CAPACITY IMPROVEMENTS FOR A DIRECT-SEQUENCE CODE DIVISION MULTIPLE ACCESS NETWORK

DAVID VICTOR BOZWARD

Doctor of Philosophy

THE UNIVERSITY OF ASTON IN BIRMINGHAM

MAY 1995

THE UNIVERSITY OF ASTON IN BIRMINGHAM

CAPACITY IMPROVEMENTS FOR A DIRECT-SEQUENCE CODE DIVISION MULTIPLE ACCESS NETWORK

by
David Victor Bozward

Submitted for the degree of
DOCTOR OF PHILOSOPHY
1995

Summary

The rapidly increasing demand for cellular telephony is placing greater demand on the limited bandwidth resources available. This research is concerned with techniques which enhance the capacity of a Direct-Sequence Code-Division-Multiple-Access (DS-CDMA) mobile telephone network. The capacity of both Private Mobile Radio (PMR) and cellular networks are derived and the many techniques which are currently available are reviewed. Areas which may be further investigated are identified. One technique which is developed is the sectorisation of a cell into toroidal rings. This is shown to provide an increased system capacity when the cell is split into these concentric rings and this is compared with cell clustering and other sectorisation schemes.

Another technique for increasing the capacity is acheived by adding to the amount of inherent randomness within the transmitted signal so that the system is better able to extract the wanted signal. A system model has been produced for a cellular DS-CDMA network and the results are presented for two possible strategies. One of these strategies is the variation of the chip duration over a signal bit period. Several different variation functions are tried and a sinusoidal function is shown to provide the greatest increase in the maximum number of system users for any given signal-to-noise ratio. The other strategy considered is the use of additive amplitude modulation together with data/chip phase-shift-keying. The amplitude variations are determined by a sparse code so that the average system power is held near its nominal level. This strategy is shown to provide no further capacity since the system is sensitive to amplitude variations. When both strategies are employed, however, the sensitivity to amplitude variations is shown to reduce, thus indicating that the first strategy both increases the capacity and the ability to handle fluctuations in the received signal power.

KEY WORDS:

Capacity, Chip Rate, Code Division Multiple Access, Mobile Radio, PMR

This is dedicated to my grandmother
Mrs D. Green
who has always helped and supported me in every way possible.

"Life is too important a thing ever to talk seriously about"

Oscar Wilde

Acknowledgements

I would like to thank my supervisor, Dr. R. L. Brewster, for his extremely helpful guidance over these last three years, during the PhD process.

Due acknowledgements are given to Roke Manor Research Ltd. for their financial support and especially to Peter Hulbert for his technical supervision.

Contents

Chapter 1 - Introduction

Chapter 2 - Direct Sequence Spread Spectrum Systems

Figures and Tables

List of Figures

List of Tables

Abbreviations

BER	Bit Error Rate
bps	Bits Per Second
BPSK	Binary Phase Shift Keying
CCH	Committee on Harmonisation
CDMA	Code Division Multiple Access
CEPT	Conference of European Posts and Telecommunications Administration
cps	Chips Per Second
CPFSK	Continuous Phase Frequency Shift Keying
CVSD	Continuous Varying Slope Delta modulation
dB	Decibel
DS	Direct Sequence
FDMA	Frequency Division Multiple Access
FEC	Forward Error Correction
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FSK	Frequency Shift Keying
GF	Galois Field
GSM	Groupe Speciale Mobile
ISI	Inter-Symbol Interference
MSK	Minimum Shift Keying
OOK	On Off Keying
OOP	Object Orienatted Programming
OQPSK	Offset Quadrature Phase Shift Keying
PMR	Private Mobile Radio
PN	Pseudorandom Noise
PSK	Phase Shift Keying
QPSK	Quadrature Phase Shift Keying
RELP	Residual Excited Linear Predictive Coding
SBC	Adaptive Suband Coding
SNR	Signal-to-Noise Ratio
sps	Samples Per Second
SS	Spread Spectrum
SQAM	Superposed Quadrature Amplitude Modulation
SVSD	Continuous Variable Slope Delta Modulation
TDMA	Time Division Multiple Access
VAD	Voice Activity Detection

Symbols

A	Amplitude
B_w	Bandwidth
b_k	Filter Coefficients
C	Shannon Channel Capacity
c(t)	Code Sequence
D	Voice Activity Detection
d	Distance
d(t)	Data Sequence
d'(t)	Received Data Sequence
E_b	Energy per Bit
F	Frequency Reuse Factor
G	Sectorisation Gain
G(f)	Spectral Density
G_r	Received Antenna Gain
G_T	Transmitted Antenna Gain
I	Cell Isolation Factor
I_{ic}	Interference Power due to Transmissions in Cell
I_{oc}	Interference Power due to Transmissions out of Cell
k	User Number
l	Correlation Variable
M	Number of Symbols
n	Length of register array
N	Number of Users
N_c	Number of Cells Adjacent to the Desired Cell
N_f	Number of Points in discrete fourier transform
N_0	Noise Spectral Density
N_s	Number of Transmitted Symbols
$n(t)$	System Noise
n_c	Number of Cells in Cluster
n_u	User Number
η	Spectral Efficiency
σ^2	Variance
$\phi(t)$	Locally generated Signal
P	Average Signal Power
p	Period of Sequence
P_e	Probability of Error

P_G	Processing Gain
P_r	Received Power
P_T	Transmitted Power
ρ	Correlation Coefficient
$Q(u)$	Gaussian Integral
r(t)	Received Signal
R_b	Data Bit Rate
S(t)	Transmitted Signal
S_u	User Duration Spacing
t	Time
T_b	Duration of Data Bit
T_c	Chip Duration
T_c'	Variation Chip Duration
T_d	Transmission Delay
t_s	Sampling Duration
τ	Code Misalignment Duration
V	Variation Factor
Y^k	Matched Filter Detector Output
$Y_\pm$	Confidence Interval
γ	Excess Phase
$\theta_d(t)$	Phase Modulation
ω_c	Carrier Frequency
ψ	Correlation Coefficient
ψ_a	Auto-Correlation Function
ψ_c	Cross-Correlation Function
Z	Probability of an Error
ζ	Sparse Bit Ratio
λ	Wavelength of Carrier

Chapter 1

Introduction

1.0 Introduction

The research contained in this document has been carried out in response to a real technological need to increase the capacity of a cellular spread spectrum system. This is achieved here using two techniques; firstly by toroidal sectorisation and secondly by varying the chip duration over one bit period. These are shown to provide an increase in the number of users which a system may handle of more than 92%.

There is a need for greater bandwidth efficiency within mobile cellular communications which has been determined by the required level of customer access. The revolution in access technology has already started, taking us beyond the strict copper wire local loop into the multimedia world with two emerging technologies; fibre optics and digital radio. The combination of emerging radio technologies [Tuttlebee92], new available frequency bands and the perceived market's size has produced a highly competitive, yet still low risk-taking, industry.

The requirement to increase the capacity may be summarised as follows; when this research was started there were about 32 million mobile workers in Europe, who were regarded as the potential market for the second generation cellular system. The overall potential market for mobile telephony in Europe is shown in figure 1.1 [PAConsulting88], with the total penetration in the year 2000 being foreseen to be around one person in fourteen (70 in 1000).

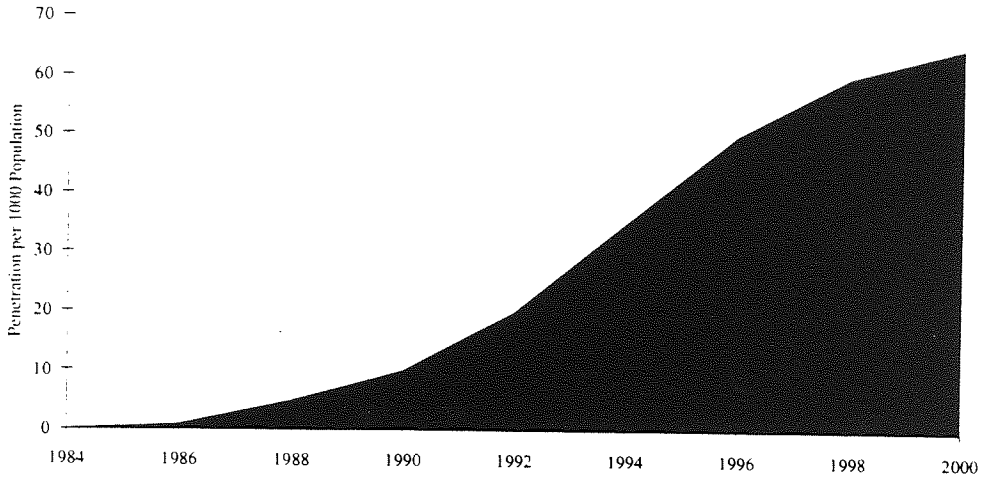


Figure 1.1
Growth of Cellular Users

The Third Generation system has been proposed [Chia92, Steele93] for roll out in the year 2000+, by which time it is forecast that the UK will have a user base of at least seven million mobile subscribers. Given that this system would be an integral part of a European-wide system and the physical limitations on cell splitting, it would be likely to require at least several hundred MHz of spectrum. There are other factors that need considering when addressing spectrum use and efficiency. Firstly, at present we are assuming an average 0.4% (5 minutes per day) usage factor which is likely to increase as the competition in the market increases and as cellular technology is accepted more into everyday lives. Secondly, the digital information era is progressing, with greater use of data and computer information transfer. Combined with high levels of coding and data compression this leaves an unknown development in the communication requirements. In order to maintain quality of speech and coverage in the Third Generation, greater bandwidth efficiency is required that is greater than any system which has currently been proposed.

1.1 Spread Spectrum

Spread spectrum was first used in the early 1950s by the military, who have continued to be its main user to date. This has led to spread spectrum techniques taking a low profile due to their distinct military advantage [Rustad90, Reed88] based on the robust immunity to interference and jamming. The major non-military use to date has been for space communications because of its ability to provide better immunity against noise and interference than any other modulation technique.

The recent upsurge in open discussions of spread spectrum techniques occurred when the cellular telephone authorities and service providers looked at better ways of utilising their overcrowded spectrum during the research and design of the Second Generation of cellular telephone systems. They are suited to the mobile environment which is severe, with propagation conditions varying rapidly both spatially and spectrally, especially at the higher frequencies, where these conditions become even more severe.

A spread spectrum signal has the ability to co-exist with many types of signal, and with variations of itself, thus enabling 'forward compatibility'. It is therefore ideally suited for the ever increasingly crowded part of the radio spectrum allocated to cellular mobile radio. The user density can thus be increased as and when new methods become available. One avenue that is also being researched is the use of spread spectrum as a secondary system, sharing the same bandwidth with a primary system. The primary could be any transmission from low-usage government bands to broadcast stations and line-of-sight data trunk systems. This enables two services to co-exist in the same bandwidth with a minimum of interference. In the First and Second Generations, frequency planning has become one of the major ongoing problems, especially in an ever-changing cellular network with changing frontier boundaries and an increasing number of multi-service providers.

It is envisaged that CDMA techniques will come into their own in these harsh environments, when other multiple access techniques may no longer be capable of reliable communications. These environments range from battle grounds and emergency situations, to the utilisation of higher and higher frequency bands. In rural areas, where mobile user densities are low, CDMA allows operators to share these areas without the complex frequency planning required with other technologies.

Spread spectrum has been applied to optical systems and has shown promising results [Petrovic91, Bozward93/1] due the extremely large bandwidth. Data rates in these systems

are in the order of many thousands of Giga bits per second and are free from any planning restrictions on bandwidth.

1.2 The Design of a Spread Spectrum Signal

The process of demodulation or data recovery is done in the receiver typically by either matched filter detection or by correlation, both of which have been covered theoretically in [Ziemer92]. The latter process achieves its performance by replicating the transmitted signal at the receiver and presuming that the noise accumulated during transmission is sufficiently small for the two signals to match/correlate, thus enabling the data to be determined. Thus techniques which predict the noise induced when traversing the medium, such as interference cancellation (section 3.3.9), increase the signal-to-noise ratio. This can, therefore, be seen as the ratio of one of the known signal amplitudes against the other signals in the spread spectrum system. These other signals have amplitudes which can not be predicted or about which no knowledge can be gained.

Shannon stated [Shannon49] that by using optimum coding, to provide data transmission over a noisy channel a maximum data rate C could be achieved. To approach the theoretical maximum information capacity bound [Crepeau91], which is known as the Shannon limit, we must transmit more information within the signal of which the communicating pair have a priori knowledge. This has been done classically in DS-CDMA systems by spreading the signal using a Psuedorandom Noise (PN) [Davenport58] code.

The received spread spectrum signal is mapped from a large set of signal components spread over a wide band into a small, or even single-valued, set occupying, ideally, just enough bandwidth to pass the baseband modulation components. This process of mapping is usually referred to as correlation and is the most important process in reception of code division multiple access communications. This process both confirms when the wanted signal is present within the noise components received and rejects the unwanted components which have been injected into the system by other transmissions. The correlator in a spread spectrum receiver is that point beyond which interfering signals cannot be permitted to pass, at least not in their original form, otherwise a narrowband interfering signal could enter the post-correlation circuits and masquerade effectively as a desired, properly de-spread signal. The prime purpose of the correlator therefore is to match a local reference signal to the desired incoming signal and thereby reproduce the embedded information-bearing carrier as an output. This is achieved when maximum auto-correlation occurs at the output and only occurs when the code is at the zero shift position.

When the incoming desired signal is exactly matched with the receiver's reference, no noise is generated by this process within the received signal. Then the received signal-to-noise ratio is determined by the noise contributed by each user in the system and the cross-correlation properties of the spreading code. At the correlator's output, the unwanted signals will appear multiplied by the local code. The noise contributed, therefore, is determined by the interference users' transmitted powers and the system processing gain. Factors that are considered to increase the capacity by decreasing the system noise, such as voice activity detection, sectorisation and various adaptive processing techniques, which will be discussed in Chapter Two.

The main process of detection is the correlation of two signals, one generated locally and the over received over the medium. The correlation of the signals in the receiver is very important and, therefore, the design of the signal needs to take into account this process. The requirements are, therefore, to construct the spread spectrum signal so that:-

1. The auto-correlation has a very high peak at the zero shift point and a very small value elsewhere. This ensures that code synchronisation and acquisition is achieved within a minimum time.

2. The cross-correlation value is small, with any correlation run being as small as possible. This ensures that the bit-error-rate is kept to within the required limits.

1.3 Additional Information within the Transmitted Signal

The previous sections discussed the mechanism of correlation which enables code division multiple access to be performed. The major element in this process is, therefore, the ability to generate a local replica of the transmitted signal. This requires that both the transmitter and receiver have a prior agreed set of parameters, such as frequency, bandwidth, data/chip rates, signal powers, type of code and its starting position and other protocols which enable reliable communications to take place. The manner in which DS-CDMA communications is completed requires the use of an extended bandwidth greater than the minimum otherwise necessary to do so, therefore additional prior information could be added to the spread spectrum signal in three basic ways;

- **Phase** Demodulation correlates the locally generated signal with those received. Varying the phase makes these received signals more individual in their nature.

Changing the phase at a greater rate increases the bandwidth used by the system, which reduces the spectral efficiency.

- **Amplitude** Additional randomness may be added by varying the amplitude and hence the instantaneous power transmitted. This will tend to decrease the gain achieved by the use of power control. If this is to be used, therefore, the average power must still be kept to a level which resembles that of the power control.

- **Frequency** The process of demodulation when seen in the frequency domain is one of matching the time-varying spectra which can be translated into frequency components. Those at the outer edges of the spectrum provide the information which details the transitions, which is why MSK reduces these transitions and thus the bandwidth used. By varying the centre frequency, the components which match are reduced.

These attributes may be altered in a pre-determined way, enabling both the transmitter and the receiver to predict the signal to a greater extent and, therefore, enable a better signal-to-noise ratio to be obtained at the demodulation stage within the receiver. It is essential that these systems are synchronised so that the variation may be used to provide a gain within the desired system. Synchronisation in a DS-CDMA system is a very important consideration for reliable communications. This research, therefore, does not consider synchronisation as it has already been extensively researched [Palsule86, Jovanovic88, Polydoros81 & 84, Cheng90].

1.4 Thesis Overview

Chapter Two of this thesis introduces the fundamental mechanics of spread spectrum, with a particular emphasis on direct sequence. It then investigates various types of code and their correlation properties, this being one of the important factors that needs to be considered in the reception of spread spectrum signals. This chapter moves on to discuss mobile architectures used within mobile networks and how these determine the capacity of a spread spectrum system. The capacity of a spread spectrum system is derived for both architectures under consideration, namely PMR and Cellular. Results are presented from the simulation of a PMR network to confirm the theoretical derivation. The chapter then goes on to examine techniques which have been implemented or proposed to increase the capacity of a DS-CDMA system.

Chapter Three investigates the use of frequency reuse, frequency management and sectorisation. The chapter goes on to develop toroidal sectorisation which provides an increase in system capacity of around 50%, it then discusses the benefits which the technique may provide and compares the results obtained with those from cell clustering techniques.

A model is developed in Chapter Four to simulate the various DS-CDMA networks which in the following chapters will be used to examine the various strategies under consideration. The code for this model is provided in Appendices C and D for reference, while in this chapter detail of the construction and implementation are considered.

Chapter Five and Six introduce techniques which were found to increase the capacity of a cellular spread spectrum system by varying either the chip duration or amplitude of the spread spectrum signal. To examine the viability of these techniques, various modulation schemes and spreading sequences incorporating Gaussian noise and orthogonal coding are implemented to provide a system error-rate. The techniques are analysed with regard to system parameters such as error probability, error statistics and the change in the spectral efficiency. Varying chip duration is shown to provide an increase in the capacity of around 60%, whereas varying the amplitude is shown not to. In varying the amplitude it is shown that the first technique considerably reduces and sometimes eliminates the effects of amplitude variations and therefore the near-far effect. This is seen as a major benefit of this technique in conjunction with the increase in system capacity.

Finally, Chapter Seven concludes with a review and discussion of the work covered in this thesis. The two methods which are introduced are assessed and an indication of how they may be developed further is presented. Current capacity enhancing aspects are discussed to provide indications of further areas of research.

Chapter 2

Direct Sequence Spread Spectrum Systems

2.0 Introduction

In this chapter, an introduction to the fundamentals of spread spectrum communications and the techniques used within this technology are presented. The essential components of a direct sequence spread spectrum system are shown in figure 2.1. It consists of a modulator, which combines the incoming data and the carrier frequency, the output of which is then multiplied with the spreading code to form the transmitted output. This signal traverses a channel and, to extract the transmitted data from the signal, the spreading code is removed and the signal is then de-modulated.

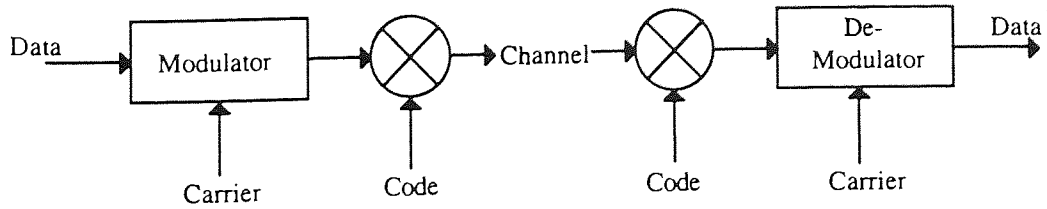


Figure 2.1

Simplified Spread Spectrum System

The chapter next considers the spreading codes used, as these are paramount in the performance of code division multiple access systems. It then moves on to develop the capacity of cellular and PMR DS-CDMA systems. An investigation of the various methods of increasing the capacity of such a DS-CDMA system is then carried out.

