operand ripple carry adders. The dedicated carry paths usually make sophisticated adder configurations such as the Carry Bypass and Carry Look-Ahead adders unnecessary. For the 4000XL family of chips with a "-09" speed grade, the fastest ripple carry adders run at 139 MHz for 8 bits, 115 MHz for 16 bits, 98 MHz for 24 bits and 86 MHz for 32 bits [22]. While impressive, these results may not be enough for quadrature demodulator designs with high filter orders that require adders that are both fast and wide.

Because the Dedicated Carry Logic is so fast compared with the rest of the propagation characteristics in the Xilinx FPGA, it makes sense to take advantage of it as much as possible. Three alternative adder architectures that are based on the ripple carry will be considered: the Carry Select Adder, the pipelined ripple carry adder, and a proposed delayed-carry adder chain.

4.6.1 Carry Select Adder

The Carry Select Adder configuration is a prime candidate to accelerate the additions in the filter. In this adder configuration, the addition is broken among a number of stages each dealing with a fraction of the total number of bits to be added. At all stages except the least-significant one, two adders are active at any one time, each assuming a different value for the carry from the previous stage ('0' for one adder, '1' for the other). Once the carry out from the previous stage is available, it controls a multiplexer to select the appropriate present stage adder output. All adders at all stages are ripple carry adders. An example for a 32-bit Carry Select Adder segmented in two stages is presented in Figure 4-12.

The advantage of the Carry Select Adder is that the delay on its critical path is much reduced when compared to the ripple carry adder, since the carry doesn't need to propagate as far. It also doesn't require pipeline registers. However, it requires more silicon real-estate for the extra adder and multiplexer at every stage.

4.6.2 Pipelined Ripple Carry Adder

If chip area is limited, the Carry Select Adder loses its appeal. In such a case, a pipelined ripple carry can offer the same performance [23]. The disadvantage is an increased latency and a requirement for extra registers. For quadrature demodulators, increased latency is not a concern, and registers are available at low cost in the targeted FPGAs.

As for any other pipelining strategy, the idea is to break the addition into different stages and to add a level of registers at each stage. The addition in each stage is delayed in time by one clock cycle from the previous stage. The carry from one stage is also delayed, and fed to the next stage as it begins processing its data. The results of each stage are also registered as many times as required to ensure that the adder output bits are synchronized. Figure 4-13 illustrates the method for a 32 bit adder segmented into two 16-bit adder stages.

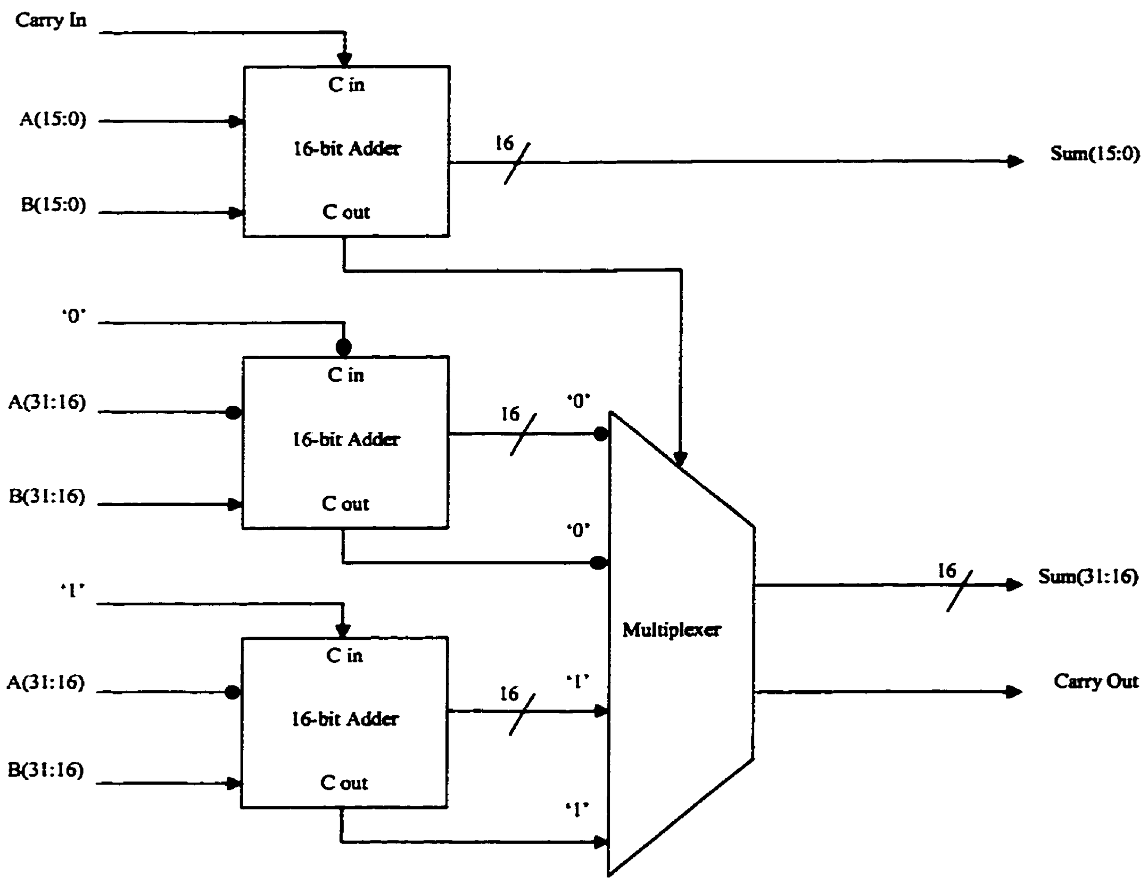

Figure 4-12 - Carry Select Adder Example

In the first stage of the adder, the least significant halves of the two operands are added together. The sum is registered, together with the last carry out. The most significant halves of the two operands are registered as well. In the second stage of the adder, the registered most significant halves of the operands are added together with the carry out from the previous stage. The most significant half sum is then combined with the registered least significant half sum for the final result.

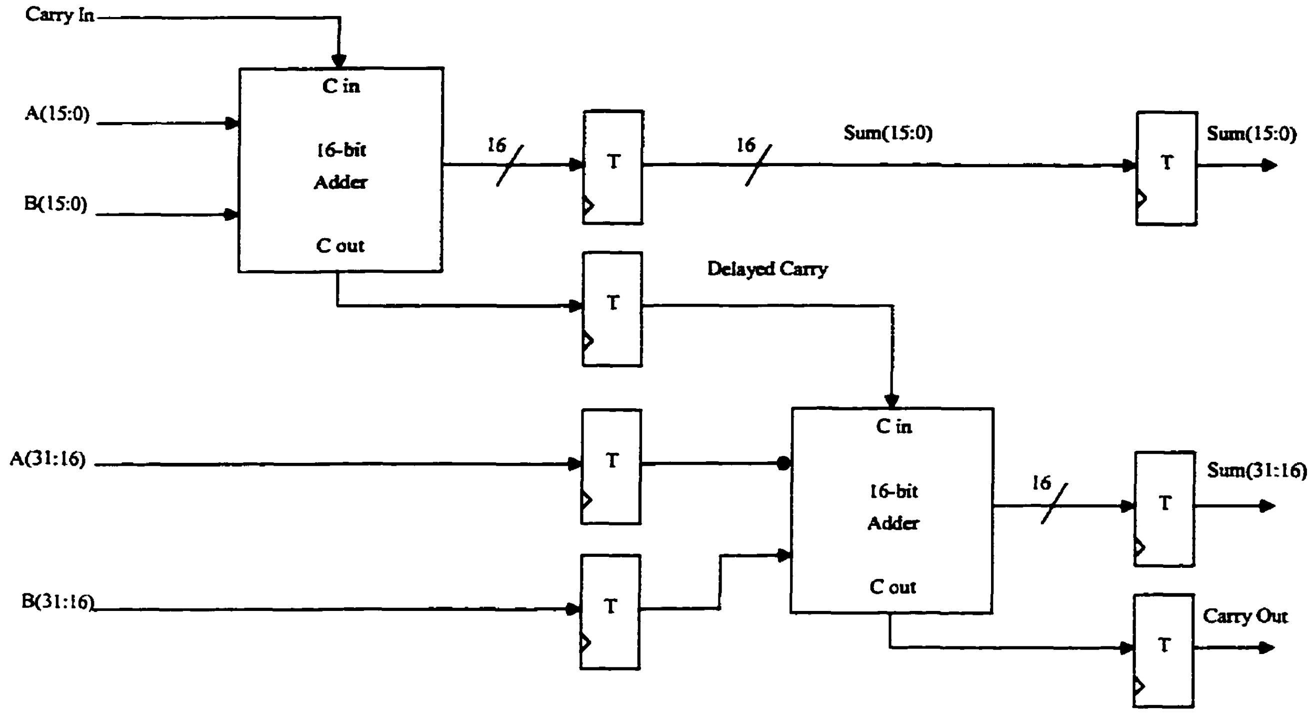

Figure 4-13 - Pipelined Ripple Carry Adder Example

The advantage of the pipelined ripple carry adder is that the processing rate is now only limited by the speed of one of the adder stages (as for a Carry Select approach with two-level segmentation). With the previous Xilinx specifications given, and for the example illustrated above, this means that a 32-bit adder could run at almost the 16-bit adder speed of 115 MHz, with a small reduction due to the extra routing delay between the stages and through the registers. The main disadvantage, other than an increased latency, is an obvious increase in the storage requirements. For the present example, 33 extra registers are required to delay the most significant portion of the operands and the delayed carry. In general, for an *N*-bit adder segmented into *k* stages of equal size, the extra register requirement grows as the square of the stage size *N/k*. A 32-bit adder segmented into 4 stages of 8 bits would require 147 extra registers.

In the case of an FPGA implementation, this means that a great number of extra CLBs would have to be dedicated to registering data without performing any processing, which is far from an efficient device utilization. A high packing ratio could probably not be attained. The pipelined ripple carry adder should therefore not be considered to increase design performance of the quadrature demodulator.

4.6.3 Delayed-Carry Chain

Although considered too expensive in overhead, the pipelined ripple carry adder concept opens up an interesting alternative, only alluded to in [23]. In the case of FIR filters implemented with one of the transposed forms, the whole pipelined adder chain can be segmented into a number of adder sub-chains. A simple example with a 4th order filter is shown in Figure 4-14, with a segmentation into two stages.

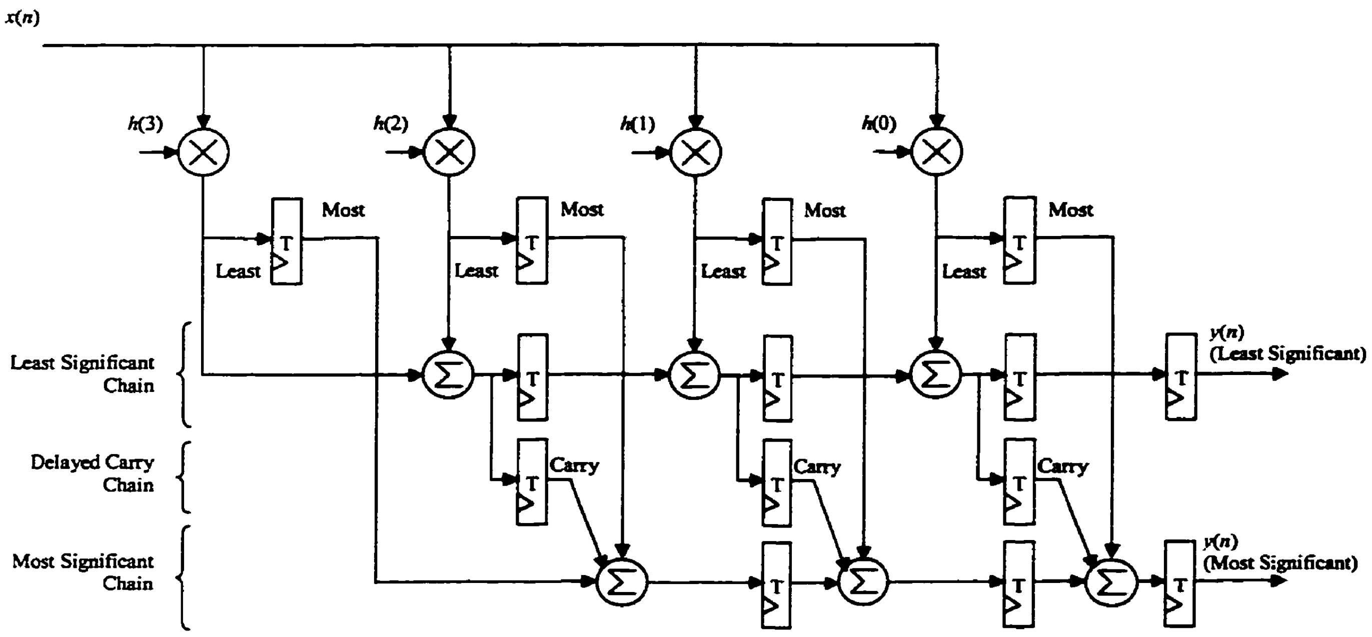

Figure 4-14 - Pipelined Delayed-Carry Adder Chain Example (Transposed Form)

Each adder in the adder chain is decomposed into a number of stages corresponding to portions of the data to be processed. Results from each adder in a stage are not synchronized but passed immediately along the chain. Adder stages of more significant levels must receive the previous level's carry out before proceeding, and the carry is therefore delayed as in the pipelined ripple carry adder. Synchronization is done only at the end of the adder chain, with the consequence that a very high device utilization density can be reached. Except for the end of the adder chain, no CLBs need to be reserved for registering only.

The input data to the adder chains must be properly segmented and skewed in time. In the example of Figure 4-14, it is seen that the multiplier outputs are divided into most and least significant halves. Each half is fed to its corresponding adder chain, but the most significant data is first delayed by one clock cycle. This is necessary to ensure that the carry from the least significant chain arrives at the same time as the most significant multiplier result to be added in the most significant adder chain.

The pipelined delayed-carry adder chain concept described here for the transposed form could be easily adapted to the adder tree structure of the direct form shown in Figure 4-8. In either case, the adders can be segmented into as many stages as required, down to the case where they reduce to half- and full-adders.

