NOTE TO USERS

Page(s) not included in the original manuscript and are unavailable from the author or university. The manuscript was scanned as received.

40,43,88

This reproduction is the best copy available.

UMI®

Wideband Multi-Level QAM MODEM with Programmable Bit Rate and Programmable Intermediate Carrier Frequency

Qi Wang

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Applied Science at

Concordia University

Montreal, Quebec, Canada

August 2003

Your file Votre référence
ISBN: 0-612-91134-9
Our file Notre référence
ISBN: 0-612-91134-9

ABSTRACT

Wideband Multi-Level QAM MODEM with Programmable Bit Rate and Programmable Intermediate Carrier Frequency

Qi Wang

Quadrature Amplitude Modulation (QAM) has been predominant in current wireless digital communications, such as satellite and microwave radio communications. Due to diverse wireless communication applications, a digital QAM modem with diverse bit rates (R) and intermediate carry frequency (IF) is of great interest. The main objective of this thesis is to design, analyze and implement a multi-level QAM modem with programmable bit rates and programmable IF, thus not only meeting different requirements, but also dramatically reducing the complexity of the following analog Radio Frequency (RF) block.

In the thesis, the digital QAM modem is built based on the fundamentals of Digital Communication Systems (DCS) and Digital Signal Processing (DSP). The underlying relationship between signal resampling and its filter impulse response is studied and analyzed. A new signed-array structure for 2's complement multiplication is introduced in order to improve the regularity and flexibility of hardware design. The modem analysis and comparisons are shown in terms of performances and structures. Hardware design techniques are developed in-depth. The modem model is coded in VHDL (VHSIC Hardware Description Language) and synthesized using Synopsys tools, and its performance is evaluated. The regular structure and portable VHDL coding of the modem can satisfy diverse applications implemented in different FPGAs or ASICs.

Acknowledgment

The subject Wideband Proprammable Modem that my supervisor Prof. Yousef Shayan particularly chose for me is immensely challenging and inspiring. The experience of doing this research could never have been more rewarding and fruitful to me. Particularly at the beginning, I would like to sincerely thank Prof. Yousef Shayan, my thesis supervisor, for his valuable guidance and priceless advice in carrying out this research from its start to its end. He introduced me to this exciting area of Wideband Multi-Level QAM Modem, which is market-oriented and of great importance in modern digital communications, and was a constant source of inspiration. Without his encouragement and precious advice all the time, this research would have been impossible to be done.

I would also like to extend my heart-felt appreciation to Dr. Kort and Mr. Obuchowicz for their help and valuable suggestions whenever I needed it.

My wife has always been a stimulus of my research. I would not smoothly have been through this hard time without her long-time encouragement and support.

I would like to express my honest gratitude to my colleagues and friends namely Haiqing Zhang, Wei Wu, Yanjie Wang, Yang Xiao and other colleagues from the Graduate Student Office. These individuals have provided boundless energy and selfless assistance during my graduate work and have seen me through this most important moment of my life as a student of Concordia University. Without their support, this would not have been a smooth, memorable ride.

Table of Contents

List of Figures

List of Tables

List of Acronyms

(Alphabetical Order)

Acronym	Meaning
ASIC	Application Specific Integrated Circuit
AWGN	Additive White Guassian Noise
BER	Bit Error Rate
CPA	Carry-Propagation-Adder
CSA	Carry-Save-Adder
CSA	Carry-Select-Adder
DAC	Digital-Analog Converter
DCS	Digital Communication System
DSP	Digital Signal Processing
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
IF	Intermediate Frequency
IIR	Infinite Impulse Response
ISI	Intersymbol Interference
KCM	Constant Coefficient Multiplier
LPF	Low Pass Filter
LUT	Look-Up Table
MF	Matched Filter
M-QAM	M-ary Quadrature Amplitude Modulation

MS ..... Memory Size

PSTN ..... Public Service Telephone Network

QPSK ..... Quadrature Phase Shift Keying

RF ..... Radio Frequency

RCA ..... Ripple-Carry-Adder

RAM ..... Random Access Memory

VHDL ..... VHSIC Hardware Description language

VLSI ..... Very Large Scale Integration

WDM ..... Wideband Digital Modem

Chapter 1 Introduction

Based on different applications, in modern communication systems, diverse modulation techniques, such as Phase Shift Keying (PSK), Frequency Shift Keying (FSK) and Quadrature Amplitude Modulation (QAM), are well developed and widely deployed. However, with the ever-growing market demands, the development of communication systems is now facing more and more rigorous constraints:

- Available bandwidth
- Permissible power
- Reliable error performance

Due to its bandwidth & power efficiency, QAM technology captures more attention and is extensively applied to high-speed digital communication systems. Currently, the commercial applications of QAM technology cover many applications such as satellite and microwave radio communications.

A typical functional transmitter/receiver diagram for Quadrature Amplitude Modulation is shown in Fig 1-1which consists of a QAM modem and a Radio Frequency (RF) unit. The RF unit is usually implemented by analog technology since its high radio frequency cannot be generated by current digital hardware technology.

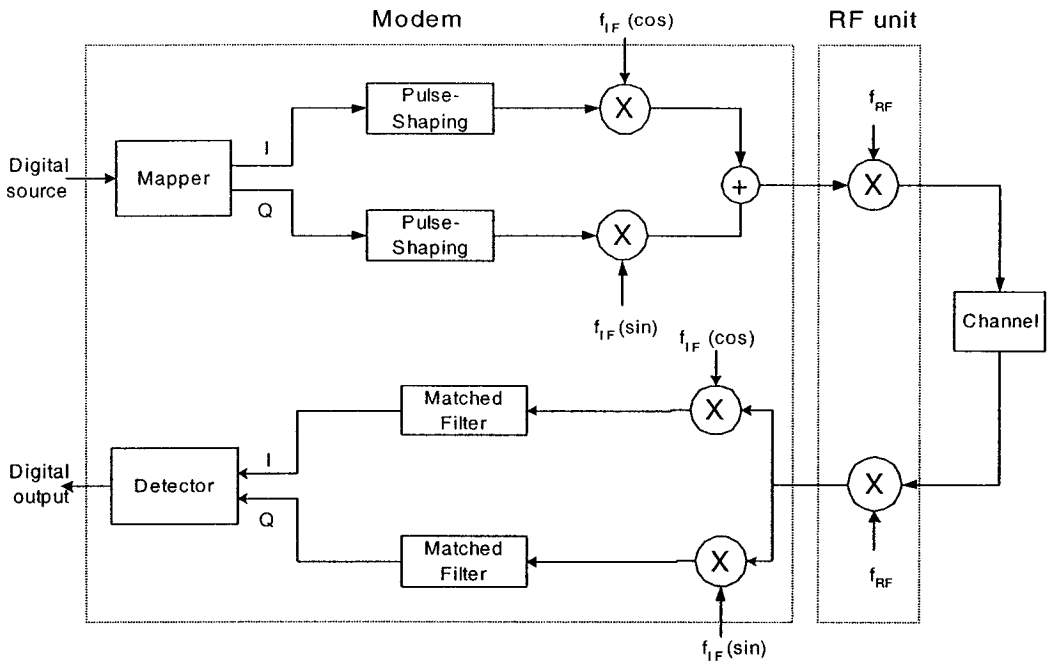


Fig 1-1 Typical QAM transmitter/receiver diagram

1.1 Literature Survey for QAM Transmitter/Receiver Implementation

Currently, there are mainly 2 implementation approaches for a digital QAM modem: analog and digital. In analog implementation, there exist the following inevitable drawbacks due to analog circuit inaccuracies:

1. In-phase & Quadrature (I & Q) imbalance from magnitude and phase point of view.
2. In-phase & Quadrature (I & Q) imbalance from Direct Current (DC) offset and drift point of view.

The errors associated with these drawbacks are illustrated in Fig 1-2. An ideal quadrature demodulator will have a perfect circle with its center at the origin (0,0) on I/Q plane. DC offsets in the analog demodulator circuit cause the center to move from the origin.

Elliptical characteristics on I/Q plane are due to gain imbalance and quadrature phase error.

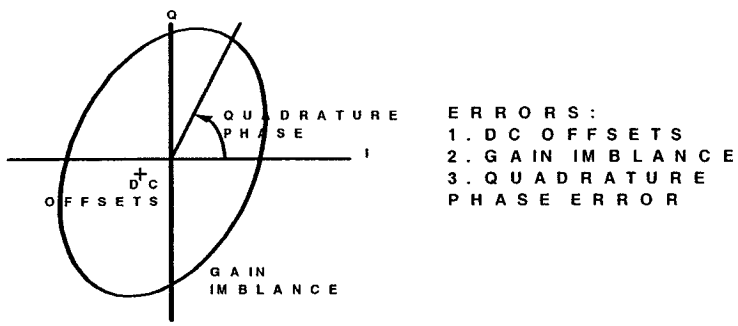


Fig 1-2 Errors in analog quadrature demodulation

To avoid the above disadvantages caused by analog hardware, there is a clear trend toward implementing part of the system digitally. The section of the system designed digitally can vary from radio to radio. Therefore, the location of the Digital-to-Analog Converter (DAC) varies. For example, Fig 1-3 [1] shows 2 different approaches to implement a digital transmitter. In Fig 1-3 (a), the transmitter is implemented by digital baseband filters and analog I/Q modulators. Two DACs are located prior to I/Q modulators in this case. However, to implement I/Q modulators digitally, as shown in Fig 1-3 (b), leads to less expensive cost, more stable results and better system peerformance. In this case, a DAC is applied right after digital I/Q modulators. The dash-lined sections in Fig 1-3 are digitally implemented.

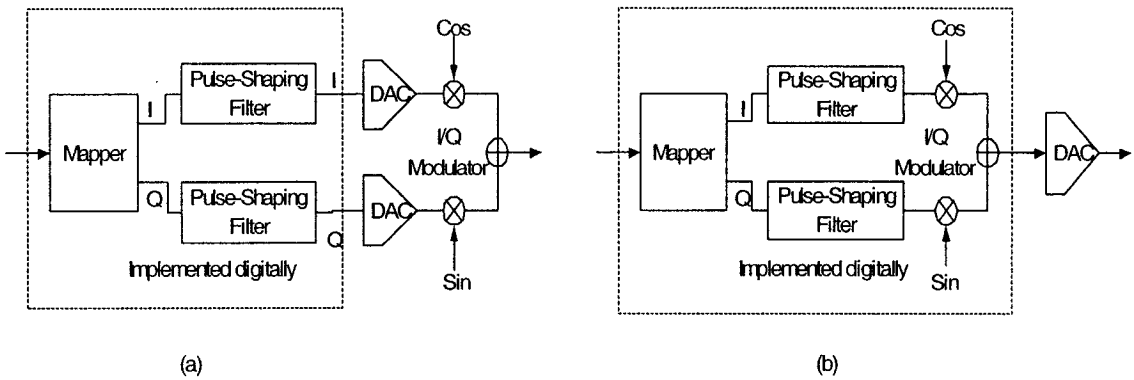


Fig 1-3 Digital implementations of a transmitter:

(a) Analog IF; (b) Digital IF

In fully digital modems, many designers have developed a simple approach with $f_c = R_s$ (R_s: baud rate, f_c: IF carrier frequency) [2, 3, 4]. This approach has taken advantage of the inherent relationship between pulse-shaping filters and IF modulators; thus hardware for the transmitter implementation could be significantly reduced. This approach is typically used in some specific applications where the baud rate R_s is fixed. However, the above method is not suitable for general-purpose commercial modems since they may be used for different applications. In this thesis, we will design and implement such a general-purpose modem considering the following requirements:

- Variable input data rates (R) for a wide variety of customer applications
- Fixed and selectable IF carrier frequency so as to reduce the complexity of analog RF
- Low bit error rate (P_B) for high quality communications

1.2 Thesis Preview

The objective of this thesis is to design and implement a wideband digital modem based on QAM modulation, which features:

- Fully digitized
- Multi-level QAM modulation (QPSK/16-QAM)
- Variable and high-speed input bit rates
- Fixed but selectable IF
- Single chip modem (ASIC or FPGA)

In the course of the design, considerable success is achieved. An approach is explained for having selectable and fixed carrier frequency. A new array-based structure for 2's complement multiplication is proposed and employed in hardware implementation. Pipelining and parallel processing is widely used for high-speed implementation of a pure digital QAM modem system.

The design and implementation of the modem consists of 2 steps. The first step focuses on the system architectural design and the system-level simulation by using MATLAB simulation software. The design specifications and parameters are determined and optimized at this step. The second step concentrates on the hardware design by using VHDL (VHSIC Hardware Description Language). At this step, the hardware structure of the system is optimized and the test-bench for design verification is built. Finally, as an example, the hardware implementation based on Xilinx virtex II FPGA is illustrated. Fig 1-4 shows the design and implementation flow of the modem.

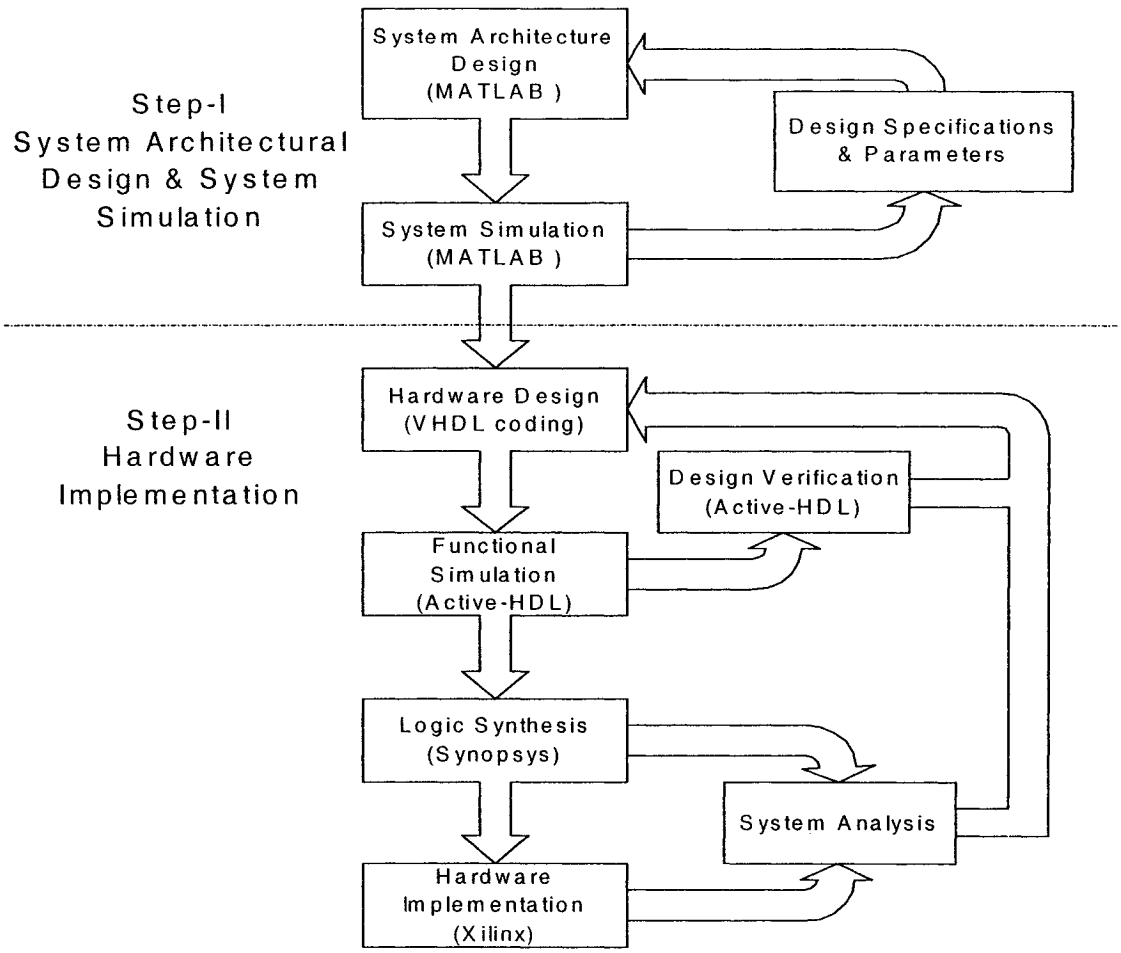


Figure 1-4 Design & implementation flow of the modem

1.3 Thesis Organization

The thesis is organized as follows:

a. In Chapter 2, a thorough introduction is given to Digital Communication Systems (DCS) and Digital Signal Processing (DSP) fundamentals and corresponding algorithms as to the modem. Subjects include discrete sampling and 2's complement representation, digital signal processing and its applications, interpolation and decimation, finite precision and quantization effects.

b. In Chapter 3, the modem fundamental architectures will be built using MATLAB simulation tools. At this stage, the system structure and functionality is simulated and verified, and the corresponding parameters such as filter orders and word length are optimized and determined.

c. In Chapter 4, hardware design techniques for the modem, such as constellation mapping, high-speed adder, poly-phase filters are developed. Furthermore, a new signed-array structure for 2's complement multiplication is introduced and detailed. Also, pipelining & parallel processing are discussed.

d. In Chapter 5, the detailed structure of each block is built using VHDL. Some optimizations are adopted to improve the performance. The functional verification is made using ALDEC Active-HDL. Design analysis and logic synthesis is done by Snopsys Design Compiler and Synopsys Analyzer. The system hardware implementation targeted to Xilinx virtex II FPGA is accomplished by using Xilinx Design Manager. At this stage, the overall system specifications will be given.

e. Chapter 6 summarizes the whole work and main contributions of the thesis.

Chapter 2 Modem Fundamentals

As discussed in Chapter 1, a purely digitally implemented QAM modem has the advantages in terms of cost and performance; thus it would be more preferable in modern communication system designs. However, unlike analog signals, digital signals are processed in a different way, called Digital Signal Processing (DSP), which is concerned with the representation, transformation and manipulation of signals and the information they contain. In this chapter, some basic but important DSP fundamentals including signal sampling and binary number representation are first introduced. Particular attention is paid to pulse-shaping and matched filter (MF) concepts. Nyquist filters and zero inter-symbol interference (ISI) are discussed in this section. The signal re-sampling section clearly manifests the inherent relationship between the digital filter and its input signal. The relation of the symbol rate and digital intermediate frequency (IF) is also discussed. Furthermore, the concept of interpolation and decimation are introduced for pure digital modems with fixed IF. At the end, the quantization effects in digital systems are explained.

2.1 Review of Basic DSP Fundamentals

Compared to analog systems, digital signals feature two characteristics: time-discrete and amplitude-discrete (a set of finite values). In this section, we give a brief discussion to these 2 major aspects that arise in processing and implementing digital systems. A more detailed analysis for signal re-sampling and finite-precision effects is given in section 2.2 and 2.5.

2.1.1 Signal Sampling

A discrete-time system takes an input of numbers and produces an output sequence of numbers. These number sequences $x(n)$ are often samples of a continuous function of time, where $x(n) = x_a(nT)$, which represents the value of an analog signal $x_a(t)$ at equally spaced times $t_n = nT$, and n is an integer. To simplify the discussion, T is usually set to 1 unless otherwise specified.

It is well known that a continuous signal with maximum frequency f_m can be restored without information loss if it is sampled by the sampling rate f_s larger than or equal to $2f_m$ (*Nyquist Sampling Theorem*). The impulse response of an ideal Nyquist low-pass filter (LPF) in time domain and its frequency spectrum can be expressed as:

$$h(t) = W \cdot \sin c(Wt) \tag{2.1}$$

$$H(f) = \begin{cases} 1 & |f| \le \frac{W}{2} \\ 0 & elsewhere \end{cases} \tag{2.2}$$

where $\frac{W}{2}$ is the bandwidth of LPF.

For a digital filter, we often use $h(n)$ and $H(e^{jw})$ to represent the filter impulse response and frequency spectrum, respectively. Hence, the digital filter response $h(n)$, sampled from its analog impulse response function $h(t)$, should comply with *Nyquist Sampling Theorem.* In real systems, the sampling rate that is used to obtain $h(n)$ from $h(t)$ is usually performed at a rate that exceeds the *Nyquist minimum bandwidth* by a factor of 4 or more.

By using MATLAB, Fig 2-1 shows different $h(n)$ sequences that can be obtained from the same analog impulse function $h(t)$. This has been achieved by up sampling at rates of 2, 4, and 8, where f_m is the cutoff frequency of $\sin c(t)$.

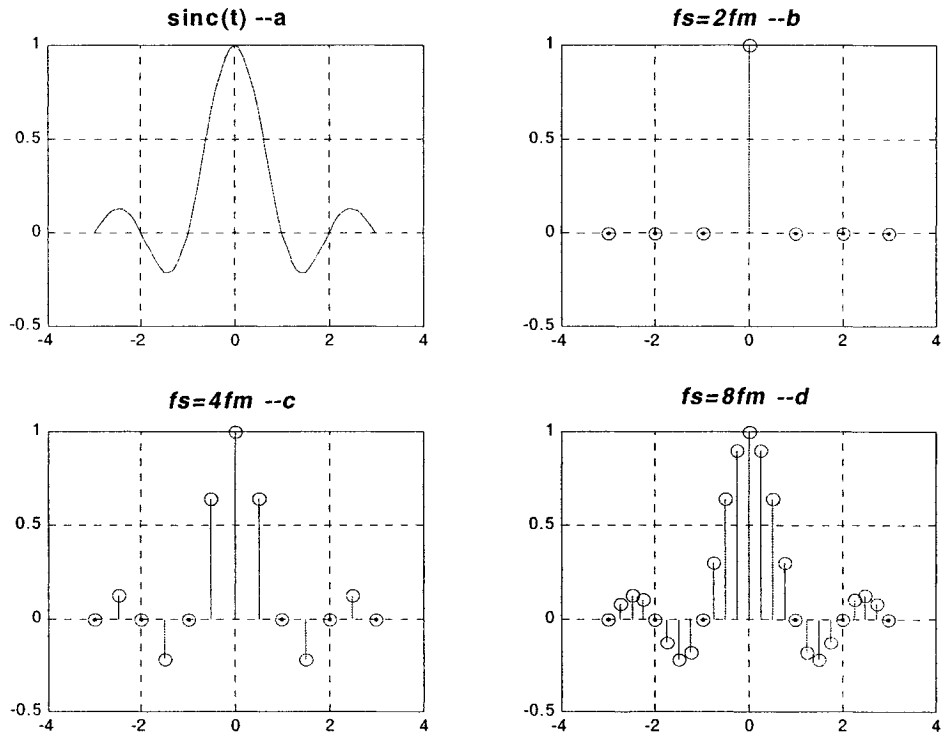


Figure 2-1 Impulse response of LPF with different up sampling rates
a. analog function *h(t)=sinc(t);* b. *h(n)* with sampling rate $f_s=2f_m$;
c. *h(n)* with sampling rate $f_s=4f_m$; d. *h(n)* with sampling rate $f_s=8f_c$

2.1.2 Binary Number Representations

Generally, we assume that signal values and system coefficient have infinite precision in theoretical analyses of discrete-time systems. However, when implementing digital systems, the signal values and system coefficients should be represented with finite precision since they need to be stored in finite-length registers. The digital hardware implementation is based on the binary-number representation. Currently, 3 number representation formats, which are signed magnitude, 1's complement and 2's complement, are available. But 2's complement representation is the most common for digital system designs because its subtraction can be performed with addition circuitry. In 2's complement, a real number with infinite-precision can be represented as [5]:

$$x = X_m \left(-b_{m-1} + \sum_{i=0}^{m-2} b_i 2^{i-(m-1)} \right) \tag{2.3}$$

Where X_m is an arbitrary scale factor and the b_i is either 0 or 1. b_{m-1} is referred to as the sign bit and m is an integer. In Eq (2.3), $0 \leq x \leq X_m$ when $b_{m-1}=0$; otherwise, $-X_m \leq x \leq 0$. In hardware implementation, this arbitrary number x must be represented with finite-precision using a finite bit length (say l bits, $l < m$). So Eq. (2.3) can be modified to:

$$\hat{x} = Q_l[x] = X_m \left(-b_{l-1} + \sum_{i=0}^{l-2} b_i 2^{i-(l-1)} \right) = X_m \hat{x}_l \tag{2.4}$$

In this case, the fractional part of the quantized number can be represented as:

$$\hat{x}_L = b_{l-1} b_{l-2} \cdots b_0 \tag{2.5}$$

Obviously, $\hat{x}$ in Eq (2.4) is the quantized result of x. This leads to a quantization error that can be expressed as:

$$\varepsilon = x - Q_l[x] \tag{2.6}$$

More detailed analysis for quantization effects is given in section 2.5.