2.1 The Spreading Principle

Spread spectrum may be defined [Pickholtz82, Dixon76, Dixon94] as:-

> *"The transmission in which the signal occupies a bandwidth in excess of the minimum necessary to send the information. The band spread is accomplished by means of a code which is independent of the data. Reception is accomplished when the synchronised receiver code is used for de-spreading and subsequent data recovery."*

The spread spectrum system distributes the transmitted energy over a wide band and, therefore, the signal-to-noise ratio at the receiver input is low. The gain provided by the distinct code enables the receiver to operate successfully. The spreading waveform is controlled by a Pseudorandom Noise (PN) sequence or pseudo noise code, which is a binary sequence that is apparently random but may be reproduced deterministically by intended users, therefore pseudo-random. There are two main methods of spreading the spectrum, frequency hopping and direct sequence. This research is only concerned with the direct sequence code division multiple access form of communications. Figure 2.2 is a schematic diagram showing the processes of spreading and de-spreading in a DS-CDMA system, the parameters of which are defined below.

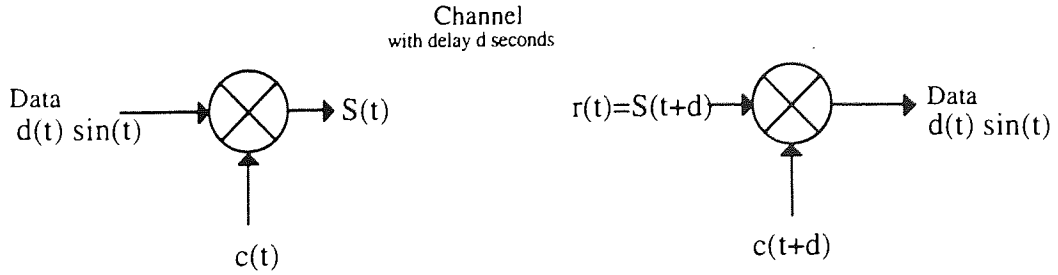


Figure 2.2
Spread Spectrum System Model

Figure 2.3 shows how the processes of spreading and de-spreading take place in the time domain. Spreading takes place by multiplying the modulated information-bearing signal

d(t)sin(t) (for the purposes of illustration one data bit period is longer than the period shown, therefore a continuous sine waveform is shown) by a baseband Pseudo-Noise random sequence **c(t)** to produce the transmitted waveform **S(t)**. When a number of spread spectrum users gain access using different codes then the arrangement is called Code Division Multiple Access (CDMA). In such a system the users' transmission are stacked upon one another in the frequency domain, and thus are named as user stacks. At the receiver, the delayed version of the transmitted waveform which is received as **r(t)** is correlated with the locally generated code **c(t+d)** to produce **d(t)sin(t)** provided that the codes are aligned taking into consideration the transmission delay in the system. This is then de-modulated and the data recovered.

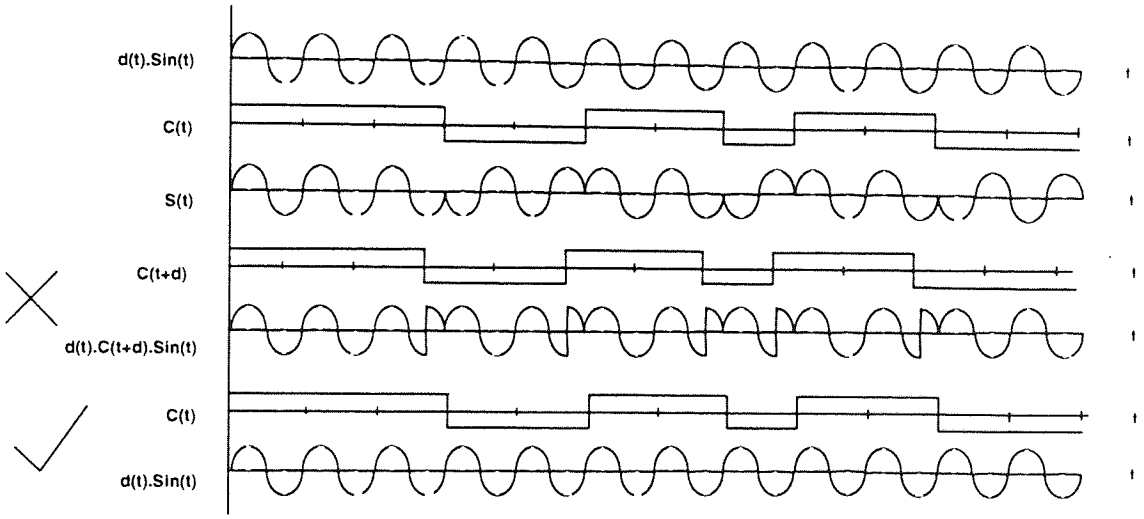


Figure 2.3
Binary Code Spreading and De-Spreading

If the two codes are not aligned *i.e.* the receiver code is not a delayed version of the transmitted spreading code sequence, or where the delay is not equal to that of the transmission, the transmitted signal is effectively multiplied by another spreading code. When the two codes are aligned, the original data is recovered as shown in figure 2.3. Thus all other codes in the multiple access scheme except for the wanted signal are all reduced in power at the correlator with only the wanted signal being de-spread to the required signal-to-noise ratio.

2.1.1 Processing Gain

The figure of merit usually used in spread spectrum systems is the processing gain [Gilhousen91, Qualcomm91]. The processing gain is defined as the time-bandwidth product, which is determined by multiplying the data bit duration by the system bandwidth

used. The Processing gain is therefore a readily estimated quantity if the bandwidth employed in a system is known and the information rate is available, which is usually the case. Therefore, in a spread spectrum system the process gain P_G is:-

$$P_G = B_w T_b = \frac{B_w}{R_b} \qquad 2.1$$

The RF bandwidth B_w is that of the transmitted spread spectrum signal and the information rate R_b is the data rate in the information baseband channel. In direct sequence spread spectrum, a superiour method for calculating the process gain is to assume that the bandwidth is that of the main lobe of the $[sinc\ x]^2$ direct sequence spectrum signal, which is 0.88 times the bandwidth-spreading code clock rate when using BPSK modulation. In Chapter Three the processing gain is shown to provide an indication of the number of simultaneous users the system can sustain within a given error rate, this relationship is given in the capacity equation provided in equation 2.27.

2.1.2 Channel Model

The transmitted signal $S^k(t)$ for some user k may be represented [Skaug85]as

$$S^k(t) = A^k c^k(t) d^k(t) \cos(\omega_o t + \alpha^k) \qquad 2.2$$

where A^k is the signal amplitude, ω_o is the carrier frequency and α^k represents the phase of the k^{th} carrier signal. d^k is the k^{th} user's data signal, such that

$$d^k(t) = \mathbf{d}^k = \ldots d_{-1}, d_0, d_1, \ldots \qquad 2.3$$

where $\mathbf{d}^k$ is a sequence of unit amplitude positive and negative pulses of duration T_b. The code sequence $c^k(t)$ used for bandwidth expansion is similarly given by

$$c^k(t) = \mathbf{c}^k = c_0, c_1, \ldots c_{p-1} \qquad 2.4$$

where $\mathbf{c}^k$ is ± 1 and p is the period of the sequence. The actual transmitted sequence for a system using sequence inversion keying is thus

$$\ldots, d_{-1}c^k, d_0c^k, d_1c^k, \ldots \qquad 2.5$$

The received signal for the case of one user k in a system of N multiple access users and with thermal noise $n(t)$ is given by

$$r(t) = n(t) + \sum_{k=1}^{N} A^k c^k(t-\tau^k) d^k(t-\tau^k) \cos\left[\omega_0 t + \left(\alpha^k - \omega_0 \tau^k\right)\right] \qquad 2.6$$

The matched filter receiver recovers the data by correlating the above with $c^k(t)\cos(\omega_0 t)$ to give the output

$$\begin{aligned} Y^k &= \int_0^{T_b} r(t) c^k(t) \cos(\omega_0 t) dt \\ &= \frac{1}{2} A^k d_1^k T_b + \int_0^{T_b} n(t) c^k(t) \cos(\omega_0 t) dt \end{aligned} \qquad 2.7$$

at $t = T_b$. This is achieved by ignoring the double frequency component of $r(t)\cos\omega_0 t$ and putting α^k and τ^k to zero because of synchronisation. If the matched filter is implemented digitally, the input will be sampled, usually after the carrier is removed. The low-pass implementation usually involves a quadrature technique because no restoration of the coherence carrier phase is feasible. The input signal is first heterodyned by the use of two sinusoids at carrier frequencies which are in quadrature phase as shown in figure 2.4. The signals are then passed into matched filters, the filter outputs are squared and added, and the square roots are taken to form the final output. If it is assumed, therefore, that α^k and τ^k can be set to zero because of synchronisation, the analysis is restricted to the upper branch shown in figure 2.4.

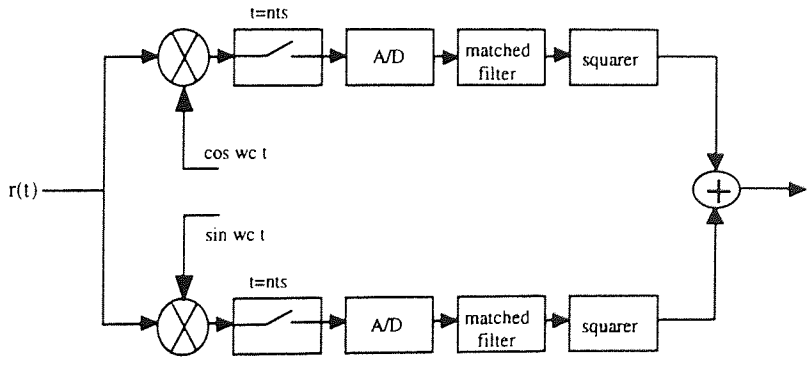


Figure 2.4

Digital Matched Lowpass Filter Demodulator

The digital filter impulse response is then derived by sampling the impulse response of the analogue filter. In this case Y^k can be written as

$$Y^k = \sum_{l=0}^{p-1} r(lt_s)c_l^k$$
$$= \frac{1}{2}A^k d_i^k p + \sum_{l=0}^{p-1} n(lt_s)c_l^k \qquad 2.8$$

where $1/t_s$ is the sample rate, $n(lt_s)$ is the sampled low-pass version of $n(t)$. This equation will be used for the derivation of the different correlation parameters.

2.2 Pseudorandom Noise Sequences

The previous section introduced the principal features governing spread spectrum communication, one very important one being the spreading sequence. The quality of communications in a code division multiple access system is predominantly determined by the cross-correlation function of the codes used for spreading. Ideally a purely random binary sequence should be used to provide very low cross-correlations and off-phase auto-correlations but, by definition, it could not be generated at the local receiver. We use, therefore, a Pseudorandom Noise sequence. This involves careful consideration in selecting the correct code as there are now many criteria which must be taken into account when choosing this sequence of $\pm 1's$. Only binary sequences are covered here and further information may be found in [Suehiro88] and [Kumar91] concerning non-binary code generation.

2.2.1 Code Criteria

The Pseudo-Noise sequences produced for spread spectrum systems must possess certain characteristics:

Ease of Implementation. It should be possible to produce the required code at the desired rate with an adequate length to provide the randomness required. This has caused the implementation to centre around the use of shift registers for the delays with logic gates in the feedback path.

One-Zero Balance. The number of 1's and 0's in a code should, if possible, be equal. This causes the direct-current component of the pulse sequence to be zero.

One-Zero Distribution. The distribution of 1's and 0's in a code will influence the output spectrum, a uniform distribution appears to have a "noise like" or evenly distributed spectrum. The interference becomes less when the chip waveform has a flatter spectrum.

Low Code Repetition Rate. The frequency with which a "pseudo statistic" sequence of 1's and 0's repeats itself should not fall into the required information (modulated) frequency range, since this could cause interfering effects. Normally, it is placed below the lowest modulation frequency.

Good Auto-Correlation Behaviour. The code must be arranged so that it can be identified uniquely by the authorised receiver (auto-correlation). It is also important that a similar code does not cause a positive identification in the decoder circuit, even when this signal is weaker.

Good Cross-Correlation Behaviour. When fed into an identification circuit that compares it to another code (cross-correlation), the received code should not produce an output signal except when the codes correspond. The auto-correlation and cross-correlation behaviour of the codes will be discussed further in section 2.2.3.

Practical and efficient implementation techniques for Pseudo-Noise sequences centre around the use of shift-registers, although other techniques are available such as using chaotic circuits [Bernstein89]. These range in complexity from a single linear feedback shift register to non-linear multiple shift register sets. The maximal linear code sequences (often called M-sequences or PN codes) are widely used for general use for communications and ranging and exhibit the above characteristics.

2.2.2 Classification of Codes

Spreading sequences may first be split into two areas, linear and non-linear codes. Most radio spread spectrum communications have centred around linear codes. The term linear refers to the feedback path between the registers.

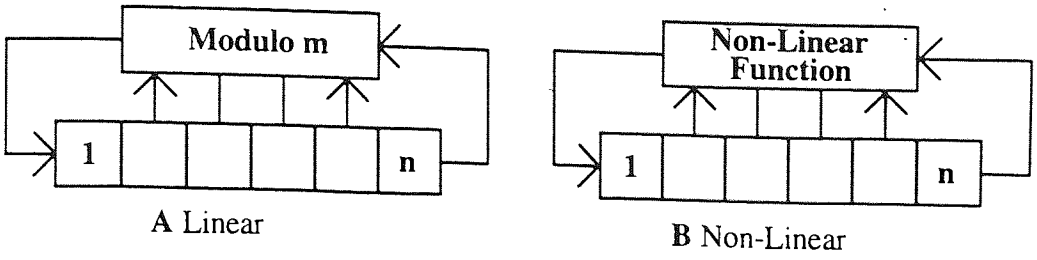


Figure 2.5
Classification of Codes

Figure 2.5 above shows schematically the two types of code generators. **A**, the linear generator has a feedback which takes the outputs of the registers and produces an output which is determined at that instant in modulo m logic. **B**, the non-linear generator may use past register values and any modulo calculation. In a binary system there are only two mutual operations permitted in GF(2), addition and multiplication.

2.2.2.1 Maximal Sequences

Maximal codes are, by definition [Peterson81, Dixon76] the longest codes that can be generated by a given shift register or delay element of a given length. In binary shift register sequence generators, which are the only type considered here, the maximum length sequence has a length p of $2^n - 1$ chips, where n is the number of shift register stages. Many of the properties required in the above section are inherent in maximal codes and are outlined below.

1. A maximal length sequence contains one more logic 1 than logic 0. The number of ones in the sequence is $(p+1)/2$.
2. The modulo 2 sum of an m-sequence and any phase shift of the same sequence is another phase of the same m-sequence.
3. If a window of width n is slid along the sequence for p shifts, each n-tuple except the all zero n-tuple will appear exactly once.
4. The periodic autocorrelation function $\psi_a(l)$ is 2 valued and is given by

$$\psi_a(l) = \begin{cases} 1.0, & l = ip \\ -\dfrac{1}{p}, & l \neq ip \end{cases} \qquad 2.9$$

where i is any integer and p is the sequence period.

5. Defining a run as a sub-sequence of sequential symbols taken from within the m-sequence, then, for any m-sequence, there are

1. 1 run of ones of length n
2. 1 run of zeros of length n-1
3. 1 run of ones and 1 run of zeros of length n-2
4. 2 runs of ones and 2 runs of zeros of length n-3
5. 4 runs of ones and 3 runs of zeros of length n-4

:

:

n. 2^{n-3} runs of ones and 2^{n-3} runs of zeros of length 1

The proofs for these are not covered here but can be seen in [Dixon76, Ziemer85, Torrieri92] and are used in the definition of the auto-correlation and cross-correlation of maximal length codes.

2.2.2.2 Gold Codes

Gold [Gold67] describedan analytical technique which tells how to select combinations of m-sequences with a specified upper bound on the cross-correlation function. Gold codes are generated by the modulo-2 addition of a pair of different maximal linear sequences of the same length. The two code generators are started with initial conditions offset by various amounts to give different output code sequences. Figure 2.6 shows a typical configuration of a Gold code generator Any two-register Gold code generator of length n can generate $2^n - 1$ maximal-length sequences of period $2^n - 1$ plus the two maximal base sequences. A multiple-register Gold code generator can generate $(2^n - 1)^r$ non-maximal sequences of length $2^n - 1$, plus r maximal sequences of the same length, where r is the number of register sets and n is the register length.

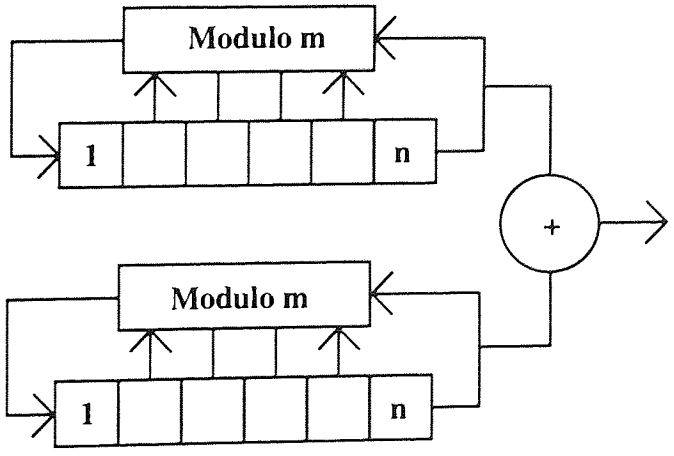


Figure 2.6
Gold Code Generator

The cross correlation between the codes is uniform and bounded between Gold codes available from a given generator. Generation of proper Gold codes requires that the codes need to be properly chosen. Gold codes are chosen by the following algorithm.

1. Select a polynomial of the proper degree from a table (e.g. [Peterson81]) (an n-stage shift register requires an n^{th} degree polynomial).
2. Read the number (k) in the polynomial roots column associated with the polynomial selected.
3. If the code generator has an odd number of stages, then calculate 2k+1. If the number of stages is even, calculate 2(k+2)/2+1.
4. The number calculated in step 3 is the polynomial root of a second code that completes a preferred pair.

2.2.2.3 Sparse Non-Linear Codes

An n-stage binary shift register has $(2^{2n})^2$ possible combinations of feedback/feedforward connections. Most of these are of the non-linear feedback type, which are difficult to analyse and usually exhibit undesirable properties. The term *non-linear* here referrs to the binary operation which combines two elements of the code together to form another. The most serious shortcoming of linear feedback is that only 2n successive outputs are required to determine the feedback connections and the initial state of an n-stage register, leaving no doubt about the entire sequence.

One common class of non-linear generators [Groth71, Key76] has linear feedback determined by a primitive characteristic polynomial as used in linear generators but with non-linear logic on only two stages to provide the output. The generators, shown in figure 2.7, are also known as sparse code generators due to the code having a much greater number of zeros than ones. These generators are commonly used in optical CDMA networks [Chung89] to reduce optical interference.

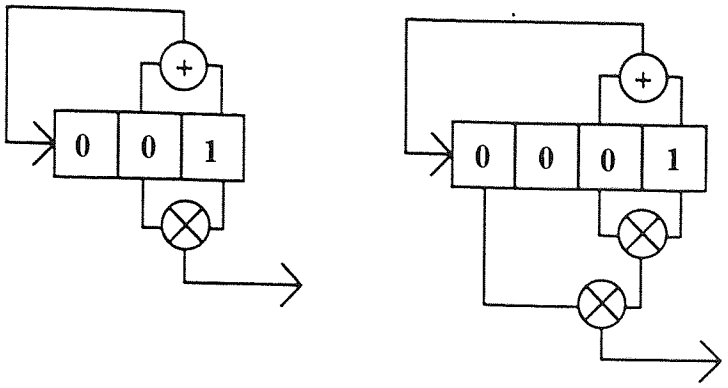


Figure 2.7
Non-Linear Code Generators

These generators have linear equivalents which require a greater number of stages of registers to produce the same codes. The output from these generators is the product of different phases of the same sequences.

2.2.2.4 Bent Sequences

These sequences are a family of non-linear sequences which achieve Welch's lower bound [Welch74] on simultaneous cross-correlation and auto-correlation magnitudes. They are so named because they are mathematically bent [Olsen82, Yarlagadda89] to achieve the correlations required. These sequences, therefore, have been specially developed for a low cross-correlation between codes in a multiple access environment.

The sequence generators are easy to randomly initialise into any assigned code and hence can be rapidly 'hopped' from sequence to sequence for CDMA operations. The codes are non-linear in that the order of the linear difference equation satisfied by the sequence can be orders of magnitude larger than the number of register elements in the generators that produced it.