The system overhead has two components: the additional registers in the multiplier block required to delay the arrival of operands to the adder chain, and the additional registers in the adder chain itself. The first overhead component depends on the value of the filter coefficients, since the number of registers required will depend on the width of the multiplication product. If a multiplication product is expressed with a number of bits smaller than the adder chain bus width at this stage, then only one bit needs to be carried for sign extension.

For the adder chain, the overhead can be calculated as a function of the number of segmentation levels, S, the order of the filter N, and the number of bits in each level, n_s. This value is assumed to be constant for every segmentation level, but a generalization could be made. First, N - 1 registers are required per segmentation level for the delayed carries. Second, the overhead due to the final synchronization is equal to (0 + 1 + 2 + ... + (S − 1)) × n_S. The total adder chain overhead in registers, *R*, is therefore equal to:

$$\begin{aligned} R &= S\cdot(N-1)+n_s\cdot(0+1+2+\ldots+(S-1)) \\ &= S\cdot(N-1)+n_s\cdot\frac{S(S-1)}{2} \\ &= \frac{n_s}{2}S^2+S(N-1-\frac{n_s}{2}) \end{aligned} \tag{4-4}$$

It is seen that for the adder chain the overhead is proportional to *N* and to the square of *S*.

The increase in processing rate significantly depends on the ripple carry adder timing characteristics of the target chip. An adder characterization study was performed for different X4000 families of FPGAs, and the results are presented in Appendix B. Disregarding routing delays to the adder, the latency for different ripple carry adders is approximately linear with a coefficient *k*, for adders wider than 4 bits. Applying the delayed-carry concept to an adder chain reduces the widest adder widths from n_0 to (n_S + 1). Given that the initial latency was T_o, the new latency is therefore approximated to:

$$T_s = T_0 - k(n_0 - n_s - 1), \quad n_0 > n_s + 1 \tag{4-5}$$

The latency decrease can be expressed as a ratio to the initial adder latency:

$$\frac{k(n_0 - n_s - 1)}{T_0} \tag{4-6}$$

Alternatively, the increase in processing rate is expressed as the ratio of the new processing rate to the initial processing rate:

$$\begin{aligned} speedup &= \frac{1}{T_s}\times\frac{T_0}{1} \\ &= \frac{T_0}{T_0 - k(n_0 - n_s - 1)} \\ &= (1-\frac{k}{T_0}(n_0 - n_s - 1))^{-1} \end{aligned} \tag{4-7}$$

It should be obvious from this relation that the increase in processing rate will be greater if k is large and if the number of segmentation levels is increased such that n_s gets very small in relation to n_0. From the results of Appendix B, the value of k gets progressively smaller for the faster X4000 FPGA families, and smaller as well for faster speed grades within a family. The increase in processing rate for a digital filter in which the delayed carry adder chain is implemented would therefore be greatest for the slowest FPGA families, and for cases where the initial adder chain width is widest. This last condition will occur most likely for high filter orders.

The delayed-carry adder chain has a few additional advantages other than speed. A high device density is maintained because few CLBs are reserved for their flip-flops only. A digital filter to which this approach is applied also benefits from facilitated placing and routing, for X4000 FPGAs. This is a consequence from the fact that using the dedicated carry logic requires ripple carry adders to be placed as a column for maximum speed. Segmenting large adders into smaller ones makes their placement easier. Finally, as shown in Figure 4-14, an additional advantage comes from the regular structure that can be easily repeated across an FPGA, as discussed in section 4.5.3.

4.7 Analysis of the Alternative Filter Architectures

Various alternative filter architectures were described, and some of their relative merits and disadvantages were identified. A summary of these merits is presented in Table 4-1. It is difficult to make an exact comparison between most of the architectures, however, because the set of specific coefficients chosen for a filter will have a direct impact on whether a certain architecture is preferable over another one. For example, if a set of coefficients exhibits a lot of redundancy, as described previously, then it would make sense to use a transposed form with multiplier block. Alternatively, in most cases coefficient symmetry should be exploited to reduce the number of multipliers in half, and to reduce input data fanout in the transposed forms.

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
Direct	standard[1]	No	No	No	$\propto N$	No	No	1
	linear phase	No	Yes	No	$\propto N$	No	No	1
Direct Pipelined Ver. I (Adder Tree)	standard	No	No	No	$\propto N$	Yes	No	1
	linear phase	No	Yes	No	$\propto N$	Yes	No	1
Direct Pipelined Ver. 2	standard	No	No	No	$\propto N$	Yes	Yes	1
	linear phase	No	Yes	No	$\propto N$	Yes	Yes	1
Transposed	standard	No	No	No	No	Yes	Yes	N+
	linear phase	No	Yes	No	No	Yes	Yes	$N/2$+
Transposed w/ Multiplication Block	standard	Yes	No	Yes	No	Yes	Yes	See 2
	linear phase	Yes	Yes	Yes	No	Yes	Yes	See 2
Transposed w/ Pipelined Input	standard	No	No	No	No	Yes	Yes	N
	linear phase	No	Yes	No	No	Yes	Yes	$N/2$
Cascade	not considered							

Table 4-1 - Filter Architecture Alternatives

Notes:

1. N is the filter order.

2. For the Transposed Form with Multiplication Block, the input data fanout depends on the coefficients and on whether pipelining of the multipliers is used or not.

[1] 'Standard' refers here to the basic, non-linear phase architecture, e.g. Figure 4-1.

The "best" architecture is also technology dependent. In the present case, FPGAs are specifically targeted, and the case has already been made with respect to the advantages of two-operand, ripple carry adders. This specific argument definitely favors the transposed forms in general, as it does the direct form with adder tree. Version II of the pipelined direct form is also rich in two operand adders, but it suffers from disadvantages that have already been highlighted.

The use of FPGAs should also favor architectures that enhance device utilization density. A high density has two immediate advantages: a smaller device may be required for a given filter design, and routing paths will be shorter from sub-block to sub-block. This latter point implies a reduction in routing capacitance, which will lead to lower power dissipation and lower propagation delays.

Decomposing the design into smaller blocks with a regular structure enhances density, because these sub-blocks can be neatly stacked one beside the other. The first architecture to favor a regular block structure is the pipelined transposed form, which *doesn't* exploit coefficient redundancy because the multipliers are part of the sub-blocks. Secondly, all transposed forms using the pipelined delayed-carry adder chain, with the multipliers kept outside of the repeated sub-blocks, also favor a regular block structure.

The alternate approach to increase device utilization density is to ensure as many resources as possible are used within each and every CLB. For a Xilinx 4000 series FPGA, this means that the two-operand adder with registered output is a very efficient building block, since it uses the F and G function generators, the dedicated carry logic circuitry, and the output flip-flops. A very poor CLB utilization comes from the utilization of the register elements only. In that respect, all direct forms are inefficient, since the delayed input data chain uses a great number of flip-flops from otherwise unused CLBs. This problem is exacerbated in high filter orders.

4.8 Decision

Based on this analysis, the preferred filter architecture for quadrature demodulation implemented in Xilinx FPGAs is based on a transposed form. Since linear-phase filters will be used, the architecture of Figure 4-5 is the best candidate. If the coefficient set exhibits enough redundancy, or if a considerable amount of pipelining is required in the multipliers, then the use of a multiplier block is warranted and the architecture of Figure 4-7 should be selected. Finally, if the adder chain has a wide bus width, due to either the specific set of coefficients or simply to the high order of the filter, and if a high data rate is required, then the pipelined delayed-carry adder chain form shown in Figure 4-14 is the most suitable. It must be noted that the multiplier block, symmetry utilization and pipelined delayed-carry adder chain are all "orthogonal" to each other, which means that they can be used alone or in conjunction with others, as most appropriate depending on the specific filter coefficients.

Chapter 5

Detailed Design Descriptions

5.1 Introduction

In this chapter, a detailed description of four quadrature demodulator designs will be given. The designs follow decisions made in previous chapters on implementation approach, multiplication realization and architecture selection. Two of the designs were implemented and tested in a Xilinx X4010E-3 chip, while the other two were only simulated. Each design reflects a different set of specifications, and together the four designs demonstrate the viability of using FPGAs for high performance quadrature demodulation. The four designs are similar in the sense that they are all based on the same building blocks.

5.1.1 Overview of Designs Considered

The first three designs described here are based on the polyphase filter approach discussed in section 2.3.3. The input signal bandwidth is limited to f_c, the prototype filter is a quarter-band filter, and the outputs of the in-phase and quadrature channels are decimated by four.

Three variations of this design were considered. For the first one, the sampling frequency was set at 213.3 MHz, as described in section 2.5, which implies a filter processing rate of 53.3 MHz. The adders in the filter adder chains were implemented as simple ripple-carry adders, because the targeted FPGA family was fast enough for this processing rate. It was assumed that the output signal from the Analog-to-Digital Converter (ADC) was de-interleaved outside of the FPGA. Consequently, four input ports and a single clock were required. This design was implemented and tested in a Xilinx 4010E-3 chip, and simulations were run for other FPGA families and speed grades.

For the second variation, the goal was to demonstrate the delayed-carry adder chain discussed in section 4.6.3, and to verify the speed increase and overhead costs when compared to the basic design. A 4010E-3 chip was again targeted, and it was assumed that all signal de-interleaving was done off-chip.

The third variation of the polyphase-filter design is identical to the second one, with the exception that it was assumed that data from the ADC would come in two interleaved streams. The de-interleaving into four streams, one for each of the polyphase sub-filters, was done inside the FPGA, as was gray code-to-2's complement conversion. A suitable ADC was selected, together with appropriate logic level translators to convert the ECL I/O logic levels used by the ADC to TTL logic levels used by the FPGA. The de-interleaving process requires very fast I/Os, and a 4000XL family chip was selected. The extra logic also required the selection of a larger chip, the 4013. This design was not implemented but was simulated.

The last design described here is based on the Low-Pass Filter Approach discussed in section 2.3.1. This design maximizes input signal bandwidth and consequently only a single decimation by two is allowed. It is suitable for any prototype filter design, but specifically for the half-band and third-band cases which correspond to a maximum output decimation factor of 2. This design was not implemented, because optimized filter coefficients were not available. As such, it is described only in general terms.

5.1.2 Filter Coefficients and Frequency Response Characteristics

For the polyphase filter approach designs, signed digit filter coefficients identical to those of a previously realized quadrature demodulator implemented in Gallium Arsenide gate array technology [6][24] were used. The coefficients were obtained using a two stage search strategy similar to that described in [25].

Although a relatively small number of signed digits was used, good performance was obtained, in part through the use of an optimization criterion concerning frequency response matching of the I and Q filters [26]. This optimization criterion dramatically improves system performance for quadrature demodulation filtering over a standard design approach such as the Hamming window method. The phase error resulting from the frequency response mismatch between the in-phase and quadrature channels is smaller than the error resulting from the effects of quantizing the input signal to 8-bit resolution.

The resulting prototype filter is a quarter-band, linear-phase FIR filter of order 29. Six coefficients are null, and eighteen are equal to a single power of two. Four coefficients are represented with four signed digits each, and one is represented with only three signed digits. The prototype filter impulse response is given here, with a normalization factor of 256:

{-1, -2, 0, 4, 8, 8, 0, -16, -32, -32, 0, 64, 141, 206, 232, 206, 141, 64, 0, -32, -32, -16, 0, 8, 8, 4, 0, -2, -1}

The normalization factor was applied so that all coefficients would be integers. This has no impact on the system output other than a known gain, which must be accounted for when interpreting the resulting data. However, it greatly simplified the design since integer arithmetic could be used throughout.

A normalized frequency and phase response plot of the prototype filter is given in Figure 5-1. The pass band matching properties for the resulting I and Q filters are discussed further in [6].

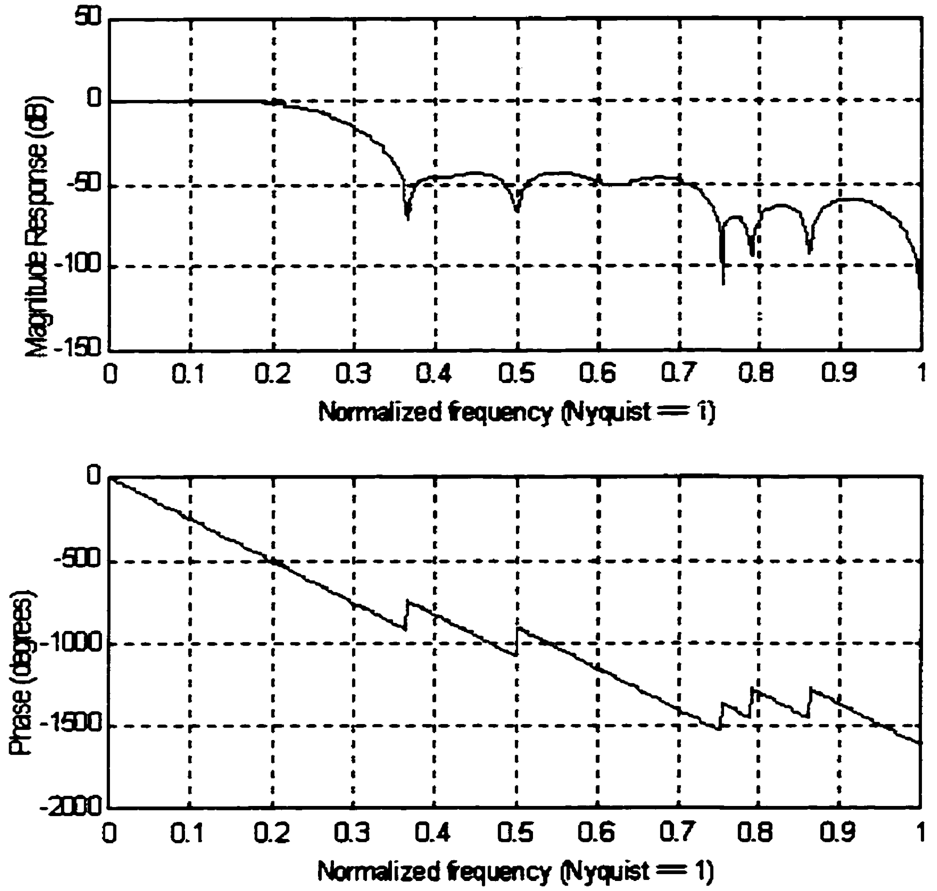

Figure 5-1 - Prototype Filter Frequency and Phase Response

Table 5-1 lists the mapping of the coefficients to the four polyphase sub-filters, according to equations (2-23), (2-24), (2-28) and (2-29). It is noted that all zero coefficients are mapped to sub-filter I_1, that sub-filter I_0 is an odd-length symmetric filter, and that sub-filters Q_0 and Q_1 are non-symmetric even-length filters. It is also noted that the coefficients for the two Q sub-filters can be divided by two, which reduces the implementation cost further.