2.2 Pulse Shaping and MF Concept

In wireless communication systems, the frequency bandwidth is always limited because of signal transmission channel bandwidth constraint and government regulations. However, a non-processed digital signal stream is a bit stream whose spectral content usually extends from DC up to infinity. It must be converted to a waveform that is compatible with the government requirements and transmission channel. Pulse-shaping is such a process that can transmit a binary representation (rectangular pulse) to a baseband waveform (Nyquist pulse) whose bandwidth can meet the above requirements. And matched filtering is its paired process in the demodulator side used to provide the maximum signal-to-noise power ratio at the output for a given transmitted signal waveform.

2.2.1 Pulse-Shaping

The more compact the signaling spectrum is made, the greater is the number of users that can simultaneously be served. For wireless communication systems, it is essential to reduce the required system bandwidth as much as possible since it means the greatest profit to communication service providers. However, there is a basic limitation to such bandwidth reduction. Once a system operates at smaller bandwidths, the pulses would spread in time domain and result in interference among adjacent symbols. This effect refers as to *Intersymbol Interference* (ISI). Hence, a pulse-shaping filter should be:

- with the bandwidth as small as possible
- with zero ISI

According to Nyquist [7], the theoretical minimum system bandwidth needed in order to detect R_s symbols/s, with zero ISI, is $R_s/2$ Hz. This is called *Nyquist Bandwidth Constraint.* The impulse response and frequency spectrum of an ideal Nyquist filter is shown in Fig 2-2. In hardware implementation, a Nyquist filter with ideal rectangular-shaped frequency transfer function is not realizable since its corresponding time-domain pulse is with infinite length.

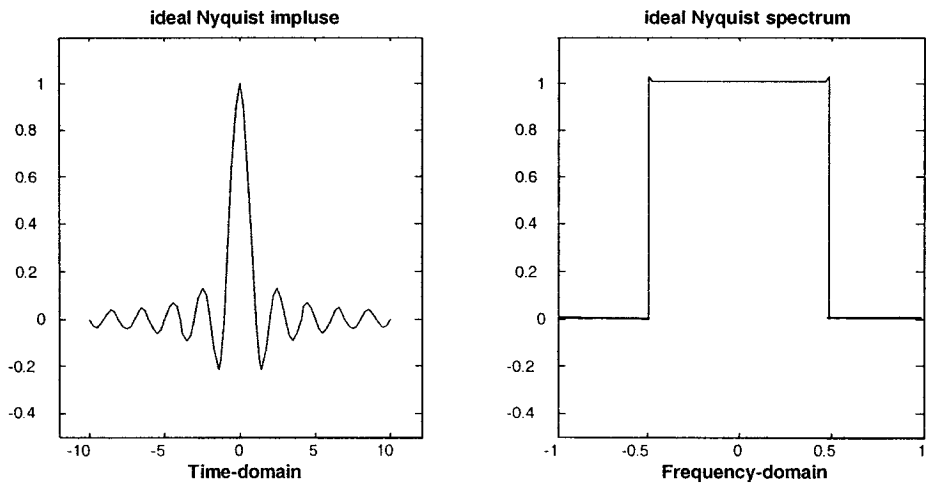


Figure 2-2 Ideal Nyquist Filter

Since ideal Nyquist filters are not realizable, a prudent decision should be made by which the requirement of zero ISI is met and the system bandwidth is reasonably small but greater than *Nyquist bandwidth constraint.* Raised cosine filter is one of Nyquist filters that can meet the above requirements. Its frequency transfer function can be expressed as follows:

$$H(f) = \begin{cases} 1 & |f| < 2W_0 - W \\ \cos^2\left(\frac{\pi}{4}\frac{|f| + W - 2W_0}{W - W_0}\right) & 2W_0 - W \le |f| \le W \\ 0 & |f| > W \end{cases} \qquad (2.7)$$

where W is absolute bandwidth and $W_0 = 1/2T$ represents the minimum Nyquist bandwidth. The roll off factor is defined as $r=(W-W_0)/W_0$, $0 \leq r \leq 1$. The difference of $W-W_0$ is termed "excess bandwidth".

A convenient model for a typical baseband digital system and its frequency transfer function $H(f)$ can be illustrated as Fig 2-3, where, $H_t(f)$, $H_c(f)$, $H_r(f)$ are transfer functions for the transmitter, channel, and receiver, respectively.

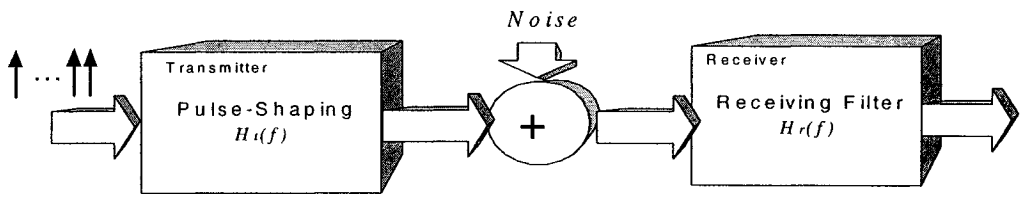


Figure 2-3 Typical baseband digital system

Without noise appearance, the transfer function for the channel is set $H_c(f)=1$. Thus, the system transfer function $H(f)$ can be rewritten as: $H(f) = H_t(f)H_r(f)$. The receiving filter is often configured to compensate the distortion by both the transmitter and the channel. The signal-to-noise ratio (*SNR*) could be optimized if $H_t(f) = H_r(f) = H(f)^{1/2}$ [5] which is called "*square-root raised cosine filter*". In such a system, the transmitting filter and receiving filter are identical. Although the impulse response of the square-root raised cosine filter, called "*square-root Nyquist pulse*", doesn't exhibit zero ISI, the product of their transfer functions will give rise to zero ISI if a square-root raised cosine filter is used both at the transmitter and at the receiver [10].

2.2.2 Matched Filter Detection

The maximum signal-to-noise ratio could be obtained when the impulse response of the receiving filter is the mirror image of the transmitted signal *s(t)*, delayed by the symbol duration T [10]. This kind of the receiving filter is then called "*matched filter*". Its impulse response can be expressed as:

$$h_r(t) = \begin{cases} ks(T-t) & 0 \le t \le T \\ 0 & elsewhere \end{cases} \tag{2.8}$$

where k is constant.

For digital systems, the above matched filter is called "*sampled matched filter*". Its function can also be expressed as:

$$h_r(n) = \begin{cases} ks(M-n) & 0 \le n \le M \\ 0 & elsewhere \end{cases} \tag{2.9}$$

where M is an integer.

In digital communication systems, MF detection is always required since the maximum SNR should be obtained. Meanwhile, zero ISI is essential for the whole system as well. Fortunately, the square-root raised cosine filter meets these 2 requirements if a root-square raised cosine filter is used both at the transmitter and the receiver.

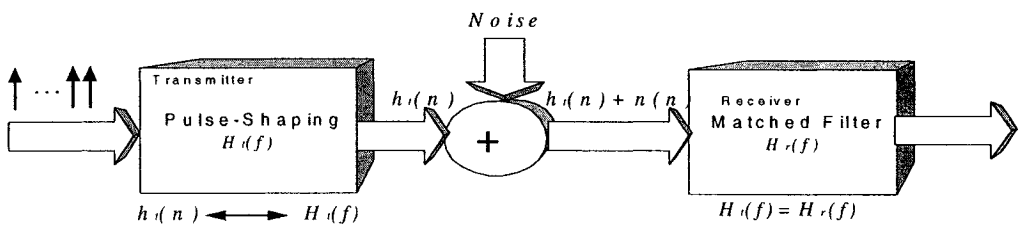


Figure 2-4 Typical matched detection system

As shown in Fig 2-4, the input of the transmitting filter is processed as an impulse sequence, so the transmitted signal $s(n)$ is exactly $h_t(n)$. And the impulse response of the square-root raised cosine filter is even symmetrical; thus $h_t(n) = h_t(-n)$. Clearly, since the transmitting filter and the receiving filter are identical, the following equation can be established:

$$h_r(n) = h_t(n) = h_t(-n) = s(-n) \tag{2.10}$$

In Eq(2.9), integer M means time delay; it doesn't affect any other characteristics. Hence, it is obvious that using square-root raised cosine filter at both transmitter and receiver could satisfy zero ISI and MF detection.

2.2.3 Signal Re-sampling

In digital communication system design, digital filter coefficient $h(n)$ is pre-defined and stored in memory. If the transversal structure is used, the pulse-shaping filter consists of 2 parts: an up-sampler and a low-pass filter (LPF) shown as Figure 2-5. In poly-phase structure (refer to section 3.4), the up-sampler and the low-pass filter are combined. In the following, the underlying relationship between input $x(n)$ and the digital filter impulse response $h(n)$ will be discussed.

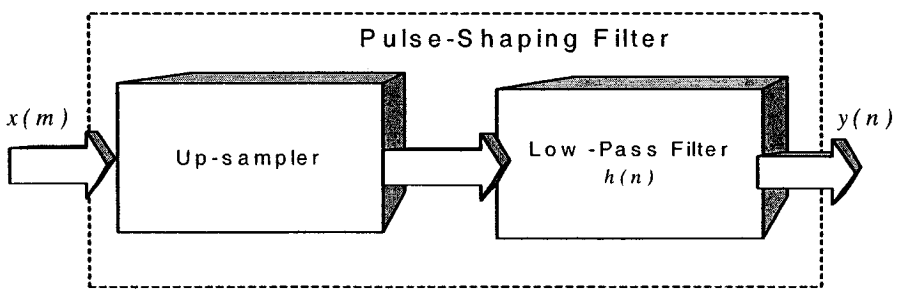


Figure 2-5 Pulse-shaping filter

Assume that the input $x(n)$ with symbol rate R_s goes through a filter. According to Nyquist, the theoretical minimum filter bandwidth is $R_s/2$ for detecting R_s symbols/s without ISI. The following equation could be obtained:

$$f_m = \frac{R_s}{2} \tag{2.11}$$

where f_m is the filter bandwidth.

The above filter can be implemented using digital techniques and sampled waveforms. If such a digital filter has N sampled waveforms per symbol duration, then the filter needs to process N times in one symbol duration T ($1/R_s$). Since $x(n)$ inputs only one symbol in one symbol duration, the processing match between input $x(n)$ and the filter coefficients $h(n)$ must be made. Up-sampling, sometimes called "zero-padding", is such a process which stuffs corresponding zeros in input sequence $x(n)$.

If the sampling frequency of the filter is f_s, the number of zeros stuffed between 2 input symbols yields:

$$N-1 = f_s/R_s - 1 \tag{2.12}$$

For example, if a digital filter has 4 sampled waveforms per symbol duration, the zero-stuffed number is N-1=3. And then the input sequence x_0, x_1, x_2 ..... is pre-processed by the up-sampler as: x_0 ,0 ,0 ,0 x_1, 0, 0, 0, x_2, 0, 0, 0..... before going through the following LPF.

2.3 Digital IF f_c and Symbol Rate R_s

So-called quadrature modulation is a method of combining two signals into a single channel by modulating a cosine wave on I channel and a sinusoid wave on Q channel,

thereby doubling the effective bandwidth. For the digital IF modulator, its IF waveform is pre-defined and stored in the memory.

Once the digital QAM modem is designed, the sampling rate at each stage is determined. In Fig 2-6, the sampling rate at each stage of the modulator is shown, where $k=\log_2(M)$ (M: QAM symbol set size), n is the up-sampling rate of the pulse-shaping filter.

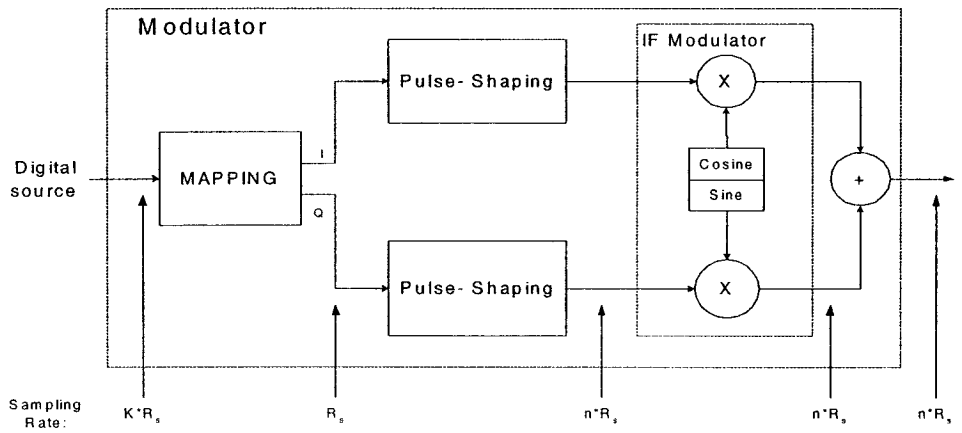


Figure 2-6 Modulator sampling rates at each stage

Carrier wave (sine and cosine) coefficients are pre-defined and stored in memory. If the carrier frequency is f_c, its coefficients could be obtained by up-sampling with sampling rate $f_s \geq 2f_c$. In real systems, such sampling rate usually reaches a factor of 4 or more. Assuming that m carrier samples are taken every carrier period ($m= f_s/f_c$), and Fig 2-6 reveals that $f_s = n \cdot R_s$; therefore, the carrier frequency f_c could be obtained as follows:

$$\frac{R_s}{f_c} = \frac{m}{n} \tag{2.13}$$

Usually, n=4 and $m \geq 4$ are taken in real systems. Clearly, if $m \geq n$, $R_s \geq f_c$, the carrier frequency f_c will never be higher than the symbol rate R_s. However, in most of communication systems, the carrier frequency is to be fixed at a much higher frequency than the symbol rate. Therefore, the interpolation and the decimation are then introduced to make it possible.

2.4 Interpolation and Decimation

A common use of interpolation and decimation in DSP is for sampling-rate conversion. Suppose a digital signal $x(n)$ is sampled at an interval T_1, and we wish to obtain a signal $y(n)$ sampled at an interval T_2. Then the techniques of decimation and interpolation enable this operation, providing the ratio T_1/T_2 is a rational number i.e. L/M. Sampling-rate conversion can be accomplished by L-fold expansion, followed by low-pass filtering and then M-fold decimation. It is important to emphasize that the interpolation should be performed first and decimation second to preserve the desired spectral characteristics of $x(n)$. Furthermore by cascading the two in this manner, both of the filters can be combined into one single low-pass filter shown in Figure 2-7 [5]. Furthermore, if the ratio T_1/T_2 is an integer, only an interpolator or decimator is needed.

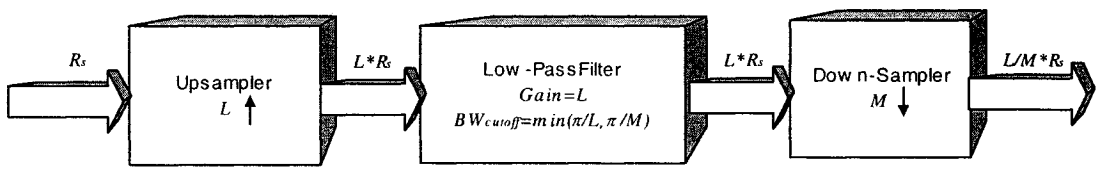


Figure 2-7 Filter with rational factor *L/M*

Actually, the pulse-shaping filter discussed above is an interpolator whose up-sampling rate is an integer. The low pass filter in Fig 2-7 is set with Gain of L and Cutoff Bandwidth of $min(\pi/L,\pi/M)$. Cutoff Bandwidth is required to achieve anti-aliasing and

anti-imaging [6, 7]. Gain of L is used to keep interpolation key values unchanged after filtering.

With interpolators and decimators added, a real pure digital modem for general-purpose and its symbol rate at each stage are shown in Fig 2-8. Where L and M are the interpolation and decimation factors. The focus of this thesis will be on the dash-lined sections in Fig 2-8.

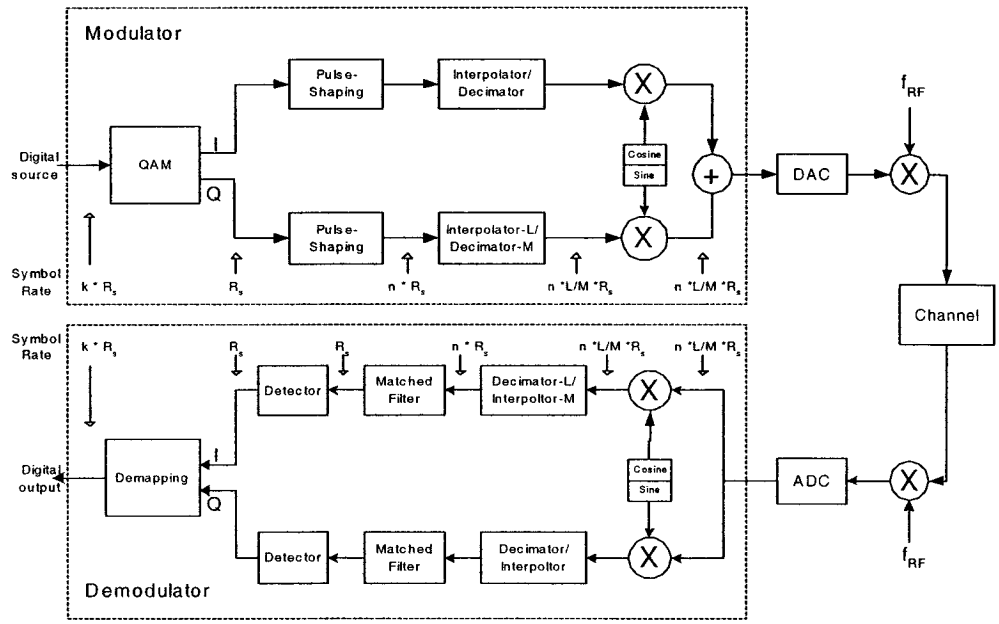


Figure 2-8 Modem and its sampling rates

With interpolation and decimation, the relationship between symbol rate R_s and the carrier frequency f_c in the transmitter side could be expressed as follows:

$$f_c = \frac{L}{M}\frac{n}{m}R_s \tag{2.14}$$

where L and M are interpolation and decimation factors, respectively; n is pulse-shaping filter up-sampling rate and m is the number of carrier samples per carrier period. Since L

and M could be changed as required, the carrier frequency f_c can be set fixed and much higher than the symbol rate R_s.

2.5 Finite-Precision Numerical Effects

In hardware implementation, digital system inputs, coefficients, and results are always stored in finite-length registers. Consequently, either overflows or quantization errors could be introduced in digital binary number representation. In the following sections, some techniques are discussed to help designers choose appropriate methods to meet both system and hardware implementation requirements.

2.5.1 Signal Scaling

If the amplitude of a signal in a fixed-point implementation exceeds the dynamic range, overflows will arise and the output signal could be extremely distorted. Meanwhile, if this amplitude is set very low when being processed through a digital filter, the filter will be inefficiently operating and the signal-to-noise ratio will be very poor. Hence, a suitable signal scaling needs to be used to optimize signal levels.

There are several ways to estimate signal levels. One is to use test signals obtained from a real target. Scaling is based on maximum sample values at each node. This method is seldom used if the system is large and thus calculation is complicated.

Another method, that is widely used, is to calculate the most maximum sample values based on the system transfer function and impulse response. There are 2 norms used for the estimation [13, 14,16]:

- Safe scaling: to choose appropriate λ to make $v(n) \leq M$.

$$v(n) \leq M \sum_{k=0}^{\infty} |\lambda f(k)| \tag{2.15}$$

where λ is the scaling constant; $v(n)$ is variable at node k; $f(k)$ is the transfer function from the input to node k; and the input amplitude is bounded by M.

- L_p norm:

$$|F_k(z)|_p = \left[\frac{1}{2\pi} \int_{-\pi}^{\pi} |F_k(e^{jw})|^p \, dw \right]^{\frac{1}{p}} \tag{2.16}$$

where p is positive integer, and $F_k(z)$ is the transfer function from the input to node k.

2.5.2 Truncation and Rounding

Once the register length is assigned, the set of machine representable numbers is determined. Assume that the assigned word length is l bits. Then, any number of m bits long ($m>l$) should be quantized before being processed. There are 2 approaches normally used for quantization: truncation and rounding. Truncation is to truncate all the least significant bits of n-bit long number that can't be accommodated in l-bit long registers. Rounding is to round to the nearest machine-representable number by adding 1 at the position $l+1$ and then truncating the m-bit long number to l bits.

Obviously, if a number x is quantized, an error ε is then introduced given by [11]:

$$\varepsilon = x - Q[x] \tag{2.17}$$

where $Q[x]$ denotes the quantized value of x.For 2's complement number, quantization error ε_T of truncation could be expressed as:

$$0 < \varepsilon_T < q \tag{2.18}$$

where q is quantile interval.

And quantization error ε_R of rounding could be expressed as:

$$-\frac{q}{2} < \varepsilon_R < \frac{q}{2} \tag{2.19}$$

In hardware implementation, quantization has to be done, but quantization error acts as noise that degrades the system performance. Hence, word length of system registers should be chosen very carefully.

Chapter 3 The Modem Simulations

A typical QAM modem simulation model is constructed using MATLAB software based on Fig 2-8 block diagram [8, 9]. The system performance is evaluated based on modem Bit Error Rate (BER). This chapter mainly discusses evaluation of different parameters in the design of digital filters since they play the most important role in the system. The modem employs four filters which are pulse shaping, matched filter, interpolator and decimator. Advantages of FIR filters in communication systems are first briefed. Section 3.2 and 3.3 analyze the effects of filter orders (the pulse-shaping & matched filter, interpolation & decimation filter) on error performance. Finite word length is discussed in section 3.4. The main design parameters are finally given based on appropriate compromises between hardware complexity and error performance.

3.1 Why FIR Filters

There are 2 types of digital filters: Infinite Impulse Response (IIR) filters and Finite Impulse Response (FIR) filters. Currently, FIR filters are widely used in most of real-time communication systems. Compared to IIR filters, FIR filters are:

- easily designed to be "linear phase" to avoid signal distortion
- simple to implement using fractional arithmetic.
- computationally more efficient when doing multi-rate applications (interpolation &decimation).

Accordingly, FIR filters have been chosen for our system. Performance of an FIR filter is mainly determined by its filter order and the coefficient word length. The order of an FIR filter and the coefficient word length should be as large as possible if a designer wants to build up a filter as precise as possible. On the other hand, a larger FIR filter order or its coefficient word length requires more resources in hardware implementation, such as chip area, power consumption, etc. Thus, it is important to make the filter order or coefficients word length as low as possible. Hence, necessary trade-offs have to be made on both sides.

3.2 Pulse-Shaping Filter & Matched Filter Order

As mentioned in Chapter 2, in our design, a square-root raised cosine filter is used as the pulse-shaping filter on the transmitter side and the matched filter on the receiver side. The impulse response of the square-root raised cosine filter is:

$$h(t) = 4r \cdot \frac{\cos((1+r)\pi t/T) + \dfrac{\sin((1-r)\pi t/T)}{4rt/T}}{\pi\sqrt{T}\,((4rt/T)^2 - 1)} \qquad (3.1)$$

where r ($1 \geq r \geq 0$) is roll-off factor, T is symbol period. In our design, the roll-off factor r is set 0.5 for the square-root raised cosine filters.

The impulse response of an ideal, causal LPF filter extends from 0 to ∞. An FIR filter is an approximation of an ideal filter. Its length is determined by examining its error performance. The order of an FIR filter is affected by the filter symbol span and the up-sampling rate. The order can be expressed as:

$$N - 1 = 4 \cdot s \tag{3.2}$$

where N is the number of filter coefficients, N-1 is the filter order, s is the symbol span. For the square-root raised cosine filter, the up-sampling rate is set to 4.

By bypassing the interpolator/decimator, all the square-root raised cosine filter parameters could be examined and optimally chosen based on BER. For M-ary QAM modulation, there is an approximate equation expressed as follows [11]:

$$P_B \approx \frac{2(1 - L^{-1})}{\log_2 L} Q\left(\sqrt{\left(\frac{3\log_2 L}{L^2 - 1}\right)\frac{2E_b}{N_0}}\right) \tag{3.3}$$

where $Q(\cdot)$ is Q function [11], and L represents the number of amplitude levels in one dimension ($L=M^{1/2}$). Fig 3-1 and 3-2 show QPSK and 16-QAM error performances with pulse-shaping and matched filter symbol span s=6, 8, 10, in presence of *Additive, Gaussian White Noise* (AGWN) channel.