2.2.3 The Correlation Functions

In an asynchronous DS-CDMA radio network, the received signal-to-noise ratio is determined by the noise contributed by each user in the system and the cross-correlation properties of the spreading code. The auto-correlation function is important for synchronisation and tracking the received code. It is therefore important to understand these properties in system design. The following investigation covers only maximal length sequences, [Chapman91, Kumar91, Lunayach83, Skaug80].

2.2.3.1 Auto-correlation Parameters

Maximal length shift register sequences are binary sequences of length $p = 2^n - 1$,where n is the number of shift register stages. A general representation of the autocorrelation function is

$$\psi_a = \frac{A-D}{p} = \frac{p-2D}{p} \qquad 2.10$$

where A is the number of places where the code $c_0,\ldots,c_{p-1}$ with period p and its cyclic shift agree, and D is the number of places where they disagree, so that A+D=p.

An m-sequence has the property that a period of the sequence contains 2^{n-1} 1s and $2^{n-1}-1$ 0s because there are 2^{n-1} odd numbers between 1 and $2^{n-1}-1$ with binary representation ending in 1, and $2^{n-1}-1$ even numbers in the same range with binary representation ending in 0. The state of the shift register runs through these numbers so that, in a window of width n slid along an m-sequence, every $2^n - 1$ non-zero binary n-tuple is to be seen once and only once. Then, if we apply the property that the sum of an m-sequence and a cyclic shift of itself is another m-sequence so that $c + c_i = c_j$, the auto-correlation becomes

$$\psi_a(i) = \frac{(2^{n-1}-1)-(2^{n-1})}{p} \qquad 0 < i \le p-1 \qquad 2.11$$

The autocorrelation may be defined for an m-sequence as

$$\psi_a(0) = 1 \qquad i = 0$$
$$\psi_a(i) = \frac{-1}{p} \qquad 0 < i \le p-1 \qquad 2.12$$

Another factor which is of interest in analysing the codes, and especially in determining the auto-correlation function, is the distribution of runs of 1s and 0s. A run is defined to be a maximal string of consecutive identical symbols. In any m-sequence, one half of the runs have length 1, one quarter have length 2, one eighth have length 3 and so on, as long as these fractions give an integral number of runs. In each case the number of runs of 0s is equal to the number of runs of 1s.

When the receiver is l code elements (chips) out of phase with the data bit edges as will be the case, at least initially, before the data fills the matched filter, the output is given by

$$\begin{aligned} Y^k &= \frac{1}{2} A^k \{ d_i(c_0^k,\ldots,c_{p-l-1}^k),[n(0),\ldots,n(p-1)] \}c^k \\ &= \frac{1}{2} A^k \psi_a^k(l) + [n(0),\ldots,n(p-1)]c^k \end{aligned} \qquad 2.13$$

and the non-periodic auto-correlation function is given by:-

$$\psi_a^k(l) = \sum_{j=0}^{p-l-1} c_j^k c_{j+l}^k \qquad 0 < l < p \qquad 2.14$$

For any $l \neq 0$ it is desirable that $\psi(l)$ is small compared with p to avoid false synchronisation. For applications over multi-path channels, knowledge of the non-periodic auto-correlation is also important. For a two path model the correlation process may look like

$$Y^k(l) = \frac{1}{2} A^k \left[d_i(c_0^k,\ldots,c_{p-1}^k) + d_{i-1}(c_{p-1}^k,\ldots,c_{p-1}^k) + d_i(c_0^k,\ldots,c_{p-l-1}^k) \right] c^k \qquad 2.15$$

This is equilivant to:-

$$Y^k(l) = \frac{1}{2} A^k \left[p + d_{i-1}\psi_a^k(p-l) + d_i\psi_a^k(l) \right] \qquad 2.16$$

where $\psi_a^k(p-l)$ is defined from equation 2.14 when l is replaced by $(p-l)$, so that

$$\psi_a^k(p-l) = \sum_{j=0}^{l-1} c_j^k c_{j+p-1}^k \qquad 2.17$$

The first term on the right-hand side of equation. 2.15 is the even (periodic) auto-correlation function resulting from the propagation path to which the receiver locally generated code is aligned. The next two terms are due to the second propagation path which has been delayed by lt_c sequence chips. The system under consideration uses binary spreading and data sequences producing two cases which must be considered. If $d_{i-1} = d_i$ then the sum of the last two terms is also an even (periodic) auto-correlation function. Thus the even correlation function is related to the periodic auto-correlation function by

$$\psi_{ae}^k = \psi_a^k(l) + \psi_a^k(p-l) \qquad 2.18$$

When $d_{i-1} \neq d_i$ the odd correlation φ^k is given by :

$$\psi_{ao}^{k} = \psi_{a}^{k}(l) - \psi_{a}^{k}(p-l) \qquad 2.19$$

The performance of the spread spectrum system in a multi-path situation will thus depend on the spreading code characterised by its even and odd auto-correlation properties. The way in which codes should be optimised will depend on the performance criteria chosen for the system analysis.

2.2.3.2 Cross-correlation Parameters

If the spread spectrum system is expanded to contain N users at the same carrier frequency in a code division multiple access system, the received signal is defined in equation 2.6. The signal may then be correlated with $c^{k}(t)\cos\omega_0 t$, $k \in [k]$, which for the digital realisation, gives the output

$$Y^{k}(l) = \frac{1}{2}A^{k}d_{i}^{k}p + \sum_{r=1,r\neq k}^{k}\frac{1}{2}A^{r}\{d_{i-1}^{k}\left[c_{0}^{r},\ldots,c_{p-l(r)-1}^{r}\right] + d_{i}^{k}\left[c_{p-l(r)}^{r},\ldots,c_{p-1}^{r}\right]\}c^{k}\cos\beta^{r} + \sum_{l=0}^{p-1}n(lt_{s})c_{1}^{k} \qquad 2.20$$

where $l(r) \in [0,1,2,\ldots,p-1]$ and β^{r} is the phase angle between the k^{th} carrier and carrier r. In proceeding between these equations it is assumed that $\alpha^{k} = 0$ and $\tau = 0$. Making use of a non-periodic cross-correlation functions $\psi_{c}^{kr}(l)$ and $\psi_{c}^{kr}(p-1)$, defined similarly to the non-periodic auto-correlation, we get

$$Y^{k}(l) = \frac{1}{2}A^{k}d_{i}^{k}p + \sum_{r=1,r\neq k}^{k}\frac{1}{2}A^{r}\left[d_{i-1}^{k}\psi_{c}^{kr}(l) + d_{i}^{k}\psi_{c}^{kr}(p-l)\right]\cos\beta^{r} + \sum_{l=0}^{p-1}n(lt_{s})c_{1}^{k} \qquad 2.21$$

Again there are two cases to be considered for each of the $[k-1]$ interferences influencing the k^{th} user. If $d_{i-1}^{r} = d_{i}^{r}$ for some $r \in [k-1]$, where $[k-1]$ is identical to $[k]$ without the k^{th} user, then the interference will depend on

$$\varphi_{ce}^{kr} = \psi_{c}^{kr}(l) + \psi_{c}^{kr}(p-l) \qquad 2.22$$

which is called the even or periodic cross-correlation. If $d_{i-1}^{r} \neq d_{i}^{r}$ then the interference will depend on

$$\varphi_{co}^{kr} = \psi_{c}^{kr}(l) - \psi_{c}^{kr}(p-l) \qquad 2.23$$

which is called the odd cross-correlation.

2.2.3.3 Comparison of Cross and Auto Correlation Functions

The cross-correlation of the codes is done over a bit period because a decision must be made on the data bit value. This leads to a partial cross-correlation of the spreading codes, which have a period very much longer than one bit period and, therefore, determine the receiver signal-to-noise ratio at the point of de-modulation and data recovery. The partial cross-correlations for the codes used are given in section 5.3.2. Here it is aimed to show how, by careful code selection, the cross-correlation parameter may be reduced or at least be controlled. Table 2.2 compares the three code sequences discussed in the previous sections for correlation and other important code parameters. One important sequence named Kasami [Kasami71] which was not implemented due to time constraints is used in the table to aid in the comparison.

The bent and Kasami codes possess the same class size and maximum correlation value. There are more values of n for which we can construct Kasami sequences than bent sequences. n must be a multiple of 4 for the bent sequence construction. The correlation bound of the Gold code is significantly inferior to either of the Bent and Kasami codes, but the class size is the square of the other two (Bent and Kasami). Gold codes have their best correlation magnitudes when n is odd, and it is essentially $\sqrt{2}$ times worse when n is even.

	Period	Size of Class	Maximum Correlation Value	n	Maximum Linear Span	Range of Sequence Imbalance
Bent	2^n-1	$2^{n/2}$	$2^{n/2}+1$	4m	$\leq \sum_{i=1}^{n/4}\left(\frac{n}{i}\right)$	1
Kasami	2^n-1	$2^{n/2}$	$2^{n/2}+1$	2m	$3n/2$	$2^{n/2}-1$
Gold	2^n-1	2^n+1	$2^{\frac{n+1}{2}}+1$	2m+1	2n	$\left[1,2^{\frac{n+1}{2}}+1\right]$
Maximal	2^n-1	$\frac{2^n-1}{n}$	$2^{\frac{n+1}{2}}+1$	m	n	1

Table 2.1

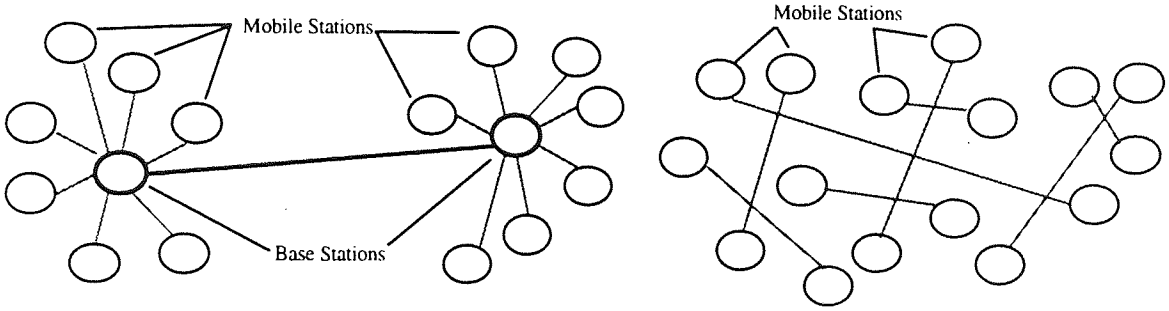
Comparison of Bent, Kasami, Gold and Maximal Sequences

2.3 Mobile Architecture

Within the radio environment there are only two architectures currently under development, these being cellular and private mobile radio:

Cellular This is where all mobile stations communicate with one or more fixed base stations. The calls are then trunked to the appropriate base station for communication with the second user.

Private Mobile Radio Where all users within the localised mobile network (net) may communicate with each other over either one channel or a small number of channels.



Cellular Network **Private Mobile Radio**

Figure 2.8
Mobile Architecture

These networks are illustrated in Figure 2.8, the left hand side being the cellular case and the right the PMR. It can be seen that there are distinct differences between the two topologies. The cellular system may provide cheap, effective spectral usage to a mass market who require to communicate over large mobile areas, using inexpensive equipment. PMR provides a single allocated channel to a licensed set of customers who can afford to pay for this privilege. Communications are confined to a designated area within this pre-defined user group.

These two environments exhibit different degrees of sophistication to the system designer and network provider. Cellular systems are provided for large numbers of mobile users and thus a larger degree of sophistication can be incorporated into the handset, which may then be mass produced. PMR is usually provided for small user groups on an individual basis and therefore additional amounts of sophistication are directly financed by the service provider and user.

2.4 Capacity of a CDMA System

In this section the capacity equations in terms of the number of users N is developed for both cellular and PMR systems from the received signal-to-noise ratios.

2.4.1 Cellular Capacity

The capacity of a cellular CDMA system is bounded due to the base station having to communicate with every active mobile user in its cell. The base station then has an interference limit at which a number of mobile stations may communicate within a given adequate performance signal. Figure 2.9 illustrates this point, where each cell has a central base station and is split into three sectors. Each cell receives interference from its own subscribers and from those in neighbouring cell's. The diagram shows this with different shading for each cells transmissions.

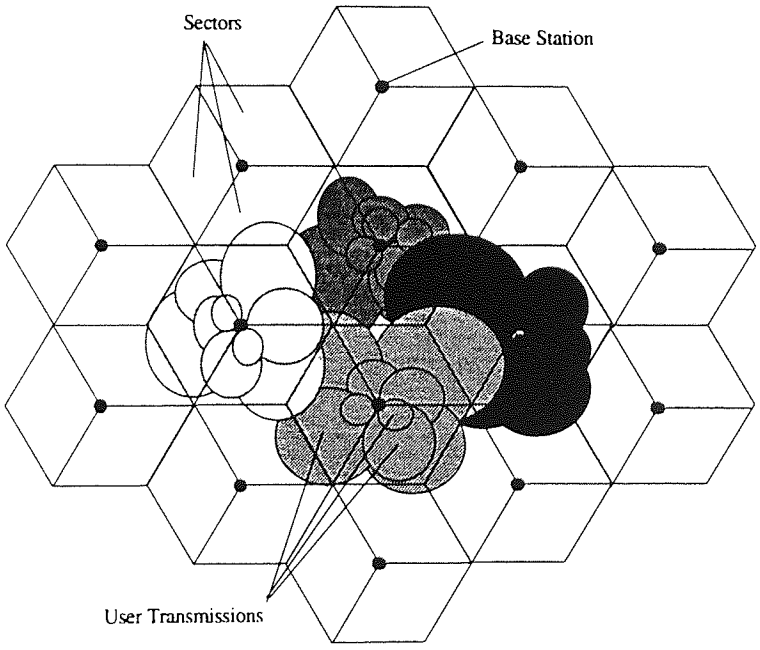


Figure 2.9
Transmission Patterns for a DS-CDMA Cellular System

Here we introduce a single radio cell capacity equation [Gilhousen91, Qualcomm92] to provide an indication of the parameters that dictate typical system performance. The network to be considered is configured in a star, *i.e.* each user communicates with the central cell base station. Each user of the CDMA system occupies the entire allocated spectrum, employing a direct sequence spread spectrum waveform. In a system with N users the signal-to-noise ratio at the base would be;

$$SNR = \frac{S}{(N-1)S} = \frac{1}{N-1} \qquad 2.24$$

This may be converted into the bit energy per noise spectral density by dividing the numerator by the information rate R_b, and dividing the noise by the total bandwidth, B_w. *i.e.*

$$\frac{E_b}{N_0} = \frac{\frac{S}{R_b}}{(N-1)\frac{S}{B_w}} = \frac{\frac{B_w}{R_b}}{N-1} \qquad 2.25$$

The ratio $\frac{B_w}{R_b}$ is the processing gain which will be denoted as P_G. The above equation does not take into account the background noise-to-signal ratio $\frac{N_0}{E_b}$. The basic equation for capacity in terms of the number of users in the system can now be written as:-

$$N = 1 + \frac{\frac{P_G}{E_b}}{N_0} - \frac{N_0}{S} \qquad 2.26$$

This equation assumes perfect power control (*i.e.* all signals arrive at the base station with equal power) and that the cross-correlation function will not significantly degrade the Bit Error Rate (BER). In a large multi-user system the performance is predominantly determined by the noise produced by other users and, as such, the background noise term can be neglected. This equation can be expanded to cover additional system features such as voice activity detection (D), Frequency Reuse (F) and Sectorisation (G) gains which are explained in the next section. Thus,

$$N \approx \frac{P_G . D^{-1} . F . G}{\frac{E_b}{N_0}} \qquad 2.27$$

This equation is seen to have the system gains in the numerator while the required performance signal is in the denominator. Therefore,

$$N = \frac{\text{System Gains}}{\text{Adequate Performance Signal}} \qquad 2.28$$

The equation in this form [Bozward93] shows that, to increase the capacity of the system, we must either decrease the Adequate Performance Signal required or increase the system gains. Classically this has been done by increasing the processing gain of the system but, as indicated above, there are other methods that may be used.

2.4.2 Private Mobile Radio Capacity

PMR capacity is a little more complex than that of the cellular case, in which everyone is communicating into one point. In a PMR network every user is communicating directly with another user, with whom they have selected to transfer information. A much larger geographic area is used in this network than that of a single cell in a cellular system. Figure 2.10 indicates the range of transmission footprints within a PMR network. Here it can be seen that the range of powers which cause the interference is larger, therefore making the system capacity smaller than for the given cellular equivalent.

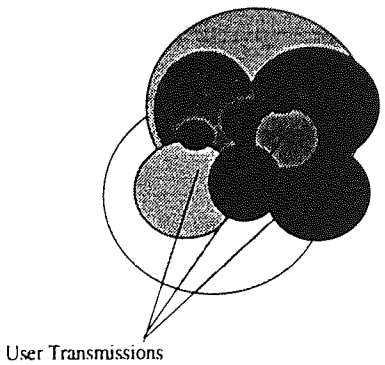


Figure 2.10
Transmission Patterns for a DS-CDMA PMR System

To simplify how we look at this scenario [Bozward93], let us consider this analogy. A DS-CDMA PMR system may be considered to be the same as if there are many people having many different conversations in one room. The walls of the room are considered to be the boundary of the group and no conversations may occur through the walls. Each person has a two way conversation with one other person, each person having to pause during speech to listen to the other person and to take in breath. At any one time, less than half the people in the room are speaking while the others are either pausing or listening [Brady68]. If we consider the case where there are only two people in the room who are having a conversation, their conversation will not be interrupted by any other conversation within

the room (because there is no one else there) and the only noise is contributed by noise generated externally. Now adding one more pair will increase the internal noise in the room but due to speech pauses and the individual personal speech traits, their conversations may still continue unaffected. The loudness required to converse will depend upon the distance between communicating partners and the positions of the interfering conversing pair. *i.e.* if each pair is situated in the corner of the room close together, they may only require a whisper whereas if they had to communicate over the entire distance of the room then the loudness would have to increase. Adding more people in conversation will increase the internal noise until a limit is reached whereby adding one more pair will produce a situation where none of the conversing pairs may communicate; this is the capacity of this environment. This limit is derived from the size of the room, the positioning of the users within the room, their personal speech traits and the speech pause ratio. The two extremes which are easily visualised are the standard classroom environment where everyone is paired into tables and conversation within that pair is very difficult to break or even interfere. The other is in the dealer room of a stock exchange where everyone is shouting, quoting prices and selling stocks. Here both the verbal and visual elements of communication are important to overcome the interference of others and get the message over without misunderstanding.

When we consider the capacity of a PMR system, the loading or noise presented to any one mobile station is not paramount, since all users communicate on an individual basis without the aid of a central base unit. The user density and the distance between communicating pairs become the important factors that determine the received user interference. The maximum user density therefore occurs when the distance between communicating pairs is small compared to the distance between these pairs (interference users). This general problem is known as the Near-Far effect, which occurs when a high powered transmitter is near and has a stronger signal than the gain provided by the system which is required to demodulate the data within a given probability of error. In cellular systems it is reduced by the use of power control, but in a PMR system this problem will still be prominent even with power control as the variation in distances is larger.

DS-CDMA PMR power control is harder to accomplish due to the range of distances (and therefore powers) involved and due to the communications being propagated directly to another mobile unit. Base stations have the advantage of being fixed and able to use larger amounts of power and have very little size restriction in comparison to the mobile which normally has to fit into the hand.

The amount of interference received can, therefore, be seen to be determined by the number of interference communicating pairs, the distance over which they communicate and the distance between the wanted receiver and the interference transmitter. The received user generated noise is bounded by the size of the network and the maximum power used within it, together with the total number of transmissions in the system at any one time.

To derive the maximum number of users in a PMR DS-CDMA system, the signal received is determined by the propagation equation [Ziemer92] shown below.

$$P_r = \frac{P_T G_T G_R}{(4\pi d)^2}\lambda^2 \qquad 2.29$$

where P_r is the power at the receiver, P_T is the power at the transmitter, G_T is the gain of the transmitting antenna, G_R is the gain of the receiving antenna, d is the distance between the communicating pair and λ is the wavelength of the carrier. This equation is used for a line of sight transmitter to receiver, the other transmissions within the network must now be included.