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
0	-1	0	-2	4
1	8	0	8	-16
2	-32	0	-32	64
3	141	232	206	206
4	141	0	64	-32
5	-32	0	-16	8
6	8	0	4	-2
7	-1			

Table 5-1 - Prototype Filter Coefficient Mapping

5.2 Polyphase Filter Approach: Basic Design

5.2.1 General Overview

The first implemented quadrature demodulator design follows the polyphase filter approach discussed in section 2.3.3. The input signal bandwidth is limited to f_c, the prototype filter is a quarter-band filter, and the outputs of the in-phase and quadrature channels are decimated by four. The target sampling frequency is 213.3 MHz, as described in section 2.5, which implies a filter processing rate of 53.3 MHz. This in turn implies a maximum processing delay on the critical paths of 18.75 ns. An additional goal was to implement the design in a Xilinx 4010E-3 chip, which was readily available, and to perform hardware testing.

In order to simplify the implementation and to ensure that the design would fit in the X4010, it was assumed that the data would be supplied to the FPGA in four de-interleaved streams. A block diagram of the design is given in Figure 5-2.

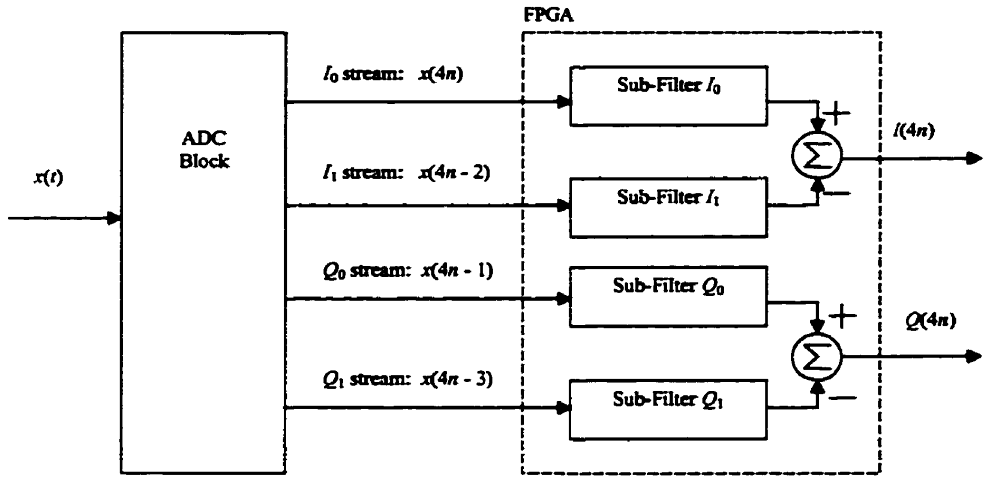

Figure 5-2 - Top Level Block Diagram (Basic Design)

The design is effectively broken up into 4 major blocks, corresponding to each of the four sub-filters defined in Table 5-1. The output from sub-filter I_1 is subtracted from the output from sub-filter I_0, and similarly for the Q sub-filters. Each sub-filter block receives its input directly from the ADC block which is outside of the FPGA. It is assumed that the ADC block performs de-interleaving into the four data streams in addition to analog-to-digital conversion, and that its data is in two's complement representation.

The four sub-filter blocks are very similar in structure, although filter I_1 is reduced to a single multiplication followed by adequate registering for proper synchronization with the other sub-filters. As discussed in Chapter 4, the transposed form with multiplier block was selected for the filter architectures. A block diagram applicable to either of the sub-filters is given in Figure 5-3.

For the implementation, there is an important advantage to breaking the design into four well-defined blocks. Each block's design can be optimized independently, and the placement in an FPGA is also simplified. Since there is no communication between the blocks, other than for the output of the I and Q sub-filter pairs, this translates to reduced communications across the chip, and hence to a design that can run faster.

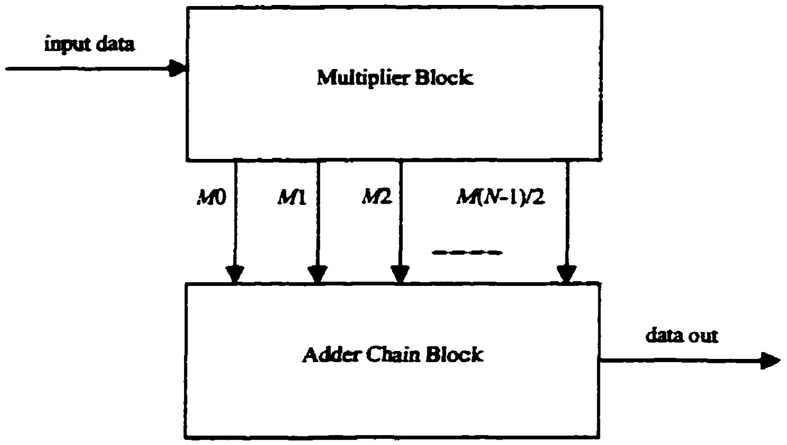

Figure 5-3 - Sub-Filter Block Diagram

5.2.2 Multiplier Block Description

The four multiplier blocks are very similar in structure. In each case, they implement only one multiplication requiring more than one addition. The other partial products are simple shifted replicas of the input data. Since the additions that implement the multiplication are pipelined, the input data must also be pipelined by the same amount so that it can be used for the partial products corresponding to power-of-two coefficients. In order to minimize the overall computation effort, no sign inversion of the input data was performed and all trivial multiplication results are positive. To properly implement negative coefficients, the partial products were added or subtracted, as appropriate, in the adder chain blocks.

Figure 5-4 below shows the structure of the multiplier block Q_0 sub-filter, which implements multiplication by 206 and delays the input data for the power-of-two coefficients. The Q_1 multiplier block is identical, and the two I sub-filter multiplier block are very similar, although they implement multiplication by different coefficients. The multiplier results corresponding to power-of-two coefficients are simple shifts of the input data, and are taken care of by shifting the connections to the adder chain block by the appropriate number of positions. The multiplication result $206x$ is more difficult to calculate and requires three adders-subtracters since the coefficient $h(3)$ has four non-zero digits.

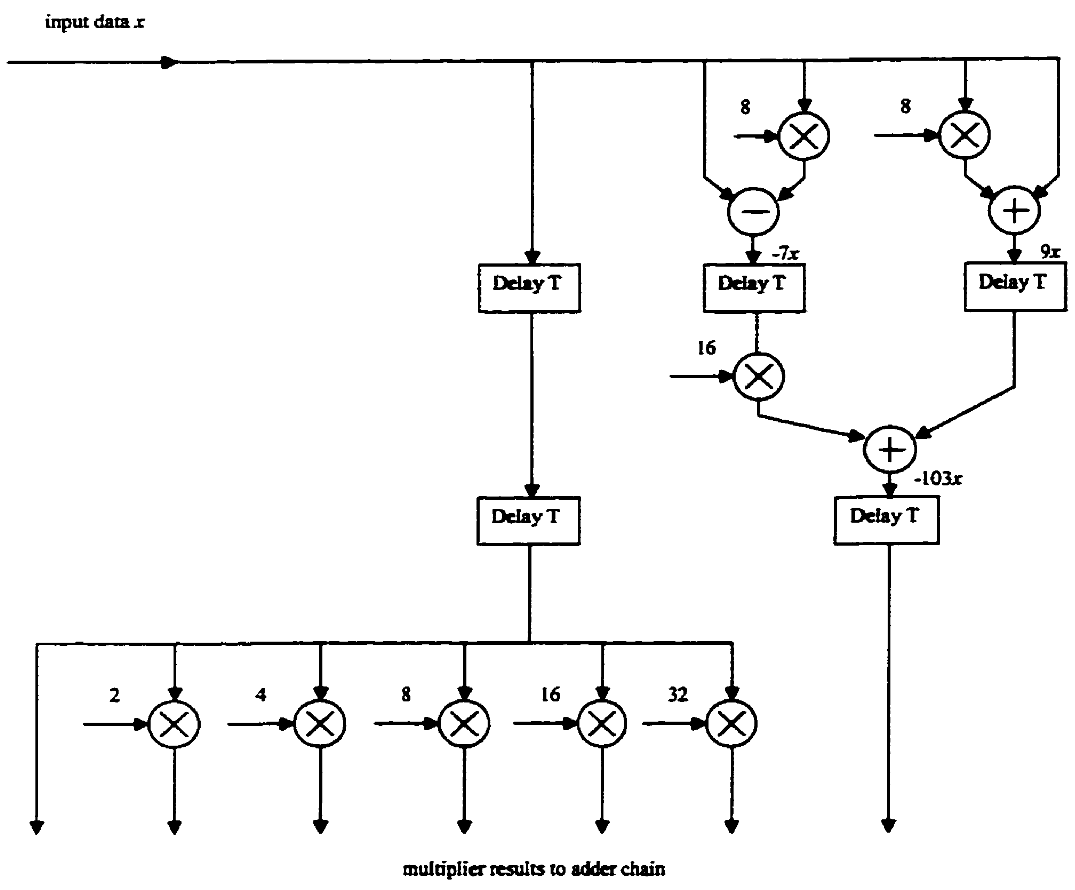

Figure 5-4 - Q_0 Sub-Filter Multiplier Block

In order to reduce the necessary computations, the product $-103x$ is calculated instead of $206x$. The sign inversion is acceptable, as long as it is taken into account in the design of the adder chain. There, the corresponding adder must be changed into a subtracter. There is no timing impact to this change since subtraction and addition operations have exactly the same latencies in Xilinx FPGAs. Dividing the coefficient by two also has no consequence if the result is interpreted correctly. Calculating the negative of the desired product allows a reduction in computation since it becomes possible to reduce the number of added bits at each multiplication stage. Consider the following typical example to illustrate this point.

Given an integer p and two four-bit numbers x and y represented in two's complement, the addition $x + (2^p \times y)$ for $p = 3$ is done as follows:

$$\begin{array}{cccccccc}
x_3 & x_3 & x_3 & x_3 & x_3 & x_2 & x_1 & x_0 \\
+y_3 & y_3 & y_2 & y_1 & y_0 & 0 & 0 & 0 \\
\hline
\end{array}$$

which means that only four bits need to be added for the eight bit result. Similarly, the subtraction $x - (2^p \times y)$:

$$\begin{array}{cccccccc}
x_3 & x_3 & x_3 & x_3 & x_3 & x_2 & x_1 & x_0 \\
-y_3 & y_3 & y_2 & y_1 & y_0 & 0 & 0 & 0 \\
\hline
\end{array}$$

is in fact implemented as follows:

$$\begin{array}{cccccccc}
x_3 & x_3 & x_3 & x_3 & x_3 & x_2 & x_1 & x_0 \\
\bar{y}_3 & \bar{y}_3 & \bar{y}_2 & \bar{y}_1 & \bar{y}_0 & 0 & 0 & 0 \\
+\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
\hline
\end{array}$$

and again only four bits need to be added, with a carry in position 3, for the eight bit result. Such a simplification is not possible, for the operation $(2^p \times x) - y$ which requires that all eight bits be added with a carry in position 0:

$$\begin{array}{cccccccc}
x_3 & x_3 & x_2 & x_1 & x_0 & 0 & 0 & 0 \\
\bar{y}_3 & \bar{y}_3 & \bar{y}_3 & \bar{y}_3 & \bar{y}_3 & \bar{y}_2 & \bar{y}_1 & \bar{y}_0 \\
+\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\hline
\end{array}$$

Therefore, if applicable, the subtraction operands should be reversed to simplify calculations. The product corresponding to coefficient $206 = 2 \times (16 \times 7 - 9)$ is such a case. The first partial product is $9x = 8x + x$. The second partial product is $7x = 8x - x$, but instead $-7x = -8x + x$ is calculated. Finally, calculating $-103x = 9x - 16 \times 7x$ requires less effort than calculating $103x = 16 \times 7x - 9x$. It is unimportant whether $-103x$ or $+103x$

is passed to the adder chain block, as long as the proper operation, addition or subtraction, is performed there.

5.2.3 Adder Chain Block Description

The three sub-filters I_0, Q_0 and Q_1 have similar adder chains composed of two-operand registered adders. The adder operands are the previous adder's output and a multiplier block output. However, there are significant differences between the three sub-filters. The Q sub-filters' impulse responses are non-symmetric and have even length, while the I_0 sub-filter has an odd-length, symmetric impulse response. The value and order of the filter coefficients also make the three adder chains different with respect to their bus widths at every stage. Figure 5-5 illustrates the adder chain block for the Q_0 sub-filter.

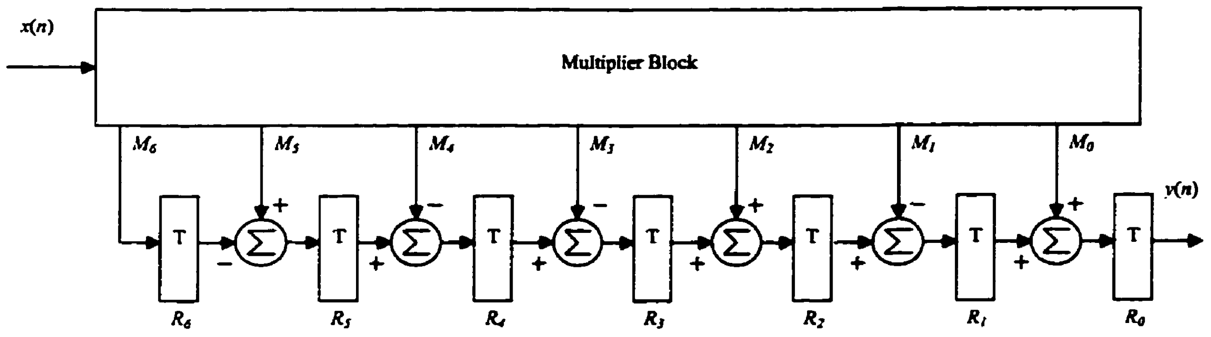

Figure 5-5 – Q_0 Sub-Filter Adder Chain Block Diagram

A filter bus width analysis was performed for each sub-filter, as described in Appendix A. The aim of the analysis was to calculate the number of bits required in the adders and registers of the adder chain to prevent overflow.