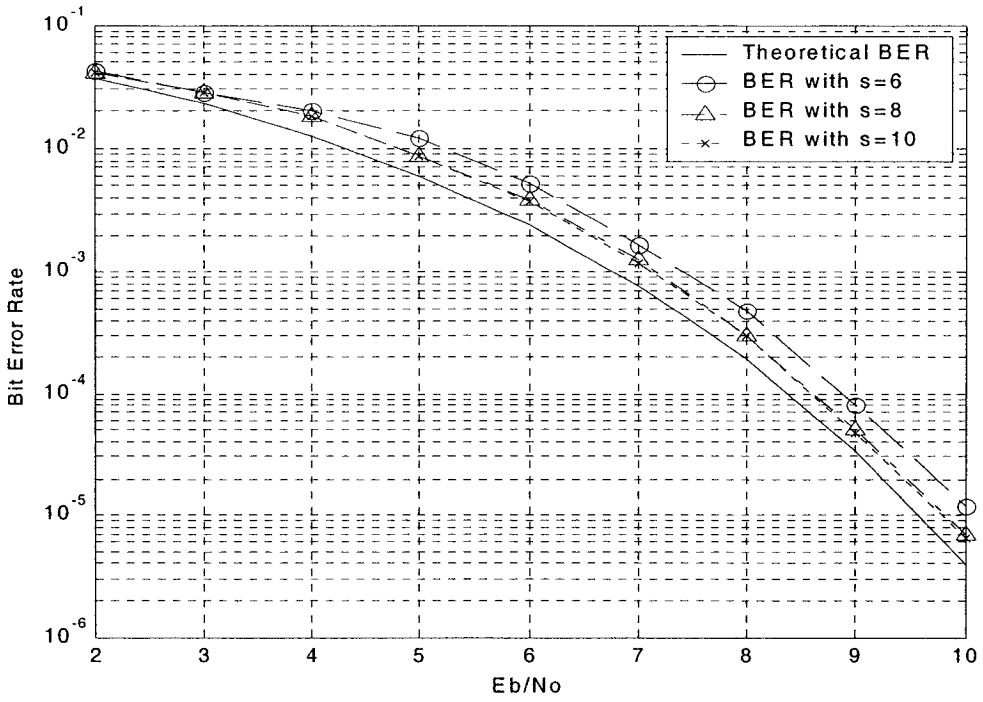


Figure 3-1 QPSK error bit rates with different symbol spans

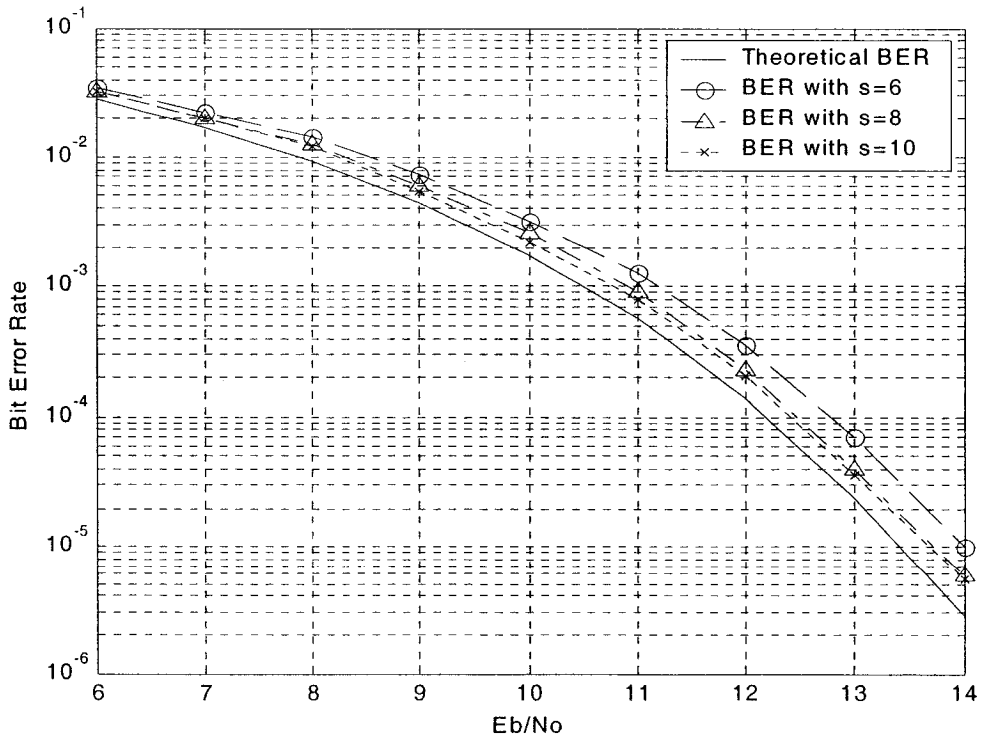


Figure3-2 16-QAM error bit rates with different symbol spans

Fig 3-1 and 3-2 reveal that a symbol span larger than 8 doesn't help much on the improvement of QPSK/16-QAM error performance but increases hardware. So the symbol span of the square-root raised cosine filter is set to 8. Thus the filter order is $4s$=32. Figure 3-3 and 3-4 are frequency and impulse response of the pulse-shaping/matched filters, respectively. The filter parameters are as follows.

- Number of filter coefficients: N=33 (filter order: 32)
- Up-sampling rate: 4;
- Roll-off factor: r=0.5

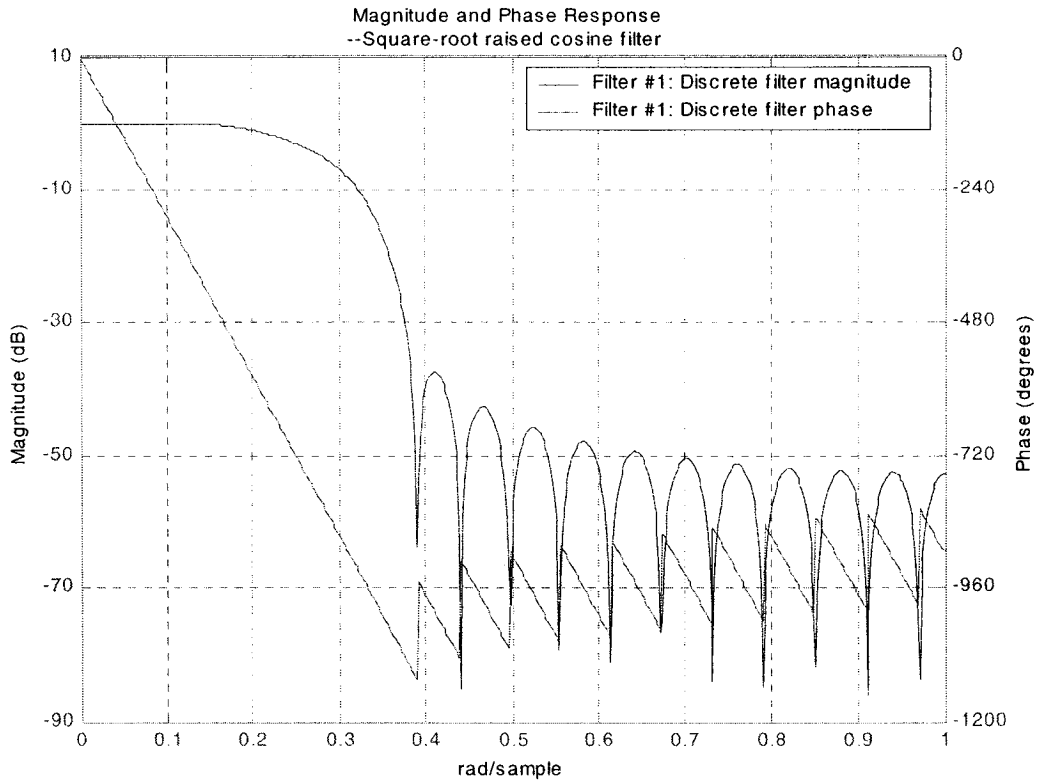


Figure 3-3 Frequency response of Square-Root Raised Cosine filter

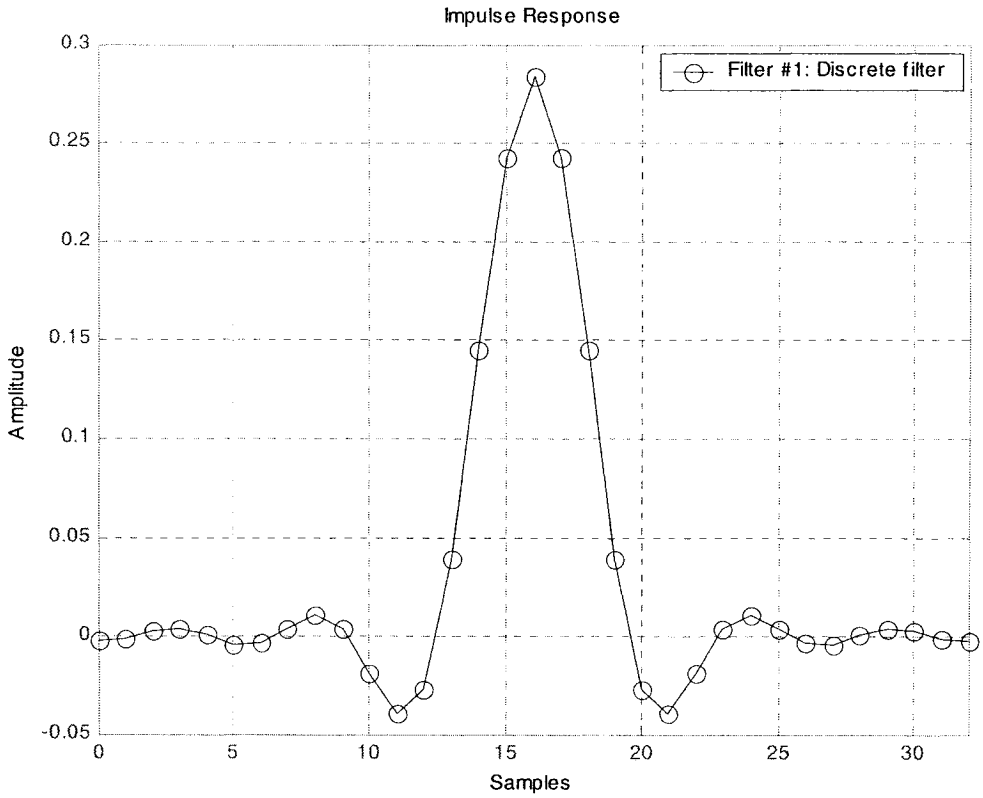


Figure 3-4 Impulse response of Square-Root Raised Cosine filter

3.3 Interpolation Filter & Decimation Filter Order

In chapter 2, we have discussed interpolation with rational factor. To simplify hardware implementation, an integer interpolation is employed in our design.

The interpolator with factor L on the transmitter side makes the filter up- sampling rate changeable in order to make the sampling match between the pulse-shaping filter and the complex mixer. On the receiver side, a decimator with the same frequency spectrum needs to be inserted between IF recovery block and the matched filter. Table 3-1 shows the system BERs with different interpolation factors L and different symbol spans s at E_B/N_0=14 dB for 16-QAM modulation.

Table 3-1 16-QAM error bit rates through interpolation/decimation

(E_B/N_0=14dB)

L	S=4		S=6		S=8	
	N	P_B	N	P_B	N	P_B
bypassed		$5.8*10^{-6}$		$5.8*10^{-6}$		$5.8*10^{-6}$
2	9	$1.1*10^{-5}$	13	$6.12*10^{-6}$	17	$6.05*10^{-6}$
4	17	$1.0*10^{-5}$	25	$5.9*10^{-6}$	33	$5.8*10^{-6}$
8	33	$1.0*10^{-5}$	49	$5.8*10^{-6}$	65	$5.8*10^{-6}$
16	65	$1.0*10^{-5}$	97	$5.8*10^{-6}$	129	$5.8*10^{-6}$

where the number of coefficients $N = s*L+1$; P_B is Bit Error Rate.

As seen in Table 3-1, the symbol span of 6 is the best tradeoff from error performance and hardware implementation point of view. This result is also verified at other values of E_B/N_0. Once the symbol span for the interpolator/decimator is determined, the frequency spectrums with different interpolation/decimation factor L/M could be plotted in Fig 3-5.

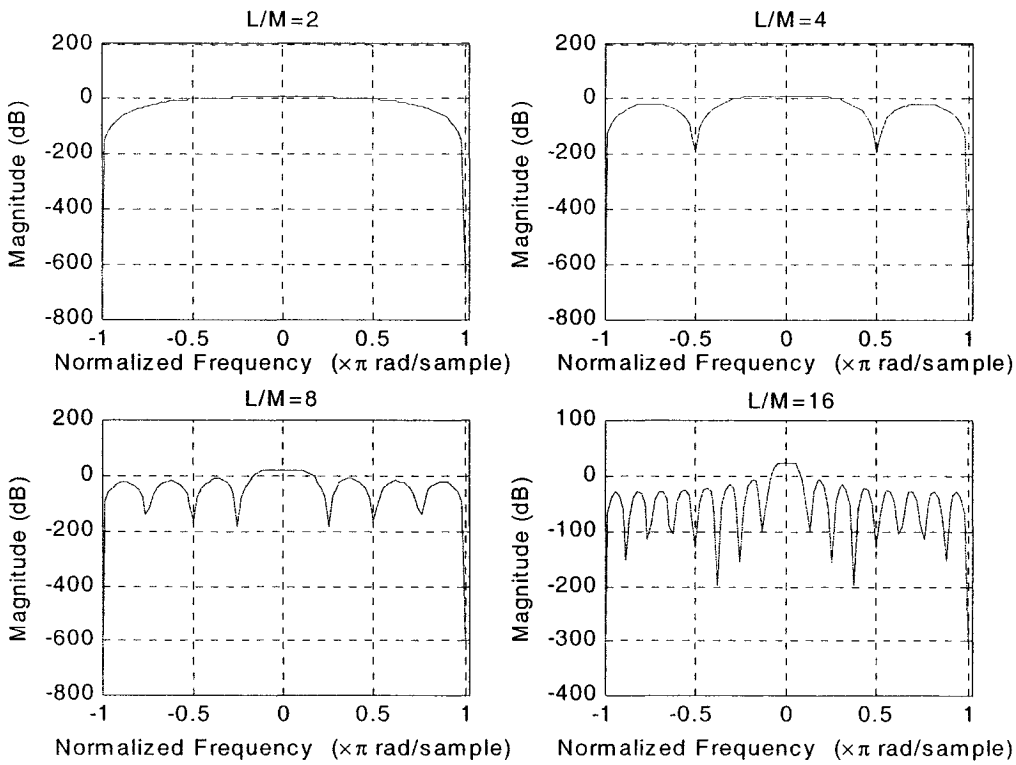


Figure 3-5 Frequency response of interpolator with different factors

3.4 Finite Word Length

The previous discussion regarding filters is based on infinite precision. However, all hardware implementation numbers such as filter coefficients, products, and sums are stored in finite-length registers. Hence, necessary quantization and signal scaling are required. In section 2.1.2, we have discussed that an arbitrary number can be represented with a scale factor X_m ($X_m > 1$) and a binary fractional part. In fixed-point computations, the scale factor is assumed $X_m = 2^k$. A value k implies that the binary point is located between b_{L-1-k} and $b_{L-1-(k+1)}$ for an L-bit number $b_{L-1}b_{L-2} \ldots b_0$ where b_{L-1} is the sign bit. Signal scaling helps to determine parameter k; quantizing technique helps to determine appropriate word length L.

To simplify hardware design and reduce design time, the system is simulated under the following assumptions:

1. Number representation: 2's complement
2. Quantization mode: rounding
3. Signal scaling mode: safe scaling

3.4.1 Signal Scaling

When processing digital filter, either signal amplitude or filter coefficient values can be scaled to avoid overflows. Since QPSK and 16QAM are employed in our design, the dynamic range of signal amplitude on I/Q channel is set from -3 to 3, which is obtained from the worst case (16QAM-constellation mapping).

On the transmitter side, the input of the pulse-shaping filter upper bounded by 3 is represented by 3 bits in 2's complement. To find the optimal filter coefficients without overflows, according to Eq (2.15), we need to make:

$$\sum_{k=0}^{n-1} |f(k)| \leq 1 \tag{3.1}$$

However, for the pulse-shaping filter and the interpolation filter, there is an efficient way to choose coefficients big enough without overflows. As we know, an interpolation filter input is stuffed with L-1 zeros before filter processing. This means that only $1/L$ filter coefficients (one out of L sub-filters) are in computation each time. Hence, Eq (3.1) could be modified as:

$$\sum_{k=0}^{\frac{n-1}{L}} |f(Lk + m)| \leq 1 \tag{3.2}$$

where L is the interpolation factor, and m is a positive integer less than L. For instance, the pulse-shaping filter with 33 coefficients stuffs 3 zeros in its input before processing. To optimize its coefficients with no overflows at the filter output, we only need to guarantee the following condition:

$$Max\left(\sum_{k=0}^{8}|f(4k)|, \sum_{k=0}^{7}|f(4k+1)|, \sum_{k=0}^{7}|f(4k+2)|, \sum_{k=0}^{7}|f(4k+3)|\right) \leq 1 \tag{3.3}$$

Based on Eq (3.3), the pulse-shaping coefficients can be chosen. The maximum sub-filter sum for pulse-shaping filter coefficients is upper-bounded by 0.73. For the interpolation filter, the same rule can be applied. Meanwhile, we need the interpolation filter with gain equal to L in order to remain key values unchanged.

The boundary of each block in the transmitter side can be shown as follows:

- Magnitude of QAM mapping output: $M_Q \leq 3$

- Magnitude of pulse-shaping filter output: $M_P \leq M_Q$ *0.73 = 2.19 (0.73: the maximum sub-filter sum for the pulse-shaping filter)
- Magnitude of interpolation filter: $M_I \leq M_P$ *1.39 = 3.04 (1.39: the maximum sub-filter sum for the interpolator)
- Magnitude of IF modulator: $M_C \leq M_I$ *1 (1: the maximum sine or cosine values)
- Magnitude of adder: $M_A \leq M_C$ *1.414 = 4.3 (1.414: the maximum (sin$x$+cos$x$) value)

In the receiver side, the received signal needs to be converted to digital signal via ADC. Since the input signal amplitude is known, the decimation filter coefficients are chosen based on Eq (2.15). For the matched filter, the same coefficients as the pulse-shaping filter are selected. The boundary of each block can also be shown as follows:

- Magnitude of IF recovery block: $M_R \leq 4.3$ (after ADC)
- Magnitude of decimation filter: $M_D \leq M_R$*0.5*1.25 = 2.69 (1.25: $1/L*\sum|f(k)|$; 0.5: caused by LPF)
- Magnitude of matched filter: $M_M \leq M_D$*2.76 = 7.42 (2.76: $\sum|f(k)|$)

3.4.2 System Quantization

An L-bit number's precision depends on its fractional part. If the scale factor of the number is determined, the only way to make the number more precise is to extend L until the requirements are met. Generally, coefficients, products and sums can have different word lengths based on design requirements. In our design, the word length L is set identical. It is examined using MATLAB simulation model and determined based on the system performance. In Fig 3-6, 3 BER comparisons with the word length L equal to 8,

12 and 16 bits are plotted. Note that all simulations are made for 16-QAM modulation scheme since it is more vulnerable than QPSK.

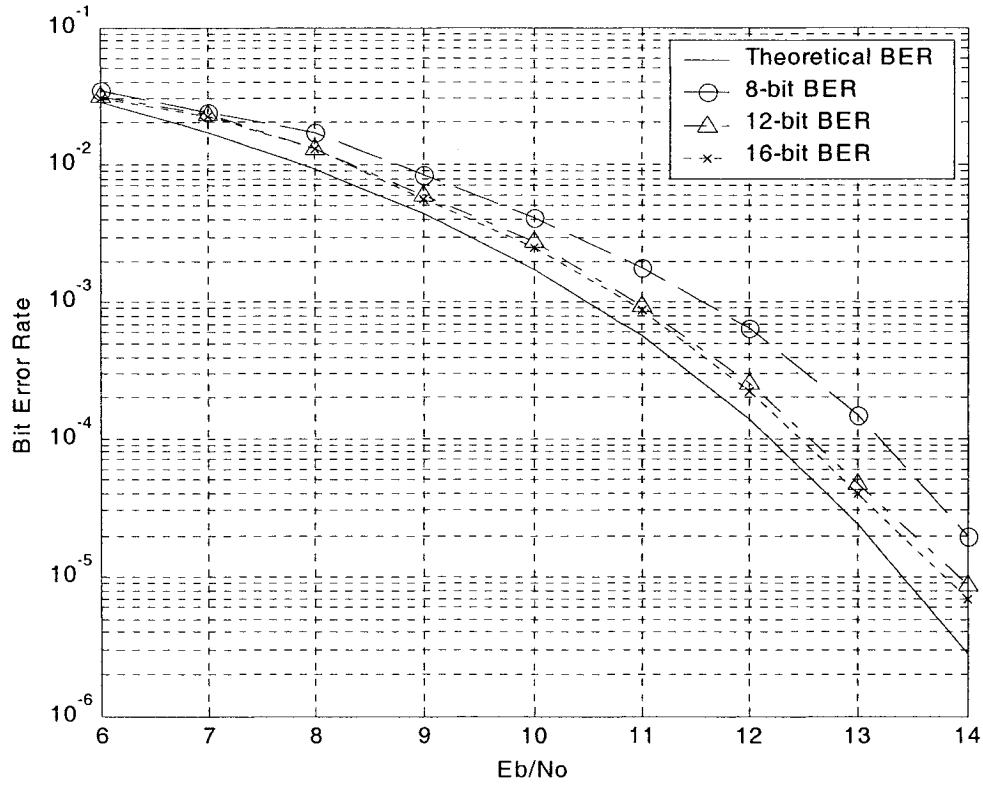


Figure 3-6 16-QAM BERs with different word lengths

By changing the interpolation factor, almost the same results are obtained. Therefore, the word length L of 12 bits is chosen for future hardware design. Hence, the word length of each block for the transmitter and receiver is given as Table 3-2 and 3-3:

Table 3-2 Word lengths for each block of the transmitter

	Mapper	Pulse-shaping filter	Interpolator	IF modulator	Adder
Coefficient	[3 0]	[12 11]	[12 11]	[12 11]	--
Input	bit-in	[3 0]	[12 9]	[12 9]	[12 9]
Output	[3 0]	[12 9]	[12 9]	[12 9]	[12 8]

In term [a b], a stands for the total word length, b stands for the fraction length.

Table 3-3 Word lengths for each block of the receiver

	IF recovery	Decimator	Matched filter	Detector	Demapper
Coefficient	[12 11]	[12 11]	[12 11]	--	--
Input	[12 8]	[12 8]	[12 9]	[12 7]	[3 0]
Output	[12 8]	[12 9]	[12 7]	[3 0]	bit-out

In order to make a comparison, the quantized and non-quantized frequency spectrums of the square-root raised cosine filter are plotted in Fig 3-7.

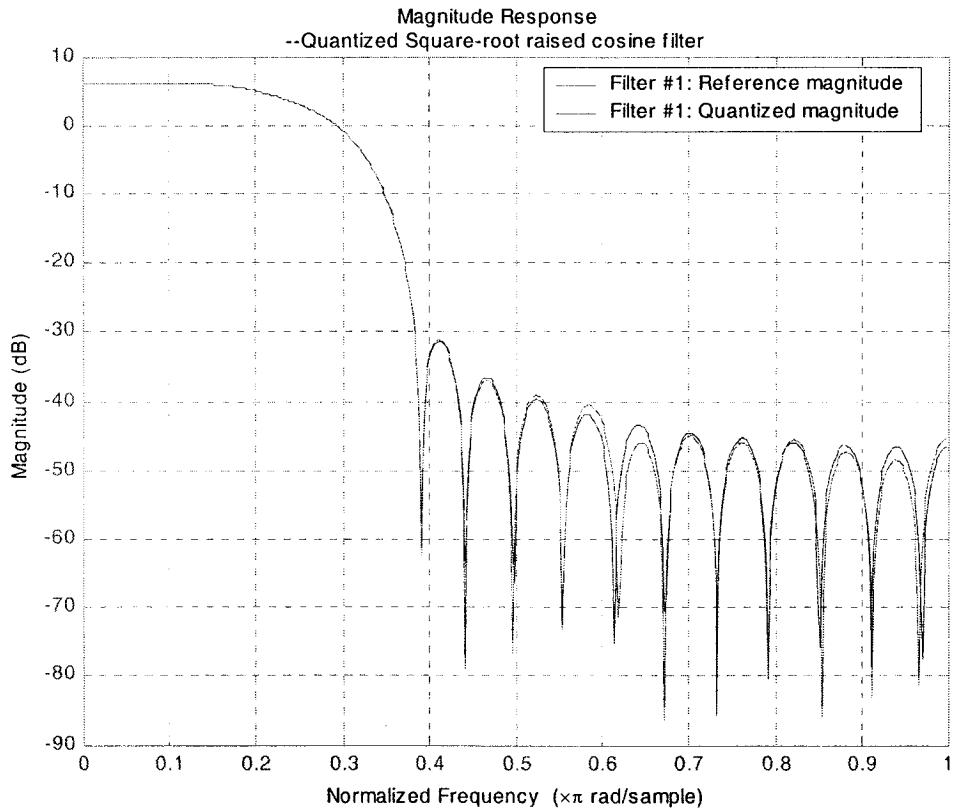


Figure 3-7 Frequency response of the quantized square-root raised cosine filter

As well, the quantized frequency spectrums of the interpolation filters with different interpolation factor are plotted in Fig 3-8.