If receiver **1b** is communicating with wanted transmitter **1a** having a series of interference communicating users (**2a, 3a, 4a,...** onwards), the equation now becomes.

$$P_{r1b} = \left[\frac{P_{T1} G_T G_R}{(4\pi d_{1a,1b})^2}\lambda^2 + \frac{P_{T2} G_T G_R}{(4\pi d_{2a,1b})^2}\lambda^2 + \ldots\right] \qquad 2.30$$

where $d_{2a,1b}$ is the distance between the interference user **2a** and receiver **1b**. To include all the possible user interference from the N-1 users, we may write;

$$P_{r1b} = \left[\frac{P_{T1} G_T G_R}{(4\pi d_{1a,1b})^2}\lambda^2\right] + \frac{G_T G_R}{16\pi^2}\lambda^2 \sum_{n=1}^{N-1}\left[\frac{P_{Tn}}{d^2_{1n,1b}}\right] \qquad 2.31$$

This equation has been written so that the two parts of the right hand side of the equation are the signal (left of the plus sign) and noise (right of the plus sign). From this, the signal-to-noise ratio at the receiver which has been generated by the N-1 communicating users may be determined:

$$\frac{S}{N}=\frac{\frac{P_{T1}G_TG_R}{(4\pi d_{1a,1b})^2}\lambda^2}{\frac{G_TG_R}{16\pi^2}\lambda^2\sum_{n=1}^{N-1}\left[\frac{P_{Tn}}{d^2_{1n,1b}}\right]}=\frac{\frac{P_{T1}}{d^2_{1n,1b}}}{\sum_{n=1}^{N-1}\left[\frac{P_{Tn}}{d^2_{1n,1b}}\right]} \qquad 2.32$$

In the above equation it is assumed all users are communicating using the same type of antennae, which in a PMR network may be the case. To convert this into the energy-per-bit to noise spectral density ratio we must multiply by the bandwidth-data rate ratio.

$$\frac{E_b}{N_0}=\frac{\frac{P_{T1}}{d^2_{1n,1b}}\cdot\frac{B_w}{R_b}}{\sum_{n=1}^{N-1}\left[\frac{P_{Tn}}{d^2_{1n,1b}}\right]} \qquad 2.33$$

The bandwidth-data rate ratio was defined in equation 2.1 as the processing gain in a DS-CDMA system. To calculate the maximum number of users the above equation requires rearranging. To simplify the equation further, the network mean power and distance used within the area can be calculated and used to produce $\frac{\overline{P_N}}{\overline{d_N^2}}$, which can then be used in the PMR network capacity equation.

$$N=\frac{P_G\cdot\frac{P_{T1}}{d_1^2}}{\frac{E_b}{N_0}\cdot\frac{\overline{P_N}}{\overline{d_N^2}}} \qquad 2.34$$

The sectorisation gain G, which represents basically the reduction in the noise received by having a directional antenna, is not used in PMR schemes as omni-directional antennas are used. A PMR network must, therefore, be a factor of at least 2.55 lower in capacity, with all other parameters being equal. To put this equation in terms of the cellular capacity, we must combine equations 2.27 and 2.34 to form the following;

$$N=\frac{P_G\cdot D^{-1}}{\frac{E_b}{N_0}}\cdot\frac{\frac{P_{T1}}{d_1^2}}{\frac{\overline{P_N}}{\overline{d_N^2}}} \qquad 2.35$$

The two ratios can be rearranged into,

$$\frac{\frac{P_{T1}}{d_1^2}}{\frac{\overline{P_N}}{\overline{d_N^2}}} = \frac{P_{T1}}{\overline{P_N}} \cdot \frac{\overline{d_N^2}}{d_1^2} \qquad 2.36$$

The capacity is determined by the ratio of average noise power to communication power multiplied by the ratio of average noise distance to communicating distance. The parameters which will affect the capacity equation given above may be summarised as;

- **Area of the Network** This will determine the maximum power used within the network and is set by the geographical size and terrain within which communications are required to be achieved. The processing gain will be required to offset this value when high powers are used in the immediate vicinity of another receiver, which is commonly known as the Near-Far problem.

- **Number of Users** In the cellular case the number of users or the user density is closely related to the processing used due to each user providing an increase in the interference noise floor. This is still the case in a PMR network and therefore may vary as the type of network use changes.

- **Distribution of Users** In a practical system the users are not normally distributed evenly over the entire network coverage area. This becomes very clear with motorway or pedestrian traffic, where the users are forced within a physical area, therefore the distribution of stations will depend upon the type of user mobility, *i.e.* walking, office or transport. Each have their own characteristic distribution patterns.

These factors will cause the users error probability within the network to change as a function of the distance from the edge of the network. The PMR system has been simulated and these results have been compared with the values provided by the theoretical equations.

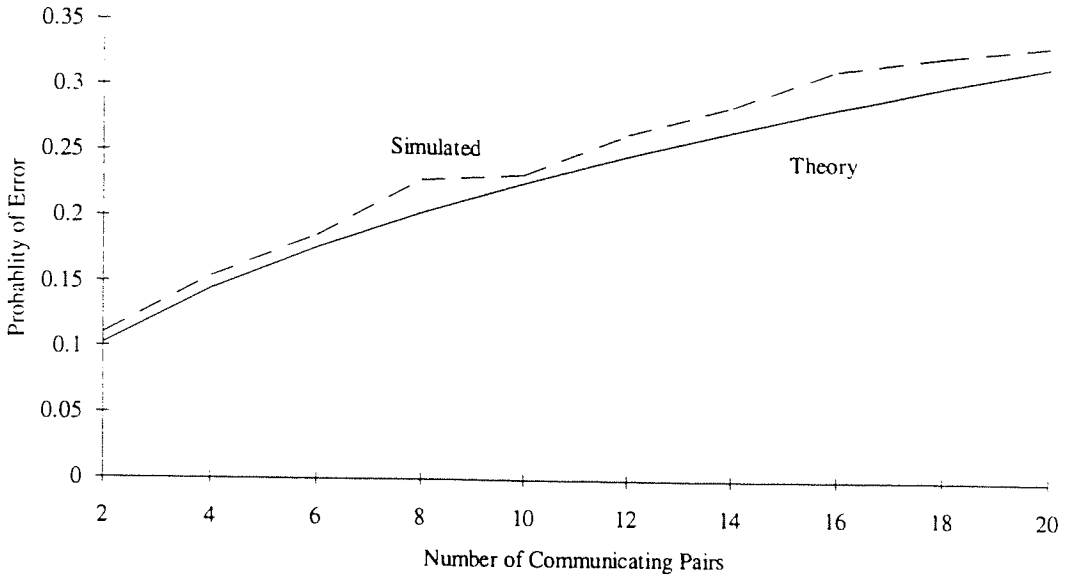


Figure 2.11
A Comparison of Simulated and Theoretical Results for a High Density PMR Network with a Processing Gain of 40

There is a slight discrepancy between the theoretical and simulated results which has arisen from the use of non-orthogonal spreading codes in the simulation. The use of the expression high density used in the figure 2.11 caption relates to the distance ratios, with a value of 10 or greater being dense. The results provided are only a system average and therefore a varying degree of BER within the network can be expected depending upon the statistics indicated in equation 2.36.

2.5. Approaches to Increasing the Capacity of a DS-CDMA Channel

There are a number of ways in which the capacity, however measured, may be increased in a DS-CDMA system and further research is continuing on many fronts [Maurubayashi92, Salmasi91]. The approaches have been split into two, the first delivered in this chapter are those broadly related with the signal design and which are used as a base for the research presented in chapter five. The second set using a frequency management approach are dealt with in the next chapter and lead to an introduction of toroidal sectorisation.

2.5.1 Higher levels of Modulation

The introduction of higher levels of modulation for a constant bandwidth provides a gain in the capacity of any system. A variety of modulation methods have been proposed [Murota81, Webb92, Wan92] for the cellular and PMR environments with an increasing amount of bandwidth efficiency. The decrease in the required bandwidth enables the system to utilise a higher processing gain and/or encoding, such as forward error correction, as is discussed in section 2.5.2.

2.5.2 Error Control

There are two error control techniques which are commonly employed together in spread spectrum systems to ensure that a reliable transfer of information is obtained over the digital communication system [Berlekamp80, Shaft77, Wiggert88, Viterbi79, Wang92]. The first is interleaving, which forces the errors to be in the correct statistical groupings and is used in conjunction with the second, Forward Error Correction (FEC). Using FEC the transmission error rate can be reduced to a desired level without increasing the transmission power above the value for which the capacity was calculated. The coding gain of the system is the difference between the E_b/N_o required to achieve a specified BER without coding and that required to achieve the same with coding. This increases the complexity and bandwidth in terms of bit rate utilised by the system. Performance of the error correction codes is closely related to the noise characteristics of the channel. These characteristics range from error-rate to the statistical relationship between errors and, as a basis, can be considered as a bursty or random error channel. Thus FEC codes are chosen on the channel characteristics over which they are to be used. The predominant factor for error generation in a CDMA channel is the presence of other users' contributions to the noise floor, unlike other channels that are primarily Gaussian in nature. This contribution may be assumed to be Gaussian when bit interleaving is employed. The most effective codes which are recommended [Wang92] for CDMA are the powerful lower rate codes which can provide a larger coding gain than 1/2 rate codes, or the more complex codes with a constraint length longer than length 7.

2.5.3 Orthogonal Coding

In section 2.1.1, the capacity of a cellular system was determined assuming that cross-correlation of the codes does not occur. In practice, the received signal contains many

signals which have been spread using similar codes and which will have differing levels of correlation if the codes are not orthogonal. There have been many methods introduced [Bottomley93, Pahlavan90] to provide varying degrees of orthogonallity to the spread transmitted signal. These may be split into two methods, firstly, by adding another code, which may be called a signature along with the spreading code [Elhakeem92, Lam92, Miller92, O'Farrell91, Rothaus93]. This may be done within a group so as to achieve a correlation value within that group or for the entire network depending on what code is used. Secondly, special codes may be used which achieve Welch's bound [Welch74] such as the Walsh-Hadamard code [Viterbi90] and Bent [Olsen82] codes.

2.5.4 Higher Processing Gain

Spread spectrum communications are able to operate by obtaining a system processing gain. This gain is directly related to the coding used as the code length must usually be longer than the message length for reasonable security and therefore longer than the number of chips per bit. To communicate over high signal-to-noise links, it is advisable to use large processing gains. However, this potentially increases the time taken to achieve synchronisation. By the use of special codes, a synchronisation channel or the use of signal processing [Fair92, Polydoros81] at the receiver it may be possible to overcome this problem. Increasing the number of users and keeping the processing gain constant will produce an increase in the interference received by all users within the system. To reduce this effect of additional users transmissions, we may either increase the spreading rate, and therefore the bandwidth used by the system, or use a spreading code which is orthogonal to all the other codes used in that system.

2.5.5 Power Control

Power Control is important in a multi-mobile station environment to keep user interference within the cell and neighbouring cells to a minimum. Each communicating pair controls their transmitter powers for optimum propagation and BER characteristics. There are basically two methods of power control, open and close loop. Both may be used within one system to provide a fast and accurate control of the power in the system. Perfect power control is, therefore, assumed in the capacity equation. If this is not implemented then there will be a resulting reduction in the capacity of the system [Hulbert93, Simpson93, Viterbi93]. Experience has shown that this may require a dynamic range of control of the order of 80 dB.

2.5.6 Voice Activity Detection

In a typical full-duplex two-way voice conversation, shown in figure 2.12 [Brady68, Gilhousen91, Pickholtz91] the duty cycle of each voice is approximately 35% to 40%.

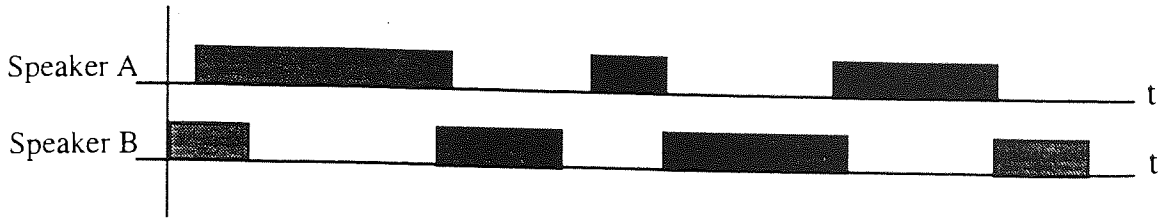


Figure 2.12
Speech Duty Cycle

During the periods that the mobile station is not transmitting the received interference to other stations will be reduced. In practice this may be slightly less since code synchronisation and background comfort noise are sent over the air interface. This can provide a capacity increase through a Voice Activity Detection (VAD) gain of approximately 2.5, plus a reduction in the transmitter average power by a similar factor.

Depending upon the system and the type of information to be conveyed, this factor may change. As an example, a reduction in system performance would occur when measures are taken to ensure traffic security which requires a constant data stream to be conveyed. Modern Codecs are capable of speech transmission using channel rates as low as 8 Kbps, tolerating an error rate up to 0.1%. These transmission rates are likely to reduce even more with further research and rates of 3.8 Kbps have been reported.

2.5.7 Diversity Gain

CDMA wideband modulation reduces the severity of fading [Allpress92, Wan92]. Fading is not completely eliminated as the signals occasionally may not be able to be processed independently. To combat the effects of multipath fading, diversity is required at the receivers. There are three major types of diversity, time, frequency and space.

1. Time Diversity is obtained by the use of interleaving and error correction coding on the data bits, the common form being repeated transmissions over a period of time.
2. Frequency Diversity is obtained through the wideband nature of the signal. The fading bandwidth is usually considered as being of the order of KHz in a mobile system.

3. Space or Path Diversity may be obtained by multiple antennae, multipath processing and through simultaneous links to more than one base station.

2.5.8 Interference Cancellation

The central base station that is receiving all communications within the cell can cancel out individual mobile transmissions since it knows their PN codes and all transmissions are on the same frequency. This may be done in the forward and reverse directions [Goldberg86] at the cell base station. In the forward direction, (cell to mobile), the base station may subtract its own transmitted signal from that received to provide a signal that only contains those transmissions from other stations. This is shown in figure 2.13.

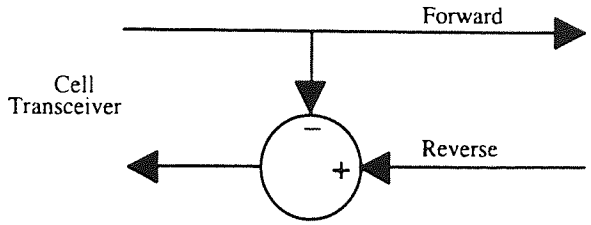


Figure 2.13
Forward Direction Cancellation

In the reverse direction, (mobile to cell), once communications have begun, each known user's signal may be subtracted from the incoming transmissions to leave only the one required for demodulation. This could be extended in the base station network to include the neighbouring base station transmissions, since this accounts for 36% of the interference contributed. The second and higher tiers of cells contribute less than 2% and would not provide a worthwhile gain to justify implementation.

To a limited extent this could also be done by the mobile station, although processing power here is limited by the battery life and size of the unit. The mobile unit could cancel out the pilot, synchronisation, paging and access channels that are known to it from the base stations it currently monitors.

2.5.9 Adaptive Processing

When a mobile station is communicating within a cellular environment, the wanted signal from the base station may be considered to be transmitted from directly in front and the majority of unwanted signals are radiated from other directions. This is indicated in a frequency reuse factor of 0.65. An advantage would be gained by using a directional

antenna at the mobile station to reduce received user interference. These could be provided using an adaptive antenna array [Torrieri92, Widrow85] that would track in real time the movement of the mobile in relation to the base station. These arrays may also provide such features as nulling, in order to cancel out high powered near interference signals and unwanted base stations signals. It is shown in [Lee92] that an antenna with a beam width of 30° provides an increase in the capacity of around 50% per sector.

Adaptive processing is already used to a limited extent in CDMA systems with the Rake receiver design [Fuxjaegar90, Goiser90, Grob90, Kohno90, Lehnert87, Turin84] which provides a multiple path receiver to combat multi-path fading effects. The introduction of more complicated processing [Snytkin91, Cooper90] would be unattractive in the mobile unit since both size and battery capacity are important marketing points in the cellular industry. Simple processing-constrained algorithms that will provide a positive gain for the mobile user, (in terms of the perceived noise) and for the service provider, (in terms of additional capacity) are likely to be the first to be applied to these systems. There are problems with physically accommodating in a hand portable a practical antenna design with the frequencies currently under use, but a vehicle mounted device could provide the starting point required for this technology.

2.5.10 Handover

The propagation conditions which have to be endured in a cellular system mean that coverage is not contiguous between cells. Communication can usually be maintained with at least one base station in the cellular grid, but it is advantageous to be in communication with at least two. The handover is thus called 'soft', as communication is through more than one base station at a time as the mobile moves through the cellular network. This is seen to help with the maintenance and smooth running of the system, providing an increase in the capacity in the sense that lost calls and higher powers can be resolved within the system in a very short space of time.

2.5.11 Conclusion

It was indicated in section 2.4.1, that to increase the capacity either a decrease in the required adequate performance signal level is needed or an increase the system gains. The improvement in the adequate performance signal level has been achieved classically by

increasing the amount of FEC available to a system and by determining modulation schemes that require smaller energy-per-bit per noise bandwidth for a given BER.

In a DS-CDMA system, other techniques are available to reduce the perceived noise and these have been discussed in this chapter. This may be achieved by either reducing the transmitted interference, *e.g.* voice activity detection, power control, or by reducing the interference to the receiver, *e.g.* Error control and interference cancellation. In a cellular system the benefits of these techniques are not always equal in both the forward and reverse directions.

CDMA is ideally placed to utilise the advantages available with adaptive noise reducing techniques which provide benefits in the user perceived noise and therefore increased user densities. Since the base station is interference limited when considering cellular capacity, it would seem to be the easiest and most cost effective place to implement such adaptive processing techniques.

Chapter 3

Frequency Reuse and Toroidal Sectorisation

3.0 Introduction

The single most important technique which has enabled cellular communications to be successfully implemented on one frequency band throughout the world is frequency reuse. This technique reuses the same frequency band at a distance which is determined by the allowable system interference. This chapter discusses frequency reuse and sectorisation in a DS-CDMA cellular network and then develops a sectorisation technique where the cell frequency split is determined by the mobile stations' radial distance from the base station. This has been named toroidal sectorisation [Bozward95].

3.1 Frequency Reuse

The concept of frequency reuse in the first generation of analogue mobile telephones was used to control the amount of co-channel interference to an acceptable limit using spatial separation while increasing system capacity. The received user-generated interference is

determined by the inverse fourth-power law relationship between signal strength and distance. This power-distance relationship is closer to the approximation at greater distances from the transmitter rather than near to it and, in turn, this means that the carrier-to-interference ratio deteriorates for small cells, as does the frequency reuse factor F.

In a CDMA system the frequency reuse factor F is defined [Gilhousen91] as the amount of interference received in a cell from stations operating in neighbouring cells and is given by the following equation:

$$F = \frac{I_{ic}}{(I_{ic} + N_c I_{oc})} = \frac{1}{\left[1 + \frac{CI_{oc}}{I_{ic}}\right]} \qquad 3.1$$

where I_{ic} is the power due to transmissions in the desired cell and I_{oc} is the noise generated by the N_c cells adjacent to the desired cell, which is shown on the left hand side of figure 3.1. The above equation may be interpreted into the following:

$$F = \frac{1}{1 + 6k_1 + 12k_2 + 18k_3 + 24k_4 + ...} \qquad 3.20$$

where $k_1, k_2, k_3, k_4, \ldots$ are the interference contributions from individual cells in rings 1,2,3,.. etc. shown in figure 3.1.