An important simplification was made by placing a restriction on the allowable input data to each of the filters. If the input data is allowed to take all possible two's complement values, then effectively an extra bit is required at every stage of the adder chain, for the unlikely case where $x = -2^{n-1}$. Further, allowing this input value implies a bias in the input to the ADC, and it was assumed that the input signal would have equal amplitude swings around a zero value. For the design, it was therefore decided to disallow such an input and restrict the input data to the range -(2^{n-1} - 1) to +(2^{n-1} - 1), or –127 to +127 in this case. This effectively saves one registered bit for every stage of the adder chain, and one bit for every pipeline level of every multiplier block as well. This was considered an acceptable design compromise.

Following the terminology given in Appendix A, the restriction on input data implies that $S_{neg}(i)$ will be equal to $S_{pos}(i)$, here denoted as $S(i)$. Equation (A-9), giving the number of bits required at every stage, can therefore be reduced to:

$$b(i) = \lceil \log_2(S(i)+1)+1 \rceil \qquad \textit{(5-1)}$$

with $S(i)$ given by a simplification of equation (A-6):

$$S(i) = (2^{n-1}-1) \times \sum_{k=N-1}^{k=i} |h(k)| \qquad \textit{(5-2)}$$

The results from these equations are given in Table 5-2 for each of the four sub-filters.

Stage	[illegible]	[illegible]	[illegible]	[illegible]
7	8	n/a	n/a	n/a
6	12	0	9	8
5	14	0	12	11
4	16	0	14	13
3	17	16	16	15
2	17	16	16	16
1	17	16	16	16
0	17	16	16	16

Table 5-2 - Bus Width of the Sub-Filter Adder Chains

5.2.4 Internal and External Timing Considerations

The target processing rate for the design was 53.33 MHz in each of the sub-filters and for the system outputs. The maximum delay on any path was therefore set at 18.75 ns. The two widest adders in the design are the final adder chain adder for the I_0 sub-filter and the final adder for the Q channel, and both are 17-bits wide. As per the adder characterization results reported in Appendix B, a 17-bit ripple carry adder implemented in the X4000E-3 family of chips requires 15.26 ns to produce a result, including the propagation delay of the registers supplying the operands. This therefore left 3.49 ns for the routing of the operands from the previous registers to the 17-bit adders.

The Xilinx automatic Placement and Routing tool was unable to meet the 18.75 ns constraint. It was therefore necessary to manually constrain the placement of critical components in the sub-filter adder chains and multiplier blocks. For the final placement, a maximum routing delay of 1.74 ns to the widest adders was attained, with a corresponding minimum clock period of 17 ns and maximum clock rate of 58.8 MHz.

For the targeted device, the Input-Output Block (IOB) flip-flops have a setup time of 7.0 ns and a hold time of 0.0 ns, due to the addition of a delay in the flip-flop's clock line. For the target clock rate, this leaves a full 11.75 ns to account for propagation delays from the ADC block and for any clock skews between that block and the FPGA.

5.2.5 Implementation in Other FPGA Families and Speed Grades

A timing analysis for the implementation of the design in a X4010XL device with the -09 speed grade indicates that an internal clock frequency of 102 MHz would be achievable. This implies that the design could achieve a 408 MHz sampling rate and a 102 MHz intermediate frequency, a result similar to the performance of a previous quadrature demodulator design reported in [24] and implemented in a GaAs gate array. The I/Os, however, are significantly slower and special consideration would have to be given to them.

The input flip-flop setup and hold times for the X4010XL device can be selected from three sets of values, depending on the amount of delay that is added to their clock lines. The possible choices are 0.8ns/2.0ns, 7.3ns/0.0ns and 5.8ns/0.0ns. The two latter choices may preclude operation at 102 MHz depending on the ADC block's performance and system clock skews. The first choice has a very small setup time but has a positive hold time, which may create timing difficulties.

Alternatively, implementation in a slightly larger device, the X4013XL, may solve system timing problems. This device supports significantly faster I/O rates, as do the X4036XL and X4062XL. For the X4013XL, a setup and hold time combination of 4.8ns/0.0ns can be selected. This could be an economical implementation alternative, since the X4013's array of 24×24 CLBs is the closest to the X4010's 20×20 array.

5.2.6 Final Comments on the Design

The mapping of the design requires 333 CLBs, including 589 CLB flip-flops. This represents 83.3% of the device's CLBs, and 73.6 % of its CLB flip-flops. The X4010 chip size is therefore a perfect match for this design.

The available X4010E-3 chip was packaged in an 84 pin grid array, for which the number of I/O pins is limited to 61. Given that 32 pins are required for the inputs, and that a clock and reset input signals are also necessary, this left only 27 pins for the outputs. Both the in-phase and quadrature channel outputs are 18 bits wide, so they were truncated to 12 bits for a total number of used pins of 58. The selection of 12 bits was made to simplify debugging, as the output signals fit neatly in a 3-digit hexadecimal representation. The extra 6 bits for both channels are calculated and are available inside the chip, and could be routed outside if a larger device package were selected.

5.3 Polyphase Filter Approach Design with Delayed-Carry Adder Chain

5.3.1 General Overview

The goals of this design were to increase the maximum data rate by implementing the pipelined delayed-carry adder chain described in section 4.6.3, and to demonstrate the viability of this approach. A 2-level segmentation was selected, which reduced the width of the widest adders in the system from 17 to 11 bits, including overhead. Although it would have been possible to segment the adders in three or four levels, it was considered uneconomical to do so. Further, a secondary goal of this design was to implement it in the available X4010E-3 chip and perform hardware tests. The overhead associated with an increase in the number of segmentation levels would have made it impossible to fit the design in this chip.

5.3.2 Design Structure

This design is identical in large-grain structure to the basic polyphase filter approach design. The adder chains were modified to accommodate the delayed carries, as per the example of Figure 4-14, and two modifications were made to the multiplier blocks. The first modification required that the most significant portion of the products be delayed by one clock cycle for proper synchronization in the chain. The second modification entailed ensuring that no additions in the multiplier blocks were wider than 11 bits, since wider additions would defeat the purpose of reducing the width of additions in the adder chain. This forced the "tree" structure for the multiplications by 103, 141 and 232, shown in Figure 5-4, to be replaced with a sequence of pipelined 9-bit adders.

5.3.3 Implementation Considerations and Comparison with Basic Design

Manual placement of the design blocks was again necessary to reduce propagation delays on all critical paths as much as possible. For an 11-bit adder in the X4000E family, the propagation delay is 12.92 ns, including the propagation delay through the operand registers, but excluding the net delays between the operand registers and the adder. This net delay was kept down to 1.83 ns in the worst case, for a minimum resulting clock period of 14.75ns and an equivalent data rate of 67.8 MHz. This represents an improvement in performance of more than 15% when compared to the basic design described previously. The final CLB count was 392, or 98% of the device. The CLB flip-flop count was 742, or 92.8% of the available flip-flops. Table 5-3 below compares the delayed-carry design with the basic one in terms of used flip-flops, CLBs, and performance.

[illegible]	[illegible]	[illegible]	[illegible]
critical component	17-bit-wide adder	11-bit-wide adder	
critical component delay	15.26 ns	12.92 ns	-15.3%
max. routing	1.74 ns	1.83 ns	+5.2%
minimum period	17 ns	14.75 ns	-13.2%
maximum data rate	58.8 MHz	67.8 MHz	+15.3%
CLB count	333/400	392/400	+17.2%
flip-flop count	589/800	742/800	+26.0%

Table 5-3 – Comparison of the Basic and Delayed-Carry Designs

The modest increase in CLBs compared to the flip-flop increase can be explained. In the basic design, unrelated logic is almost never packed in a given CLB. The effect is that the design density is decreased but the CLB count is increased. Reducing the design density significantly facilitates routing, although it somewhat complicates placement. In comparison, the delayed-carry design has a much higher density because it was literally impossible not to pack unrelated logic together in some CLBs. Almost every CLB in the device is used, and "airing out" the placement was not an option. Therefore, a better metric of comparison between the basic and delayed-carry designs is the number of flip-flops used.

As discussed in Chapter 4, the overhead for the delayed-carry chain would be significantly less for a design with wider adders, and the performance improvement would be much greater.

5.4 Polyphase Filter Approach Design with Data Conversion

5.4.1 General Overview

A third design was produced and simulated, but not implemented. It is based on the polyphase approach with delayed-carry. For this design, an additional requirement was that no hardware outside of the FPGA was to be used to perform data de-interleaving or representation format conversion. A "front-end" was therefore added to the previous delayed-carry design and a larger FPGA was selected to support this increased amount of logic.

A number of ADCs were considered for this design. Their first requirement was a minimum sampling rate of 213.3 MS/s. ADCs meeting this requirement normally have ECL outputs, but the FPGA families considered for the implementation require TTL or CMOS inputs. Level translation from ECL to TTL was therefore required, and this limited the maximum data rate between the ADC and the FPGA. It was therefore decided to choose an ADC with two interleaved data outputs. The fundamental difference for this design, when compared to the two previous ones, is that the input data is interleaved in two streams, each coming in at a rate of 106.67 MHz from the ADC.

The selected ADC for this design outputs data encoded with an 8-bit Gray code. Since all filtering operations are done on two's complement data, it was necessary to effect conversion inside the FPGA. In order to keep all processing in the device at the lowest possible level, the two input data streams are split into four at half the data rate prior to conversion.

The device requires only one clock at a quarter of the sampling data rate, 53.3 MHz in this case. All processing is maintained at this rate as per the previous designs. A block diagram of the design is given in Figure 5-6.

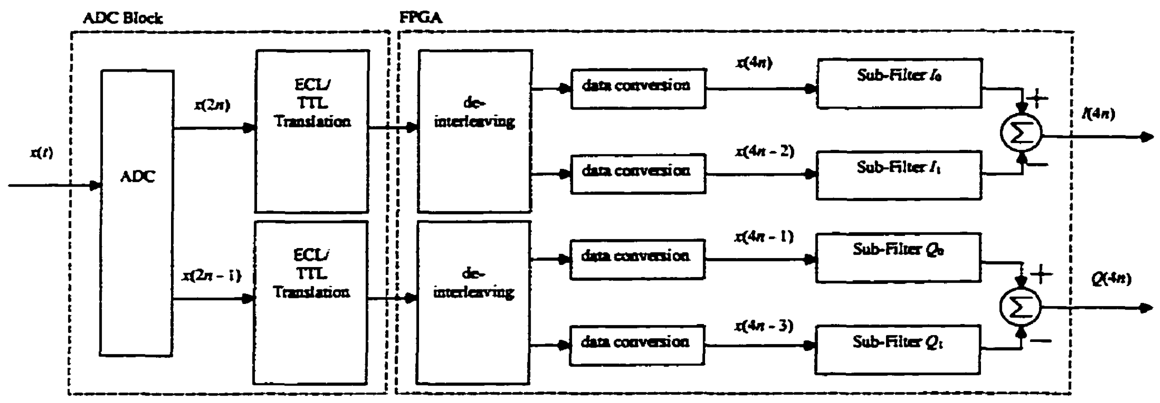

Figure 5-6 - Top Level Block Diagram (Full Design with Data Conversion)

5.4.2 Analog-to-Digital Converter (ADC) Block

The ADC block comprises an Analog-to-Digital Converter and ECL-to-TTL logic level translators for interface to the FPGA.

For this system, a sampling frequency of 213.3 MHz was necessary, with the even and odd samples being output on two different ports at a rate of 106.7 MHz each. The ADC selected is the SPT7750 from Signal Processing Technologies. It is an 8-bit, 500 megasamples per second FLASH ADC. Each ECL-compatible output port has an associated data-ready strobe signal which can be used to control a system clock.

To interface with the FPGA, it is necessary to perform logic level translation. The selected ECL/TTL level translator is the National Semiconductor 100325. Each 100325 can translate 6 signals, so three chips would be required for the 16 bits of ADC output data. One of the two remaining channels can be used to translate the FPGA clock, if necessary.

5.4.3 Data De-Interleaving Process and Timing Requirements

The two de-interleaving blocks each accept a 106.7 MHz data stream from the ADC block and split it into two streams at 53.3 MHz. Only one 53.3 MHz clock is necessary, and it is the same clock as the rest of the system. A block diagram of the in-phase channel de-interleaving block is given in Figure 5-7. The quadrature channel de-interleaving block is identical.

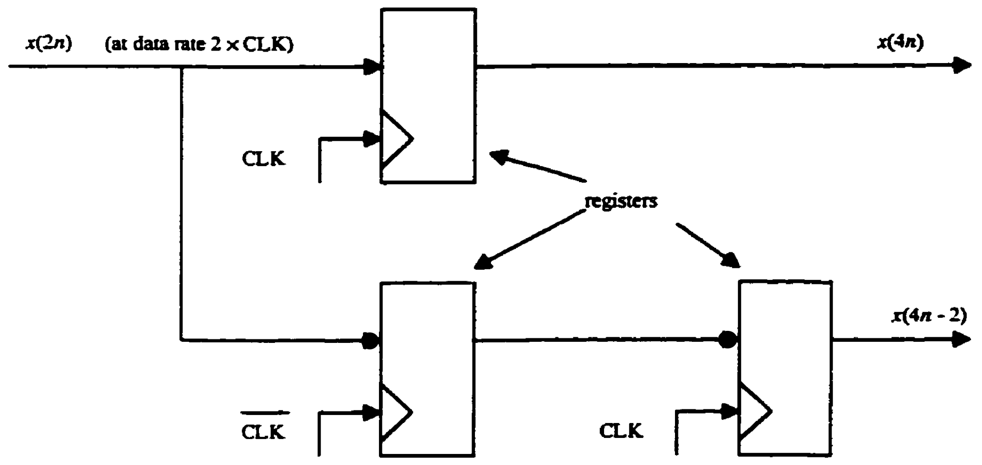

Figure 5-7 - De-Interleaving Block, In-Phase Channel

Three registers are used. In the lower path, the first register's clock is inverted. It therefore latches on data that leads the upper path data by one-half clock period. The second flip-flop synchronizes this data with the upper path's.