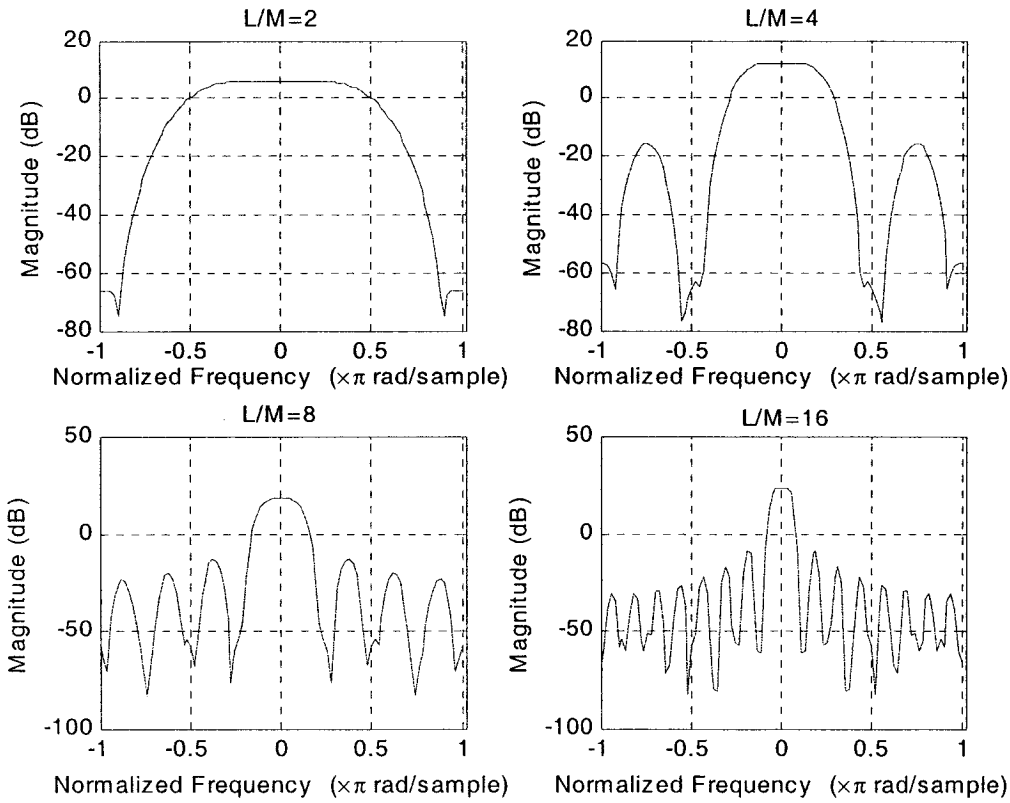


Figure 3-8 Quantized frequency response of interpolator/decimator

3.5 Coefficients and 2's Complement Representations

In our design, the blocks that are Constellation mapper, Pulse-shaping filter/Matched filter, Interpolator/Decimator and IF modualtor need coefficients pre-defined and pre-stored. The quantized coefficient values and their 2's complement representation (based on Eq (2.4)) can be listed as follows.

- In Table 3-4, the mapping output of QPSK scheme I/Q is extended 1 bit to make the same word length as that of 16QAM. This is to simplify hardware implementation.
- Table 3-5 shows Pulse-shaping/ Matched filter coefficients.

- Table 3-6 and 3-7 show the interpolator (gain=L) and decimator (gain=1) coefficients, where L=2, 4, 8, 16, respectively.
- Table 3-8 shows Cosine waveform coefficients. Since Cosine and Sinusoid have 90° offset, only Cosine waveform coefficients are tabulated.

Table 3-4 QPSK/16-QAM constellation mapping

QPSK (I/Q: 2+1-BIT)					16-QAM (I/Q: 3-BIT)				
INPUT	I		Q		INPUT	I		Q	
	Decimal	2's complement	Decimal	2's complement		Decimal	2's complement	Decimal	2's complement
00	1	001	1	001	0000	1	001	1	001
01	-1	111	1	001	0001	-1	111	1	001
10	1	001	-1	111	0010	1	001	-1	111
11	-1	111	-1	111	0011	-1	111	-1	111
					0100	3	011	1	001
					0101	-3	101	1	001
					0110	3	011	-1	111
					0111	-3	101	-1	111
					1000	1	001	3	011
					1001	-1	111	3	011
					1010	1	001	-3	101
					1011	-1	111	-3	101
					1100	3	011	3	011
					1101	-3	101	3	011
					1110	3	011	-3	101
					1111	-3	101	-3	101

Table 3-5 Pulse-shaping/ Matched filter coefficients

No.	Coefficients in decimal	Coeficients in 2's complement (12-bit)
h(0)/h(32)	-0.0049	111111110110
h(1)h(31)	-0.002	111111111100
h(2)h(30)	0.0054	000000001011
h(3)/h(29)	0.0083	000000010001
h(4)/h(28)	0.0015	000000000011
h(5)/h(27)	-0.0083	111111101111
h(6)/h(26)	-0.0073	111111110001
h(7)/h(25)	0.0078	000000010000
h(8)/h(24)	0.021	000000101011
h(9)/h(23)	0.0078	000000010000
h(10)/h(22)	-0.0376	111110110011
h(11)/h(21)	-0.0786	111101011111
h(12)/h(20)	-0.0532	111110010011
h(13)/h(19)	0.0786	000010100001
h(14)/h(18)	0.2896	001001010001
h(15)/h(17)	0.4873	001111100110
h(16)	0.5684	010010001100

Table 3-6 Interpolator coefficients

L=2 N=12 No.	Coefficients in decimal	Coefficients in 2's complement
0	0	000000000000
1	0.0117	000000011000
2	0	000000000000
3	-0.0977	111100111000
4	0	000000000000
5	0.5859	010010110000
6	0.9995	011111111111
7	0.5859	010010110000
8	0	000000000000
9	-0.0977	111100111000
10	0	000000000000
11	0.0117	000000011000

L=4 N=24 No.	Coefficients in decimal	Coefficients in 2's complement
0	0	000000000000
1	0.0078	000000010000
2	0.0117	000000011000
3	0.0093	000000010011
4	0	000000000000
5	-0.0605	111110000100
6	-0.0977	111100111000
7	-0.0845	111101010011
8	0	000000000000
9	0.2822	001001000010
10	0.5859	010010110000
11	0.8462	011011000101
12	0.9995	011111111111
13	0.8462	011011000101
14	0.5859	010010110000
15	0.2822	001001000010
16	0	000000000000
17	-0.0845	111101010011
18	-0.0977	111100111000
19	-0.0605	111110000100
20	0	000000000000
21	0.0093	000000010011
22	0.0117	000000011000
23	0.0078	000000010000

L=8 N=48 No.	Coefficients in decimal	Coefficients in 2's complement
0	0	000000000000
1	0.0039	000000001000
2	0.0078	000000010000
3	0.0103	000000010101
4	0.0117	000000011000
5	0.0112	000000010111
6	0.0093	000000010011
7	0.0054	000000001011
8	0	000000000000
9	-0.0313	111111000000
10	-0.0605	111110000100
11	-0.0835	111101010101
12	-0.0977	111100111000
13	-0.0991	111100110101
14	-0.0845	111101010011
15	-0.0522	111110010101
16	0	000000000000
17	0.1343	000100010011
18	0.2822	001001000010
19	0.4355	001101111100
20	0.5859	010010110000
21	0.7256	010111001110
22	0.8462	011011000101
23	0.9395	011110000100
24	0.9995	011111111111
25	0.9395	011110000100
26	0.8462	011011000101
27	0.7256	010111001110
28	0.5859	010010110000
29	0.4355	001101111100
30	0.2822	001001000010
31	0.1343	000100010011
32	0	000000000000
33	-0.0522	111110010101
34	-0.0845	111101010011
35	-0.0991	111100110101
36	-0.0977	111100111000
37	-0.0835	111101010101
38	-0.0605	111110000100
39	-0.0313	111111000000
40	0	000000000000
41	0.0054	000000001011
42	0.0093	000000010011
43	0.0112	000000010111
44	0.0117	000000011000
45	0.0103	000000010101
46	0.0078	000000010000
47	0.0039	000000001000

L=16N =96 No.	Coefficients in decimal	Coefficients in 2's complement
0	0	000000000000
1	0.002	000000000100
2	0.0039	000000001000
3	0.0059	000000001100
4	0.0078	000000010000
5	0.0093	000000010011
6	0.0103	000000010101
7	0.0112	000000010111
8	0.0117	000000011000
9	0.0117	000000011000
10	0.0112	000000010111
11	0.0107	000000010110
12	0.0093	000000010011
13	0.0078	000000010000
14	0.0054	000000001011
15	0.0029	000000000110
16	0	000000000000
17	-0.0156	111111100000
18	-0.0313	111111000000
19	-0.0464	111110100001
20	-0.0605	111110000100
21	-0.0732	111101101010
22	-0.0835	111101010101
23	-0.0923	111101000011
24	-0.0977	111100111000
25	-0.1001	111100110011
26	-0.0991	111100110101
27	-0.0938	111101000000
28	-0.0845	111101010011
29	-0.0708	111101101111
30	-0.0522	111110010101
31	-0.0288	111111000101
32	0	000000000000
33	0.0649	000010000101
34	0.1343	000100010011
35	0.207	000110101000
36	0.2822	001001000010
37	0.3584	001011011110
38	0.4355	001101111100
39	0.5117	010000011000
40	0.5859	010010110000
41	0.6577	010101000011
42	0.7256	010111001110
43	0.7886	011001001111
44	0.8462	011011000101
45	0.8965	011100101100
46	0.9395	011110000100
47	0.9746	011111001100

L=16 N=96 No.	Coefficients in decimal	Coefficients in 2's complement
48	0.9995	011111111111
49	0.9746	011111001100
50	0.9395	011110000100
51	0.8965	011100101100
52	0.8462	011011000101
53	0.7886	011001001111
54	0.7256	010111001110
55	0.6577	010101000011
56	0.5859	010010110000
57	0.5117	010000011000
58	0.4355	001101111100
59	0.3584	001011011110
60	0.2822	001001000010
61	0.207	000110101000
62	0.1343	000100010011
63	0.0649	000010000101
64	0	000000000000
65	-0.0288	111111000101
66	-0.0522	111110010101
67	-0.0708	111101101111
68	-0.0845	111101010011
69	-0.0938	111101000000
70	-0.0991	111100110101
71	-0.1001	111100110011
72	-0.0977	111100111000
73	-0.0923	111101000011
74	-0.0835	111101010101
75	-0.0732	111101101010
76	-0.0605	111110000100
77	-0.0464	111110100001
78	-0.0313	111111000000
79	-0.0156	111111100000
80	0	000000000000
81	0.0029	000000000110
82	0.0054	000000001011
83	0.0078	000000010000
84	0.0093	000000010011
85	0.0107	000000010110
86	0.0112	000000010111
87	0.0117	000000011000
88	0.0117	000000011000
89	0.0112	000000010111
90	0.0103	000000010101
91	0.0093	000000010011
92	0.0078	000000010000
93	0.0059	000000001100
94	0.0039	000000001000
95	0.002	000000000100

Table 3-7 Decimator coefficients

L=2 N=12 No.	Coefficients in decimal	Coefficients in 2's complement
0	0	000000000000
1	0.0059	000000001100
2	0	000000000000
3	-0.0488	111110011100
4	0	000000000000
5	0.293	001001011000
6	0.5	010000000000
7	0.293	001001011000
8	0	000000000000
9	-0.0488	111110011100
10	0	000000000000
11	0.0059	000000001100

L=4 N=24 No.	Coefficients in decimal	Coefficients in 2's complement
0	0	000000000000
1	0.002	000000000100
2	0.0029	000000000110
3	0.0024	000000000101
4	0	000000000000
5	-0.0151	111111100001
6	-0.0244	111111001110
7	-0.021	111111010101
8	0	000000000000
9	0.0703	000010010000
10	0.1465	000100101100
11	0.2114	000110110001
12	0.25	001000000000
13	0.2114	000110110001
14	0.1465	000100101100
15	0.0703	000010010000
16	0	000000000000
17	-0.021	111111010101
18	-0.0244	111111001110
19	-0.0151	111111100001
20	0	000000000000
21	0.0024	000000000101
22	0.0029	000000000110
23	0.002	000000000100

L=8 N=48 No.	Coefficients in decimal	Coefficients in 2's complement
0	0	000000000000
1	0.0005	000000000001
2	0.001	000000000010
3	0.0015	000000000011
4	0.0015	000000000011
5	0.0015	000000000011
6	0.001	000000000010
7	0.0005	000000000001
8	0	000000000000
9	-0.0039	111111111000
10	-0.0073	111111110001
11	-0.0103	111111101011
12	-0.0122	111111100111
13	-0.0122	111111100111
14	-0.0107	111111101010
15	-0.0063	111111110011
16	0	000000000000
17	0.0166	000000100010
18	0.0352	000001001000
19	0.0542	000001101111
20	0.0732	000010010110
21	0.0908	000010111010
22	0.106	000011011001
23	0.1177	000011110001
24	0.125	000100000000
25	0.1177	000011110001
26	0.106	000011011001
27	0.0908	000010111010
28	0.0732	000010010110
29	0.0542	000001101111
30	0.0352	000001001000
31	0.0166	000000100010
32	0	000000000000
33	-0.0063	111111110011
34	-0.0107	111111101010
35	-0.0122	111111100111
36	-0.0122	111111100111
37	-0.0103	111111101011
38	-0.0073	111111110001
39	-0.0039	111111111000
40	0	000000000000
41	0.0005	000000000001
42	0.001	000000000010
43	0.0015	000000000011
44	0.0015	000000000011
45	0.0015	000000000011
46	0.001	000000000010
47	0.0005	000000000001

L=16 N=96 No.	Coefficients in decimal	Coefficients in 2's complement
0	0	000000000000
1	0	000000000000
2	0.0005	000000000001
3	0.0005	000000000001
4	0.0005	000000000001
5	0.0005	000000000001
6	0.0005	000000000001
7	0.0005	000000000001
8	0.001	000000000010
9	0.001	000000000010
10	0.0005	000000000001
11	0.0005	000000000001
12	0.0005	000000000001
13	0.0005	000000000001
14	0.0005	000000000001
15	0	000000000000
16	0	000000000000
17	-0.001	111111111110
18	-0.002	111111111100
19	-0.0029	111111111010
20	-0.0039	111111111000
21	-0.0044	111111110111
22	-0.0054	111111110101
23	-0.0059	111111110100
24	-0.0059	111111110100
25	-0.0063	111111110011
26	-0.0063	111111110011
27	-0.0059	111111110100
28	-0.0054	111111110101
29	-0.0044	111111110111
30	-0.0034	111111111001
31	-0.002	111111111100
32	0	000000000000
33	0.0039	000000001000
34	0.0083	000000010001
35	0.0127	000000011010
36	0.0176	000000100100
37	0.0225	000000101110
38	0.0273	000000111000
39	0.0317	000001000001
40	0.0366	000001001011
41	0.041	000001010100
42	0.0454	000001011101
43	0.0493	000001100101
44	0.0527	000001101100
45	0.0562	000001110011
46	0.0586	000001111000
47	0.061	000001111101

L=16 N=96 No.	Coefficients in decimal	Coefficients in 2's complement
48	0.0625	000010000000
49	0.061	000001111101
50	0.0586	000001111000
51	0.0562	000001110011
52	0.0527	000001101100
53	0.0493	000001100101
54	0.0454	000001011101
55	0.041	000001010100
56	0.0366	000001001011
57	0.0317	000001000001
58	0.0273	000000111000
59	0.0225	000000101110
60	0.0176	000000100100
61	0.0127	000000011010
62	0.0083	000000010001
63	0.0039	000000001000
64	0	000000000000
65	-0.002	111111111100
66	-0.0034	111111111001
67	-0.0044	111111110111
68	-0.0054	111111110101
69	-0.0059	111111110100
70	-0.0063	111111110011
71	-0.0063	111111110011
72	-0.0059	111111110100
73	-0.0059	111111110100
74	-0.0054	111111110101
75	-0.0044	111111110111
76	-0.0039	111111111000
77	-0.0029	111111111010
78	-0.002	111111111100
79	-0.001	111111111110
80	0	000000000000
81	0	000000000000
82	0.0005	000000000001
83	0.0005	000000000001
84	0.0005	000000000001
85	0.0005	000000000001
86	0.0005	000000000001
87	0.001	000000000010
88	0.001	000000000010
89	0.0005	000000000001
90	0.0005	000000000001
91	0.0005	000000000001
92	0.0005	000000000001
93	0.0005	000000000001
94	0.0005	000000000001
95	0	000000000000

Table 3-8 IF modulator (cosine) coefficients

N=4 No.	Coefficients in decimal	Coefficients in 2's complement
0	-1	100000000000
1	0	000000000000
2	0.9995	011111111111
3	0	000000000000

N=8 No.	Coefficients in decimal	Coefficients in 2's complement
0	-1	100000000000
1	-0.707	101001011000
2	0	000000000000
3	0.707	010110101000
4	0.9995	011111111111
5	0.707	010110101000
6	0	000000000000
7	-0.707	101001011000

N=16 No.	Coefficients in decimal	Coefficients in 2's complement
0	-1	100000000000
1	-0.9238	100010011100
2	-0.707	101001011000
3	-0.3828	110011110000
4	0	000000000000
5	0.3828	001100010000
6	0.707	010110101000
7	0.9238	011101100100
8	0.9995	011111111111
9	0.9238	011101100100
10	0.707	010110101000
11	0.3828	001100010000
12	0	000000000000
13	-0.3828	110011110000
14	-0.707	101001011000
15	-0.9238	100010011100

N=32 No.	Coefficients in decimal	Coefficients in 2's complement
0	-1	100000000000
1	-0.981	100000100111
2	-0.9238	100010011100
3	-0.8315	100101011001
4	-0.707	101001011000
5	-0.5557	101110001110
6	-0.3828	110011110000
7	-0.1953	111001110000
8	0	000000000000
9	0.1953	000110010000
10	0.3828	001100010000
11	0.5557	010001110010
12	0.707	010110101000
13	0.8315	011010100111
14	0.9238	011101100100
15	0.981	011111011001
16	0.9995	011111111111
17	0.981	011111011001
18	0.9238	011101100100
19	0.8315	011010100111
20	0.707	010110101000
21	0.5557	010001110010
22	0.3828	001100010000
23	0.1953	000110010000
24	0	000000000000
25	-0.1953	111001110000
26	-0.3828	110011110000
27	-0.5557	101110001110
28	-0.707	101001011000
29	-0.8315	100101011001
30	-0.9238	100010011100
31	-0.981	100000100111
48	0	000000000000
49	-0.0981	111100110111
50	-0.1953	111001110000
51	-0.2905	110110101101
52	-0.3828	110011110000
53	-0.4712	110000111011
54	-0.5557	101110001110
55	-0.6343	101011101101
56	-0.707	101001011000
57	-0.7729	100111010001
58	-0.8315	100101011001
59	-0.8818	100011110010
60	-0.9238	100010011100
61	-0.957	100001011000
62	-0.981	100000100111
63	-0.9951	100000001010

N=64 No.	Coefficients in decimal	Coefficients in 2's complement
0	-1	100000000000
1	-0.9951	100000001010
2	-0.981	100000100111
3	-0.957	100001011000
4	-0.9238	100010011100
5	-0.8818	100011110010
6	-0.8315	100101011001
7	-0.7729	100111010001
8	-0.707	101001011000
9	-0.6343	101011101101
10	-0.5557	101110001110
11	-0.4712	110000111011
12	-0.3828	110011110000
13	-0.2905	110110101101
14	-0.1953	111001110000
15	-0.0981	111100110111
16	0	000000000000
17	0.0981	000011001001
18	0.1953	000110010000
19	0.2905	001001010011
20	0.3828	001100010000
21	0.4712	001111000101
22	0.5557	010001110010
23	0.6343	010100010011
24	0.707	010110101000
25	0.7729	011000101111
26	0.8315	011010100111
27	0.8818	011100001110
28	0.9238	011101100100
29	0.957	011110101000
30	0.981	011111011001
31	0.9951	011111110110
32	0.9995	011111111111
33	0.9951	011111110110
34	0.981	011111011001
35	0.957	011110101000
36	0.9238	011101100100
37	0.8818	011100001110
38	0.8315	011010100111
39	0.7729	011000101111
40	0.707	010110101000
41	0.6343	010100010011
42	0.5557	010001110010
43	0.4712	001111000101
44	0.3828	001100010000
45	0.2905	001001010011
46	0.1953	000110010000
47	0.0981	000011001001

Chapter 4 Hardware Design Techniques

Our system consists of several primary components. On the transmitter side, there are constellation mapper, pulse-shaping filters, interpolation filters, up converters and a waveform adder. The receiver includes down converters, matched filters, decimation filters, and QAM detector. In this chapter, the constellation mapping and its lookup table (LUT) are first discussed. Section 4.2 focuses on the poly-phase structure of digital FIR filters. In section 4.3, a high-speed carry-select-adder, carry-lookahead adder (CSA-CLA) and a new signed-array multiplier are developed. At the end, the relationships among hardware implementation, the carrier frequency f_c and the baud rate R_s are revealed.

4.1 Quadrature Signal Constellation

In the QAM modulator, the data stream is mapped to signal points on the constellation diagram by assigning proper I & Q values. Fig 4-1 shows the rectangular QPSK and 16-QAM signal constellation diagrams. Note that Gray-encoding is already employed for the constellation assignments.

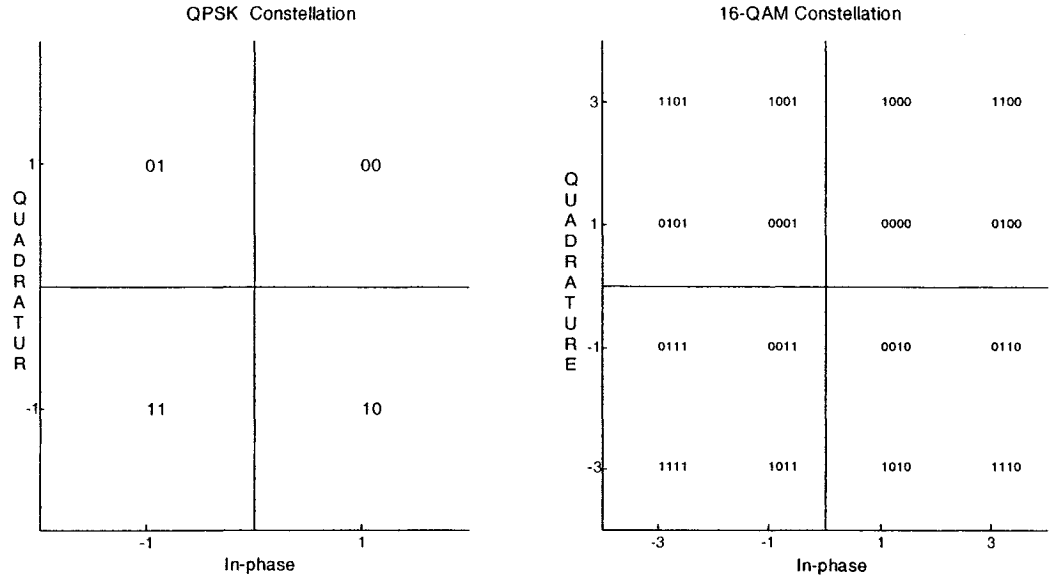
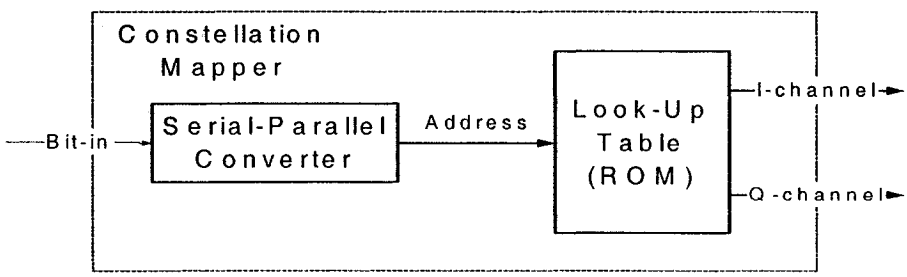


Figure 4-1 Rectangular QPSK/16-QAM signal constellations

In Fig 4-1, each number from 0 to M-1 (M=4 for QPSK and 16 for 16-QAM) represents a possible data group taking $log_2(M)$ bits at a time from the input stream. The corresponding values on the abscissa (I-channel) and the ordinate (Q-channel) are the mapped outputs. For example, if the input data group of 16-QAM is "1010", the outputs are 1 on I-channel and –3 on Q-channel. Generally, for such a set of finite points, the lookup table structure shown in Fig 4-2 is the simple, efficient and hardware saving



solution.