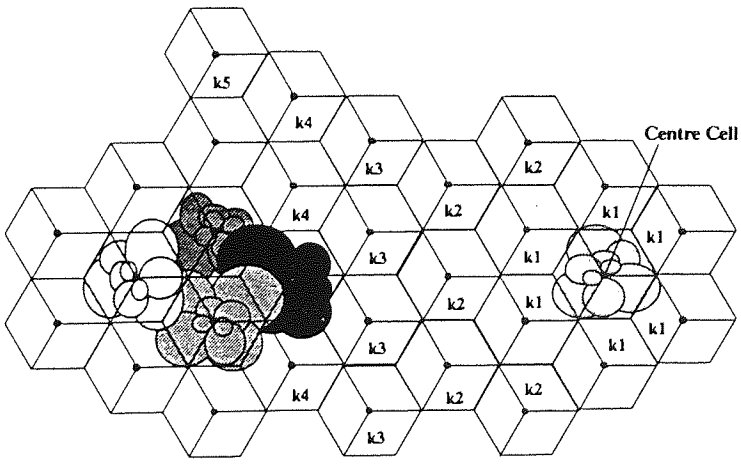


Figure 3.1
Neighbouring Cell Interference

This contribution to noise from all neighbouring cells has been calculated [Qualcomm92] to be approximately equal to 65% of the interference due to the mobile stations operating within the cell. This produces a frequency reuse factor of around 0.61.

3.2 Frequency Management

The first two generations of mobile cellular and PMR communications have all carried a single signal data rate over a fixed bandwidth and/or time duration. It is envisaged that the future needs will dictate a more flexible approach to the bandwidth and other resources which are available within the system. Figure 3.2 [Bozward95] shows the reduction in the system capacity when the total bandwidth is split into a number of DS-CDMA bandwidths.

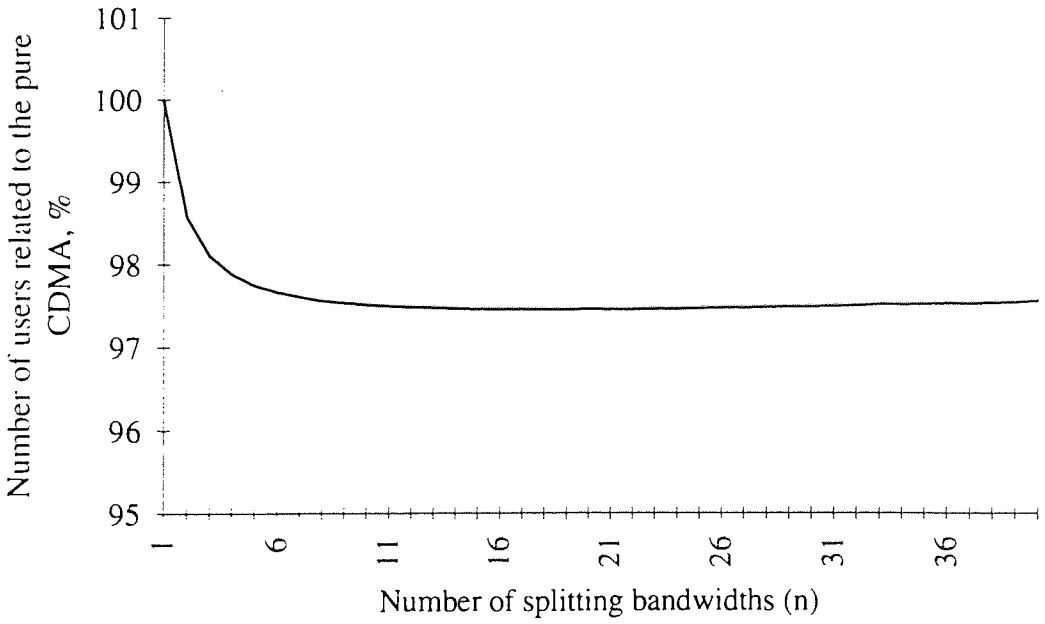


Figure 3.2
System Capacity as a Function of the Number of Splitting Bandwidths

The diagram indicates that a reduction in the system capacity of around 2.5% can be expected when the total bandwidth is split into user transmission stacks. Once the number of stacks becomes greater than 7, a reduction in the system capacity no longer occurs.

The total bandwidth of a DS-CDMA system may be allocated in several ways and figure 3.3 indicates the variety of ways in which the combination of spreading bandwidth, system data rate and the required signal-to-noise ratios can be used. **A** shows the straight case in

which all users are allocated the entire bandwidth and use the same spreading code rate. **B** shows the case where the entire bandwidth is split into sub-bands, which we will call user-stacks; here three user-stacks are shown. This case has been proposed by [Qualcomm92] for the Second Generation of mobile cellular telephones in the USA. **C** and **D** provide a mix of the first two allocations depending upon the required signal-to-noise ratio.

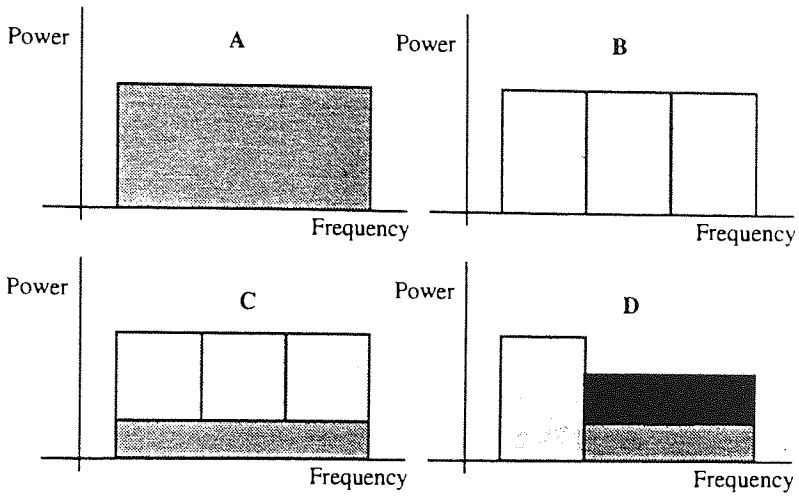


Figure 3.3

Various DS-CDMA Bandwidth Allocations

There are a number of considerations which must be put forward when the bandwidth is to be split. The most important is the reduction in the system capacity.

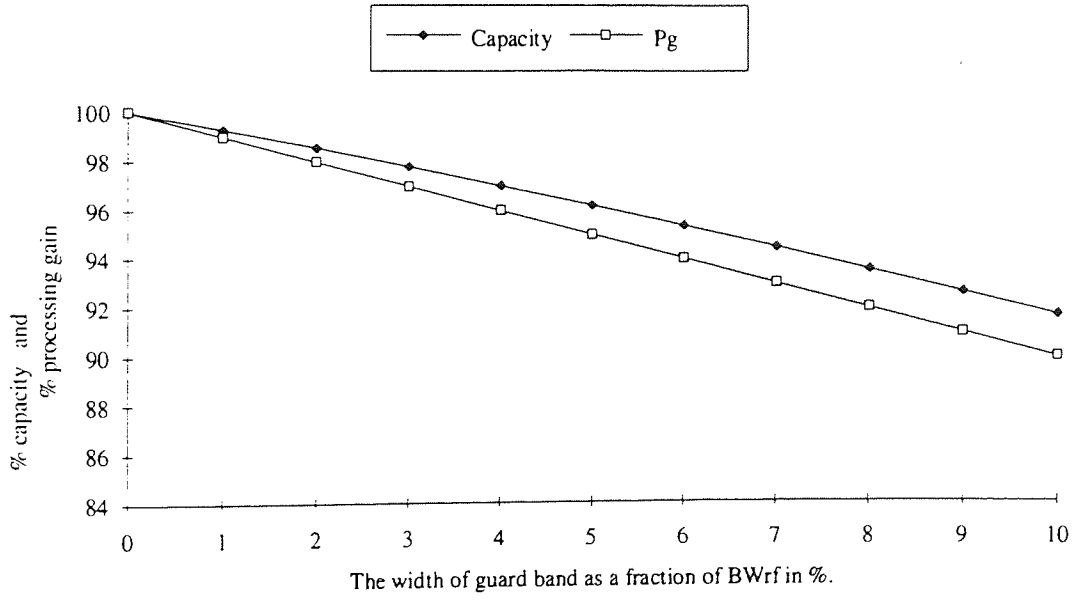


Figure 3.4

Total Capacity and Processing Gain against the Width of the Guardband

Figure 3.4 shows the relationship between the system processing gain and the total capacity against the bandwidth percentage which is allocated for use as the guardband in the system. This diagram indicates that by increasing the guardband of a radio system a reduction in the total capacity which can be occupied in that bandwidth will result.

3.3 Sectorisation

The application of frequency reuse and sectorisation in the first generation of analogue mobile telephones [Graziano86, Graziano89] has been the major advancement in cellular technology. A gain is achieved by splitting a cell into sectors, reducing the total user interference per sector and thus increasing the cell capacity by roughly the number of divisions. In practice this figure is reduced by around 15% due to sectors having to overlap and the true cell footprint being non ideal combined with sidelobe anomalies within the antenna array of the cell. This means a cell divided into three has a sectorisation gain, G, of 2.55.

In the first generation of mobile cellular systems there are two types of division, three sector division proposed by Ericsson and six sector division proposed by Motorola. Each has it's own benefits [Graziano89], the trunking efficiency is better in the three sector system due to a smaller division in system resources, catering for 27% more mobile stations. The six sector system provides an adaptive correction to traffic loads due to antenna overlap and a constant system gain relative to the sector antenna pointing direction. In this system the spatial splitting of cells was achieved by the use of directional antennas.

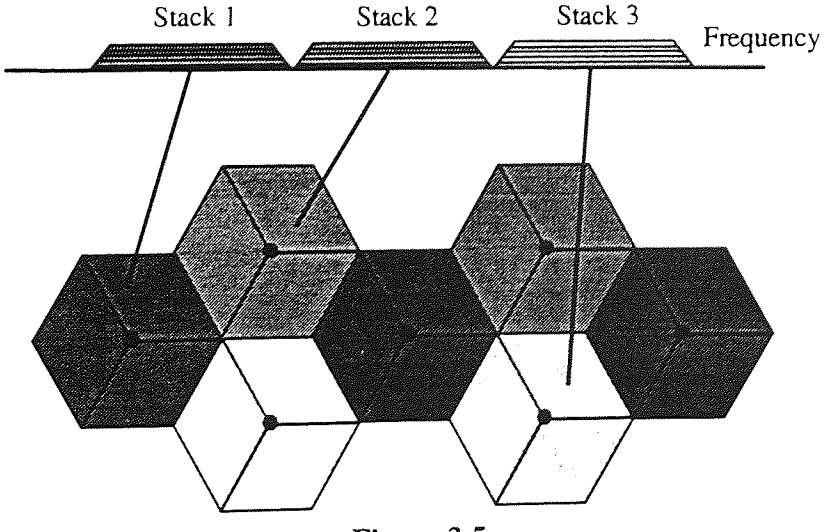


Figure 3.5
Cell Clustering System

Figure 3.5 shows this form of arc sectorisation, which splits the cells into equally sized 'cake' slices in which users communicate towards a central base station. The cell may be split in many other ways in a DS-CDMA system. Another method is to split the cell into distance rings [Bozward95], where the frequency band used is determined by the radial distance of the mobile user from the central base station. This is shown in figure 3.6, where the cell is split into three sectors or toroids (denoted in differing shaded areas) with three frequency stacks assigned to each cell.

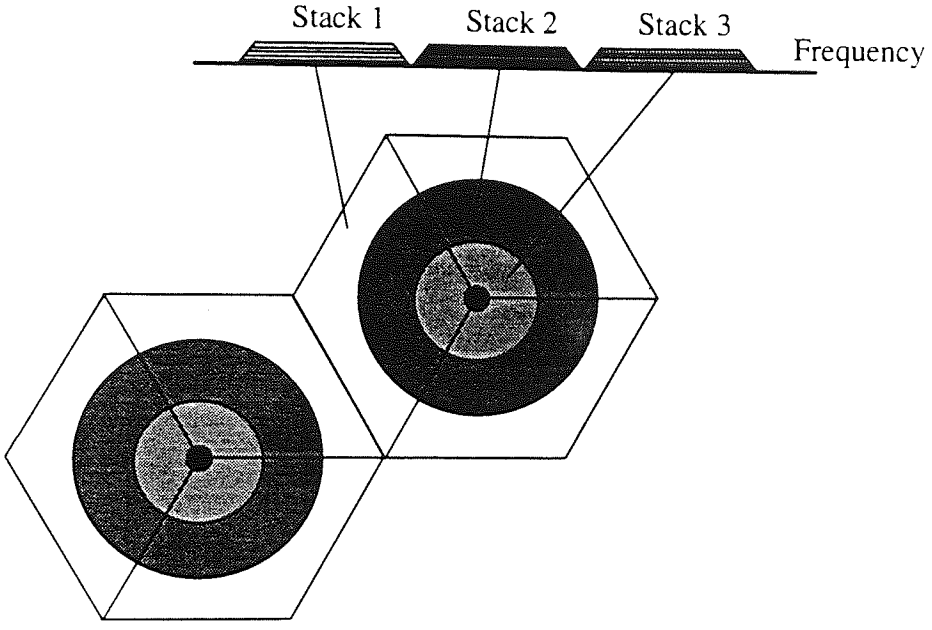
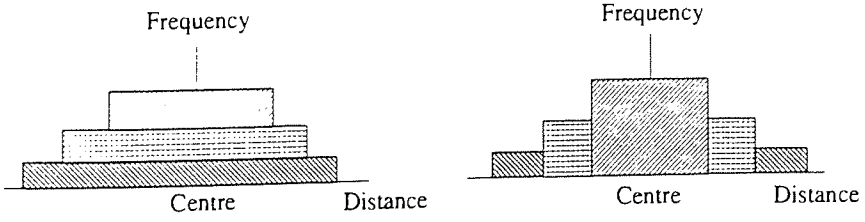


Figure 3.6
Toroidal Sectorisation System

Distance ring cell splitting may be done in one of two ways, firstly by splitting the cell into disks as shown in figure 3.7a and secondly, by using toroids as shown in figure 3.7b. The disk system uses the frequency stack from the centre of the cell up to a certain radius. The toroid system uses the frequency stack between two radii.



a. Disk Frequency Planning

b. Toroid Frequency Planning

Figure 3.7
Two Types of Banding Systems.

These methods are shown to provide a system gain over and above the standard, "all frequencies for all cells", DS-CDMA case. The advantages of this method of sectorisation are that the sectors are independent of the cell radiation pattern and hardware restraints. This form may be combined with the traditional form of sectorisation so that the cell is split into many square like segments, all being operated from one cell base station. This may be used to deliver the required capacity per square kilometre 'in real time' to the cell structure implemented, thus enhancing the system's ability to cope with any capacity 'hotspot'.

This feature can be seen in figure 3.8 which contains a typical distribution of major and minor roads with a small town and an ideal cell structure overlayed. The placing of cell base stations is normally non-ideal and therefore being able to place the centre of the cell in the heart of the small town or at the junction of major roads may be impossible.

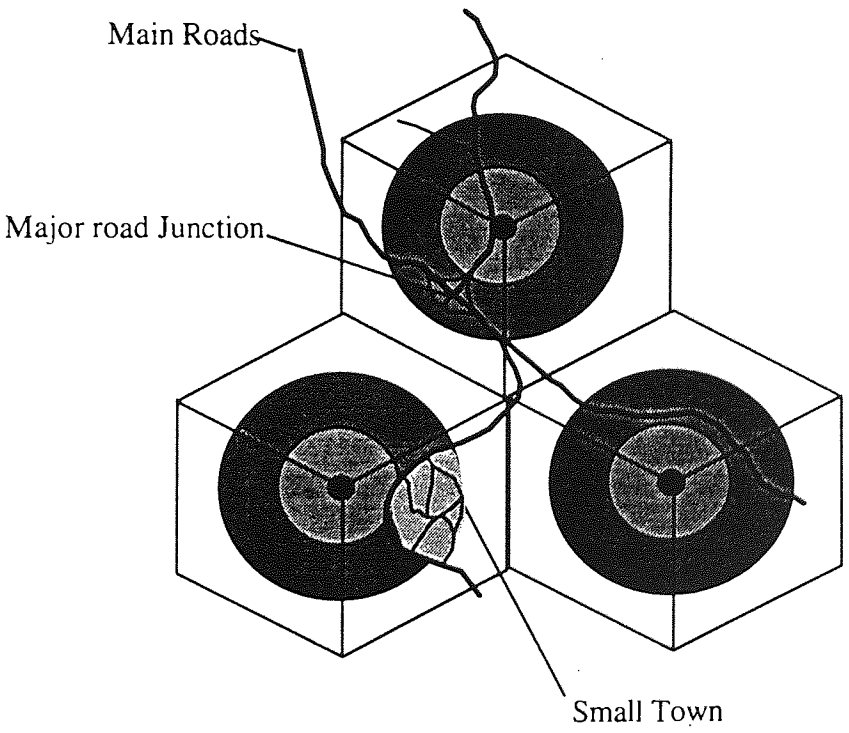
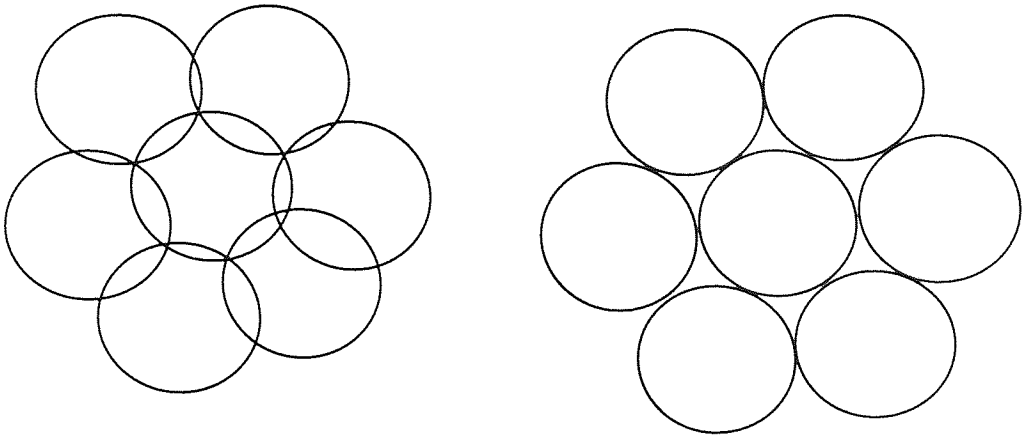


Figure 3.8

Example of Cell Clustering, Arc and Toroidal Sectorisations in Use

The use of toroidal and arc sectorisations provides an extremely powerful method of cellular frequency and capacity planning for these higher capacity requirement areas.

The model developed to compare these forms of sectorisation uses circular cells, therefore there are two cases which require consideration, overlapping cells and non-overlapping cells. These are shown in figure 3.9. In a real system the cells are of no pre-defined shape and therefore the results from this model show the two extreme situations.



a. Overlapping b. Non-overlapping

Figure 3.9
Circular Cell Model Diagram

Figures 3.10 and 3.11 illustrate the results for the two frequency allocation systems under consideration. Each diagram shows two cases, the first, when the cells overlap, covering the entire surface of the ground plane. The second case is when the cells do not overlap and only touch each other at their boundary, thus leaving uncovered areas between cells. Figure 3.10 shows the results for the case where clustering is used, here 2, 3 and 4 frequency stacks are employed in the system.