For the selected ADC, a difficulty arises in that the data on the two output ports is not synchronized, but delayed by one clock period of the sampling data rate. Since the quadrature demodulator output is clocked at a quarter of the sampling data rate, this would imply a quarter period lag between the I and Q filter outputs, and the requirement for two distinct system clocks. The outputs would also have to be re-synchronized.

Instead, both data streams are de-interleaved from a single clock which is synchronized with the ADC's 'A' data stream. This places an additional timing constraint on the sampling of the 'B' data stream, but it can be met by the selected FPGA. The following timing diagram, shown in Figure 5-8, illustrates the situation.

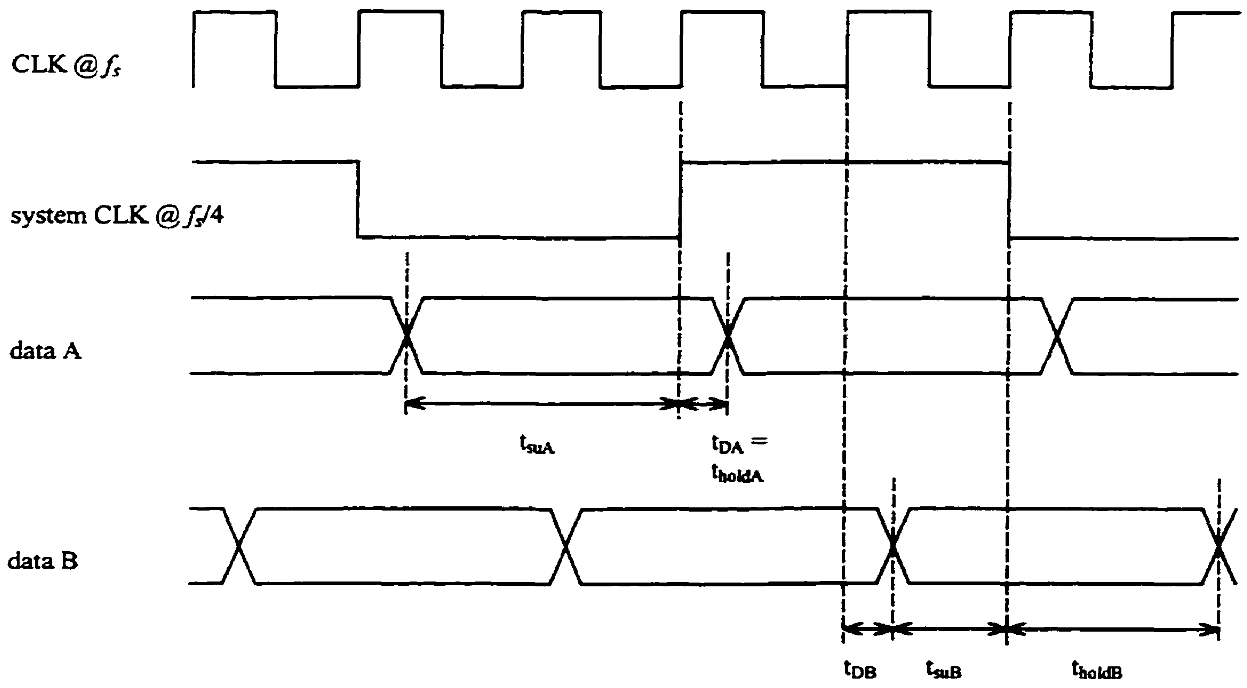

Figure 5-8 - Data De-Interleaving Timing Diagram

The propagation delays t_{DA} and t_{DB} are specified independently between 1.25 and 2.25 ns in the ADC's data sheet. With the sampling interval equal to 4.69 ns, this means that the minimum setup and hold times for the de-interleaving registers are 7.13 and 1.25 ns for data 'A'. For data 'B', the figures are 2.44 and 5.94 ns. These setup and hold times can be met by many of the X4000XL family of chips, including the X4013XL-3. For the in-phase channel, adding a delay in the input flip-flop clock line would ensure setup and hold times equal to 7.4 and 0.0 ns, adequate for data 'A'. For the quadrature channel, switching the same delay element off would lead to setup and hold times equal to 1.2 and 3.2 ns, adequate for data 'B'. Other combinations are possible, and the figures vary with chip family, size, and speed grade. The clock line delay elements are internal to the FPGA and are switched on or off during configuration.

It has been assumed here that the system clock is supplied by a device external to the FPGA. The timing diagram of Figure 5-8 also assumes that the clock signals are passed through the same ECL-to-TTL converters as the data, or through delay elements that produce identical delays. It is further assumed that all other sources of skew outside of the FPGA have been compensated for.

The timing proposed here is only one of many possibilities, although synchronization of signals at such high data rates is not simple. However, the goal here was to demonstrate that FPGAs can accommodate these high data rates if all system considerations are properly accounted for.

5.4.4 Data Conversion Process

The conversion from the ADC's output format to the one used for processing in the quadrature demodulator filers requires three conceptual steps.

First, conversion of gray code to unsigned binary is straightforward. Given that a number is gray coded with *n* bits G_n, the two's complement representation digits B_n are given by:

$$B_i = G_i \oplus G_{i+1} \oplus \ldots \oplus G_{n-1} \qquad \textbf{\textit{(5-3)}}$$

Secondly, the resulting unsigned binary number, ranging here from 0 to 255, must be converted to two's complement, from -128 to +127. This is done by adding a bias of -128, or simply by inverting the Most Significant Bit. Finally, as discussed previously it is necessary to change any occurrences of -128 to -127.

All the conversion process is easily coded in VHDL with a few statements. It is interesting to note that the realization of the converter on the FPGA is fairly slow, in the order of an 8-bit adder, because it requires communication between multiple CLBs. No pipelining was required, however, because this conversion is not in the critical path. This shows again how fast the dedicated carry logic makes ripple carry addition in X4000 series FPGAs.

5.4.5 Final Comments on the Design

The overall design requires 467 CLBs and 796 flip-flops. A 4013 chip, which has 576 CLBs, was therefore selected. In order to meet the I/O timing requirements described here, it was necessary to target the X4000XL family since the X4000E family was too slow. However, the -3 speed grade, the slowest for the 'XL chips, is still fast enough, especially the X4013XL which benefits from special I/O optimization.

The achieved data rate for this design was 77.2 MHz in a X4013XL-3 chip. The smallest package for this chip has 144 pins, and 53 of the 113 IOB are used. Although implementation in a faster chip grade would increase this data rate, the external timing considerations would have to be completely reviewed to ensure that the de-interleaving process would function correctly. Alternatively, the same front-end processing could be kept but the filters changed to those described in the basic design.

5.5 Low-Pass Filter Approach Design

5.5.1 General Overview

A fourth design is described here in general terms. It was not implemented because appropriate filter coefficients were not available. The design is suitable for any case where the decimation factor is 2, so it could accommodate half-band or third-band prototype filters.

The proposed design is made up of the following major blocks: the Analog-to-Digital Converter block, two data conversion blocks, two modulator blocks, the In-Phase Filter Block, and the Quadrature Filter Block. A block diagram is shown in Figure 5-9. The Low-Pass Filter approach described in section 2.3.1 was selected for the system. Therefore, two filters work in parallel on different data. The even samples of the input signal are processed by the in-phase portion of the system, and the odd samples are processed by the other. In each path, a data converter transforms the data from the ADC output format to two's complement representation. A modulator then multiplies the input data with the sequence {1, -1, 1, -1, ... }, effectively inverting the sign of every other sample. The resulting data is passed to the in-phase and quadrature low-pass filters.

The data converters, modulators and filter blocks can be implemented together on a single FPGA.

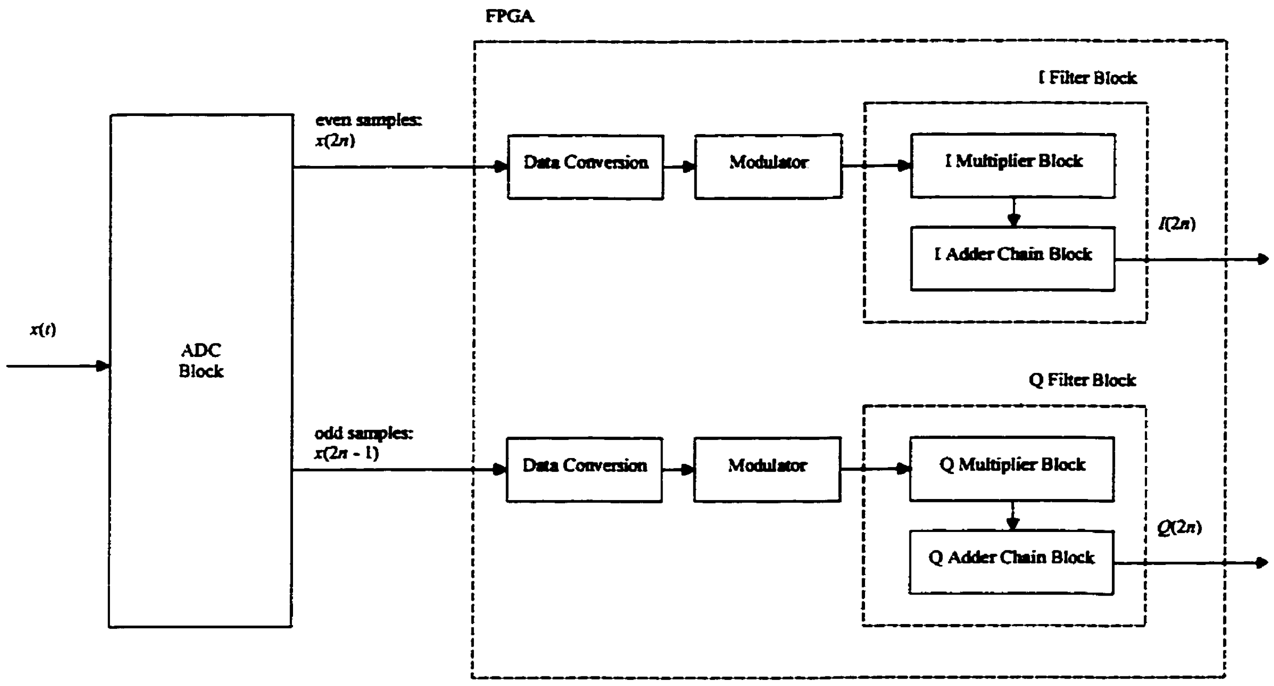

Figure 5-9 - Top-Level System Block Diagram (Low-Pass Filter Approach)

5.5.2 Analog-to-Digital Converter (ADC) Block

The ADC block is identical to the one described in section 5.4.2.

5.5.3 Data Conversion Block

The Data Conversion Block is identical to the one described in section 5.4.4.

5.5.4 Modulator Blocks

The two modulator blocks are identical, although they work on different data with different clock signals. They are fed from the Data Conversion blocks' outputs. The even samples are passed to the in-phase channel modulator and the odd samples are passed to the quadrature channel modulator. Their output is made up of two signals which have the same value except for a sign inversion, "data plus" and "data minus". The two signals are required by the multiplier block. A block diagram of a modulator block is shown in Figure 5-10.

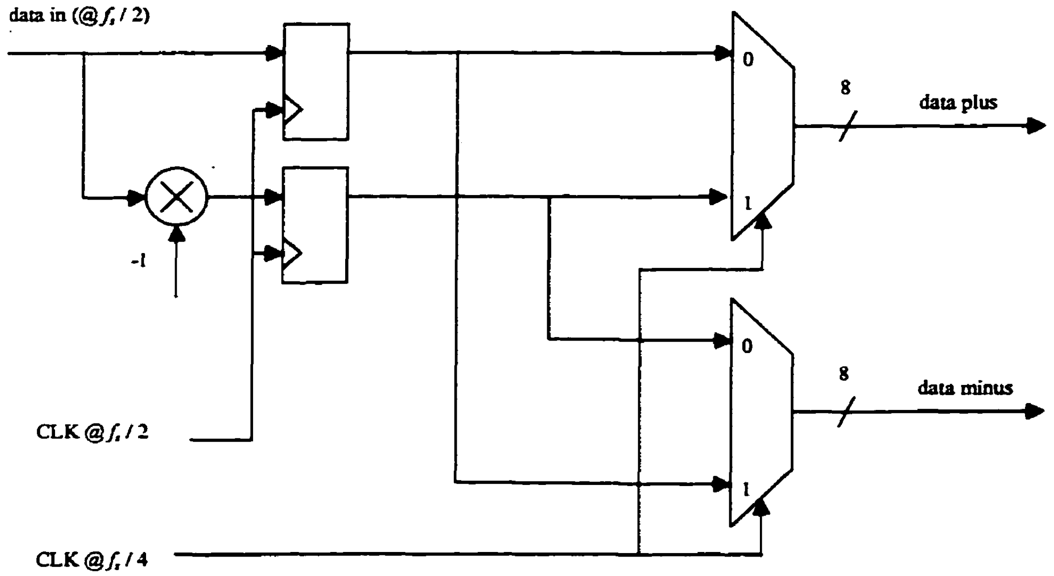

Figure 5-10 - Modulator Block

The second clock signal, with a frequency of $f_s/4$, is used to control two multiplexers. Their outputs therefore alternate between positive and negative versions of the input data.

5.5.5 In-Phase and Quadrature Filter Blocks

The in-phase and quadrature filter blocks have an identical large-grain structure, and differ only due to the different sets of coefficients for the two filters. They are decomposed into two smaller blocks, a multiplier block and an adder chain block, following the transposed form discussed in section 4.4. The structure is shown in Figure 5-11, with the modulator block output.

The multiplier block takes for input the positive and negative version of a number in two's complement representation, using 8 bits. It outputs one product of this number with each of the filter coefficients *h*(*n*). The corresponding products are labeled *Mn*. They are passed to the adder chain.