Figure 4-2 Lookup table for QAM mapping

4.2 FIR Filter Design

In the modem, the modulator has 2 types of filters, the pulse-shaping filter and the interpolation filter. The demodulator has also 2 types of filters, the matched filter and the decimation filter. Generally, designers develop these filters in different ways. For instance, the lookup table could be used for implementing the pulse-shaping filter, and the constant coefficient multiplier (KCM) could be used for implementing the matched filter. In our design, we use a classical but efficient structure called "poly-phase" to build up all the four filters of the modem. This significantly simplifies the design methodology.

4.2.1 Transversal Structure

In general, an FIR filter structure shown in Fig 4-3, is called a transversal structure or direct form as it is a direct implementation of the convolution of the input sequence and a finite impulse response. In this figure, Z^{1} stands for unit delay, and *h(0), h(1), ...h(N-1)* stand for the filter coefficients.

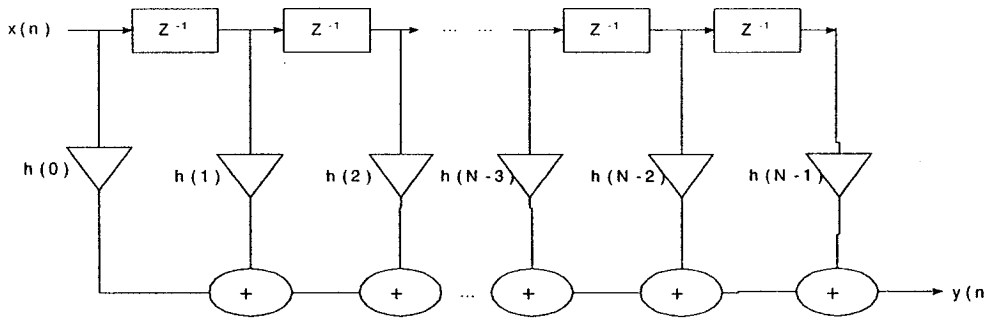


Figure 4-3 Transversal structure of a FIR filter

Figure 4-3 gives the form of signal flow, which clearly shows how signals in a digital filter are processed and combined. This represents a basis for programming signal processors or designing integrated circuits.

Implementation of decimators or interpolators based on the basic transversal structure is not efficient. Fig 4-4 demonstrates a decimator based on the transversal structure and its signal convolution process. Assuming the use of an FIR filter with N coefficients and decimation factor M, the filtering can be described by the following convolution:

$$t(n) = x(n) \otimes h(n) = \sum_{k=0}^{N-1} h(k) \times x(n-k) \tag{4.1}$$

And the down-sampling process can be expressed as:

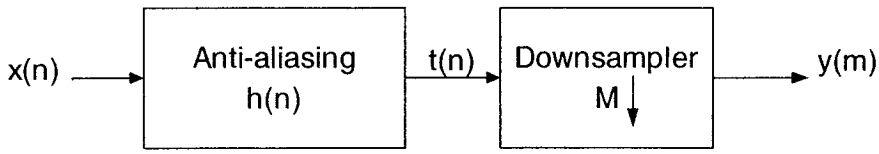
$$y(m) = t(m \cdot M) \tag{4.2}$$

According to Eq (4.2), for evaluation of $y(m)$, only the calculation of $t(n)$ for $n=m*M$ is required. Therefore, this structure is not efficient because the calculation of $t(n)$ for $n \neq m*M$ is wasteful and not used in the final output. Eq (4.3) shows the decimation as convolution followed by downsampling:

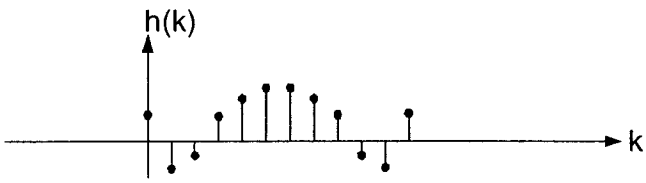
$$y(m) = \sum_{k=0}^{N-1} h(k) \times x(m \cdot M - k) \tag{4.3}$$

As an example, Fig 4-4 illustrates the convolution process corresponding to the above equations. Obviously, the multiplication of $h(k)$ with $x(1-k)$ leads to the interim value $x(1)$, which is not used later on. Hence, it is unnecessarily computed.

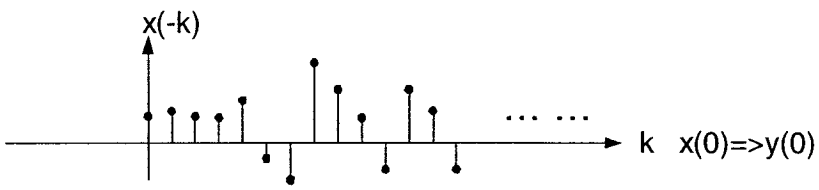
Due to different applications, many efficient structures like transposed and poly-phase are available. In our design, the filters in the modulator side are the interpolation filters, and the filters in the demodulator side are decimation filters. Hence, the poly-phase structure discussed next is the easiest but efficient structure for all the four filters.



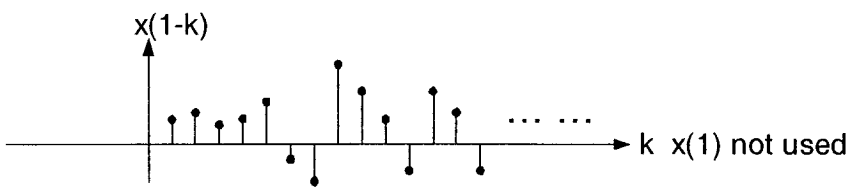
A: Decimator Model



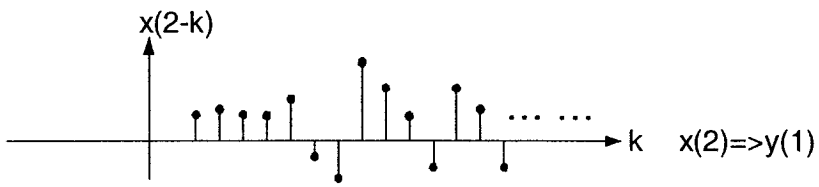
B-1: FIR filter impulse response



B-2: Reversed and shifted input signal



B-3: Reversed and shifted input signal



B-4: Reversed and shifted input signal

Figure 4-4 Decimator model (*N*=12, *M*=2)

4.2.2 Poly-phase Structure

4.2.2.1 Decimator Model

To avoid the unnecessary computation, an efficient transversal structure that combines the filter and down-sampler together called poly-phase structure can be used. Fig 4-5 is an example with down-sampling rate M=2 and filter coefficients N=12.

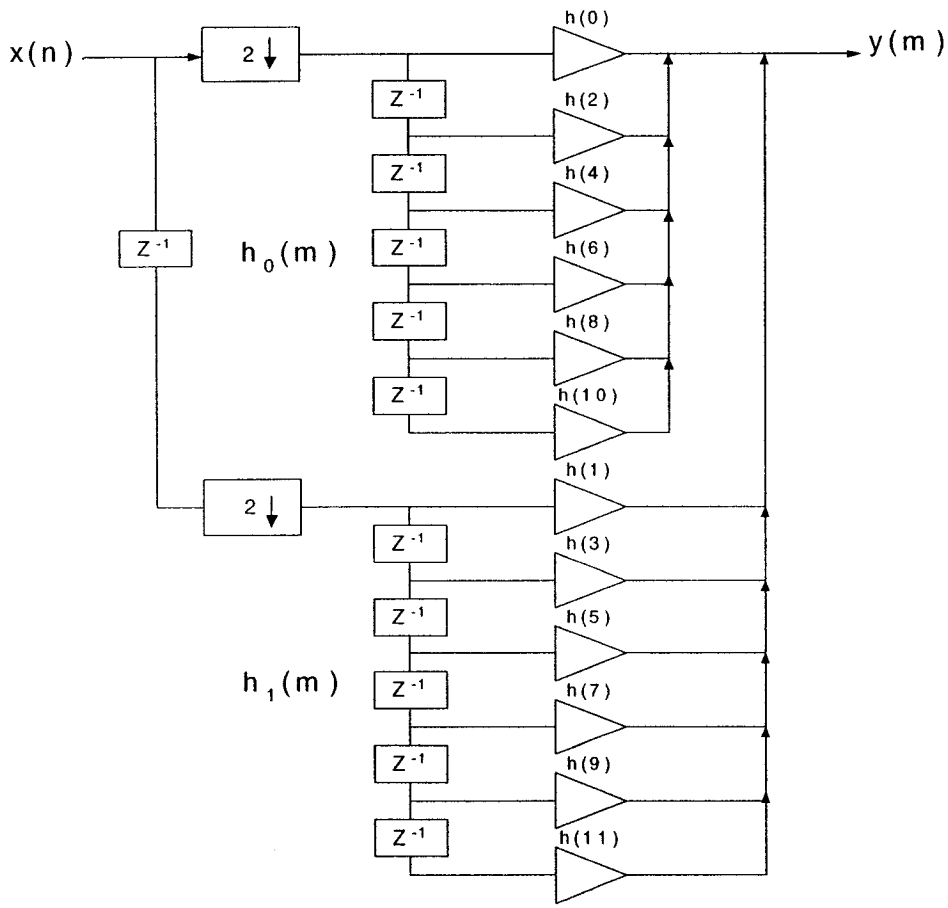


Figure 4-5 Poly-phase decimation structure (N=12, M=2)

In Fig 4-5, a traditional transversal structure can be decomposed of 2 sub-filters: $h_0(m)$ and $h_1(m)$. Subfilter $h_0(m)$ consists of $h(0)$, $h(2)$, ..., $h(10)$; subfilter $h_1(m)$ consists of $h(1)$, $h(3)$, ..., $h(11)$. In general, the decimated signal can be expressed as [12]:

$$
\begin{aligned}
y(m) &= \sum_{k=-\infty}^{\infty} h(k) \times x(m \cdot M - k) \\
&= \sum_{\lambda=0}^{M-1} \sum_{r=-\infty}^{\infty} h(rM + \lambda) \cdot x([m - r]M - \lambda) \\
&= \sum_{\lambda=0}^{M-1} \sum_{r=-\infty}^{\infty} h_{\lambda}(r) \cdot x_{\lambda}(m - r) = \sum_{\lambda=0}^{M-1} h_{\lambda}(m) \otimes x_{\lambda}(m)
\end{aligned}
\tag{4.4}
$$

where $k=r*M+\lambda; h_\lambda(r)=h(r*M+\lambda); x_\lambda(r)=x(r*M-\lambda)$.

The convolution with subsequent down-sampling can be decomposed into M independent convolutions (M sub-filters), where each convolution product is exactly assigned to one of the phase indexes $\lambda = k \bmod M$. In this method, no unnecessary calculations are performed. Since the subsequences used are basically poly-phase signals, the process is then called poly-phase filtering. A general poly-phase decimator can be plotted in an efficient way as shown in Fig 4-6 [12]. This efficient way means that the internal sampling rate could be decreased by $1/M$ if all the sub-filters process in parallel, or the memory could be decreased by $1/M$ if periodically time-varying coefficients are used.

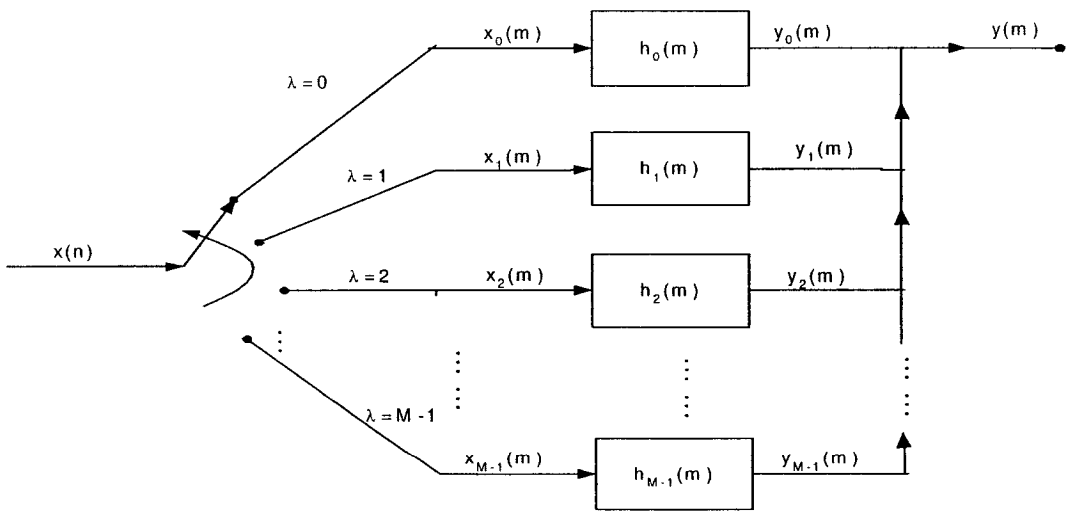


Figure 4-6 Poly-phase decimator with an input commutator rotating countclockwise

4.2.2.2 Interpolator Model

Similarly, an interpolator can also be realized by using the poly-phase structure. Its signal flow graph can be derived from that of a decimator. The directions of all signals are reversed, the inputs and outputs exchanged and the up-sampler and down-sampler swapped. As an example, Fig 4-7 shows the interpolator structure with the number of coefficients N=12 and interpolation factor L=2, which is exactly the reverse graph of Fig 4-5.

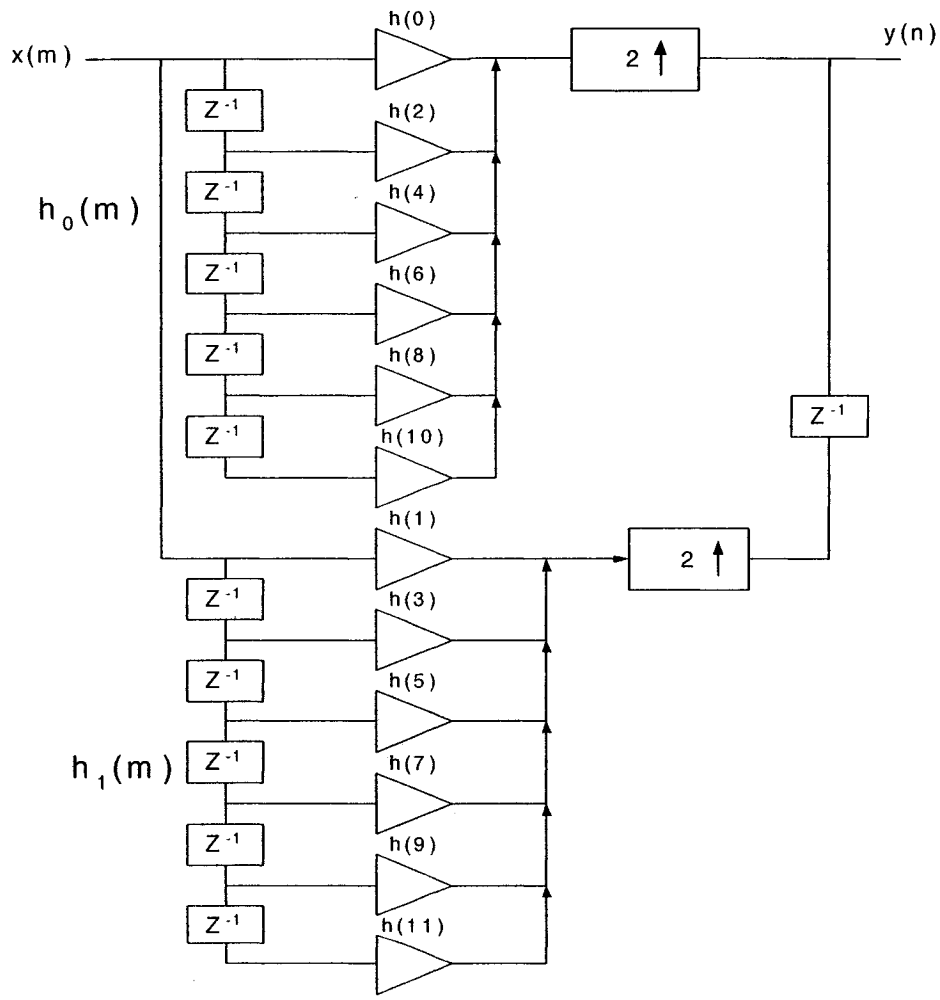


Figure 4-7 Poly-phase interpolation filter (N=12, L=2)

Generally, an interpolator with factor L can be shown in Fig 4-8.

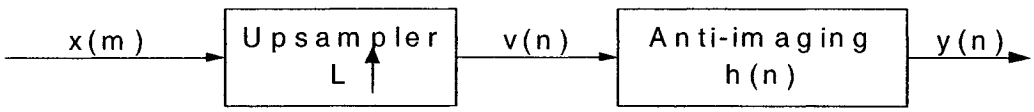


Figure 4-8 Interpolator Model

With N coefficients, the interpolator output $y(n)$ could be expressed as:

$$y(n) = v(n) \otimes h(n) = \sum_{k=0}^{N-1} h(k) \times v(n-k) \tag{4.5}$$

If the following assumptions are given:

$$N = N_s \cdot L \qquad N_s : the\ \ number\ \ of\ \ subfilter\ \ coefficien\ ts \tag{4.6}$$

$$k = r \cdot L + \lambda \qquad \lambda : subfilter\ \ index \qquad r : \text{int}\, eger \tag{4.7}$$

$$v(n-k) = v(n-rL-\lambda) = \begin{cases} x(m-r) & for \quad n = mL + \lambda \\ 0 & otherwise \end{cases} \tag{4.8}$$

Then, the general expression of the output $y(n)$ could be modified as follows[12]:

$$\begin{aligned} y(n) &= \sum_{k=0}^{N-1} h(k) \times v(n-k) \\ &= \sum_{\lambda=0}^{L-1} \sum_{r=0}^{N_s-1} h(rL+\lambda) \cdot v(n-rL-\lambda) \\ &= \sum_{\lambda=0}^{L-1} \sum_{r=0}^{N_s-1} v(n-rL-\lambda) \cdot h(rL+\lambda)_{\lambda = n \bmod L} \\ &= \sum_{\lambda=0}^{L-1} \sum_{r=0}^{N_s-1} x(m-r) \cdot h_\lambda(m) = \sum_{\lambda=0}^{L-1} x(m) \otimes h_\lambda(m) \end{aligned} \tag{4.9}$$

According to Eq (4.9), an interpolation filter with poly-phase structure can be plotted as shown in Fig 4-9.

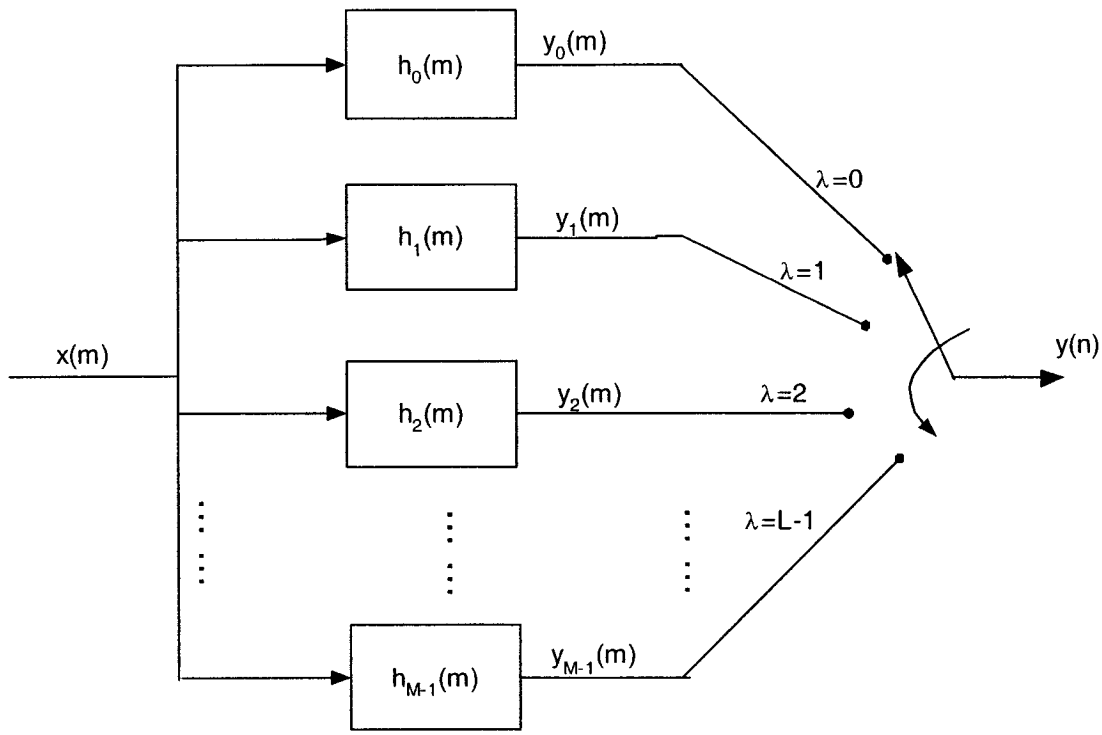


Figure 4-9 Poly-phase interpolator with an output commutator rotating countclockwise

As discussed above, the poly-phase structures can improve either speed or hardware or both. However, the basic building cells such as adders and multipliers, which will be discussed below are still needed for sub-filter implementation.

4.3 Adders & Multipliers

An adder and multiplier are two basic but critical cells in DSP-based implementation. FIR filters are constructed by adders and multipliers. Digital IF modulators are built by multipliers. Due to their wide use, building regular and modular adders and multipliers is

of great importance. In this section, a high-speed CSA-CLA adder and a new signed-array multiplier structure are introduced and developed.

4.3.1 High-Speed CSA-CLA Adder

Many types of adders such as Ripple-Carry-Adder (RCA), Carry-Look-Ahead-Adder (CLA) Carry-Select-Adder (CSA) and their combinations are widely used. In our design, a high-speed CSA-CLA adder is used to improve the whole system performance.

Addition can be viewed in terms of generate, G, and propagate, P. In hardware implementation, a 2-input addition is separated and added bit by bit as [13]:

$$\begin{aligned} G[i] &= A[i] \cdot B[i] & C[i] &= G[i] + P[i] \cdot G[i-1] \\ P[i] &= A[i] \oplus B[i] & S[i] &= P[i] \oplus C[i-1] \end{aligned} \tag{4.10}$$

where i stands for the i^{th} stage, A, B are 2 inputs and C, S are the carry and sum.

If we evaluate Eq (4.10) recursively from i=1, the following equation can be obtained:

$$\begin{aligned} C[1] &= G[1] + P[1] \cdot G[0] \\ C[2] &= G[2] + P[2] \cdot G[1] + P[2] \cdot P[1] \cdot G[0] \\ C[3] &= G[3] + P[2] \cdot G[2] + P[2] \cdot P[1] \cdot G[1] + P[3] \cdot P[2] \cdot P[1] \cdot G[0] \\ &\cdots\cdots \end{aligned} \tag{4.11}$$

This result shows that we can look ahead and calculate the carryout in advance. This is the reason why CLA adders operate fast [13].

The CLA logic becomes less regular and more complex when we look ahead further (more than 4 stages). To build a high-speed adder with large input stages, a carry select adder can be used. In this case, we duplicate 2 4-bit CLA for the cases C_{IN}=0 and C_{IN}=1, and then use a MUX (multiplexer) between each block. This structure is regular and fast at the expense of some hardware overhead. Each 4-bit CLA block is calculated simultaneously. In our design, a 12-bit high-speed CSA-CLA structure can be plotted as shown in Fig 4-10.