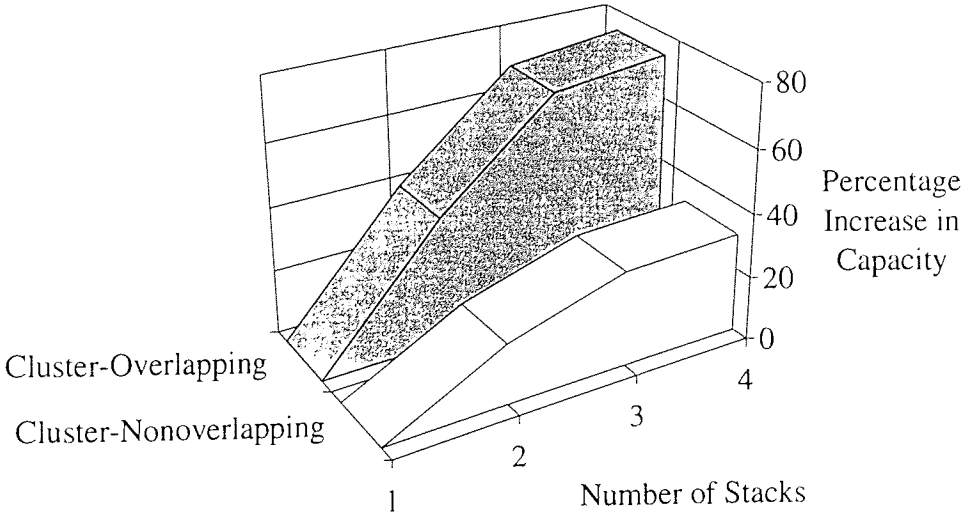


Figure 3.9
Cluster Frequency Management Techniques

Figure 3.11 shows the case where the banding frequency management technique is used; here again 2, 3 and 4 frequency stacks are used. The two methods of banding, disk and toroid, produce very similar results and, therefore, only one plot is presented.

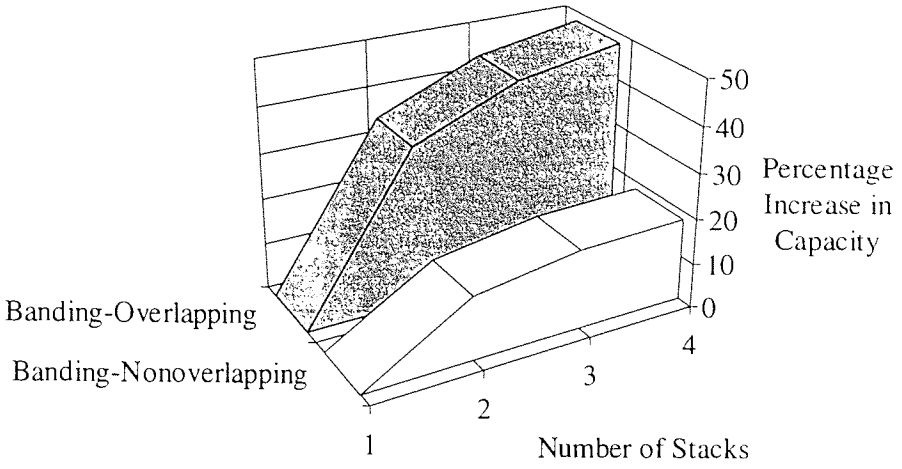


Figure 3.10
Banding Frequency Management Techniques

In summary, the inclusion of sectorisation into a DS-CDMA cellular system increases the overall network capacity whilst increasing the frequency reuse factor. The classical arc sectorisation increases the capacity by a factor of 2.55 for a three-sector system, the toroidal sectorisation increases capacity by 1.2 and the clustering by around 1.3. The frequency planning element is not as critical as in the FDMA and TDMA cases, since interference is proportional to the number of users in the stacks and to the correlation function of the spreading codes used.

3.4 Conclusion

This chapter has discussed the benefits of frequency reuse, management and sectorisation in a cellular communication system. The development of toroidal sectorisation in this chapter has provided a capacity gain of between 20 and 50 percent over and above the techniques which have been discussed in this and previous chapters. This form of sectorisation is free from hardware constrictions and may be achieved under software control on a live network as and when required. Toroidal sectorisation, therefore, though not providing the capacity improvements which can be obtained with arc sectorisation, is a very powerful network planning technique in mobile cellular communication systems.

Chapter 4

Spread Spectrum Cellular System Model

4.0 Introduction

In this chapter the design and construction of a DS-CDMA model is developed using the fundamentals and networking architecture which were presented in chapter two.

The aim of this chapter is to introduce the platform from which the results have been obtained, providing the rationale behind the system and programming decisions. The entire program listings are provided in Appendices A and B for reference and are split into two sections, the first being the library files to which all the models refer and the second being the models themselves.

4.1 Spread Spectrum Model

The system modelled is a DS-CDMA cellular system, where each mobile user is given an individual PN spreading code, which is then used to communicate over the entire common

channel bandwidth. In the DS-CDMA system under study, a number of mobile users are communicating with a fixed base station and the model concentrates on the reverse link (mobile to base) as it is considered to be this link which limits capacity. Perfect power control is assumed as all users' transmissions are received at the base station with equal power. Furthermore, we assume that the standard matched filter [Williams86] is being used at the receiver and that the spread spectrum receiver is capable of perfect carrier recovery (coherent reception) [Holmes82], acquisition, tracking, and bit timing. The spreading codes used were a maximal length sequences, a Gold code, a Bent sequence and the Qualcomm long code which were applied to random data. Figure 4.1 shows a block diagram of the spread spectrum system which has been developed.

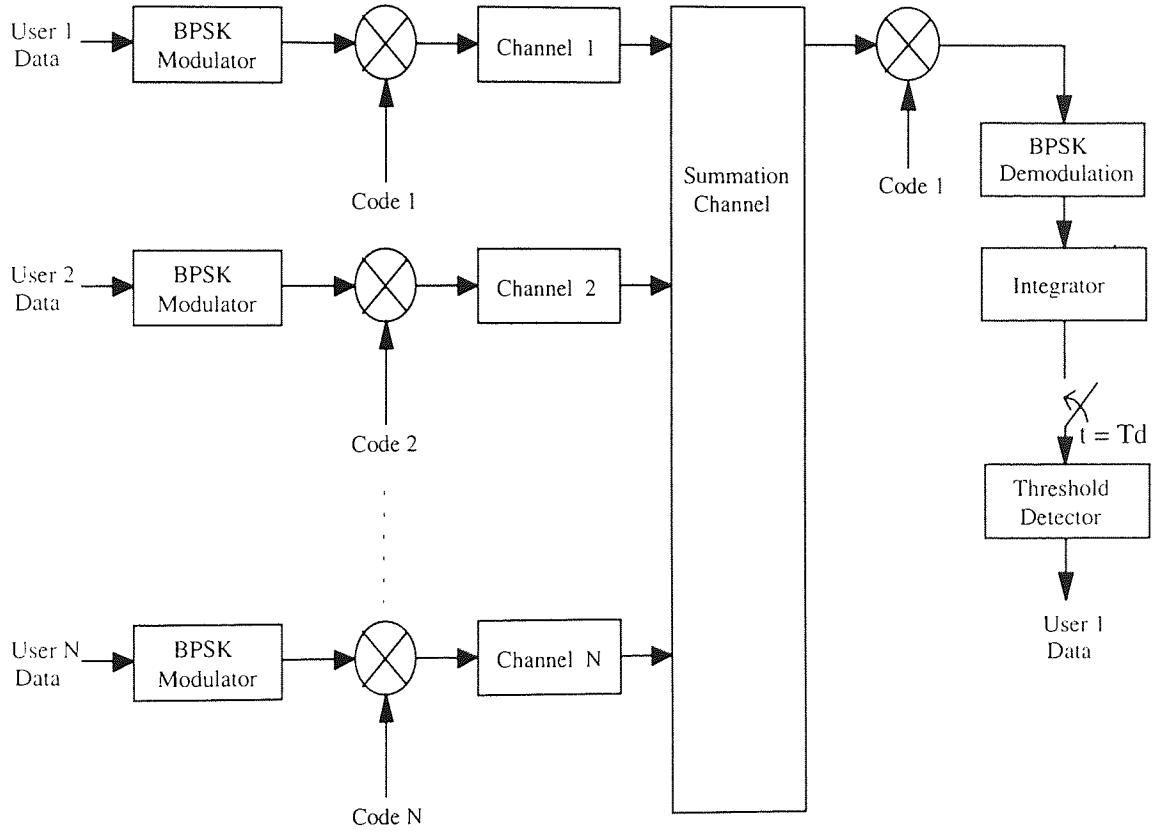


Figure 4.1
Model System Diagram

There are two main problems associated with the simulation of spread spectrum systems that are not normally present in conventional systems. One of the difficulties occurs when the objective of the simulation is to obtain the performance measure obtained from the data signal, such as the probability of error. Perhaps ironically, this difficulty is due to the very property that makes the system work, *i.e.*, the transmission bandwidth is much larger than the data bandwidth. This means that the number of samples per data symbol [Nyquist88] has to be increased by a factor of about the processing gain, which is a tremendous computational burden relative to the standard case. The best that can be done is to sample

this rapidly in the portion of the system where the bandwidth is expanded but only at a rate proportional to the data bandwidth in the portions of the system where it is appropriate.

The second difficulty has to do with simulating the code acquisition [Jovanovic88]. This is a difficulty in the same sense as above, namely, that it can be a computationally time-consuming procedure for two reasons. One is that the acquisition circuitry/logic is fairly complicated, and the other has to do with the nature of the acquisition. There are many variants of the acquisition process, but the essence of any scheme is to correlate the arriving code with the locally generated code. Since the received signal is generally buried in the noise or interference, correlation however, may need to take place over many periods of the code and this represents a significant amount of computation. Since the main purpose of the model is to investigate the different types of variations, the acquisition is assumed perfect within the system.

4.2 The System Model Developed

The fundamental purpose of the model [Huang92] is to determine the BER of a DS-CDMA cellular system under different schemes. The Monte Carlo simulation method [Rubinstein81] was used to calculate the errors within the system. The platform used for the model was the Personal Computer (PC) for the development stages of the code. For the execution of the model, SPARC workstations were used. The main reasons for this is that the development software is easier to use and more widely available for the PC's, while the SPARC offers a greater amount of processing power over longer periods with a true networked multi-tasking environment. The program was developed using C++, which is an object-orientated language that concentrates upon data structures, then the processing capabilities were added to handle those data structures once these structures have been decided. A C++ program consists of a number of objects, which may range from a single constant to an entire process. The definition of an object is a collection of information plus a description which indicates how that information is to be manipulated. Objects are organised into classes, each having its own local storage areas and some means of communication with the other objects. These communications occur by sending and receiving messages, which usually take the form of statements in the executable program. The reason for using C++ is fundamental to the activity and enables a greater ease of construction and understanding of the system which has been implemented. C++ is, as stated previously, an object-orientated language which means that we may define one set of processes within our system (*e.g.* a transmitter) and then say that every user, which itself is a process, has 'access' to these. Each similar process or procedure will be independent of

the others, yet identical in it's execution and handling of the data and other processes. This means that only one set of processes, which are called classes in Object Oriented Programming (OOP), need to be defined and therefore program development and control is made easier.

Each system user has his own set of parameters which are used to determine the transmissions which originate from and reach their transceiver. In C++, accounting for this series of related information may be done in two ways; by classes and structures. Due to the complexity required and the way the mobile units interact, the object oriented class is the best suited for this task of holding the users specific information. All users in the system have to take on board certain parameters to communicate such as frequency, chip and bit rates and sampling rates. These are specified in the program by using a structure as only one set is required per system and each user may access this one structure. By putting this structure into an array, several systems may sit next to or on top of each other to assess their impact upon one another. The latter routines are those which the mobile transceiver would normally carry out and C++ provides the best platform to simulate these multiple routines in an easy and efficient way. The user parameter routines are not all used in any one model and show the variety of routines which have to be evaluated. To show the system in a more schematic way, the diagram shown in figure 4.2 is used. The first user, *user[1]* has both transmit and receive processes while the others only require the transmit set of processes. This is due to the bit error-rate being measured by only one user.

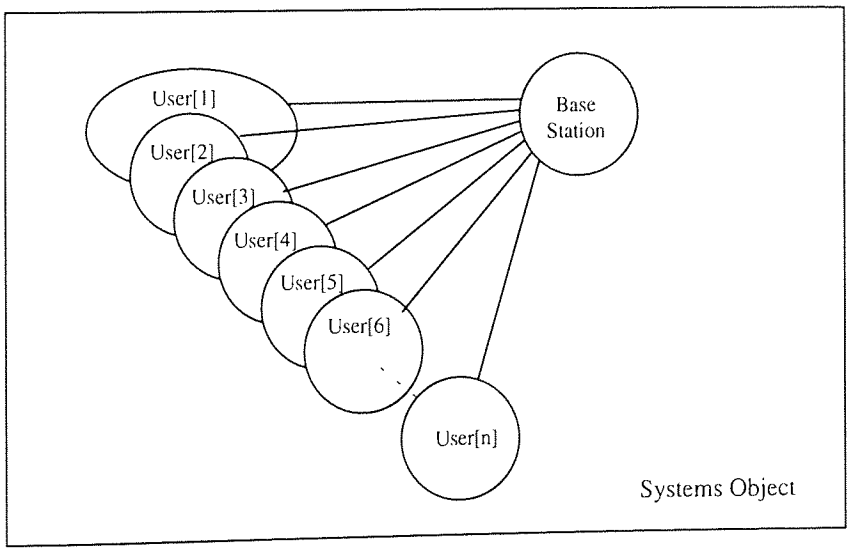


Figure 4.2
Parameters Used in the User Transceiver

These processes are the basic ones required for DS-CDMA communications and are similar to those which would be implemented in a real system. As stated above, the fundamental concept of OOP is that the program is centred around the data structures which are

required to be manipulated.. Before the processes are discussed, therefore, we must first introduce the variables which are to be used in the model. The model uses two structures of variables, one is used for the mobile system as a whole and every transmitting user in the system has to use these parameters and, secondly, every user has their own set of parameters which is individual to themselves. The following table shows the system and user variables which were implemented. There are a large number of them to enable the model to be as accurate as possible.

Object	Variable Name	Type	Description
User	*chip*	Int	Value of Spreading Chip
	data_in	Int	Value of Transmitted Data Bit
	data_out	Int	Value of Received Data Bit
	trans	Double	Transmission
	rec	Double	Reception
	mask	Int[43]	Spreading Code mask
	next_chip	Int(+)	Sample number of next chip generation
	amplitude	Int	Amplitude of user's transmission
	samples_per_v_chip	Int(+)	Number of Samples in chip
	walsh_data_in	Int[6]	Data into Walsh coder
	walsh_code_out	Int[64]	Code out of Walsh Coder
	walsh_code_in	Int[64]	Code into Walsh Decoder
	walsh_data_out	Int[6]	Data out of Walsh Decoder
System	*data_rate_bps*	Int (+)	System Data Rate
	ave_chip_rate	Int (+)	Average Chip Rate
	model_length	Long Int (+)	The Number of Bits which Model Runs
	num_of_users	Int (+)	The Total Number of Users in System
	frequency_hz	Long Int (+)	Centre Frequency of Transmission
	samples_per_second	Long Int (+)	Number of Samples in one Second
	gain	Int (+)	The System Processing Gain
	unwanted_user_amp	Int (+)	Amplitude of Interfering Noise Users
	variation	Float	Chip Variation Factor
	user_v_spacing	Int (+)	Spacing between variation values
	quick_run	Char *	Save Sample Information to Disk

Table 4.1

Object and Structure Model Parameters

Starting from the top of the table, it can be seen that each user will be assigned a *chip*, *data_in* and on the third line a *data_out* value. These are integer values which may have only the values ±1. The system transmits and receives signals and these are represented in the user object as *trans* and *rec* with an assignment of being double precision, which may have up to 15 decimal places. This is done so as to achieve the greatest degree of accuracy within the channel where all the signals are added together to form the noise components which form the received signal. The sixth variable down in the user object is a 42-bit mask, which every user in the system is given, so that the codes which are used in the spreading of the data are individual and thus their correlation is minimised. The *amplitude* parameter is used to ensure all signals are the same amplitude, by varying this the effects of amplitude distortion may be investigated. Four parameters are set aside for the Walsh orthogonal coding which will be covered later.

The system set of parameters are those values which are common to all the users within the system. There may be many different systems within the model, so that the individual user may choose over which system he requires to communicate.

Again, starting from the top, the first variable is *data_rate_bps* which is the data rate in bits per second and is assigned as a positive integer (negative data rates are not allowed). The same can be said for the majority of the variables used within this structure and it is hoped that the names explain their contents. The *model_length* is the number of bits for which the model is to be run and is defined in the program as a long positive integer. This means that the maximum model length is (8 machine bytes long) 4.29496×10^9 transmitted bits. The *num_of_users* variable denotes the number of transmitting users in the system at any one time and again can only be assigned a positive value. The next variable is the *frequency* of the transmitted signal and is in the units of hertz, with a system maximum value the same as that of the model length given above. The model is run using samples and the sampling rate, which is defined under the system set of parameters under *samples_per_second*, may be set by the model and is again a long positive integer number. Once the data and chip rates are set, the processing gain of the system may be calculated using the formulae given in chapter two. This is calculated and held in the system parameters under the variable *gain*, which is a positive integer. The next variable is the amplitude of the unwanted signals in the system, which is denoted by *unwanted_user_amp*. Only one signal is demodulated, with the error ratio and statistics being provided just for that user's transmission and assumed similar for all the other users in the system. The probability of error in long runs, as defined in [Rubinstein81], may be assumed to be the same for all users within the system, but may deviate considerably in smaller sample periods.

The *variation* is the maximum duration compared to the standard duration, which may be varied. The *user_v_spacing* is the amount of spacing given between each user's variable chip duration. These are covered in depth in the next chapter together with their effects on the bit-error ratio. The last variable covered here under the system parameters is the *quick_run* variable. The default value is 'yes', but when 'no' is used the model files all user parameters for every sample to disk.

The complete list of components within these objects contains both variables and procedures which are used in the transceiver of the DS-CDMA signals. There are no procedures contained in the system set of parameters, but all are contained in the user object, which is discussed in the following sections.

Object	Variable Name	Type	Description
User	*qum_gen()*	Int	Maximal length chip generator
	m_gen()	Int	Maximal length data generator
	gold_gen()	Int	Gold chip generator
	sparse1_gen()	Int	Sparse amplitude variation generator
	sparse2_gen()	Int	Sparse amplitude variation generator
	bent_gen()	Int	Bent chip generator
	gen_chip()	Void	Decides when to generate another chip
	gen_bit()	Void	Decides when to generate another bit
	psk_mod()	Double	PSK modulator
	msk_mod()	Double	MSK modulator
	qpsk_mod()	Double	QPSK modulator
	display_user	Void	Display users' object parameters
	triangle_chip_var()	Void	Generate Variable Chip Duration
	cosine_chip_var()	Void	Generate Variable Chip Duration
	channel()	Void	Additive Channel
	receive_sin_psk_sample()	Void	Integrate Sin Channels
	receive_cos_psk_sample()	Void	Integrate Cos Channels
	receive_msk_sample()	Void	Integrate MSK channels
	decide_bit(int)	Void	Decide on Bit Value and Compare Values
	walsh_coding()	Void	Code Walsh
	walsh_decoding()	Void	Decode Walsh

Table 4.2

Object Processes Used in Model

Table 4.2 shows the second and final part of the user object, the first part being given in the upper part of Table 4.1. Here, in the second part of the processes are outlined, with a brief description detailing their use within the model. A more detailed description is provided in the following sections. These are ordered as they would be used within the system. Not all of the processes are used in any one model but the capability to make use of them is available. In Table 4.2 the 'Type' column represents the value type which the process returns. The value given in brackets in the 'Variable Name' column, *e.g. decide_bit(int)* is the value type which must be passed to the process when it is called.

One of the major problems in such a model is the computational length (run time) and, therefore, a critical factor in designing the model is the reduction of the size of the variables used in the simulation. This consists of reviewing the attributes given to all the parameters to ensure that they have the correct value and to take the smallest possible memory space whilst still maintaining their effectiveness and accuracy within the model. A double valued parameter will take considerably longer, with 8 bytes, to move into place for the task of computation than an integer which has only two bytes. Considering that the model will run, to simulate the transceiving of many thousands of samples, a large benefit may be gained by the correct usage of variable types.

4.2.1 System Flow Chart

In this section a description of how the system was simulated is given, based upon the flow chart given in figure 4.3 which illustrates how the processes described in the above sections interact to form the model.