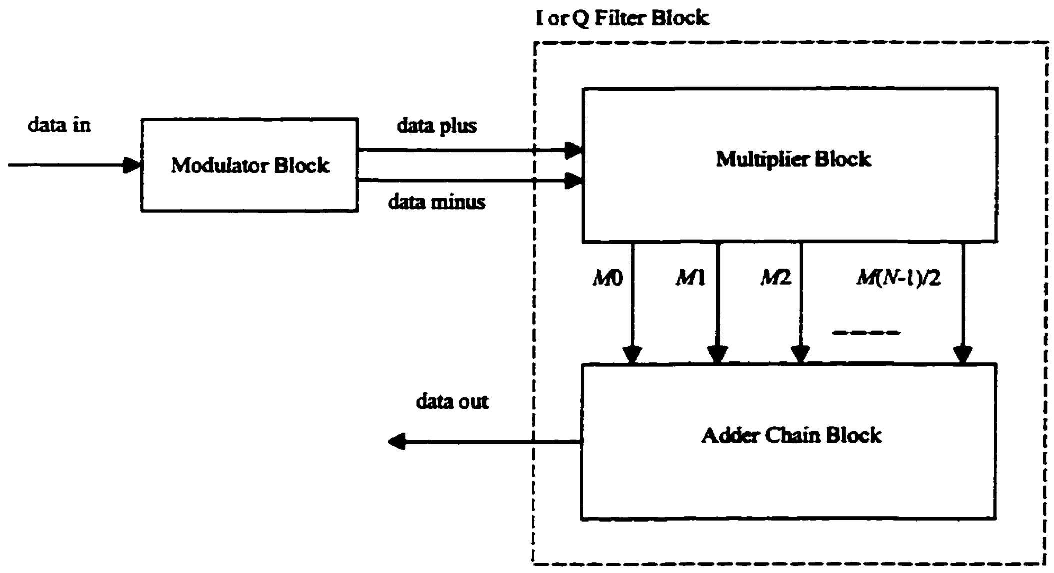

Figure 5-11 - Structure of the Filter Blocks (N even)

Supplying the positive and negative versions of the input data simplifies the design of the multiplier blocks since only additions need to be performed.

The adder chain blocks take for inputs the results of the multiplication of the filter block input data with the different filter coefficients, produced by the multiplier blocks. They produce the filter output, which is also a system output. The delayed-carry adder chain discussed in section 4.6.3 can be implemented in the adder chain blocks to maximize operating speed.

5.5.6 Final Comments

It should be obvious that this design is very similar to either of the polyphase filter designs. The most significant difference is the addition of the modulator blocks. A final subtraction between sub-filters is also unnecessary.

Given the availability of very large FPGAs holding more than 3000 CLBs, very high filter orders could be supported. Additionally, using the delayed-carry chain approach would make it possible to maintain the processing rate at the level attained for the simpler designs.

5.6 VHDL Description Considerations

VHDL was chosen to describe the quadrature demodulator designs instead of schematic entry. There are many reasons to use a Hardware Description Language (HDL) for design entry. Very complex design descriptions manageable by allowing the designers to describe objects at higher levels of abstraction. Compared with schematic entry, they also make it much simpler to modify many parts of a design simultaneously according to a given design parameter, such as a bus width. The consequence of using HDLs, however, is that the designer generally loses some control over the resulting hardware as implemented from the output of synthesis tools. The quadrature demodulator designs described here had stringent timing and performance requirements. It was therefore necessary to use a coding style that would allow as much control as possible to be kept over the resulting design description.

Consequently, it was decided early on in the design process to use a structural VHDL description. This complicates the design process in that the description is closer to one from a schematic entry. However, the design is much simpler to modify, and the advantage over higher-level descriptions (such as behavioral) is that the designer retains much more control over the implemented product. This was deemed more important in this case since high performance was desired.

5.6.1 Building Blocks

Signals are the most basic building block. They involve no processing, but defining them generally ensures that a particular line or connection will be implemented in the FPGA. There are important exceptions. When two signals carry the same information, one is normally optimized away. Similarly, a signal which has no load, inside or outside of the FPGA, is also optimized away by the synthesis or mapping tools. There are cases where such an optimization is not desired. An example would be the routing of a signal to two different blocks physically separated across a large distance in the FPGA. If only one signal driver is used, then the propagation delays can be unacceptably high. In such a case, registering the signal with two distinct register components generally ensures that two drivers will be available, even after optimization.

The next basic building block is the *n*-bit register. Its description must be made with a behavioral description that specifies whether the device latches data on a clock level or transition, and whether this transition is active high or low. Clock enables can be added in the behavioral description, as can specific set or reset values. The following code excerpt is for an *n*-bit register with asynchronous active-low reset which latches the input on the clock's rising edge.

```
process (CLK, reset)
begin
      if reset = '0' then    --asynchronous RESET active LOW
            Q <= (others => '0');
      elsif (CLK'event and CLK='1') then  --CLK rising edge
            Q <= D;
      end if;
end process;
```

The behavioral description of a register element can require a few dozen lines of code. Since a typical register is often reused through a design, it is much more efficient to encode the description only once in a parameterized entity, then to instantiate this entity as necessary later. For the quadrature demodulator designs, four distinct register elements were defined, each with its idiosyncrasies such as a different reset value.

An even more complex building block is the registered adder, which builds on the register building block. Again, many such adder descriptions were required in the designs so that various cases could be accommodated. These cases included a standard adder, a standard subtracter, an adder and a subtracter with input carry, and a specialized subtracter for the delayed-carry adder chain. The following code excerpt is for a standard registered adder with active-low asynchronous reset. The sum is stored on the clock's rising edge. Of note is the special consideration given to sign extension for two's complement addition.

```
process (CLK, reset)

variable tempA : STD_LOGIC_VECTOR (Sumwidth - 1 downto 0);
variable tempB : STD_LOGIC_VECTOR (Sumwidth - 1 downto 0);

begin

        tempA := (others => A(A'left)); -- sign extension
        tempB := (others => B(B'left)); -- sign extension

        tempA(A'left downto 0) := A;
        tempB(B'left downto 0) := B;

        if reset = '0' then    --asynchronous RESET active LOW
                Sum <= (others => '0');
        elsif (CLK'event and CLK='1') then  --CLK rising edge
                Sum <= tempA + tempB;
        end if;

end process;
```

The larger design building blocks, including the sub-filters, filters and data converters, were then put together for the different designs. This approach had the consequence that using a block that had been previously tested greatly reduced the design time.

5.6.2 Optimizing the Hardware Realization from the VHDL Description

It was found that well optimized logic could be obtained from a VHDL description once the behavior of the synthesis tool was understood. Two specific examples will be considered here. The first example concerns the realization of a simple two-operand adder with input carry. The second example deals with the specification of extra registers to reduce logic placement constraints.

A most simple building block is the registered added with input carry, which one may code in VHDL as `Sum = A + B + Cin`. However, the synthesis tool that was used, FPGA Express, incorrectly synthesizes this statement with two distinct two-operand adders instead of a single two-operand adder with input carry. In order to obtain the properly optimized logic, the operation had to be specified by the addition of only two operands. The input carry was therefore appended to the right of the LSB of each input vectors A and B by using a temporary signal. The resulting extra bit in the sum was then discarded. A similar artifice is required to properly obtain an adder with output carry or with overflow indication.

The second example concerns the pipelining of data paths. For all designs described in this chapter, two extra stages of pipelining had to be specifically declared at the input data ports. This strategy ensured that the synthesis tool used the IOB flip-flops for the first pipelining stage. Since the physical distance on the FPGA between a given input pin and the location where the data is first processed may be significant, the second level of pipelining is implemented in any CLB that is preferably closer to the data processing location than to the input pin. The interconnect between the pipeline IOB and CLB may be large and may in fact cross the whole FPGA from one side to the other, but since no processing is done the propagation delay is much lower than the delay on the critical paths. Not adding these two pipeline levels generally constrained the placement of the input pins so much that timing requirements could not be met. A similar strategy was followed for the output pins.

5.6.3 Automated Structural VHDL Code Generation

The quadrature demodulator designs described in this chapter are directly dependent on a few design parameters. These parameters include the input data format and bus width, the filter length, the selected filter architecture, and the value of the filter coefficients. For example, the description of the filter adder chain can be fully automated. First, following the equations given in Appendix A, an adder chain bus width analysis is performed for the given set of filter coefficients. The resulting adder and register widths then completely specify the bounds of each signal in the adder chain, and signal declaration statements can then be automatically generated. Similarly, interconnection statements can be automatically generated based on the filter order.

Such an automated process was investigated and developed at a rudimentary level for the quadrature demodulator designs. Many programming languages could have been used, but Microsoft Excel was selected due to its highly visual interface, ease of use and powerful text processing capabilities. A worksheet was developed and successfully used for the filter bus width analysis, and from this analysis' results signal declarations were generated. The main advantage was that modification of one of the fundamental filter parameters allowed the design description to be automatically modified. Further, it was possible to generate basic VHDL descriptions for all sub-filters from the same worksheet by simply modifying the filter coefficients.

Using this automated process, a designer could easily produce many digital filter designs based on greatly varying sets of coefficients in very little time. If timing requirements are not critical, then little extra design work is involved. However, if optimized designs are desired, manual editing was essential when going from one design to another. As a case in point, the very fine grain multiplier implementation optimizations described in section 5.2.2 were not automatically generated.

In order to obtain such fine grain optimization from a few simple design parameters, two routes present themselves. One would be a dramatic improvement of the synthesis tools, especially the provision for automatic inclusion of pipeline stages in a design. This approach would be advantageous because it could be applied to a wider range of designs. The other route is the one proposed here, where a tool would be optimized to generate structural VHDL descriptions for a specific design or class of designs. However, the basic approach that was taken here would require significantly more sophistication to produce the level of optimization attained through human intervention.

5.7 Summary

In this chapter, four quadrature demodulator designs were described. The first three designs are appropriate for the decimation-by-four case. Each one meets a different set of performance or interfacing requirements. The fourth design proposed would be appropriate for the decimation-by-two case, whether a half- or third-band prototype filter is used. Specific high-speed interfacing considerations were described. Together, the four design descriptions demonstrate the viability of FPGAs for high-performance quadrature demodulation. Finally, specific remarks concerning the VHDL description of the designs were made.

The code for the three implemented designs will be included in a technical report to be submitted at a later date.

Chapter 6

Design Verification and Testing

6.1 Introduction

In this chapter, the verification and testing of the quadrature demodulator designs are described. The verification and testing strategy is first outlined, then the test vectors used to stimulate the design are defined. The approach taken for the functional verification of the VHDL code is then explained, followed by a description of the applicability of the design realization process to design verification. Timing analysis is then discussed, and a proposed hardware testing setup is outlined.

6.2 Strategy

Design verification first entailed confirming the correctness of the VHDL description through functional simulation testing. Since this description was modular, each building block was tested independently. Major blocks made from building blocks were produced and then tested, then the correct operation of the overall design was confirmed. A second major step of design verification concerned the successful synthesis of the VHDL code, then the mapping, placing and routing of the synthesized logic in a selected FPGA device. Finally, a timing analysis of the placed and routed design confirmed whether timing requirements were met. Any problem in one of these three steps generally required a design modification and the beginning of a new verification loop.

Design testing consisted of confirming that the implemented design in a FPGA met functional and timing requirements. The designs were downloaded to a FPGA, test stimuli were applied, and output vectors were compared to expected results.

Testing of the filter characteristics per se was not done in the present research, since the goal of the produced designs was to implement quadrature demodulators based on a prototype filter with given filter coefficients. The ultimate goal of the design verification and testing process was therefore to confirm that correct computations were performed given a set of input vectors. This made this process robust and reliable, and, due to the nature of the processing, the selection of test vectors was simplified.

Testing of the FPGA device for the full range of operating conditions was not considered, since these characteristics are available from the device manufacturer.

6.3 Test Vector Selection

For both functional and device testing, the selection of an appropriate set of test vectors was crucial.

6.3.1 Fundamental Tests

The first step of verification consisted of ensuring that test vectors were received by the system, and that system outputs could be monitored. In these basic tests, the proper operation of the reset signals was also confirmed. A few randomly selected test vectors were also chosen for an initial performance assessment.

6.3.2 Impulse Response Test

This vector consists of a single pulse of unit time duration. This is a standard test used to verify digital filters. After a system reset, or after a "long" stream of zeros was input to the digital filter, the input is set to the value 'one' for one clock cycle, then set to 'zero' afterwards. The expected output is made up of the filter coefficients in sequence, one for every clock cycle. The number of clock cycles between the non-zero input and the first output corresponds to the number of pipelined stages in the filter.

This test vector is both simple and effective, and it gives a quick indication of the correctness of the filter description. However, the fault coverage of this test is limited, and its use is therefore restricted to the early design stages.

A simple variation on this test consists of extending the length of the pulse. The expected output is then a sequence of sums of filter coefficients, and the number of coefficients in each sum depends on the length of the pulse. This test variation was not used to test individual filters since its effectiveness is limited. However, it could be useful to verify the operation of selected adders in a transposed form filter's adder chain.

While this test is primarily aimed at single filters, it can also by used to test the whole quadrature demodulator design. However, it is then necessary to extend the length of the pulse, or to apply pulses of unit duration at precise times so that only one sub-filter is stimulated.

6.3.3 Extreme Outputs Test

The goal of this test is to verify that the system is able to accommodate the greatest positive and negative intermediate and final results. The input vector sequence consists of a vector whose length is the same as that of the set of filter coefficients and whose elements consist of either the extreme positive or negative values of the input data. The input vector is selected so that the signs of its elements correspond to the signs of the corresponding filter coefficients.

This test confirms whether the analysis described in Appendix A was done correctly and whether the results were correctly implemented by replicating the analysis process during the test.

6.3.4 Pseudo-Random Sequence Test

This test is more involved but its coverage is also greater. A pseudo-random sequence is applied to the input of the system under test, and the actual output sequence is compared with the expected output sequence. An exact match of these two sequences indicates a successful test. This kind of test is well suited for integrating test functionality into the system.