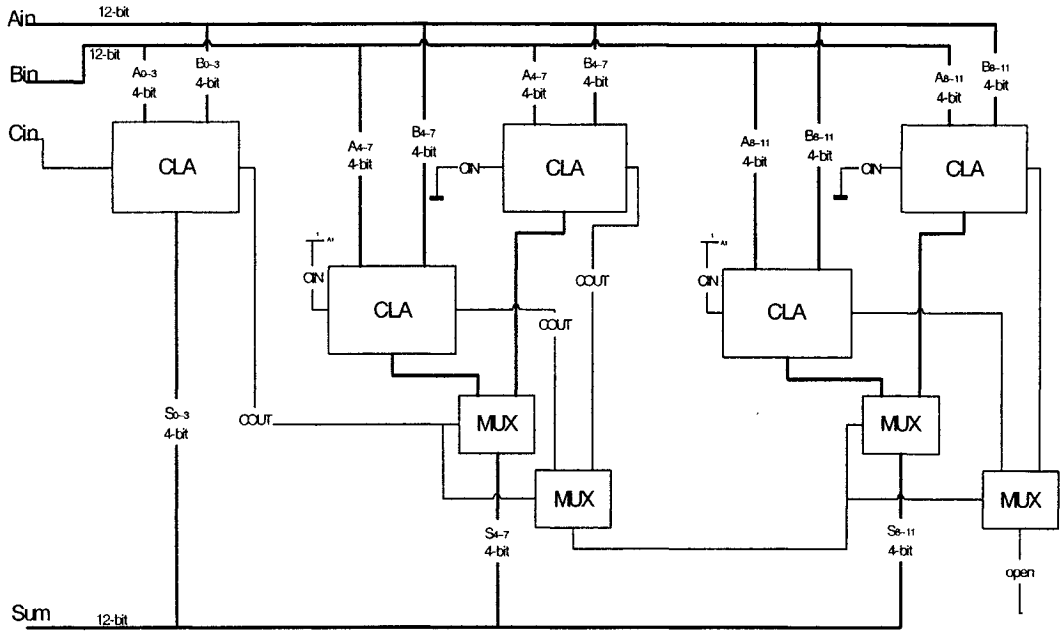


Figure 4-10 12-bit high-speed CSA-CLA adder

4.3.2 New Structure for Signed Array Multiplier

As fundamental cells in DSP-based implementation, signed multipliers are widely used in filters, mixers, etc. There are many signed multiplier structures suitable for VLSI implementation. In this section, a new structure is proposed for 2's complement multiplication and compared with other multiplier structures. It is shown that this proposed scheme has a more versatile and modular structure. Therefore, no redesign is required for different lengths of multipliers and few different types of cells are needed. Moreover, this new structure can be also applied to Booth-encoded partial products.

4.3.2.1 Currently Used Multipliers

Generally, there are 2 types of multipliers: unsigned and signed. The unsigned multiplication is very simple, just like paper-and-pencil case.

The conventional (symmetric) array structure [14] is often used for unsigned multiplication. However, in most DSP designs, 2's complement multiplication is usually required. Unlike the operands of unsigned multiplication, the multiplicand and multiplier of 2's complement multiplication have the most significant bit as their sign.

For VLSI implementation of multipliers, tree-based multipliers and array-based multipliers are well known and often used. Tree-based multipliers (Wallace tree [14, 15] and Dadda scheme [14, 16]) are focusing on decreasing the depth (time delay) of partial products processing. Array-based multipliers (Baugh-Wooley [17, 13] 2's complement array multipliers) are focusing on its modularity and regularity. However, the tree-based multipliers are less regular or modular; meanwhile, the array-based multipliers need more extra logic cells and not suitable for Booth-encoded partial products when processing a 2's complement multiplication.

In general, a multiplication consists of 2 steps: (1) Generate partial products; (2) Sum up the partial products.Generation of partial products could be processed in many ways, like conventional paper-and-pencil algorithm, Booth-encoding algorithm and Baugh-Wooley array scheme, etc. Similarly, addition of the partial products could be processed in many ways as well, such as conventional array structure, Baugh-Wooley array scheme, Wallace tree and Dadda scheme. Dadda scheme and Wallace tree can be used for both unsigned and signed multiplication. The conventional array structure is usually used for

unsigned multiplication. However, Baugh-wooley array scheme, which is a modified conventional array structure, is used for 2's complement multiplication.

In 2's complement multiplication, there are some constraints for addition of the partial products once the method of generating partial products is determined. For example, the partial products created by Booth-coding algorithm can't be applied to Baugh-Wooley array structural scheme. In general, the sign bit of partial products is to be extended to the full width of the final product. Then these extended partial products could be summed up by Wallace-tree or Dadda scheme. These schemes are not the best choices from regularity point of view when different-length multiplication is needed. Meanwhile, Baugh-Wooley array structural scheme is not the best when limited time delay is required although it is regular. Due to the drawbacks of Wallace tree, Dadda scheme and Baugh-Wooley array structural scheme, a new structure that is also derived from the conventional array structural scheme is presented in this paper. Compared to Baugh-Wooley array structure, this new structure has few types of modular cells and could reduce the number of the partial products by using booth-coding algorithm. Compared to Wallace tree and Dadda scheme, this new structure is more regular and could easily be extended to any length multiplication.

4.3.2.2 The Proposed Structure

An m-bit 2's complement number $A=(a_{m-1},a_{m-2},\cdots,a_1,a_0)$ is expressed as follows:

$$A = -a_{m-1} \times 2^{m-1} + \sum_{i=0}^{m-2} a_i \times 2^i \tag{4.12}$$

where a_{m-1} is the sign bit.

Multiplication of two 2's complement numbers $A=(a_{m-1},a_{m-2},\cdots,a_1,a_0)$ and $B=(b_{n-1},b_{n-2},\cdots,b_1,b_0)$ could be expressed as follows[18]:

$$P = AB = A\left(-b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i\right)$$

$$= A\sum_{i=0}^{n-2} b_i 2^i + (\bar{A}+1)b_{n-1}2^{n-1}$$

$$= A\sum_{i=0}^{n-2} b_i 2^i + \bar{A}b_{n-1}2^{n-1} + b_{n-1}2^{n-1} \qquad (4.13)$$

where P is the product, $\bar{A}=(\bar{a}_{m-1},\bar{a}_{m-2},\cdots,\bar{a}_1,\bar{a}_0)$ is 1's complement of $A=(a_{m-1},a_{m-2},\cdots,a_1,a_0)$ and $\bar{a}_i$ is logical inversion of a_i.

Eq. (4.13) is composed of 3 terms. The first term is the summation of $n-1$ partial products and could be expanded as follows[19]:

$$A\sum_{i=0}^{n-2} b_i 2^i = 2^0(a_{m-1},a_{m-2},\cdots,a_1,a_0)b_0 +$$

$$2^1(a_{m-1},a_{m-2},\cdots,a_1,a_0)b_1 +$$

$$2^2(a_{m-1},a_{m-2},\cdots,a_1,a_0)b_2 +$$

$$\cdots$$

$$2^{n-2}(a_{m-1},a_{m-2},\cdots,a_1,a_0)b_{n-2} \qquad (4.14)$$

And therefore,

$$A\sum_{i=0}^{n-2} b_i 2^i = 2^0(a_{m-1}b_0,a_{m-2}b_0,\cdots,a_1b_0,a_0b_0) +$$

$$2^1(a_{m-1}b_1,a_{m-2}b_1,\cdots,a_1b_1,a_0b_1) +$$

$$2^2(a_{m-1}b_2,a_{m-2}b_2,\cdots,a_1b_2,a_0b_2) +$$

. . .

$$2^{n-2}(a_{m-1}b_{n-2}, a_{m-2}b_{n-2}, \cdots, a_1b_{n-2}, a_0b_{n-2}) \qquad (4.15)$$

where a_ib_j is logical AND function of a_i and b_j.

In Eq. (4.15) multiplications by 2^i can be replaced by shifting the binary number in the bracket i bits to the left and inserting 0 at i least significant bits of the binary number. And therefore, all the terms in Eq. (4.15) will be replaced with a binary number. Next step in finding the summation would be addition of partial products which are binary numbers in 2's complement format. Rather than first extending the sign bit of each partial product to the full length of the final product, in the new structure, each sub-sum is extended only 1 bit at a time until the final stage. Therefore, we can add 1 partial product at a time to previously created carry-out and sub-sum. Addition of first two binary numbers after extending the sign bit of the first term is shown in Eq. (4.16).

$$\begin{aligned} &2^0(a_{m-1}b_0, a_{m-2}b_0, \cdots, a_1b_0, a_0b_0) + \\ &2^1(a_{m-1}b_1, a_{m-2}b_1, \cdots, a_1b_1, a_0b_1) \\ &= (a_{m-1}b_0, a_{m-1}b_0, a_{m-2}b_0, \cdots, a_2b_0, a_1b_0, a_0b_0) + \\ &(a_{m-1}b_1, a_{m-2}b_1, a_{m-3}b_1, \cdots, a_1b_1, a_0b_1, \ 0\ ) \end{aligned} \qquad (4.16)$$

Using Carry Save Adder (CSA) structure, Eq. (4.16) creates carry-out and sub-sum values. The created carry-out and sub-sum can then be added to the third partial product. As well, the sign bit of the sub-sum needs to be extended 1-bit to the left. Step by step, Eq. (4.15) can be finally evaluated using the same approach.

To obtain the final product as shown in Eq. (4.13), the carry-out and sub-sum created by Eq. (4.15) are first added to term $\overline{A}b_{n-1}2^{n-1}$ using the approach shown in Eq. (4.16).

Then, the final carryout and sub-sum are added to term $b_{n-1}2^{n-1}$ using Carry Propagate Adder (CPA) as shown in Fig 4-11. This figure shows the hardware implementation of Eq. (4.13).

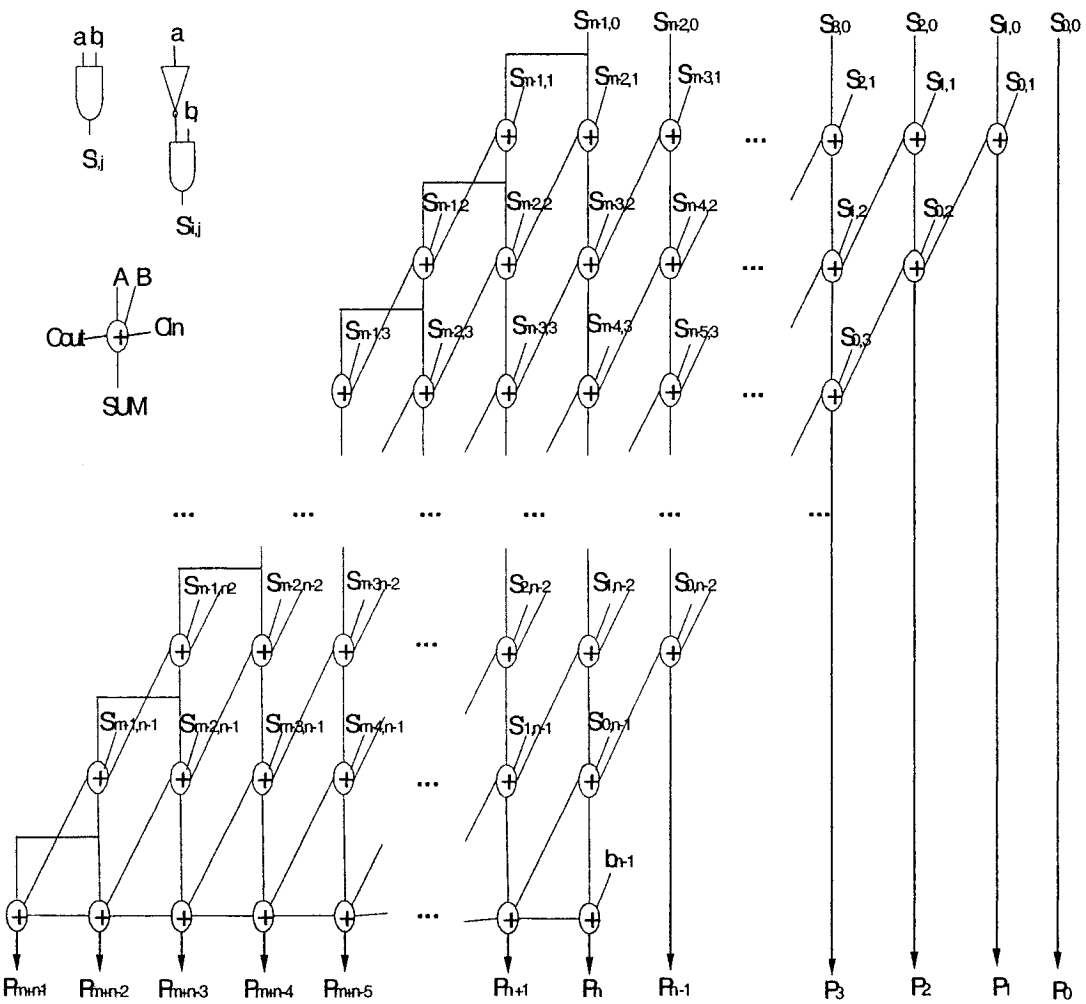


Figure 4-11 The new structure for 2's complement multiplication; Multiplicand $A=(a_{m-1},a_{m-2},\cdots,a_1,a_0)$**; Multiplier** $B=(b_{n-1},b_{n-2},\cdots,b_1,b_0)$

If the multiplication result needs to be rounded, we can easily adjust its output according to Chapter 2. Fig 4-12 shows an example of 6-bit by 6-bit signed-array multiplier with 7-bit round-off product.

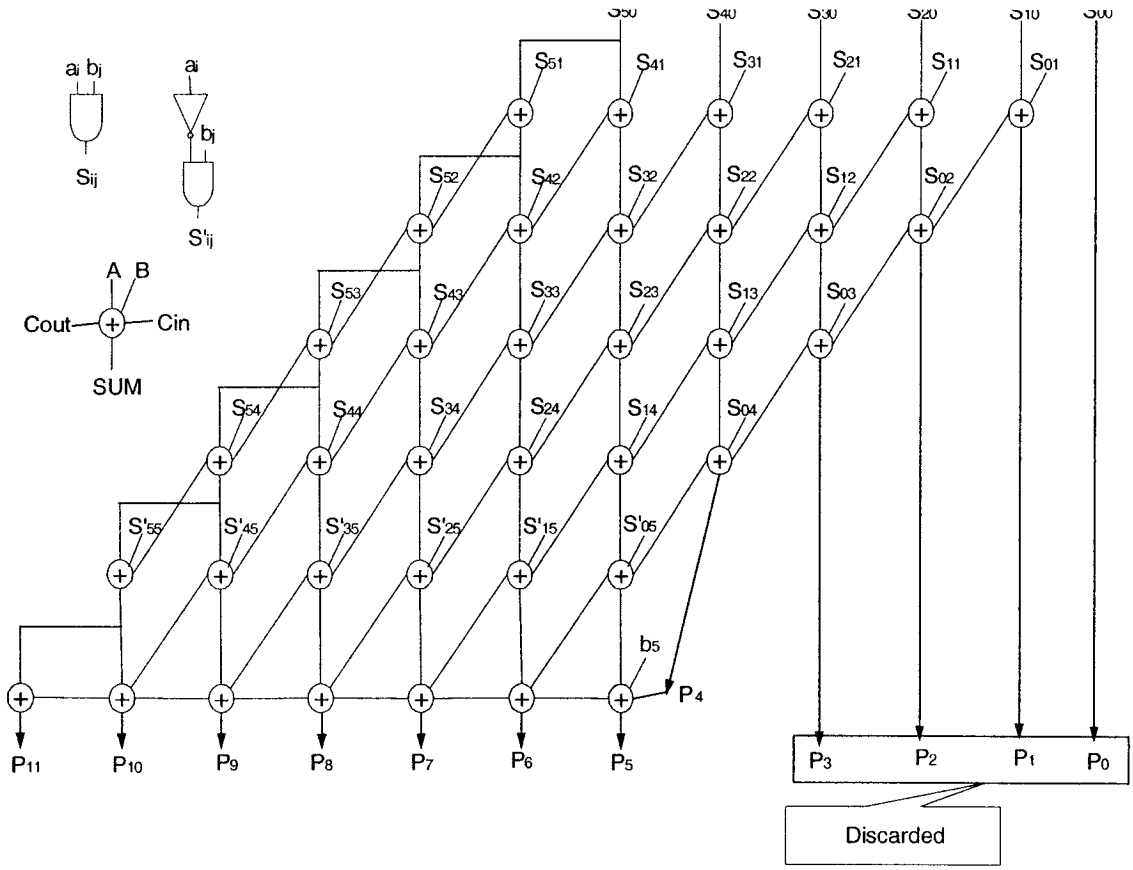


Fig 4-12 6-bit by 6-bit signed-array multiplier with 7-bit roundoff product

The new structure for Booth-encoded products:

Previously, we have discussed the algorithm for generation of direct partial products, which is quite similar to Baugh-Wooley array scheme. Furthermore, Booth-encoded partial products can be also applied to this new structure. The only difference is just replacing the direct partial products with Booth-encoded partial products. Actually, with slight modification (alignment), this structure could be suitable for the partial products created by any radix Booth-coded partial products. Fig 4-13 shows the new structure for radix 4 Booth-encoded partial products (6-bit by 6-bit).

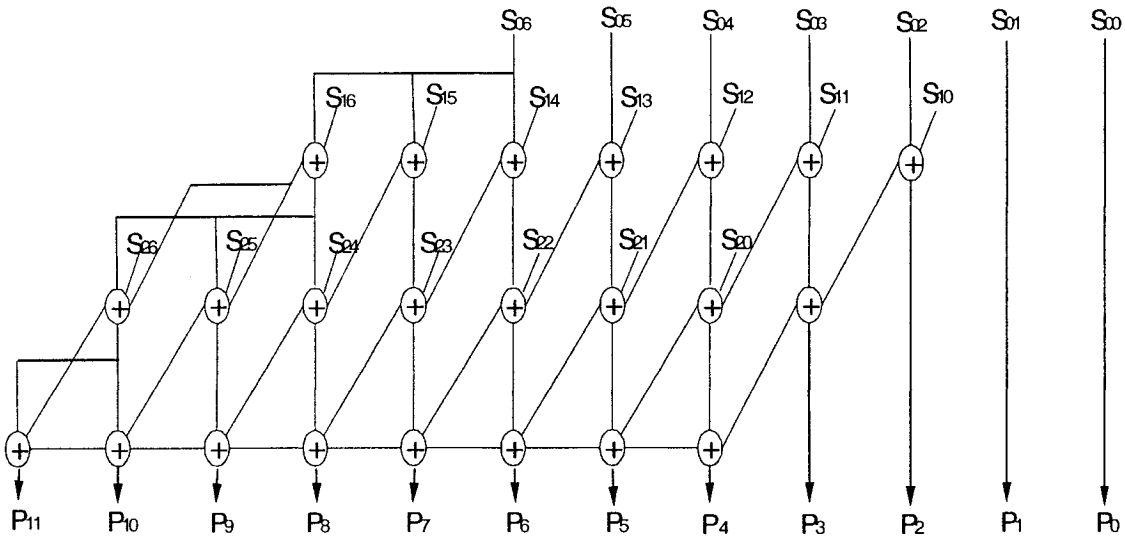


Figure 4-13 the new structure for radix 4 Booth-encoded partial products

In Fig 4-13, a 6-bit multiplier $B = b_5 b_4 b_3 b_2 b_1 b_0$ is encoded to $E = e_2 e_1 e_0$ using radix 4 booth-coding algorithm, where e_i could be any number of the set (-2, -1, 0, 1, 2). The number of partial products $S_{i6} S_{i5} S_{i4} S_{i3} S_{i2} S_{i1} S_{i0}$, therefore, is reduced to 3.

Properties of the New Structure:

Obviously, as an array-based multiplier, this new proposed structure has all properties an array scheme has, such as regularity and modularity. Moreover, rather than Baugh-Wooley array scheme, this new structure is more regular and flexible in 2's complement multiplication since it needs few types of cells and could be also applied to the partial products created by Booth-coding algorithm. These properties could save a lot on hardware implementation and make the reduction of partial products possible.

4.4 Carrier Frequency, Baud rate, and Memory Size

In chapter 2, Eq (2.12) has revealed the general relationship between f_c and R_s for modulators. For integer interpolation, this equation can be rewritten as follows:

$$m = L \cdot n \cdot \frac{R_s}{f_c} \tag{4.17}$$

where L is interpolation factor; n is up-sampling rate for the pulse-shaping filter, usually set to 4; m is the number of carrier samples per carrier period.

In digital modems, carrier samples are stored in memory. The memory size (*MS*) stands for the number of one-period-carrier sample values stored in the memory. It is clear that *MS* is proportional to L. From equation (4.17), we can calculate m, the number of samples needed to be taken from the above memory. As well, if *MS* is fixed, we can obtain the sample interval $\Delta = MS/m$. For instance, if we have 4 samples (m=4) per period and *MS*=1024, these 4 sample values are taken from 0^{th}, 256^{th}, 512^{th}, 768^{th} word of the memory. Eq (4.15) cannot guarantee that the sample interval Δ and the sample number m are always integers. However, they should be treated or rounded to nearest integers in hardware implementation. The adjustment that will be discussed below could affect IF precision. Assume that IF frequency is assumed fixed (i.e. f_c=70M Hz), some evaluation is then made on the relationship and the precision among IF f_c, Baud rate R_s, Memory Size *MS* and interpolation factor L. In following tables, f_{cr} is the nearest realizable IF frequency, obtained from the following equation:

$$f_{cr} = n \cdot L \cdot R_s / (MS / round(\Delta)) \tag{4.18}$$

$(f_c - f_{cr})/ f_c$ is the discrepancy; $round(\Delta)$ is the nearest integer of Δ; $int(m)$ the nearest floored integer of m.

Table 4-1 exploits memory size (*MS*) effect on IF f_c precision. In Table 4-1, we assume that R_s is fixed to 23 MHz and *MS* is varied at 1024 and 1024*1024. The result can be concluded as " the bigger *MS* is, the closer to theoretical f_c the real f_{cr} is". Note that *MS* does not affect the number of samples *m*.

Table 4-1 Memory Size (*MS*) effect on IF f_c precision

f_c:	70 MHz					
R_s:	23 MHz					
MS:	1024 byte					
L	*m*	int(*m*)	Δ	round(Δ)	f_{cr}	Discrepancy
1	1.3145058	1	779.13	779.00	69.98828125	0.000167411
2	2.625641	2	389.57	390.00	70.078125	-0.001116071
4	5.2512821	5	194.78	195.00	70.078125	-0.001116071
8	10.556701	10	97.39	97.00	69.71875	0.004017857
16	20.897959	20	48.70	49.00	70.4375	-0.00625
MS:	1048576 byte					
L	*m*	int(*m*)	Δ	round(Δ)	f_{cr}	Discrepancy
1	1.314285	1	797829.57	797830.00	70.00003815	-5.44957E-07
2	2.62857	2	398914.78	398915.00	70.00003815	-5.44957E-07
4	5.2571532	5	199457.39	199457.00	69.99986267	1.96184E-06
8	10.514254	10	99728.70	99729.00	70.00021362	-3.05176E-06
16	21.028718	21	49864.35	49864.00	69.99951172	6.97545E-06

Table 4-2 illustrates baud rate R_s effect on IF f_c precision. In Table 4-2, we assume that *MS* is fixed to 1024*1024 bytes and R_s is varied at 12 MHz and 23 MHz. The result can be concluded as "baud rate R_s affects the number of samples *m*". Based on the condition m>=4, the smaller is m or L, the more precise is the real f_{cr}.

Table 4-2 Baud rate R_s effect on IF f_c precision

MS:	1048576 byte					
f_c :	70 MHz					
R_s:	12 MHz					
L	*m*	int(*m*)	Δ	round(Δ)	f_{cr}	Discrepancy
1	0.6857144	0	1529173.33	1529173.00	69.99998474	2.17983E-07
2	1.371428	1	764586.67	764587.00	70.00003052	-4.35965E-07
4	2.7428595	2	382293.33	382293.00	69.99993896	8.71931E-07
8	5.4857047	5	191146.67	191147.00	70.00012207	-1.74386E-06
16	10.971467	10	95573.33	95573.00	69.99975586	3.48772E-06
R_s:	23 MHz					
L	*m*	int(*m*)	Δ	round(Δ)	f_{cr}	Discrepancy
1	1.314285	1	797829.57	797830.00	70.00003815	-5.44957E-07
2	2.62857	2	398914.78	398915.00	70.00003815	-5.44957E-07
4	5.2571532	5	199457.39	199457.00	69.99986267	1.96184E-06
8	10.514254	10	99728.70	99729.00	70.00021362	-3.05176E-06
16	21.028718	21	49864.35	49864.00	69.99951172	6.97545E-06

Table 4-3 demonstrates baud rate R_s precision effect on IF f_c precision. In Table 4-3, we assume that *MS* is fixed to 1024*1024 bytes and R_s is slightly modified at 12, 12.001, 12.011, 12.111 MHz. The result can be concluded as "baud rate (R_s) precision affects the f_c precision". The f_c precision only depends on Δ.