The model is initialised by inputting via the keyboard a set of parameters that are required to run the simulation. A default set is provided, but the program user may change these to suit their own requirements. Once these have been entered and the user is satisfied with their value, the program then goes on to calculate the complete set of required parameters as discussed in section 4.1. At this point, the program checks to ensure that the values are viable and that a reasonable simulation may be carried out. These include ensuring that the sampling rate is greater than twice the frequency of transmission, that there are less than two hundred users (defined by the maximum array size), that the variations (discussed later) are less than one hundred percent and that the processing gain is an integer by ensuring that the data and chip rates are also integers. The program will exit if negative values are given to such values as the sampling rate, frequency, data/chip rates and the number of users.

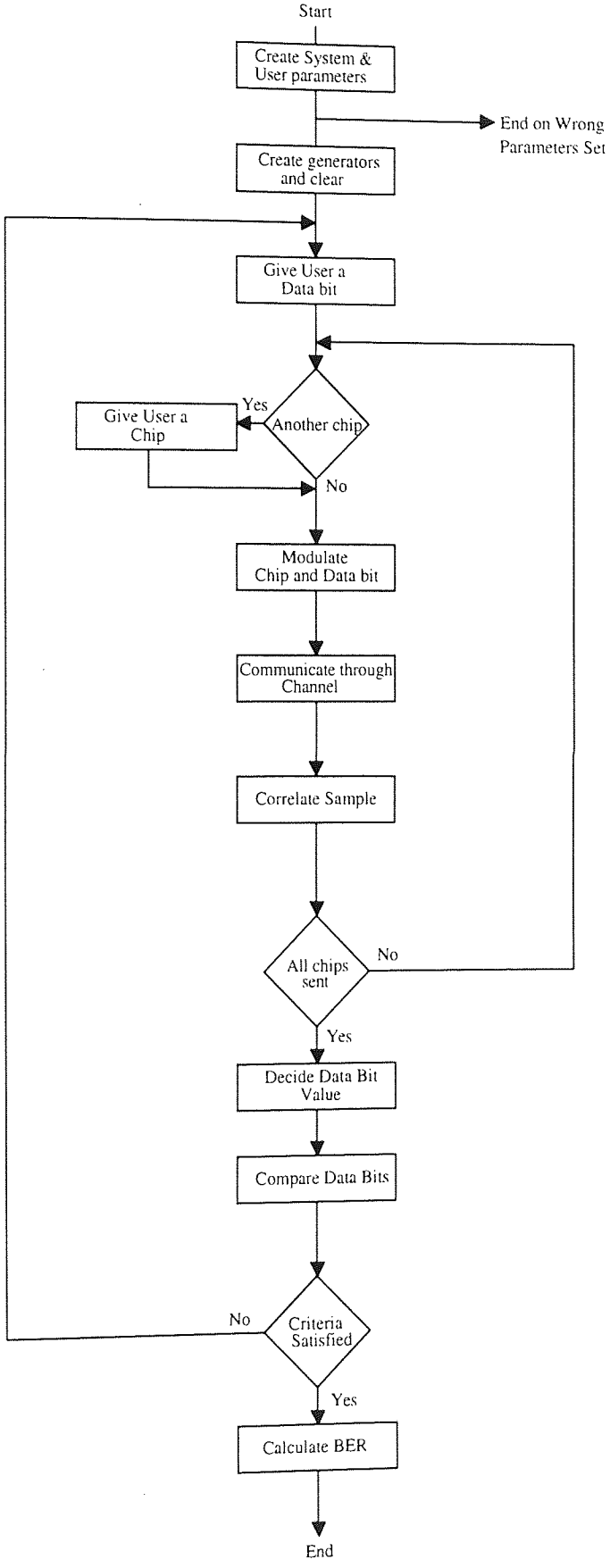


Figure 4.3

Model High Level Flow Chart

The next step is to initialise all the registers in the users' spreading code generators to a logic 1 state and then clock each through approximately 50 bits. The logic 1's ensure the correct modulo addition operation of the generators and the clocking of the bits ensures that all the logic 1's have been flushed through the register array so that the generators will operate in the random part of their sequences and, therefore, the bit error rate calculated will be that associated with the random data.

The program then goes on to provide each user with a data bit which is generated by the maximal length sequence generator. This generator produces a long code which provides data bit values to all the users *data_in* parameter. The model may work in the synchronous or asynchronous data bit mode. In either mode the actual samples are periodic and the progression is around the flow chart (shown in figure 4.3) will be in a synchronous manner, since the period between samples and the sampling rate are common throughout the system. For each transmitting user this is accomplished by calculating the sample at which the next data bit is required to be generated. The same may done for the chip durations.

The next part of the program, after the generation of the data bits for all the users, is the determination of whether the transmitting user requires a new chip value. Upon initial operation of program the first process is the generation of a chip using one of the three chip generators as discussed in section 4.2.2.

Once all the users have been assigned a data bit and a chip value, the model can now proceed to moving samples from the transmitter to the receiver by first modulating the chip and the data values onto the carrier frequency. These transmitted signals are then passed onto the channel, which produces a single user received value as its output. This value is the summation of all the users' transmitted samples in the system and it is this value which forms the received signal sample for each user. This is then correlated with the users locally generated sample value. If, at this sample, the end of the data bit has not been reached, then the program moves on to generate more samples and chips until the last sample required for that particular data bit has been received and correlated.

The outputs from the correlator over one bit period are added and at the end of the data bit a decision is made regarding the value of the bit. If the data bit value which has been generated at the transmitter is the same as that which has been produced after the decision process, the received bit is not in error and thus no error has occurred during transmission.

The transmission and reception of data bits and their comparisons is continued until the total number of data bits considered is equal to the model length set by the user at the beginning of the program. To satisfy the criterion set in [Fishman78, Rubinstein81, Jeruchim92] regarding Monte Carlo simulation the model length can be set for a given number of errors to be received and, therefore, a certain accuracy to be reached within the simulation. At this final point the BER is calculated by the number of bit errors that have occurred divided by the model length in bits. The statistics of the simulation are then output to the screen and to an ASCII file on disk for a hard copy and for further consultation.

Each user will move around the flow chart functions independently of any other users' state or values, but all will move at the same constant rate determined by the sampling rate of the system. The allocation of the data/chip values and all decision making takes place instantaneously within the system. In the simulation, only one signal is received and demodulated to produce the data, which is then compared to produce a BER. This considerably reduces computation in the model, but relies on the error probability being consistent for all users. The probability of error may be assumed to be the same for each user when the model length in bits is long or the number of errors is large and, therefore, the effects of bursty errors has been averaged over a long period.

4.2.2 Code Generators

In the system under consideration the data and spreading codes are required to be pseudo-random in nature and the spreading code to be as in accordance with section 2.3.1. The data must also contain a good distribution of ones and zeros to ensure that the system is not biased towards any one data bit value. The data throughout the model is generated using a long maximal length PN code. Three register lengths were used to verify that the sequence length is not related to the system BER; these were 2^{39}, 2^{15} and 2^{5}. The sequences generated were based on the primitive polynomial $4{,}000{,}000{,}000143_8$, 100003_8 and 45_8 respectively. These are expressed in Peterson's notation [Ziemer92]. It was found that the BER was not affected by the code length of the data generator.

The flow chart given in figure 4.4 illustrates the generation of a single bit in a 2^{15} maximal length code. Subsequent calls continue to generate the code. Similar charts may be constructed for the other codes used. The generator has 15 registers, n being set equal to this value. The next step is to sum (in modulo-2) those register bits which generate the wanted code. Here the registers are 1, 2 and 15. The n^{th} register, number 15, is converted into bi-polar form so that it may be modulated later.

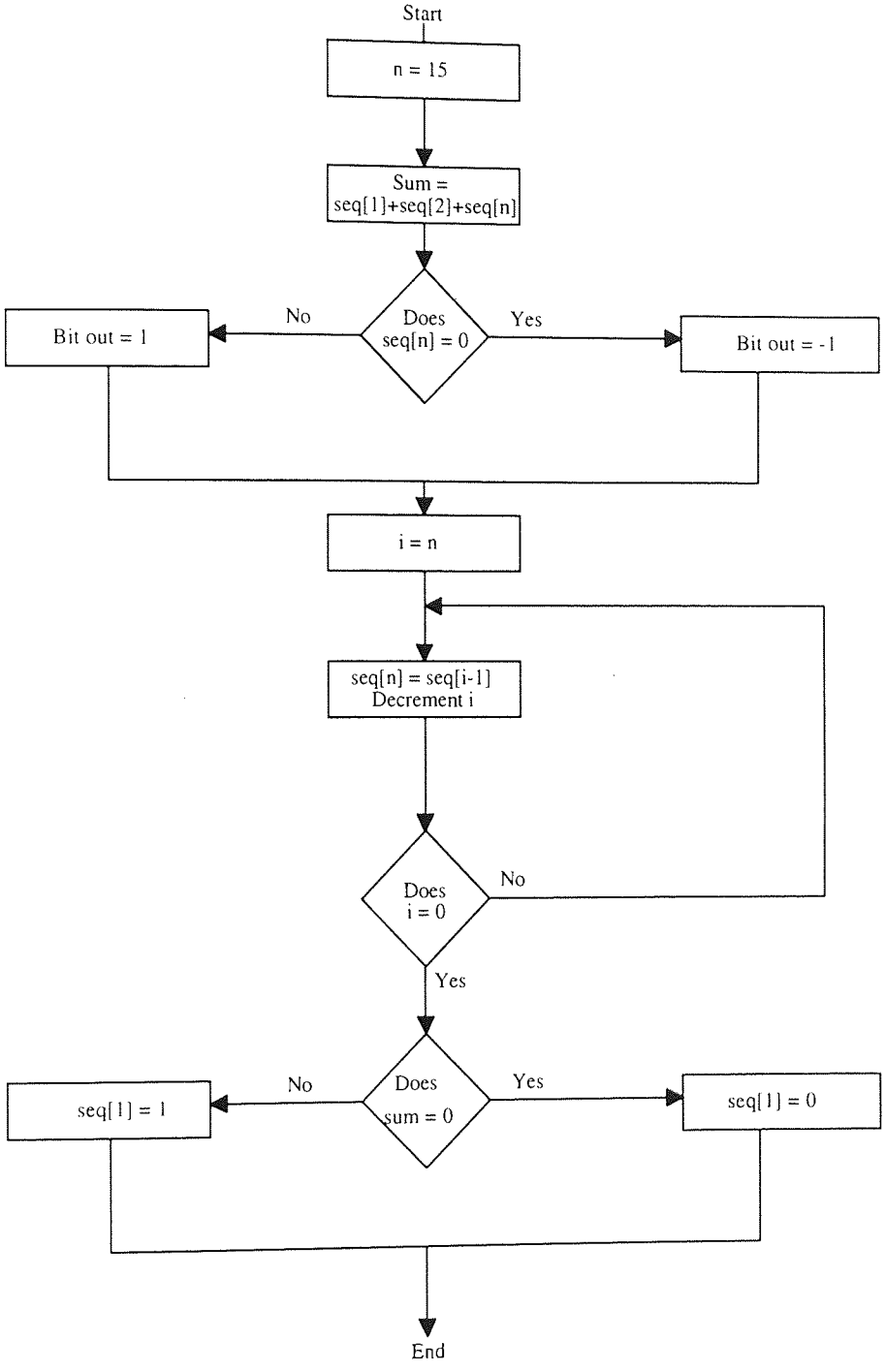


Figure 4.4

Maximal Length Data Generator Flow Chart

Four spreading sequences were used, firstly, a maximal length code of degree 39, secondly, the Qualcomm long spreading code of degree 42, thirdly, a Gold code of degree 13 and, finally, a Bent code of register length 12. To ensure that the end user has an unique code sequence, these codes are made individual by the inclusion of a user mask. This mask is generated by converting the user transmission number to decimal and then converting it

into a binary array. The first sequence generator was used for both random data and for spreading code generation.

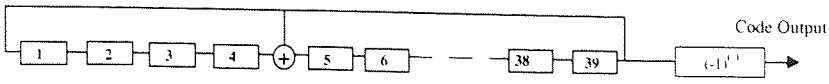


Figure 4.5
Maximal Length Random Data and Spreading Code Generator

Each code generator, as shown in figures 4.5 to 4.8, includes a logic translator before the code output which converts the 0/1 logic. This is used in the conversion of the codes into 1/-1 format, which may be used in the phase modulators. The second PN spreading code generator used was derived from the Qualcomm standard long code generator, which is of degree 42 [Qualcomm92]. This code is added to a user mask which has been derived from the user's transmission number and is used as a mask to make each code individual. This code was implemented since it is used in an existing system and provides a known error response with which to compare results. The code is non-maximal and is not a composite of two maximal sequences *i.e.* a Gold code. The code generator with the polynomial $753{,}000{,}000{,}000{,}001_8$ is shown in figure 4.6.

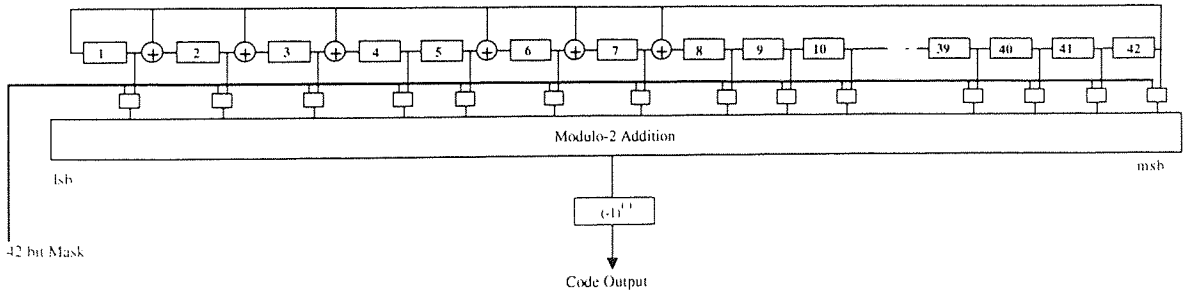


Figure 4.6
Qualcomm's Spreading Code Generator

To achieve correct operation in a code division multiple access system, different codes are required and, to provide this from a single maximal length sequence, the spreading code is determined by masking the output of the generator with a user defined value. This provides each user with different code sequences. In the system under consideration the mask is determined from the user number. This is converted into binary and distributed over the binary word in order to provide a degree of randomness within the codes generated simulating This simulating as near as possible, a real system.

The third spreading sequence used was the Gold code generator which provides a code length of 8191. The sequences generated were based on the primitive polynomials 20033_8

and 34333_8. This, and the maximal length generator, is of the *user parameter* class since each mobile station has an individual generator in order that the codes may be random and independent.

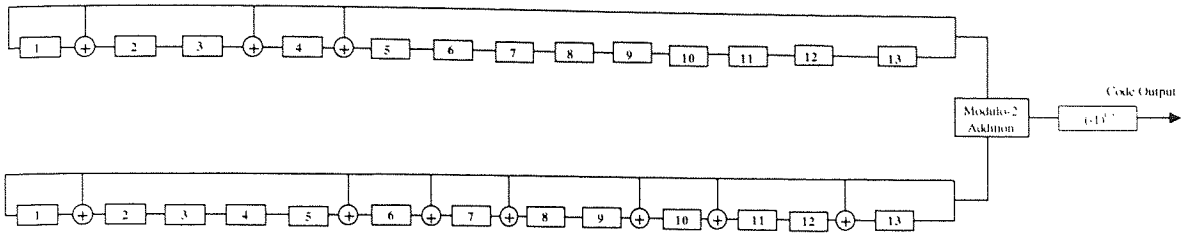


Figure 4.7
Gold Spreading Code Generator

The starting values of the generators, reading least significant bit first, are 0110110110111 for the top register bank and 1111111111110 for the lower bank of registers, as shown in figure 4.7 above. These generators provide a maximum correlation value of 129 and a class size of 8193, which is derived using the theory summarised in section 2.3.3.3.

The fourth spreading code used were Bent sequences. The generator used is shown in figure 4.8 and is based on the same code which is specified in [Olsen82]. The sequences are generated using the primitive polynomial 10123_8 [Ziemer92].

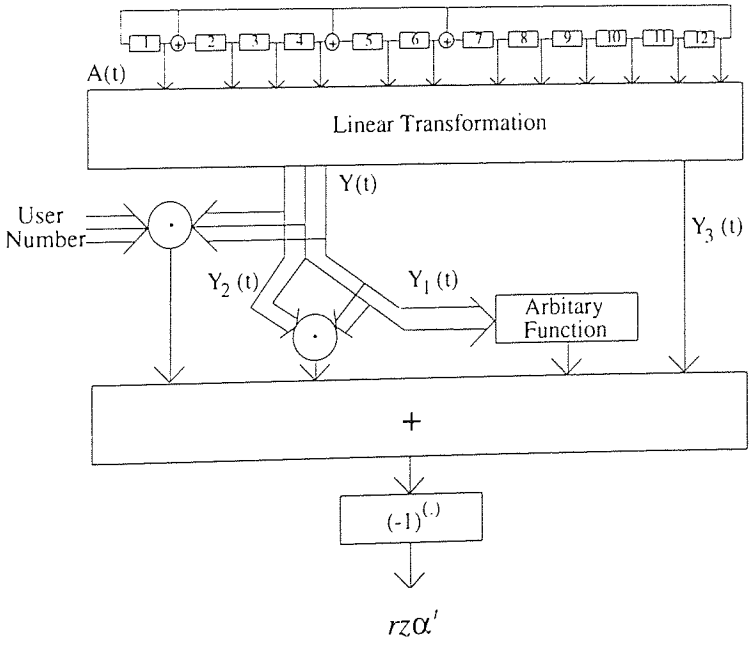


Figure 4.8
Bent Sequence Spreading Code Generator

The *Y* parameters given in figure 4.8 can be defined as;

$$Y(t) = \left[\frac{Y_1(t)}{Y_2(t)}\right] = LQ^T A(t) \qquad 4.1$$

and

$$Y_3(t) = (Q^T C)^T A(t) \qquad 4.2$$

For the bent generator shown above, the linear maximal length generator output sequences are bent using the following co-ordinate transformation matrix in a manner which is defined in equations 4.1 and 4.2.

$$L = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \qquad 4.3$$

$$Q^T = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \qquad 4.4$$

and

$$(Q^T C)^T = [1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1] \qquad 4.5$$

This generator produces a code length of 4095, with a maximum correlation value of 65. The number of sequences in the class equal to 64.

4.2.3 The Transmitter

The transmitted signal in a DS-CDMA system is determined by the modulation technique used in conjunction with the data and chip values. A number of modulation systems [Ziemer92] were used and these included PSK, QPSK and MSK [Pasupathy79]. The transmitting algorithms available in the modulator set of programs are *psk_mod()*, *qpsk_mod()* and *msk_mod()*, respectively.

The simulation procedures are as expected in many respects, except that time is in integer sampled value which is divided by the chip rate, in chips per second, and by the number of samples per chip interval. This ensures that the system has the correct number of samples per chip/frequency cycle interval. The phase-shift-keying function returns the product of the chip and data bits, together with the carrier frequency and sampling rate components.

The PSK algorithm takes only one data bit at a time while the QPSK and MSK modulator take two. This is achieved by the procedure by having two paths through the modulator; this is similar to that used in a hardware implementation. The MSK algorithms require that the phase shifts between the chip and the data bits is continuous, to reduce the required bandwidth used during the transitions. The *msk_mod()* procedure implements the equation given in [Ziemer92].

4.2.4 The Channel

The additive channel sums all the user transmissions and this value is received at the base station. All the transmissions have the same power/amplitude value, therefore the system is said to have ideal power control. All user's transmissions are added and the sum is the received signal. This channel does not add Gaussian noise or any delays into the signals. This channel is, therefore, the ideal condition and the process is named *channel()* in the simulation program.

4.2.5 The Receiver

The receiver is far more complicated when compared with the transmitter. To reduce the complexity, the vast processes which simulate the circuitry which is required for acquisition and synchronisation of the spreading codes has been eliminated by ensuring that both the

transmitter and receiver spreading codes are the same *i.e.* they are synchronised. The effects of synchronisation are not part of this research.

The process of demodulation produces the recovered data $d'(t)$ and when implemented by a correlator it can be expressed as follows:-

$$d'(t) = \int_{t=0}^{T_b} r(t)\cos(\omega t)\,dt \qquad 4.6$$

This is shown schematically for BPSK in figure 4.9.