This test is well adapted to the present system, since its function is to process a sequence of input numbers. A suitably long input data sequence of pseudo-random numbers can give a high level of confidence that the appropriate arithmetic operations are performed. It must be noted, again, that the goal here is not to test the implemented filter performance, but rather to test that a particular filter has been correctly implemented in hardware.

Three methods were used to generate pseudo-random sequences and to calculate the expected system outputs with them as stimuli. Microsoft Excel worksheets were first used for basic tests with relatively short sequences, as were MATLAB script files. The random numbers were generated from Excel's or MATLAB's internal pseudo-random sequence generators. Secondly, a MATLAB script was again used, but with a customized random sequence generator scratch-built from a Linear Feedback Shift Register (LFSR) simulator. Thirdly, for hardware tests, a LFSR generator design was coded in VHDL, then synthesized, mapped, placed, routed and downloaded to an FPGA that was used to stimulate the Device Under Test (DUT).

6.3.5 De-Interleaving and Data Conversion Tests

The tests described in the previous sections were aimed at verifying the digital filter operations. It was equally important to test the design's front end.

For the de-interleaving process, timing diagrams were carefully plotted based on the ADC's specifications. Test vectors were then constructed to reproduce the ADC's behavior, and applied to the DUT. The four resulting data streams were then compared to the expected output, and so was the overall system output.

For the data conversion process, it was possible to devise a complete test since only 256 cases had to be covered.

6.4 Functional Verification of VHDL Code

Coding of the three polyphase filter based designs of Chapter 5 was divided according to the designs' major blocks. In all cases, the code for the four sub-filter blocks was tested independently. The in-phase and quadrature channel sub-filter pairs were then tested together. The data de-interleaving and conversion blocks were also tested independently. Finally, the whole design was combined and tested as one unit.

Code testing was an integral part of the design process. This way, errors were quickly found, identified and corrected. The modularity of the design enhanced testability at every step. The test vectors described in section 6.3 were used in each case. For the pseudo-random sequence test, a test length of 20 000 vectors was used.

6.5 Synthesis, Mapping, Placing and Routing Verification

After successful functional testing of the VHDL code, the next steps in the design process led to implementation in a FPGA device. Other than hardware testing, no further testing was done via the application of input stimuli and the comparison of output data with calculated values. However, an important part of the verification process was to ensure that the design could be implemented in the target device.

The first step consisted of the synthesis of the VHDL code into a netlist describing, among other things, the interconnections between building blocks such as pins, buffers, gates and adders. In some cases, errors were found in the design description because certain VHDL constructs could not be synthesized. For example, synthesis tools will not allow a signal to have multiple drivers. Other errors involved different VHDL standards used by the design description environment and the synthesis tool. Xilinx' *FPGA Express* synthesis tool was used.

Mapping of the synthesized netlist was the following step. This consisted of translating the netlist into functions implemented by Configurable Logic Blocks (CLBs) of the specified family of devices. The tool used was Xilinx' *MAP*. A possible design error reported at this step was the selection of a device too small to accommodate the design's logic. In such a case, the description of the design was modified to reduce the amount of logic required, or a larger device was selected.

Placing of the mapped CLBs on a target device and routing of interconnections between them was the next step, and this was done by a combination of manual intervention and the action of Xilinx' automated Place-and-Route tool *PAR*. The main error at this step was the impossibility to meet a timing requirement. Many problems could lead to this error. The most fundamental one was the existence of logic function whose delay was greater than the minimum clock period specified. In this case, it was necessary to break the logic into pipelined blocks, which required a modification of the VHDL code.

The most common problem, however, was the impossibility to route interconnections given a certain block placement. There were three main approaches to alleviate this problem. The first one was to insert an additional level of pipelining. This gave greater flexibility in bringing a signal from one location to another. Reproducing a signal so that it was generated from two distinct CLBs was another possibility, for cases where a signal had a very large fanout. Alternatively, manual placement of CLBs such that signals had as short a path as possible to travel generally solved the problem.

The visual inspection of placed CLBs was another method to verify the correct description of the design. The *Floorplanner* tool from Xilinx allows this to be done, in addition to allowing location constraints to be manually specified. By tracing a given signal across the FPGA, it was possible to verify that intended building blocks had correctly been synthesized from the VHDL description. This method also allowed a detailed study of the synthesis tool behavior to be done. In general, the errors found this way would have affected the performance of a design, not its functional correctness. The example of the two-operand adder with carry-in was given in 5.6.2.

6.6 Timing Analysis and Identification of Critical Paths

The EDA tools used for the quadrature demodulator designs allow the specification of timing requirements as part of the synthesis options, or through a user constraint file prior to mapping. The specifications take the form of maximum propagation delays between groups of selected logic, such as between any two flip-flops in a data path, or from the clock pin to any flip-flop in the device.

The place-and-route tool uses the timing specifications to guide its placement and routing choices, and will report on whether the specifications have been met or not. The Xilinx Timing Analyzer can then be used on the resulting design file to analyze each data path in turn. All paths not meeting the constraint can then be easily identified.

When timing constraints were not met, two main options were available. First, if a logic block's delay alone was longer than the timing constraint, this block was broken up into a number of pipelined sub-blocks. Similarly, if the timing constraint could not be met due to the excessive fanout on a given signal, this signal had to be divided among multiple drivers in the VHDL description.

Whenever a timing constraint was not met, however, it was because of the limitations of the automated placement and routing tool. In such a case, it was necessary to manually constrain the placement of communicating CLBs so that they would be physically close to each other.

6.7 Hardware Testing

The selection of FPGAs as a target technology meant that it was possible to realize and test the designs in a reasonable time frame. Various hardware test setups available at the Royal Military College, the Communications Research Center, and at the Canadian Microelectronics Corporation were considered. Testing at the CRC or CMC would have involved the use of the Integrated Measurement System (IMS) test fixture. This in turn would have required the wiring of a test board with the appropriate FPGA device. Since time was limited, an alternative avenue was selected.

The availability of FPGA demonstration boards with X4010E-3 chips at RMC motivated the design of a test setup shown in Figure 6-1. The stimulator used to test the DUT is constructed from a separate FPGA of the same family and speed grade. It implements a Linear Feedback Shift Register that is initialized to a known value, and generates a known pseudo-random output sequence to implement the test described in section 6.3.4. A logic analyzer collects the DUT's output data and stores it for off-line comparison with the expected output sequence. If the two sequences are identical, the test is a success.

A separate FPGA demonstration board with a X4003E-3 chip was used for the stimulator. The selection of the same device family and speed grade ensures a compatible interface between the stimulator and DUT and that the stimulator will support testing at the DUT's highest clock frequency. However, actual hardware testing was limited because of a lack of an adequate clock generator that could support frequencies in excess of 20 MHz. Various sources were investigated with no success. Still, functional testing was successful at 20 MHz, and the test setup would be simple to reproduce once an adequate clock generator is available.

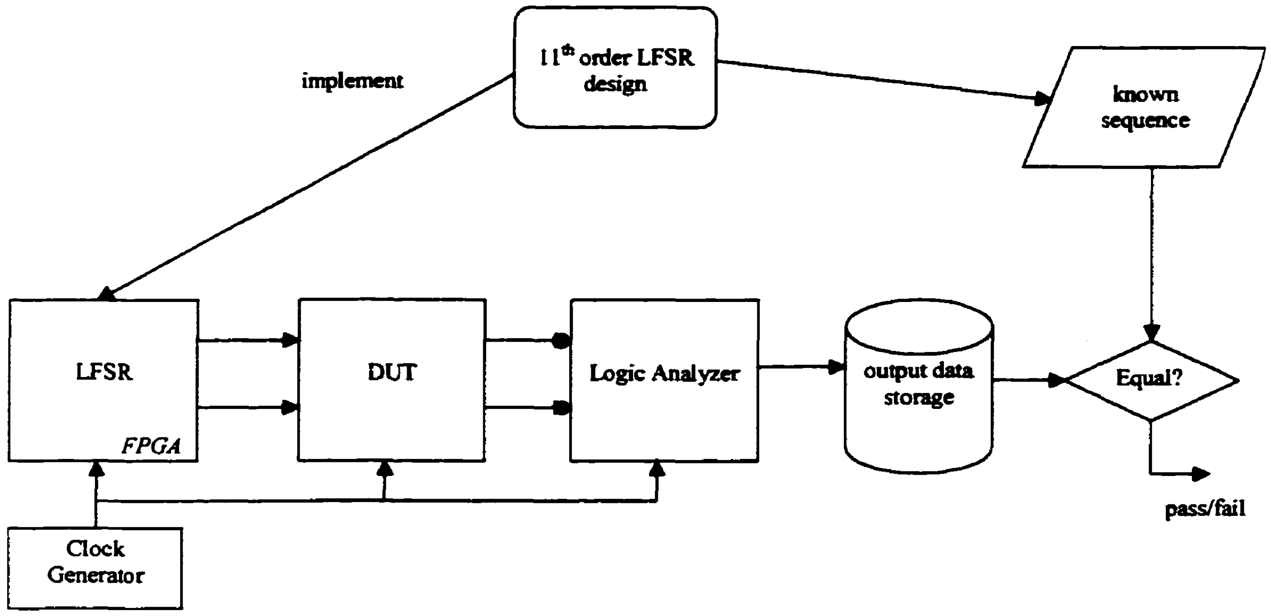

Figure 6-1 - Hardware Test Setup

6.8 Summary

In this chapter, the issues of design verification and testing were discussed. The verification and testing strategy was outlined, as were specific considerations relating to test vector selection and the integration of the verification and design implementation processes. A proposed FPGA hardware test setup was presented.

Chapter 7

Conclusions and Recommendations

7.1 Conclusions

This thesis has addressed issues relevant to the design and implementation of wide-band digital quadrature demodulators in Field-Programmable Gate Arrays. Fundamental principles for the digital realization of quadrature demodulators were discussed and different theoretical approaches were first presented.

Due to their ubiquity in digital filter designs, the problem of implementing constant-coefficient multipliers in FPGAs was given special attention. The popular Look-Up Table approach for multiplication greatly simplifies the design process, but requires significantly more chip resources and somewhat increases critical path latency. The viability of optimizing coefficients by reducing the number of signed digits required to represent them was confirmed for FPGA implementations as well as for other ASIC technologies.

The selection of a filter architecture that maps well to FPGA Configurable Logic Blocks is a major issue. In general, the transposed form architecture was found to increase design density when compared to the direct form architecture. It was also suggested that future quadrature demodulation filter designs should include coefficient redundancy as a hardware cost optimization criterion. A technique that exploits the speed of the X4000's dedicated carry logic, the delayed-carry adder chain, was proposed to keep the system's critical path delay constant regardless of the filter order.

Four quadrature demodulation design examples meeting different sets of specifications were used to demonstrate the viability of using FPGAs for wide-band digital quadrature demodulators. Processing and interfacing rates above 100 MHz were demonstrated in the faster Xilinx X4000 FPGA families. The designs were limited more by the speed of the I/Os than by the achievable internal rates. This implies that it may be desirable to de-interleave data outside of the FPGA.

Very high device utilization was obtained; two of the designs use 85% and 98% of the device's CLBs while maintaining near-maximal data rates. This resulted from the largely local nature of the intercommunications inherent to the selected system architecture, and by the careful imposition of manual constraints on the placement of CLBs.

The issue of providing designer control over the implementation of a design from a VHDL description was considered, and examples of coding style to enhance this control were given. An automated digital filter design process was briefly investigated and was found to decrease design time. Suggestions were given on how to improve the process.

Finally, the issues of design verification and testing were discussed. In addition to functional testing, the merging of the design verification process into the design implementation process was considered, and cases where the design's optimization could be improved were presented. A hardware test setup was proposed utilizing an FPGA based stimulator separate from the Device Under Test.

7.2 Recommendations for Future Work

Many promising avenues for further research have been identified.

A detailed analysis of the delayed-carry adder chain should be undertaken, and its applicability to other ASIC technologies should be investigated. The case was made that utilizing an FPGA CLB solely for its flip-flops wasted valuable resources. Such is not the case for custom or gate array ASICs, and the overhead costs presented here would probably be significantly smaller. A search for such an analysis was made with no success.

It may be possible to devise analytical formulae to describe the implementation cost of a particular digital filter based on the value of its coefficients, the desired filter architecture, and the target technology. Such results would facilitate the evaluation of different coefficient sets when comparing possible quadrature demodulator designs. Additionally, it is believed that the issue of coefficient redundancy has not been exploited for the design of quadrature demodulator filters in general, or for FPGA implementations in particular. This would therefore be a promising research area.

A quantitative analysis of power consumption by the filter architectures studied here should be performed to determine the validity of the quantitative assessments that were made. Again, specific considerations should be given to FPGA-relevant issues, such as the use of low supply voltage FPGAs.

Variations on the proposed designs are possible. Prime candidates include the duplicated polyphase filter approach, for which the data-interleaving process would be more complicated, and the low-pass approach with a third-band prototype filter. For the existing designs, the inclusion of Built-In Self Test (BIST) functionality may prove useful. The development of a VHDL description of BIST circuitry that could be easily included to any quadrature demodulator design would facilitate the implementation of this design enhancement. An interesting feature for many FPGAs is the availability of an internal clock that could be used to run the BIST circuitry completely independently from the outside. Wave Pipelining [27] should be investigated, as it could push FPGA performance beyond what conventional synchronous designs can accomplish.

Novel FPGA architectures with increased CLB performance and resources will likely be available in the near future. While the direct re-implementation of the existing designs is possible, the full exploitation of advances in FPGA devices may require new implementation approaches.

Finally, much more remains to be done with Electronic Design Automation tools. The automation of VHDL design description for digital filter designs was briefly mentioned, and is seen as a promising area of research. Alternatively, the development of a synthesis tool with automatic pipelining inclusion should be (and may in fact be) a major area of development for the EDA tool industry. Significant improvements remain to be made with the automatic Placement and Routing Tools, especially for designs with a regular block structure such as digital filters.

References

[1] B. Von Herzen, “Signal Processing at 250 MHz using High-Performance FPGAs”, *IEEE Transactions on VLSI Systems*, Vol. 6, No. 2, June 1998.

[2] J. Lee, "Effects of Imbalances and DC Offsets on I/Q Demodulators", DREO Report No. 1148, Dec. 1992.