Table 4-3 Baud rate R_s precision effect on IF f_c precision

MS:	1048576 byte					
f_c:	70 MHz					
R_s:	12 MHz					
L	*m*	int(*m*)	Δ	round(Δ)	f_{cr}	Discrepancy
8	5.4857047	5	191146.67	191147.00	70.00012207	-1.74386E-06
R_s:	12.001 MHz					
L	*m*	int(*m*)	Δ	round(Δ)	f_{cr}	Discrepancy
8	5.4861639	5	191130.74	191131.00	70.00009555	-1.36501E-06
R_s:	12.011 MHz					
L	*m*	int(*m*)	Δ	round(Δ)	f_{cr}	Discrepancy
8	5.4907316	5	190971.61	190972.00	70.00014319	-2.04555E-06
R_s:	12.111 MHz					
L	*m*	int(*m*)	Δ	round(Δ)	f_{cr}	Discrepancy
8	5.5364503	5	189394.77	189395.00	70.00008682	-1.24032E-06

Table 4-4 and 4-5 demonstrates baud rate R_s and memory size MS effect on IF f_c precision. In Table 4-4 and 4-5, we modify MS and R_s but keep Δ as an integer. The result can be concluded as "the f_c precision only depends on Δ". If original Δ is integer, the real f_{cr} is equal to the theoretical one no matter how much MS or R_s is.

Table 4-4 R_s & MS effect on IF f_c precision

MS:	1048576 byte					
f_c:	70 MHz					
R_s:	10 MHz					
L	m	int(m)	Δ	round(Δ)	f_{cr}	Discrepancy
1	0.5714286	0	1835008.00	1835008	70	0
2	1.1428571	1	917504.00	917504	70	0
4	2.2857143	2	458752.00	458752	70	0
8	4.5714286	4	229376.00	229376	70	0
16	9.1428571	9	114688.00	114688	70	0
R_s:	20 MHz					
L	m	int(m)	Δ	round(Δ)	f_{cr}	Discrepancy
1	1.1428571	1	917504.00	917504	70	0
2	2.2857143	2	458752.00	458752	70	0
4	4.5714286	4	229376.00	229376	70	0
8	9.1428571	9	114688.00	114688	70	0
16	18.285714	18	57344.00	57344	70	0
MS:	1024 byte					
f_c:	70 MHz					
R_s:	10 MHz					
L	m	int(m)	Δ	round(Δ)	f_{cr}	Discrepancy
1	0.5714286	0	1792.00	1792	70	0
2	1.1428571	1	896.00	896	70	0
4	2.2857143	2	448.00	448	70	0
8	4.5714286	4	224.00	224	70	0
16	9.1428571	9	112.00	112	70	0
R_s:	20 MHz					
L	m	int(m)	Δ	round(Δ)	f_{cr}	Discrepancy
1	1.1428571	1	896.00	896	70	0
2	2.2857143	2	448.00	448	70	0
4	4.5714286	4	224.00	224	70	0
8	9.1428571	9	112.00	112	70	0
16	18.285714	18	56.00	56	70	0

On hardware design, a designer wants to make *MS* for IF as small as possible. Upon the above discussion, *MS* needs to be made big enough for certain IF precision when L is big or Δ is non-integer. However, it is known that the *MS* could be made very small if Δ is integer. Hence, to simplify hardware implementation, the following assumptions are made in our design:

1. Integer interpolation: $L=2^d$, d: $0 \leq d \leq 4$, integer
2. Selectable IF: $f_c = 4L/m * R_s$, $R_s \leq f_c$

Chapter 5 Modem Hardware Implementation

VHDL (VHSIC Hardware Description Language), as a hardware description language, is used mainly for the development of FPGAs and ASICs (Application Specific Integrated Circuits). Tools for synthesis of VHDL code into a gate-level netlist were well developed. Based on the technologies discussed previously, a hardware model for the system could be established by using VHDL [19]. However, the need for high-speed real-time communication presents significant hardware design challenges. In this chapter, pipelining & parallel processing, and multi-clock control are presented to enhance the system performance. In section 5.2, VHDL hardware implementation is discussed in terms of behavioral and structural modeling. A test-bench is built for system verifications by using Active-HDL. The design is elaborated and synthesized using Synopsys Design Compiler and Synopsys Analyzer. As an illustration, the system is implemented into a Xilinx Virtex-II FPGA using Xilinx Design Manager, and the characteristics are given at the end.

5.1 Pipelining & Parallel Processing

Previously, we have discussed pure combinatorial logic. If such a pure combinatorial logic is used to build an FIR filter or a communication system, the throughput rate will not meet the requirements of most communication systems in practice. Throughput rate is the key factor dictating the system performance. Real-time signal processing requires extensive concurrency using pipelining and parallel processing in order for increasing the system throughput rate. Pipelining is to reduce the effective critical path by introducing latches along the critical data path. Parallel processing is to increase the sampling rate by replicating hardware so that several inputs can be processed in parallel and several outputs can be produced at the same time. Fig 5-1 shows the system structures of pipelining and parallel processing.

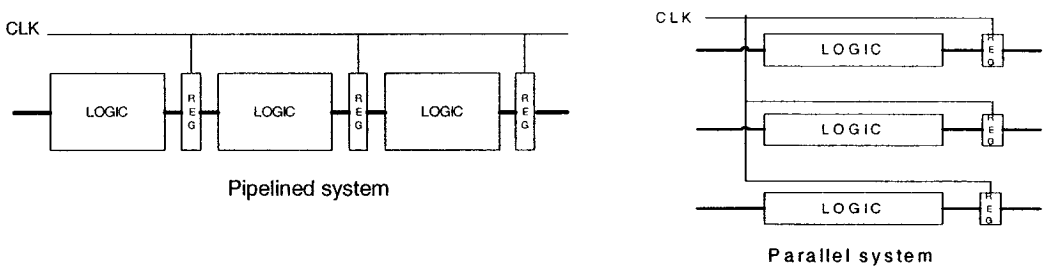


Figure 5-1 Pipelining and parallel system structure

Our Modem is a complicated, multi-clocked system. Both pipelining and parallel processing techniques are widely deployed to enhance the system performance. However, based on different types of hardware, the implementation strategies for pipelining and parallel processing are different. For instance, a FA (full adder) targeted into Xilinx Virtex II is about 2.5 times faster than that targeted into Xilinx 4000 family. It could be much faster if ASICs are used. Thus, further pipelining and parallel processing is needed for low-speed hardware to reduce time delay.

5.2 VHDL Implementation

VHDL is a hardware description and simulation language that was not originally intended for synthesis. Therefore, many hardware description and simulation constructs are not supported by synthesis tools. To guarantee that design is synthesizable, any design model created by VHDL code should follow hardware design guidelines.

5.2.1 VHDL Design Strategies

A digital system can be described at different levels of abstractions [20, 21], such as behavioral level, structural level or hybrid. Design Process starts with behavioral description, decomposing the high level of constructs into more precise functional units, then mapping these units into physical elements. In general, an efficient and effective VHDL design features the following characteristics:

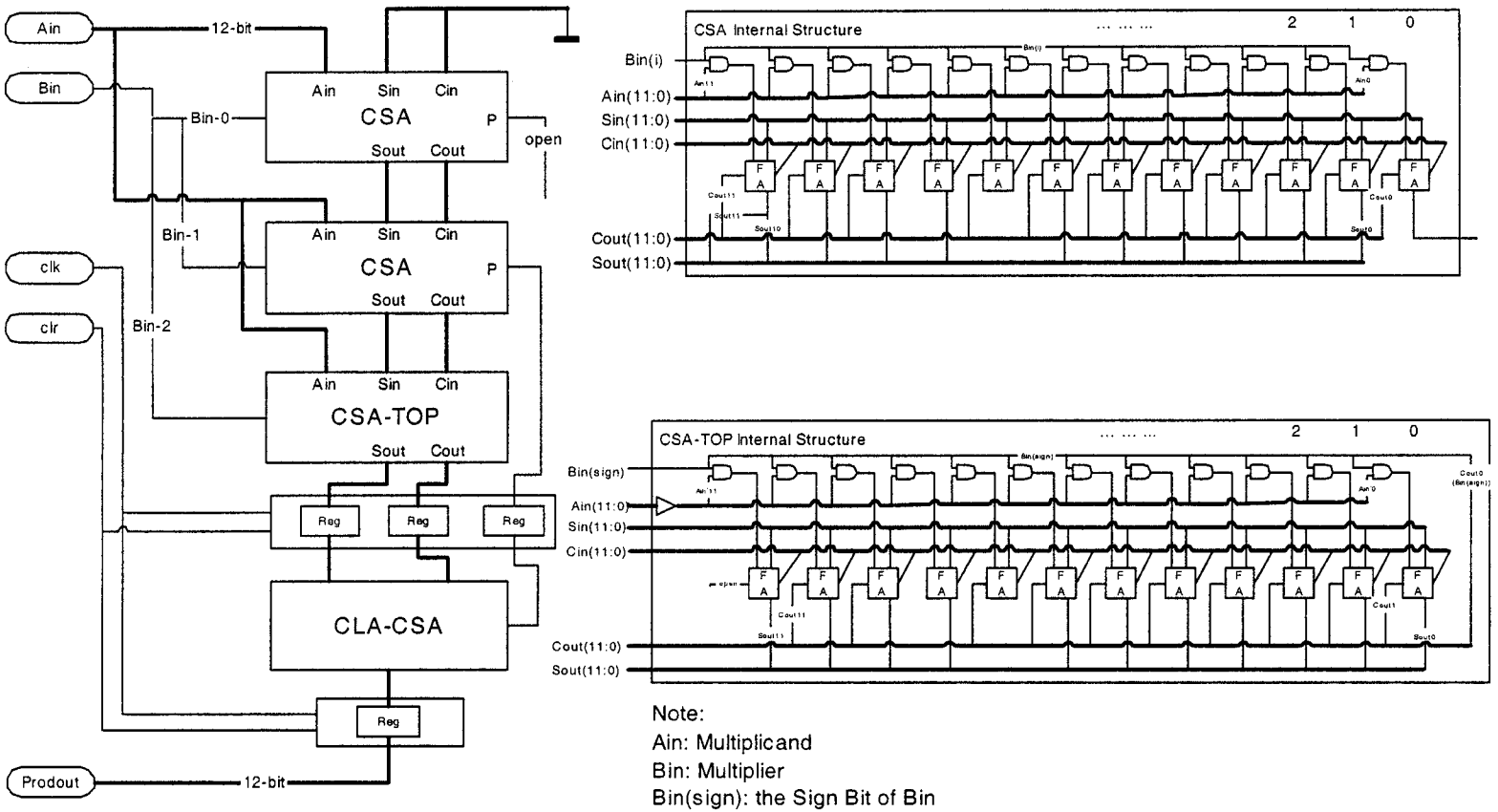
- Hierarchy: A repeated process of dividing a large system into smaller subsystems until the complexity of subsystems at an appropriately comprehensible level of detail.
- Regularity: Divide the hierarchy into similar building blocks whenever possible.
- Modularity: Well defined behavioral, structural and physical interface.
- Locality: Internals of the blocks are unimportant to any exterior interface.

5.2.2 Primitives, Components and Top-Level System

In hierarchy and structural design, components at the lowest level of the hierarchy are described as pure behavioral models, using the basic logic operators defined in VHDL. At the intermediate-level, components are supposed to be defined earlier (e.g. in package) and can be described as a structural, behavioral or hybrid model. The top-level design is usually described as a structural model.

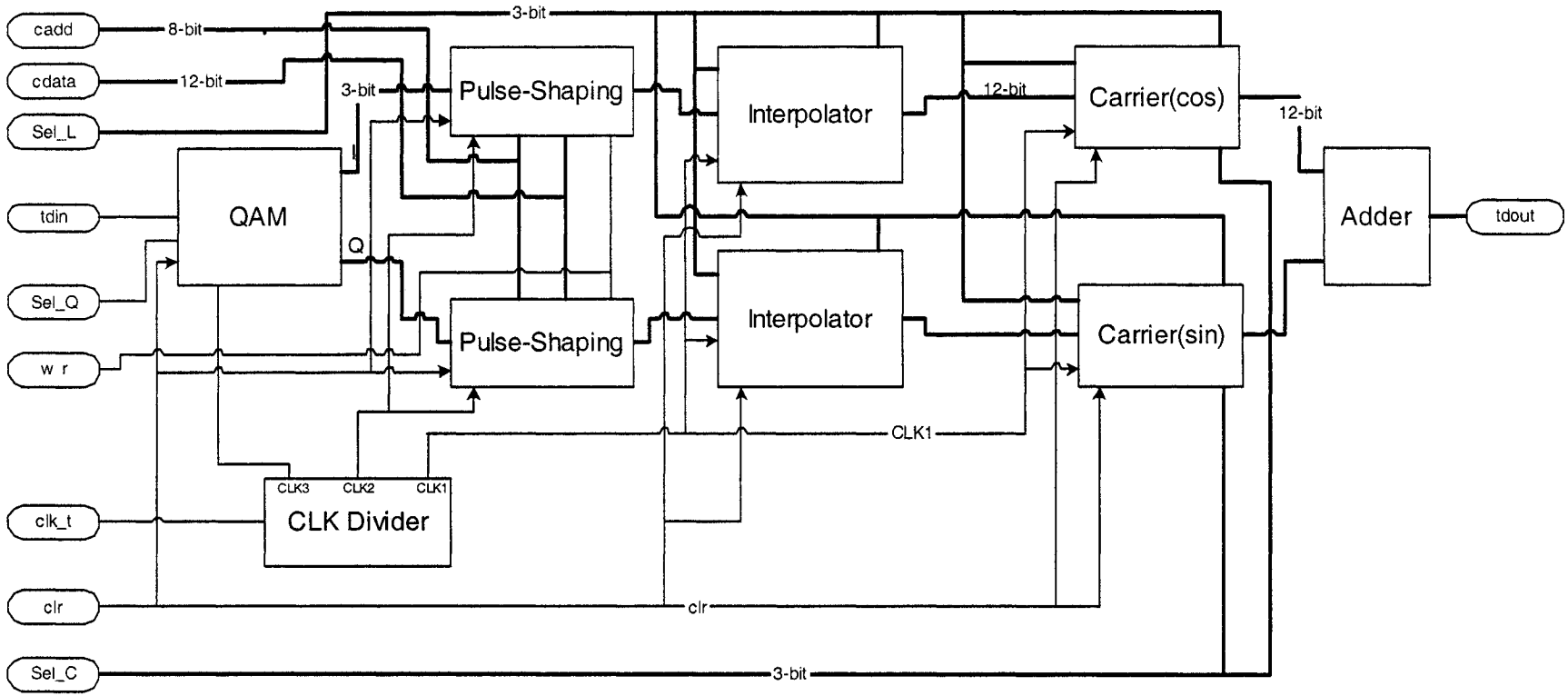
As a primitive component at the lowest level of the hierarchy, the high-speed CSA-CLA 12-bit adder is not further pipelined, described in pure behavioral level as shown in Appendix A. Its corresponding diagram is exactly the same as Fig 4-10. Apart from the primitive level, structural design techniques may be applied everywhere else in a hierarchical design. In Fig 5-2, one of the intermediate-level components, 12-bit by 3-bit multiplier, is described based on the hybrid approach: the behavioral model (data flow) for pipelining, and the structural model (port map instantiation) for intermediate carry save adders and the final CSA-CLA adder. In Appendix A, the VHDL code is given for this 12-bit by 3-bit multiplier.

A hierarchical description of a system is expressed in terms of subsystems interconnected by signals. Each subsystem may in turn be expressed as sub-subsystems interconnected by signals, and so on. This is a powerful technique to help hardware designers be able to design complicated systems time efficiently. In our design, high-regularity and high-modularity are well satisfied. A 12-bit CSA-CLA adder is the lowest hierarchical-level primitive, and a 12-bit by 3-bit multiplier and a 12-bit by 12-bit multiplier are regularly built. They are used as components for further designs of large subsystems. Based on DCS & DSP fundamentals, these 3 basic components are widely instantiated in many subsystems such as pulse-shaping filters, interpolation filters, IF modulator and others wherever possible. In addition, an interface is designed for the pulse-shaping/matched filter to load coefficients if necessary. After every subsystem is described, the top-level modulator and demodulator are finally described as a pure structural model shown in Fig 5-3 and Fig 5-4.



Note:
Ain: Multiplicand
Bin: Multiplier
Bin(sign): the Sign Bit of Bin
FA: 1-bit Full Adder
CLA-CSA: 12-bit High Speed Adder
CSA: Carry Save Adder
CSA-TOP: Carry Save Adder for the Significant Bit of the Multiplier

Fig 5-2 12-bit by 3-bit roundoff multiplier



Note:

Sel_Q:QAM selector (QPSK/16QAM)

Sel_L:Interpolator selector(0~4)

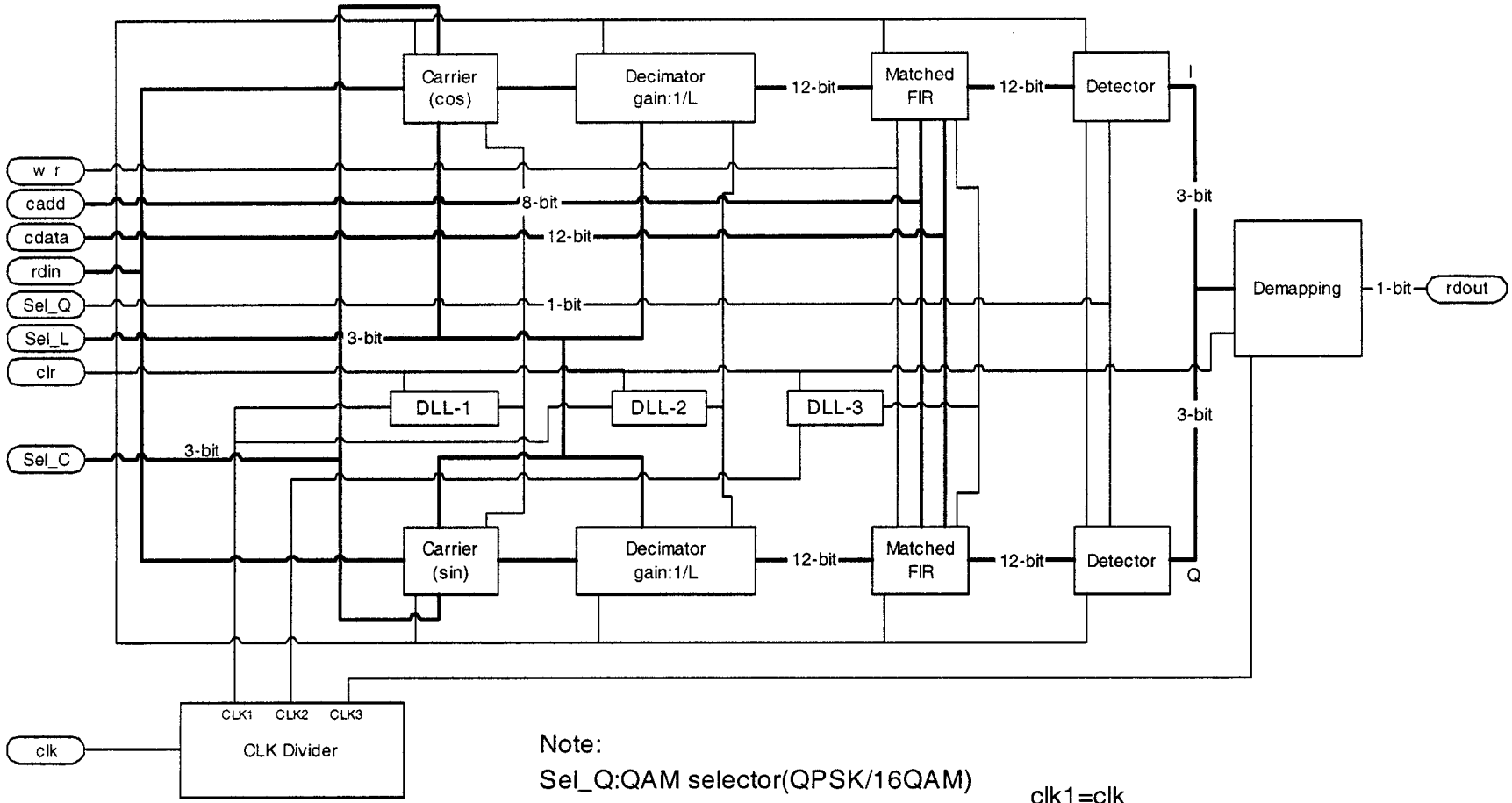
Sel_C:Fc/Rs selector(0~4), <=Sel_L

clk1=clk

clk2=clk1/L=clk/L

clk3=clk2*log2(M)/4=clk*log2(M)/4L

Figure 5-3 Modulator diagram



Note:

Sel_Q:QAM selector(QPSK/16QAM)

Sel_L:Interpolator selector(0~4)

Sel_C:Fc/Rs selector(0~4), <=Sel_L

DLL: Synchronization Module

Decimator: gain:1/L

clk1=clk

clk2=clk1/L=clk/L

clk3=clk2*$\log_2(M)/4$=clk*$\log_2(M)/4L$

Figure 5-4 Demodulator diagram

5.3 Clock Distribution and System Throughputs

In our system, the input data stream runs serially at a certain bit rate and there exists multi-rate DSP blocks inside the modem. Three different clocks are needed for serving different DSP blocks simultaneously. Moreover, QAM modulation modes and the interpolation ranges are adjustable, so the sub-clocks have to be able to adjust correspondingly. The relationship of 3 sub clocks and the global clock is listed as follows:

$$\begin{aligned} CLK_1 &= CLK \qquad CLK : global \quad clock \\ CLK_2 &= \frac{CLK}{L} \qquad L : \text{int } erpolation \quad factor \\ CLK_3 &= \frac{CLK * \log_2 M}{4 * L} \qquad M : QAM \quad symbol \quad set \quad size \end{aligned} \tag{5.1}$$

According to Eq (5.1), the sampling rate at each stage could be tabulated as Table 5-1:

Table 5-1 The Modem sampling rate at each stage

Modulator	Input	QAM output	Pulse-shaping output	Interpolator output	Complex mixer output	Adder output
Sampling rate	$k*R_s$	R_s	$4*R_s$	$4*L*R_s$	$4*L*R_s$	$4*L*R_s$
Demodulator	Input	IF recovery output	Decimator output	Matched filter output	Detector output	De-QAM output
Sampling rate	$4*L*R_s$	$4*L*R_s$	$4*R_s$	R_s	R_s	$k*R_s$

Where R_s and L are the baud rate and interpolation factor, respectively; $k = \log_2 M$.

5.4 System Verification

A traditional approach to verify the functionality of a design is to use a vendor specific Simulation Control Language (SCL). For simple designs like the QAM mapper and adders, this method is much faster since the SCL is tailored to the vendor's environment. However, when verifying the complicated operation of the Unit Under Test (UUT), SCL lacks automatic verification, portability, and interaction between UUT and UUT's environment [21]. Our design is a complicated communication system. The system verification needs to be processed under the following conditions:

- Multiple modulation modes: QPSK/16-QAM
- Multiple interpolation factor (L): 1, 2, 4, 8, 16
- Selectable carrier frequency factor ($\frac{f_c}{R_s}$): 1, 2, 4, 8, 16

An appropriate VHDL testbench for the system verification is built in our design. Based on the above conditions, the testbench includes 3 more components besides the system itself,:

- Stimulus generator: to create a random bit stream as the modem input
- Delay Module: to hold up its input by the modem latency.
- Comparator: to compare the system output with its delayed input and report if any error occurs.

As discussed above, the testbench diagram for our design could be plotted in Fig 5-5.

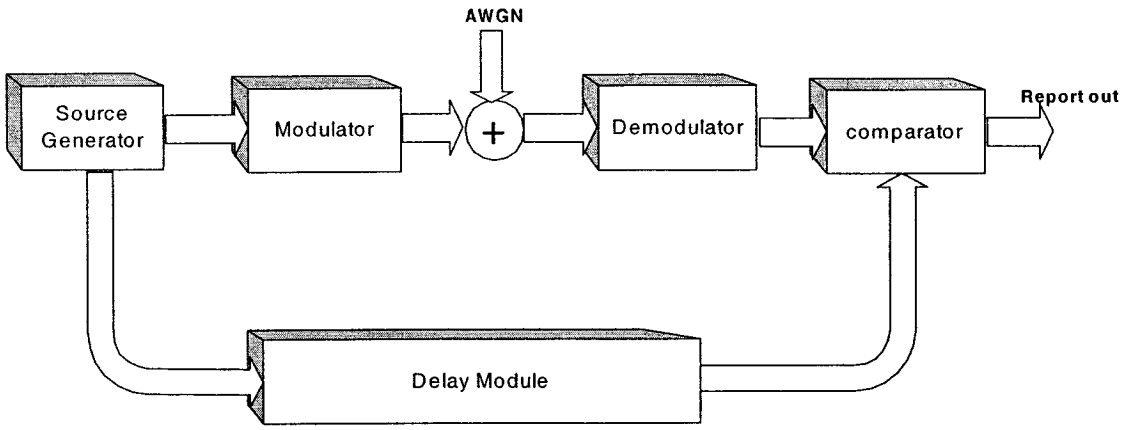


Figure 5-5 The modem Testbench

Based on the above testbench, our system is simulated using ALDEC Active-HDL simulator with randomly fed 100,000 bits for each case.