[3] A.V. Oppenheim and R.W. Schafer, *Digital Signal Processing*, Prentice-Hall, 1975.

[4] R.E. Crochiere and L.R. Rabiner, *Multirate Digital Signal Processing*, Prentice-Hall, 1983.

[5] G. Zhang, D. Al-Khalili, R. Inkol, Saper, “A Novel Approach to the Design of I/Q Demodulation Filters”, *IEE Proceedings on Vision, Image and Signal Processing*, Vol. 141, No. 3, pp. 154-160, June 94.

[6] R. Inkol, L. Désormeaux, and V. Szwarc, “Proposed Design Improvements for the Coherent Processor ASIC”, DREO Technical Memorandum 13/94, October 1994.

[7] M.G. Bellanger, J.L. Daguet and G.P. Lepagnol, "Interpolation, Extrapolation, and Reduction of Computation Speed in Digital Filters", *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol. 22, No. 4, August 1974.

[8] M.G. Bellanger, G. Bonnerot and M. Coudreuse, "Digital Filtering by Polyphase Network: Application to Sample-Rate Alteration and Filter Banks", *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol. 24, No. 2, August 1976.

[9] Inkol, R. Duplicated Polyphase Architecture Block Diagram. Private communication.

[10] R. J. Inkol. "Novel FIR Filter Designs for Digital Quadrature Demodulation", to appear in Proceedings of Canadian Conference on Electrical and Computer Engineering, May 1999.

[11] V. Anastassopoulos, T. Deliyannis, , "Efficient Implementation of N^{th}-band FIR Filters Based on a Simple Window Method", *IEE Proceedings*, Vol. 137, Pt. G, No. 4, pp. 302-308, Aug. 1990.

[12] V.C. Hamacher, Z.G. Vranesic, and S.G. Zaky, *Computer Organization*, 4th Ed. McGraw-Hill, 1996.

[13] G.W. Reitwiesner, "*Binary Arithmetic*", Advances in Computers, Vol. 1, F.L. Alt, Ed. Academic Press, 1960.

[14] D.R. Bull and D.H. Horrocks, "Primitive Operator Digital Filters", *IEE Proceedings-G*, Vol. 138, No. 3, pp. 401-412, June 1991.

[15] A.G. Dempster and M.D. Macleod, "Constant Integer Multiplication Using Minimum Adders", *IEE Proceedings-Circuits, Devices, Systems*, Vol. 141, No.5, pp. 407-413, Oct. 1994.

[16] D. Li, "Minimum Number of Adders for Implementing a Multiplier and Its Application to the Design of Multiplierless Digital Filters", *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, Vol. 42, No. 7, pp. 453-460, July 1995.

[17] A.G. Dempster and M.D. Macleod, "Use of Minimum-Adder Multiplier Blocks in FIR Digital Filters", *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, Vol. 42, No. 9, pp. 569-577, Sep. 1995.

[18] K. Chapman, "Constant Coefficient Multipliers for the XC4000E", Xilinx Application Note 054, version 1.1, 11 December 1996.

[19] B. New, "Using the Dedicated Carry Logic in XC4000E", Xilinx Application Note 013, Version 2.0, 4 July 1996.

[20] L.C. Ludeman, *Fundamentals of Digital Signal Processing*, John Wiley & Sons, 1986.

[21] L. Désormeaux, Communications Research Center, private communication, June 10th, 1998.

[22] "Speed Metrics for High-Performance FPGAs". Xilinx Application Brief XBRF015, version 1.0, November 1997.

[23] L. Dadda and V. Piuri, "Pipelined Adders", IEEE Transactions on Computers, Vol. 45, No. 3, March 1996, pp. 348-356.

[24] M.O. Esonu and D. Al-Khalili, "Design and VLSI Implementation of a Coherent Processor ASIC", Department of Electrical and Computer Engineering, Royal Military College of Canada, March 1994.

[25] H. Samueli, "The Design of Multiplierless Digital Data Transmission Filters with Powers-Of-two Coefficients", *Proceedings of IEEE Telecommunications Symposium*, Sep. 1990.

[26] R. J. Inkol, R. Clouston, M. Herzig and R. H. Saper, "A New Approach to the Design of Multiplierless FIR Digital Filters for Quadrature Demodulation", Proceedings of Canadian Conference on Electrical and Computer Engineering, May 1996.

[27] E.I. Boemo, S. López-Buedo and J.M. Meneses, "Some Experiments About Wave Pipelining on FPGAs", *IEEE Transactions on VLSI Systems*, Vol. 6, No. 2, June 1998.

Appendix A

Filter Bus Width Analysis

A filter bus width analysis must be performed as part of the design of a digital filter. The analysis described here assumes that the filter is implemented following the transposed form. The aim of the analysis is to calculate the number of bits required in the adders and registers of the adder chain to prevent overflow.

To obtain this number of bits, the greatest possible positive and negative sums must be calculated for every stage in the chain. The greatest number of bits required to represent either sum is then the bus width for this stage's adder and register.

Each adder in the adder chain has two inputs. The first one is the partial sum up to this point, coming from the previous adder in the chain, and the second input is the result of the multiplication of the present input data with a filter coefficient.

Let $S_{pos}(i)$ and $S_{neg}(i)$ be the greatest positive and negative sums that can exist at the i^{th} adder stage. Let $M_{pos}(i)$ and $M_{neg}(i)$ be the greatest positive and negative results that can come from the i^{th} multiplier, whose coefficient is $h(i)$. Note that in either case, i actually runs from N down to 0, where N is the filter order, to follow the order of the filter coefficients. $S_{pos}(i)$ is given by:

$$S_{pos}(i)=\begin{Bmatrix}0 & ,i=N\\ S_{pos}(i+1)+M_{pos}(i) & ,N-1\geq i\geq 0\end{Bmatrix}$$
$$=\sum_{k=N-1}^{k=i}M_{pos}(k) \qquad \textit{(A-1)}$$

and $S_{neg}(i)$ is given by

$$S_{neg}(i)=\begin{Bmatrix}0 & ,i=N\\ S_{neg}(i+1)+M_{neg}(i) & ,N-1\geq i\geq 0\end{Bmatrix}$$
$$=\sum_{k=N-1}^{k=i}M_{neg}(k) \qquad \textit{(A-2)}$$

The values of $M_{pos}(i)$ and $M_{neg}(i)$ depend on the value of the coefficient $h(i)$ and on the greatest positive or negative values that the input data can take. This value depends on the selected number representation and on the sign of $h(i)$. For two's complement representation, the greatest positive and negative numbers that can be represented with n bits were given in Table 3-1 as $+(2^{n-1} - 1)$ and $-(2^{n-1})$. For a negative coefficient, the greatest positive multiplier result is obtained from taking the greatest negative input number, and the greatest negative multiplier result is obtained from taking the greatest positive input number. The situation is reversed for a positive coefficient. The expression for $M_{pos}(i)$ is therefore:

$$M_{pos}(i)=\begin{Bmatrix}h(i)\times(2^{n-1}-1) & ,h(i)\geq 0\\ h(i)\times(-2^{n-1}) & ,h(i)<0\end{Bmatrix}$$
$$=|h(i)|\times(2^{n-1}-\frac{\text{sgn}[h(i)]+1}{2}) \qquad \textit{(A-3)}$$

where:

$$\text{sgn}[x]=\begin{Bmatrix}1 & ,x\geq 0\\ -1 & ,x<0\end{Bmatrix} \qquad \textit{(A-4)}$$

Similarly, the expression for $M_{neg}(i)$ is given by:

$$M_{neg}(i)=\begin{Bmatrix} h(i)\times(-2^{n-1}) & ,h(i)\geq 0 \\ h(i)\times(2^{n-1}-1) & ,h(i)<0 \end{Bmatrix}$$
$$=-|h(i)|\times(2^{n-1}+\frac{\text{sgn}[h(i)]-1}{2}) \qquad \textbf{\textit{(A-5)}}$$

Substituting equation (A-3) into equation (A-1) gives the expression for $S_{pos}(i)$:

$$S_{pos}(i)=\sum_{k=N-1}^{k=i}|h(k)|\times(2^{n-1}-\frac{\text{sgn}[h(k)]+1}{2}) \qquad \textbf{\textit{(A-6)}}$$

and substituting equation (A-5) into equation (A-2) gives the expression for $S_{neg}(i)$:

$$S_{neg}(i)=\sum_{k=N-1}^{k=i}-|h(k)|\times(2^{n-1}+\frac{\text{sgn}[h(k)]-1}{2}) \qquad \textbf{\textit{(A-7)}}$$

From Table 3-1, the number of bits b required to represent a given value V in two's complement can be calculated as:

$$b=\begin{Bmatrix} \lceil \log_2(V+1)+1 \rceil & ,V\geq 0 \\ \lceil \log_2(|V|)+1 \rceil & ,V<0 \end{Bmatrix} \qquad \textbf{\textit{(A-8)}}$$

Therefore, the number of bits $b(i)$ required to represent any sum at stage i in the adder chain is equal to:

$$b(i)=\max\{\lceil \log_2(S_{pos}(i)+1)+1 \rceil, \lceil \log_2(-S_{neg}(i))+1 \rceil\} \qquad \textbf{\textit{(A-9)}}$$

From this expression, a measure of the complexity of the adder chain can be found. The number of registered bits R_b in the adder chain is equal to the sum of every $b(i)$:

$$R_b=\sum_{i=0}^{i=N-1}b(i) \qquad \textbf{\textit{(A-10)}}$$

Appendix B

Characterization of Ripple-Carry Adders in Xilinx FPGAs

Detailed simulations were performed with the implementation of ripple-carry adders in three Xilinx FPGA families, for all available speed grades. These simulations were necessary because the Xilinx databooks do not provide timing information that is as accurate as the one obtained from the Timing Analyzer tool.

In all cases, the adders were described with sufficient levels of pipelining in order to properly isolate them from IOB performance and chip size considerations. Figures were obtained for the latency of addition for various adder sizes, from 4 to 64. The targeted FPGA families were the X4000E, X4000XL, and X4000XLA.

In all cases, the latency of the adder itself was considered, as was the delay of the registers supplying the adder operands. The routing delay from these registers to the adder depend on their relative placement and on the amount of routing resources used in this particular area, and they were therefore not included in the figures obtained.

As expected, it was found that the latency of ripple carry addition is generally linear with the width of the adder. However, some peculiar characteristics were also discovered. For the 'E' and 'XLA' families, adders of width n and $n + 1$ have the same latency, for n even. This is explained by the fact that in both cases the adders fit in the same number of

CLBs. While the mapping is identical for the 'XL' family, there is a regular latency increase for all width increases, for the –3, -2 and –1 speed grades.

The behavior of the –08 and –09 speed grades for the 'XL' family is even more peculiar, however, as the latency is found to be non-monotonically increasing. For *n* even, the latency is smaller for an adder of width *n* than it is for an adder of width $n - 1$.

Another conclusion from these simulations is that the rate of increase in adder latency, between narrow and wide adders, is not constant and depends on the FPGA family and speed grade. It was found that the faster chips had a lower rate of latency increase with adder width.

The following graph shows the adder latencies for different FPGA families and speed grades.

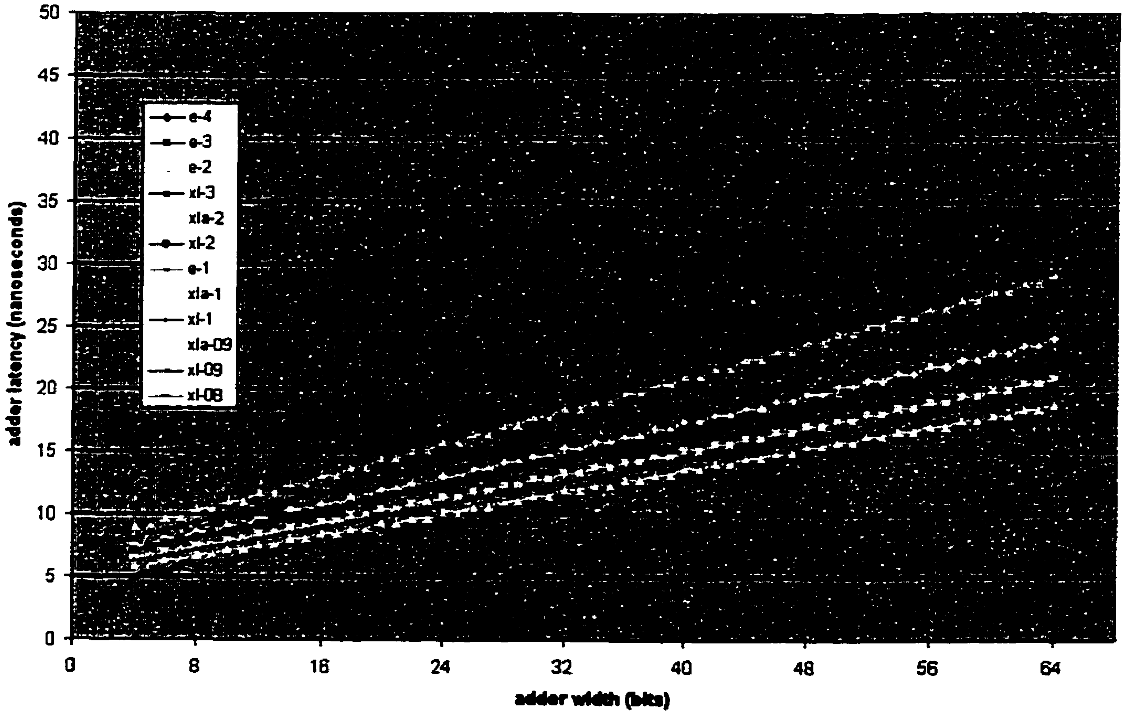

Figure B-1 Latencies for X4000 Adders

The following table gives an approximation of the rate of increase in latency with respect to adder width for different FPGA families and speed grades.

Speed [illegible]	[illegible]	[illegible]	[illegible]
-4	0.521	n/a	n/a
-3	0.390	0.293	n/a
-2	0.335	0.259	0.276
-1	0.264	0.225	0.240
-09	n/a	0.188	0.216
-08	n/a	0.167	n/a

Table B-1 Approximate Latency Increase for X4000 FPGA adders, ns/bit