5. 5 Illustrative Example Based on Xilinx Virtex II

FPGA, which features low-cost, high-efficiency and high-flexibility, is an integrated circuit that contains many identical-logic cells that can be viewed as standard components. Each logic cell can independently take on any one of a limited set of personalities. The individual cells are interconnected by a matrix of wires and programmable switches. A user's design is implemented by specifying the simple logic function for each cell and selectively closing the switches in the interconnect matrix. Complex designs are created by combining these basic blocks to create the desired circuit.

5.5.1 Effects Caused by Structures or Hardware

Generally, the main performances of a system include speed, area and power. Implementation to different target FPGAs will cause different speeds even if the VHDL code of a design is the same. As an example, Table 5-2 shows the critical path time delay

of a full adder and our design between FPGAs of Xilinx 4000 and Xilinx Virtex II. Note that it is Logic-synthesized using Synopsys Design Compiler & Design Analyzer.

Table 5-2 Timing analysis for different FPGAs

	Xilinx 4000	Xilinx virtex II
FA	3.2 ns	1.5 ns
The modem	27.26 ns (estimated)	12.78 ns

Similarly, different system structures will cause different area occupation and power consumption. Generally, a system structure is analyzed and determined at the earlier stage than hardware implementation. However, in order to verify effects caused by different structures, subsystem matched filter in our design is chosen and built in both the transversal structure and the poly-phase structure, respectively. The effects on area occupation and power consumption are shown in Table 5-3. Note that it is Logic-synthesized using Synopsys Design Compiler & Design Analyzer.

Table 5-3 Area & Power analysis for different structures of matched filter

	Transversal Structure	Poly-Phase Structure
Area (Cell)	26935	10021
Power (mW)	145.3	126.26

5.5.2 System Parameters and Features

As an example, our design is synthesized and implemented into Xilinx Virtex II FPGA chip, which uses 0.15μm / 0.12μm CMOS 8-layer metal process. Using Xilinx Design Manager, the final results measured after Placement and Routing (PAR) can be shown in table 5-4.

Table 5-4 The modem system parameters

Item	Specification	Total	Remark
Target Device	x2v6000		
Target Package	bf957		
Target Speed	-5		
Mapper Version	virtex2--D.27		
Slice No.	23,868	33,792	70%
Slice Register No.	17,387	67,584	25%
4 -input LUT No.	37,079	67,584	54%
IOB No.	57	684	8%
Maximum Throughput	33.4046 MHz		
Power	276.1512 mW		

Based on the design requirements and the above results, the system features are summarized as follows:

- Modulation Mode: QPSK/16QAM
- Input Data Rate (R) for Modem: up to 33.4046 MHz
- Interpolation Range (L): 1, 2, 4, 8 16

5.5.3 Tradeoffs to Enhance System Performance

System performance is verified in terms of power consumption, area occupation and time delay. Usually, to improve one of these parameters could make others worse. Such tradeoffs are shown as follows:

- Further pipelining and parallel processing could reduce critical path time delay but increase system latency and area.

- Using more efficient FPGAs and their embedded primitives could improve system performance but lack portability and regularity.
- ASIC design could extremely improve system speed and reduce area and power consumption but dramatically increase design complexity, cost and time-to-market.

Hence, a designer should meticulously choose his design plans based on system requirements and usable resources before starting his job.

5. 6 Commercial Applications Using ASICs

An introduction has been given in section 5.5 for a modem with programmable bit rate and programmable intermediate carrier frequency. But the previous example targeted to Xilinx Virtex II FPGA has a few constraints such as limited interpolation range L. Generally, IF frequency f_c is not a multiple of R_s in commercial applications, so a large memory (say, $\geq 10^6$ bytes) is required to meet the requirement of the frequency accuracy (say, $(f_c - f_{cr})/f_c \leq \pm 10^{-6}$).

The E-carrier system[1] in Europe and Asia and the T-carrier system[2] in North America are commonly used in Public Service Telephone Network (PSTN). Due to different traffic requirements, the E-carrier system deploys E-1, E-2, E-3, etc. The T-carrier system deploys T-1, T-2, and T-3. Assume that a modem is used to modulate T-1 (1.544Mbps), T-2 (6.312Mbps) or T-3 (44.736Mbps) shown in Table 5-5 on a fixed IF via point-to-point microwave communication, which is common for cost saving. In Table 5-5, estimations are made in terms of memory size and IF carrier frequency accuracy

[1] The E-carrier system is entirely digital, using pulse code modulation and time-division multiplexing. The E-1 carrier system is at a rate of 2.048 Mbps, and can carry 32 64-kbps channels.
[2] The T-1 digital stream is 1.544 Mbps, consisting of 24 64-Kbps channels that are multiplexed.

under the following assumptions: (1) IF $f_c = 20MHz$, $f_s \geq 80MHz$; (2) Modulation mode: 16QAM, where $R_s = R/4$; (3) Realizable frequency accuracy: $\leq 10^{-6}$.

Table 5-5 Data rates for T-carrier system transmission[3]

Technology	Speed	Application
DS1/T-1	1.544 Mbps	Large company to ISP; ISP to Internet infrastructure
DS2/T-2	6.312 Mbps	Large company to ISP; ISP to Internet infrastructure
DS3/T-3	44.736 Mbps	ISP to Internet infrastructure; Smaller links within Internet infrastructure

Table 5-6 Estimations for memory size and IF frequency accuracy

f_c:(MHz)	20					
MS:(byte)	1048576 (1024*1024)					
R_s=R/4(MHz)	0.386					
L	m	int(m)	Δ	round(Δ)	f_{cr}	Discrepancy (%)
1	0.0772	0	13582590.7	13582591	20.0000005	-2.40326E-08
2	0.1544	0	6791295.34	6791295	19.999999	4.95911E-08
4	0.3088	0	3395647.67	3395648	20.000002	-9.76563E-08
8	0.6176	0	1697823.83	1697824	20.000002	-9.76563E-08
16	1.2352	1	848911.917	848912	20.000002	-9.76563E-08
32	2.4704	2	424455.959	424456	20.000002	-9.76563E-08
64	4.9408	4	212227.979	212228	20.000002	-9.76563E-08
R_s=R/4(MHz)	1.578					
L	m	int(m)	Δ	round(Δ)	f_{cr}	Discrepancy
1	0.3156	0	3322484.16	3322484	19.9999991	4.73022E-08
2	0.6312	0	1661242.08	1661242	19.9999991	4.73022E-08
4	1.2624	1	830621.039	830621	19.9999991	4.73022E-08
8	2.5248	2	415310.52	415311	20.0000231	-1.15662E-06
16	5.0496	5	207655.26	207655	19.999975	1.25122E-06
R_s=R/4(MHz)	11.184					
L	m	int(m)	Δ	round(Δ)	f_{cr}	Discrepancy
1	2.2368	2	468783.977	468784	20.000001	-4.88281E-08
2	4.4736	4	234391.989	234392	20.000001	-4.88281E-08

To achieve IF frequency accuracy $\leq 10^{-6}$, the clock should be at least 80 MHz, the memory size at least 1 M bytes, and the minimum interpolation factor L=64. For such requirements, FPGAs with the same hardware techniques are no longer suitable because of the lack of the huge memory and the slow clock. FPGA's lag ASIC's by a factor of ten or more in density, and lack system-level facilities, such as large random access memories (RAM's) and clocks, that are needed to implement large systems [23]. For example, the maximum memory that Xilinx Virtex II platform FPGAs can provide is 336k 9-bit block SelectRAM (168*2k), which is far from enough. Furthermore, in a typical FPGA, the sum of the routing delays along the critical path comprises well over half of the total delay. Using ASICs can solve these problems. Large memory could be easily obtained, and routing delays could be dramatically reduced compared to FPGA architecture.

The critical path of our modem has only 20 gates. If the same hardware technology (0.15μm / 0.12μm CMOS 8-layer metal process) as Xilinx Virtex FPGA is used for our modem design in ASIC, the time delay could be reduced up to around 10 ns. Hence, our modem speed could increase up to 100 MHz. Furthermore, the modem speed could reach more than 300 MHz if bit-level pipelining is used.

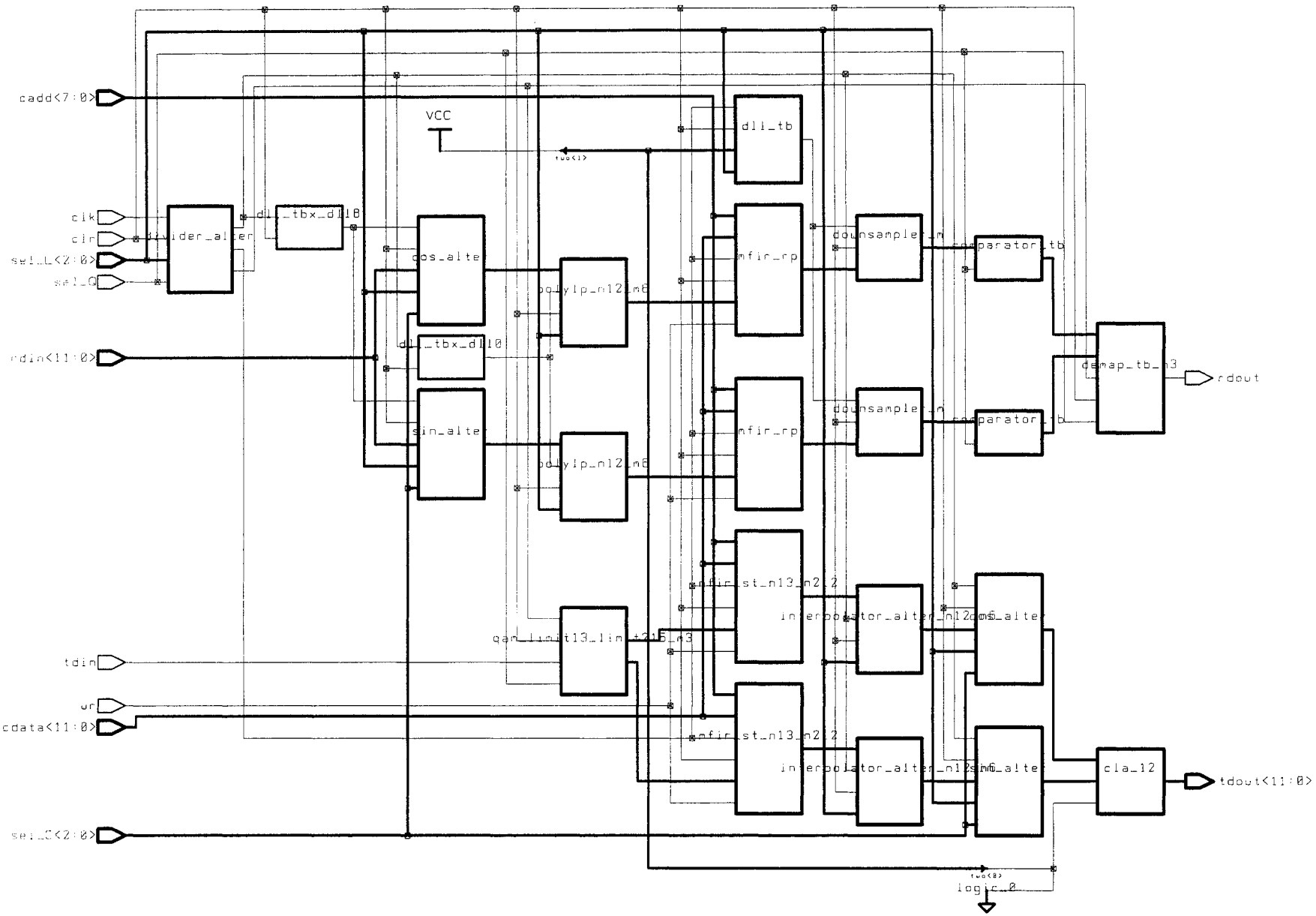
[3] Extracted from http://whatis.techtarget.com

5.7 System Schematic and Pin Descriptions

The modem system schematic elaborated by Synopsys Design Analyzer is shown in Fig 5-6. And its pin descriptions are as follows.

Table 5-7 The Modem pin descriptions

Pin name	Pin type	Description
VCC		Digital Power
GND		Digital Ground
clk_t	I	Modulator clock
clk_r	I	Demodulator clock
clr		Reset
cadd <7:0>	I	Address bus for loading filter coefficients
cdata <11:0>	I	Data bus for loading filter coefficients
tdin	I	Modulator input
wr		"cdata" write in control
sel_Q	I	Modulation mode select
sel_L <2:0>	I	Interpolation factor select
sel_C <2:0>	I	Carry frequency select
tdout <11:0>	O	Modulator output
rdout	O	Demodulator output



design: smodem	designer:	date: 7/29/2003
technology:	company:	sheet: 1 of 1

Figure 5-6 Modem schematic

Chapter 6 Conclusions

The main contribution achieved in this thesis is the design and implementation if a wideband digital modem featuring variable input data rates, programmable carrier frequency and multi-level QAM modulation. The thesis contains wireless communication fundamentals, digital signal processing, MATLAB simulation, VHDL coding and FPGA implementation as follows.

The design and implementation of WDM consists of 2 steps. The first step focuses on the system architecture design and the system-level simulation by using MATLAB simulation software. The design specifications and parameters are determined and optimized at this step. The second step concentrates on the hardware design by using VHDL. At this step, the hardware structure of the system is optimized and the test-bench for design verification is built.

FIR filters are very critical parts in digital communication systems. Their structures and parameters basically influence the system performance. Thus, much attention was paid on FIR filter design. In Chapter 2, 2 FIR structures (transversal and poly-phase) have been illustrated. Due to the computation efficiency and hardware saving, the poly-phase structure for FIR filters has particularly been detailed for the future hardware design. In Chapter 3, the MATLAB M-file has been written to verify the design functionality and

parameterize for general M-ary QAM modulation scheme. QPSK and 16-QAM schemes have been simulated and parameterized in our design.

Multipliers are of great importance as a basic building block for DSP-based implementation. In Chapter 4, a new and versatile signed-array multiplier for 2's complement multiplication has been introduced and developed. This new structure has the advantages as follows:

- Array-based, more regular and modular, suitable for design of different-width multipliers in VLSI implementation
- Applicable for other algorithms, Booth-coding etc.
- Possiblility of time delay reduction by using powerful partial product generating algorithms, e.g. Radix 4 or higher Booth-coding algorithm

The structure for programmable bit rate and programmable carrier frequency has been explained in Chapter 4. In Chapter 5, the relationship between IF f_c and symbol rate R_s has been discussed. It clearly shows that the realizable IF f_{cr} and the theoretical IF f_c is identical only on the condition of the sample interval Δ is an integer. Otherwise, the realizable IF f_{cr} can only be approximately equal to the theoretical IF f_c. Their discrepancy could be reduced by increasing the size of memory *MS* that is for storing the carrier frequency sample values.

In this thesis, it was shown that using Xilinx Virtex FPGA, our modem can work up to speed of 33 M bits/sec. However, using ASIC with proper pipelining, we can achieve bit rates of up to 300 M bits/sec.

Bibliography

[1] Agilent Technologies, *Testing and Troubleshooting Digital RF Communications Transmitter Designs,* Agilent Technologies Wireless Test Solutions Application Note 1313, literature number 5968-3578E, Jan 2002.

[2] Henry Samulei and Bennett C. Wong, "A VLSI Architecture for a High-Speed All-Digital Quadrature modulator and Demodulator for Digital Radio Applications," *IEEE Journal on Selected Areas in Communications*, Vol. 8. No. October 1990.

[3] Faisal M. Kashif, Wajahat Qadeer, and Syed Lsmail Shah, "Efficient Implementation of Quadrature Amplitude Modulation Transmitters," *IEEE INMIC 2001*, 2001.

[4] G Carle, J Tardeval, and K V Lever, "Digital Implementation of Baseband Pulse-Shaping Filters and Modulators for High-Capacity Digital Radio Systems," *Multi-Level Modulation Techniques and Point-to-Point and Mobile Radio, IEE Colloquium on*, Mar 1990.

[5] Alan V. Oppenheim, Ronald W. Schafer, and John R. Buck, *Discrete-Time Signal Processing,* Second Edtion, Pretice Hall, 1998.

[6] Steven W.smith: *The Scientist and Engineer's Guide to Digital Signal Processing,* San Diego, Calif.: California Technical Pub, c1997

[7] Nyquist, H., "Certain Topics of telegraph Transmission Theory," *Trans. Am. Inst.Elextr. Eng.,* vol. 47, Apr 1928, pp617-644.

[8] Rudra Pratap, *Getting Strated with MATLAB 5,* Oxford University Press, 1999.

[9] *Using MATLAB, Version 5,* The Mathworks, inc., 1997

[10] Bernard Sklar, *Digital Communications Fundamentals and Applications,* Englewood Cliffs, N.J.: Prentice-Hall, c1988

[11] Andreas Antoniou, *Digital Filters: Analysis and Design,* McGraw-Hill, 1979. pp87-96, 1997.

[12] N. J. Fliege, *Multirate Digital Signal Processing,* John Wiley & Sons, 1996

[13] Smith, Michael J. S, *Application-Specific Integrated Circuits*, Addison-Wesley,1997

[14] S.Y.Kung, *VLSI Array Processors,* Prentice Hall and Englewood Cliffs, New Jersey, pp. 480-487, 1988.

[15] C.S.Wallace, "A suggestion for a fast multiplier," *IEEE Transaction on Electtronic Computers*, pp. 14-17, February 1964.

[16] L.Dadda. *Some schemes for parallel multipliers*, Alta Frequenza, 34; pp. 349-356, March 1965.

[17] C.R. Baugh and B.A. Wooley, "A 2's complement parallel array multiplication algorithm," *IEEE Transaction on Computers,* C-22: pp. 1045-1047, December 1973.

[18] Wang, Q. and Shayan, Y., "A Versatile Signed Array Multiplier Suitable for VLSI Implementation", *CCECE 2003*, No. 532, IEEE Canada, May 2003.

[19] Stanley Mazor and Patricia langstraat, *A Guide to VHDL,* Kluwers Academic Publishers, 1992.

[20] D. Gajski and R. Khun, "Introduction: New VLSI Tools," *IEEE Computer*, Vol. 16, No. 12, pp. 11-14, Dec. 1983.

[21] Ben Cohen, *VHDL Styles and Methodologies,* Second Edtion, Kluwers Academic Publishers, 1999.

[22] David. M. Lewis, David. R. Galloway, Marcus van Ierssel, Jonathan Rose, and Paul Chow, "The Transmogrifier-2: A 1 Million gate Rapid-Prototyping System," *IEEE Transations On Very Large Scale Integration (VLSI) Systems,* Vol. 6, No. 2, June 1998.

Appendix A—1:

— VHDL code of 12-bit high speed CSA-CLA adder

```
--12-BIT CARRY LOOKAHEAD & CARRY SELLECT ADDER
-- without cin and cout
library ieee;
use ieee.std_logic_1164.all;
ENTITY cla_12 IS
        PORT(A,B: IN STD_LOGIC_VECTOR(11 DOWNTO 0);
        CIN:IN STD_LOGIC;
        S:OUT STD_LOGIC_VECTOR(11 DOWNTO 0));
END ;

ARCHITECTURE BEHAVIOR OF cla_12 IS

        --CLA: 4-STD_LOGIC CARRY LOOLAHEAD BLOCK
        PROCEDURE CLA(A,B : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        CIN: IN STD_LOGIC;SIGNAL S: OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
        SIGNAL GG: OUT STD_LOGIC) IS
        VARIABLE G,P:STD_LOGIC_VECTOR(3 DOWNTO 0);
        VARIABLE TEMP_C: STD_LOGIC_VECTOR(3 DOWNTO 0);
        BEGIN

                G := A AND B; P :=A XOR B;
                TEMP_C(0) :=CIN;
                TEMP_C(1) :=G(0) OR (CIN AND P(0));
                TEMP_C(2) :=G(1) OR (G(0) AND P(1))
                OR (CIN AND P(1) AND P(0));
```

```
            TEMP_C(3) :=G(2) OR (G(1) AND P(2))
            OR (G(0) AND P(2) AND P(1)) OR
            (CIN AND P(2) AND P(1) AND P(0));
            GG <=G(3) OR (G(2) AND P(3)) OR
            (G(1) AND P(3) AND P(2)) OR (G(0)
            AND P(3) AND P(2) AND P(1)) OR
            (CIN AND P(3) AND P(2) AND P(1) AND P(0));
            S <= P XOR TEMP_C;

    END;

SIGNAL GG1,GG2,GC:STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL LOW,HIGH: STD_LOGIC;
--SIGNAL OVERFLOW: STD_LOGIC;
SIGNAL S1,S2: STD_LOGIC_VECTOR(11 DOWNTO 0);
--SIGNAL S_TEMP: STD_LOGIC_VECTOR(11 DOWNTO 0);
BEGIN
    LOW <='0'; HIGH<='1';-- FIXED VALUES INITIALIZED
    CLA(A(3 DOWNTO 0),B(3 DOWNTO 0),CIN,S(3 DOWNTO 0),GC(0));

  CLA(A(7 DOWNTO 4),B(7 DOWNTO 4),LOW,S1(7 DOWNTO 4),GG1(1));
    CLA(A(7 DOWNTO 4),B(7 DOWNTO 4),HIGH,S2(7 DOWNTO 4),GG2(1));
    GC(1) <= GG1(1) WHEN GC(0)='0' ELSE GG2(1);
    S(7 DOWNTO 4) <= S1(7 DOWNTO 4)
    when GC(0)='0' ELSE S2(7 DOWNTO 4);

    CLA(A(11 DOWNTO 8),B(11 DOWNTO 8),LOW,S1(11 DOWNTO 8),GG1(2));
    CLA(A(11 DOWNTO 8),B(11 DOWNTO 8),HIGH,S2(11 DOWNTO 8),GG2(2));
```

```
S(11 DOWNTO 8) <= S1(11 DOWNTO 8)
when GC(1)='0' ELSE S2(11 DOWNTO 8);
GC(2) <= GG1(2) WHEN GC(1)='0' ELSE GG2(2);


END;
```

Appendix A–2:

—VHDL code of 12-bit by 3-bit pipelined roundoff multiplier

```
--12*3 bit signed clocked multiplication:
--12-bit roundoff output
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.component_alter.all;


entity clocked_mult_c4 is
--width of multiplicand and multiplier
 generic(m1: integer :=12; m2: integer :=3);
 port(ain: in  std_logic_vector(m1-1 DOWNTO 0);
    bin: in  std_logic_vector(m2-1 DOWNTO 0);
    prodout:out std_logic_vector(m1-1 DOWNTO 0);
          clk,clr:in std_logic);
end ;


architecture circuits of clocked_mult_c4 is --signed


 signal zero : std_logic_vector(ain'length-1 downto 0) ;
 type array_N_M is array(0 to bin'length-1)
 of std_logic_vector(ain'length-1 downto 0);
 signal s   : array_N_M; -- partial sums
 signal c   : array_N_M; -- partial carries
 signal c_reg,s_reg:
```

```
 std_logic_vector(ain'length-1 downto 0);
 signal a: std_logic_vector(ain'length-1 downto 0);
 signal b: std_logic_vector(bin'length-1 downto 0);
 signal prod1:std_logic_vector(bin'length-2 downto 0);
 signal prod2:std_logic_vector(prodout'length-1 downto bin'length-1);
 signal prod_reg:std_logic_vector(b'length-2 downto 0);
begin  -- circuits of CLOCKED_MULT_164
  zero <= (others=>'0'); --initial value reset
        stages: for i in 0 to bin'length-1 generate
                lowstd_logic: if i=0 generate
                        carry_save:csa(ain,zero,zero,bin(0),
                        c(0),s(0),prod_reg(0));
                end generate;
                middlestd_logic:
                if i/=0 and i< bin'length-1 generate
                        carry_save:csa(ain,s(i-1),c(i-1),
                        bin(i),c(i),s(i),prod_reg(i));
                end generate;
                topstd_logic: if i=bin'length-1 generate
                        carry_save:csa_top(ain,s(i-1),c(i-1),
                        bin(i),c(i),s(i));
                end generate;
        end generate;
                CLA: cla_12 PORT MAP (S_REG,C_REG,prod1(prod1'left),
                prod2);
process(clk,clr)
begin
        if clr='1' then
```

```
                prod1<=(others=>'0');
                prodout<=(others=>'0');
                c_reg <= (others =>'0');
                s_reg <= (others =>'0');
        elsif clk='1' and clk'event then
                --pipelining
    c_reg<=c(bin'length-1);
                s_reg<=s(bin'length-1);
    prod1<=prod_reg;
    prodout<=prod2;
        end if;
        end process;
end architecture circuits;
```