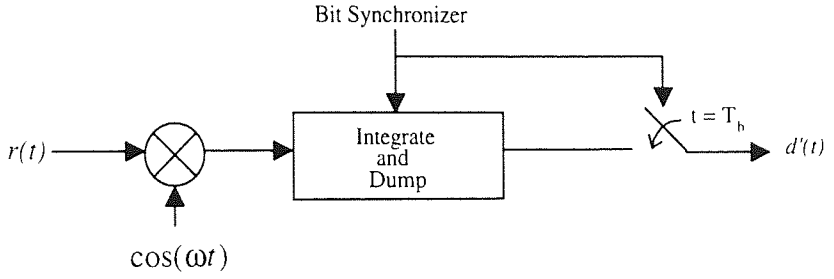


Figure 4.9
The Demodulation Process

In the digital receiver implemented, this process is accomplished in two parts; the first is the correlation of the two signals $r(t)$ and $\cos(\omega t)$, plus the addition of the samples which occur at the sampling rate. The second, which only occurs at the end of the data bit, is the decision which determines the data bit value.

The locally generated signal $\cos(\omega t)$, used above in figure 4.9, is a simplified version of the transmitted signal $s(t)$ and is produced in a similar manner to that of the transmitted signal. The receiver must correlate the wanted spread signal, which is produced locally, with that which is the received carrier of the information. The received signal may be expressed as;

$$r(t) = n(t) + \sum_{k=1}^{N} A^k c^k(t-\tau^k) d^k(t-\tau^k)\cos\left[\omega_0 t + \left(\alpha^k - \omega_0 \tau^k\right)\right] \qquad 4.7$$

The algorithms in the data recovery set of programs consist of a coherent demodulator and an integrate and dump detector as shown above. These are split into their relevant procedures, as some need to occur at the sampling rate and others at the chip and bit rates. *psk_mod*, *qpsk_mod*, *msk_mod*, *demod_sin_psk_sample* and *demod_cos_psk_sample* all process the signals at the sampling rate while *decide_bit* is used at the bit rate.

4.2.6 Orthogonal Coding

The orthogonal coding is achieved by transmitting one of the 64 Walsh symbols for a given 6 data bits. The modulation symbols are selected according to the order specified in [Qualcom92] as shown below:-

$$\text{Symbol Number} = c_0 + 2c_1 + 4c_2 + 8c_3 + 16c_4 + 32c_5 \qquad 4.8$$

where c_5 represents the last or most recent bit. The $\{64 \times 64\}$ Walsh function array is first constructed so that it may be used as a look-up table when required. The table is built up from the basic Walsh function shown in equation 4.9.

$$H_{2N} = \begin{bmatrix} H_n & H_n \\ H_n & \overline{H}_n \end{bmatrix} \quad , N \text{ is a power of 2} \qquad 4.9$$

This is then repeated to generate the required sized table. The following flow chart, figure 4.10, shows the process of constructing the $\{64 \times 64\}$ Walsh function table from an array of size 32 binary bits square.

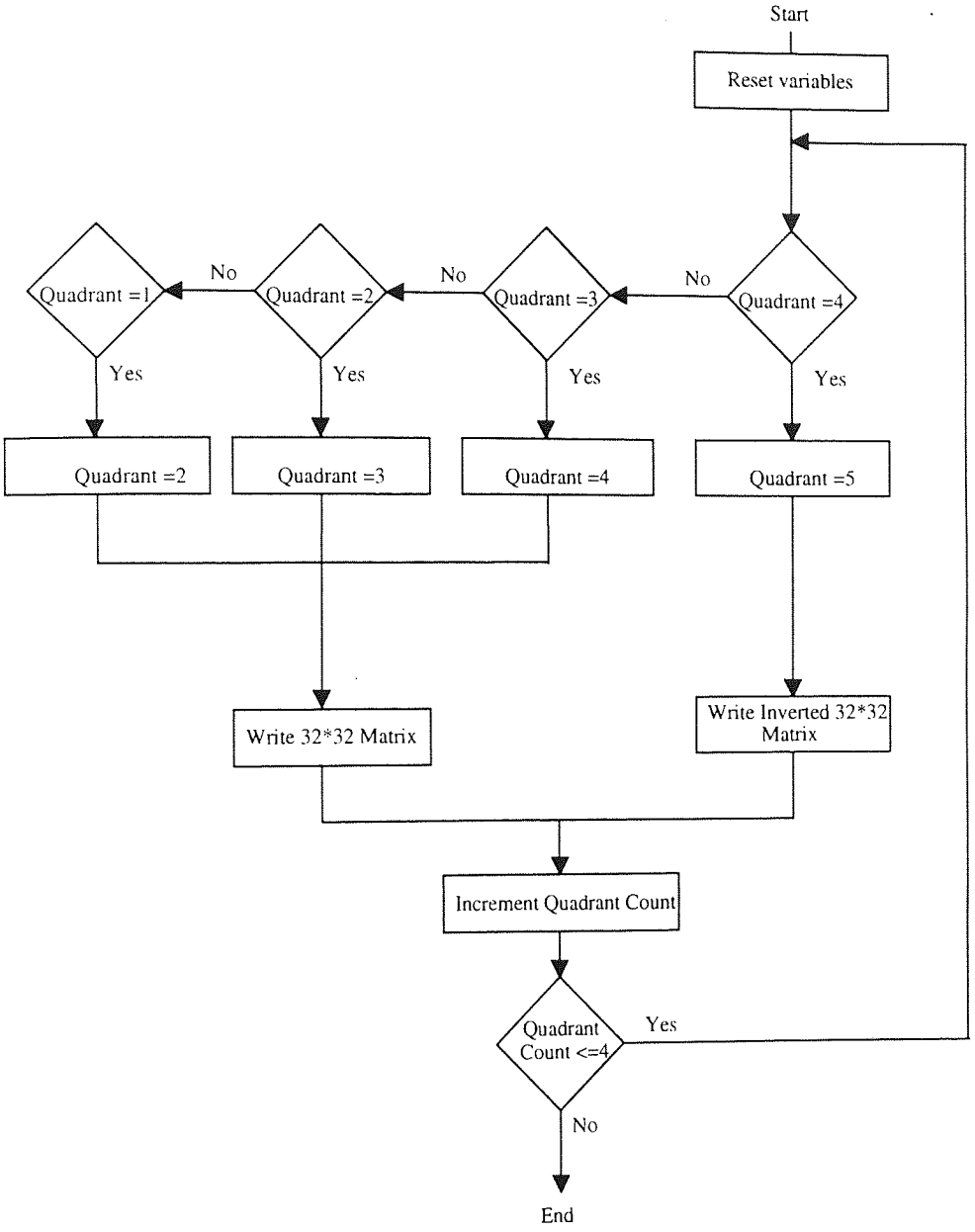


Figure 4.10

Walsh Function Table Generator Flow Chart

When six data bits have been concatenated (*walsh_data_in[6]*), they are then coded into a 64-bit Walsh symbol (*walsh_code_out[64]*) which may now be passed to the transmitting processes. When Walsh encoding is used, therefore, the system may be viewed as one which passes six data bit blocks over the communication channel. The process of Walsh encoding is one of a lookup table, whereas the process of decoding is the determination of the nearest symbol in the table to that which has been received. The following flow chart (figure 4.11) shows the process of deciding which symbol in the table is the nearest to that of the received symbol.

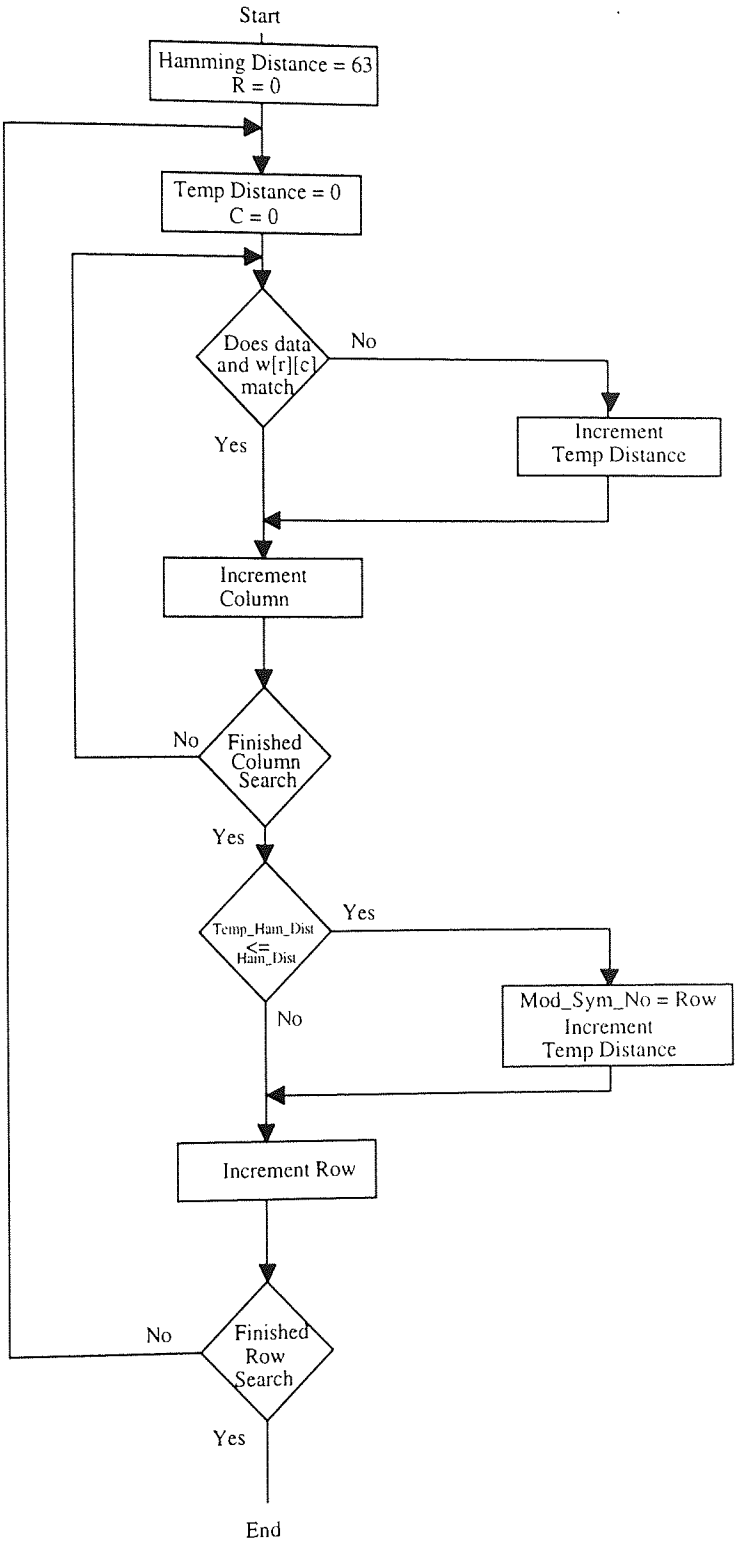


Figure 4.11
Walsh De-coding Flow Chart

The output of the above process is a modulation symbol number which may then be directly referenced to six data bits (*walsh_data_out[6]*) by using equation 4.8. The use of this Walsh code ensures that contentions due the coding never occurs at the receiver and that the Hamming distance between any two Walsh symbols is the same.

4.2.7 Gaussian Noise Generator

In a typical communication system the probability of error is determined by the bit energy per noise spectral density [Ziemer92] and is assumed to be Guassian in it's statistical distribution. In a spread spectrum system the majority of the noise is generated by other users transmissions and thus the noise can no longer be assumed to be Gaussian in nature. To provide a direct comparison to other communication systems, the inclusion of Gaussian noise into the interference path is required. This is achieved within the model by adding the transmitted signal to the output of a Gaussian noise generator to form the received signal. The amplitude of the Gaussian noise generator may be varied to provide a variation in the probability of error.

The Gaussian or normal distribution is defined as;

$$P(y)dy = \frac{1}{\sqrt{2\pi}} e^{\frac{-y^2}{2}} dy \qquad 4.13$$

The method chosen to implement this noise generator is the Box-Muller method and the algorithm is outlined in the reference [Numerical-C].

Chapter 5

Variable Chip Duration Spread Spectrum Modulation

5.0 Introduction

The development of an approach to increase the capacity of a DS-CDMA system by reducing the partial correlation has been discussed in Chapter One. Chapter Four developed a model which is used in this chapter to produce results from which a conclusion may then be drawn as to the viability of varying the chip duration.

This chapter starts by constructing a variation function that will be used to generate the varying chip duration in the DS-CDMA model. It then goes on to discuss the length of run for the model which is required to achieve a good degree of accuracy. The next section investigates the use of variations in the chip duration and compares the results from these variations with the constant chip duration results.

The variation scheme is compared in four ways to the system that does not vary the chip duration. Firstly, by using three modulation schemes, secondly, four different spreading

codes, thirdly, orthogonal Walsh coding and, finally, for a system with Gaussian noise introduced.

The results generated are presented graphically to aid analysis. Analysis of the simulated systems is carried out to determine the bandwidth efficiency, the signal spectral components, the error statistics and the probability of error. All the results presented graphically in this chapter are also given in table form in Appendix C.

5.1 Adding Variation in the Chip Duration to the Model

In this section the chip duration variation generators are described and the process by which they are used within the model is outlined. The approach taken in varying the epoch of the signal is important as the gain induced by this process should not detract from the overall system processing gain. Therefore, the relationship between the data and chip rates, in conjunction with system bandwidth, should remain unchanged.

In this scheme of varying the chip duration it is intended to change the point of the chip transition around its nominal value. The rate of change around this nominal value will be controlled by the variation function, whose duty cycle will equal one data bit. This provides a constant number of chips per data bit and, therefore, all data bits will have the same chip duration variation pattern. The error rate, therefore, will continue to be determined by the users' interference contributions and the cross-correlation properties of the codes.

The consequence of the variation function duty cycle not being equal to the bit duration is that a different number of chips per bit would result. The transitions of the data and the last chip (within the bit) would now not be synchronised. If this occurred there would be another chip added if the data changed phase. Both these effects will alter the processing gain of the system and other system parameters, such as bandwidth and the signal-to-noise ratios.

In varying the epoch of the chip duration, the signals' spectral shaping will alter, the most distinctive change being a proportional increase in the bandwidth utilised by the system. This increase or decrease in bandwidth will be proportional to the duration of the chip, whose average will remain constant. There are many functions which may be used to vary the chip duration periodically. Figure 5.1 shows three of these which have been considered.

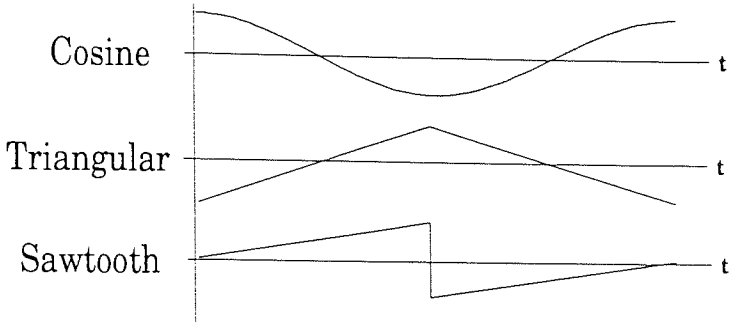


Figure 5.1
Variation Functions

Two variation function generators were created, one generates a cosine and another produces a triangular waveform as shown in figure 5.1. These where named *cosine_chip_var()* and *triangle_chip_var()* respectively and are part of the user object which is outlined in table 4.2. Figure 5.2 illustrates how the signal chip duration in this system is to be varied around an average that is equal to the number of chips per data bit using a cosine waveform. Each transmitting user is spaced over the variation period so that each user, at any one time, has a different chip duration to any other. This is shown in figure 5.2 with three users having different phases, and thus chip durations, at any one point in time.

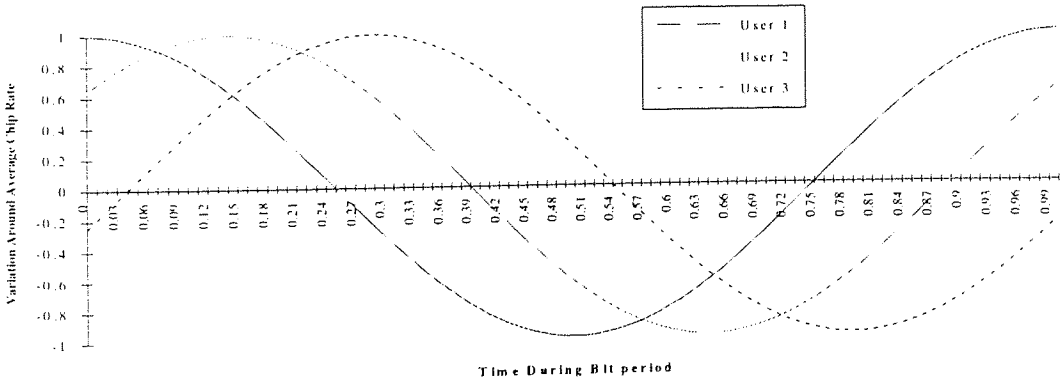


Figure 5.2
User Chip Duration Variations

The chip duration for user 1 above is translated into the chip stream shown in figure 5.3. This is shown using a PN alternating code and the variation is large for ease of demonstration.

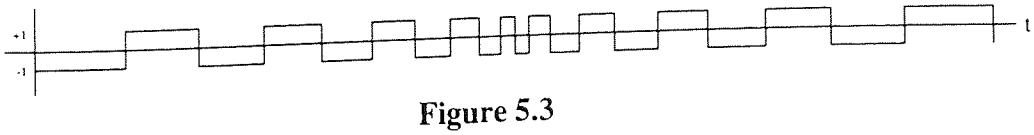


Figure 5.3
User 1 Chip Variations

The individual chip duration is derived from the average chip rate R_c and the variation V, whose units are in fractions of a chip. Figure 5.4 shows the implementation in schematic form of the transmitting section for this system. The clock for the PN generator is driven by a cosine generator that varies the Voltage Controlled Oscillator (VCO) frequency around a nominal average chip rate. The only difference from the standard or constant duration design is the inclusion of this cosine generator before the VCO.

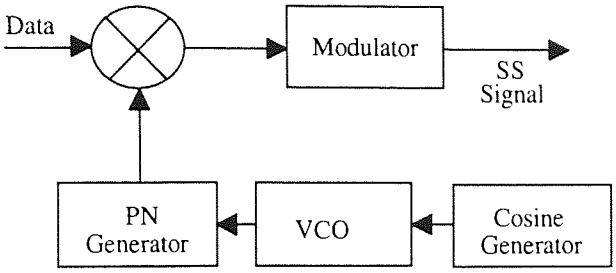


Figure 5.4

Variation Block Diagram for the Transmitter

The varying duration of the chip $T_c^{'}$ in the cosine variation function is given by the following equation;

$$T_c^{'} = T_c + V.\cos\left[\left(\frac{2\pi t}{T_b}\right) + N_u + S_u\right] \qquad 5.1$$

where N_u is the user number, S_u is the spacing between users (denoted *user_v_spacing* in table 4.1) within the variation function and T_c is nominal chip duration. These values enable the users transmissions to be individual in the duration of the chip and therefore each chip duration within the system is set to the selected value for that user and no other.

The program presented in *cosine_chip_var()* is slightly different to the equation given above as it operates in samples and not continuous time. This means that t is replaced with *chip_num* as the system must determine a different duration value for each chip. The required chip number is therefore calculated by dividing the sample number by the number of samples per chip. The output of the equation is in multiples of the carrier frequency cycle period. This ensures that the changes of the chip and data values occur at the point of carrier frequency intersection on the amplitude line, reducing spurious frequency emissions and ensuring no overall d.c. components.

5.2 Model Parameters Used

For a meaningful set of results, the correct selection of the user and system parameters are required, as described in detail in chapter four. The process of measuring the error-rate is a slow one, as explained in Rubinstein81, and it is only a statistical average over many transmitted bits which is calculated. The error rate can be seen to converge to it's average over many errors. Typical patterns which the model has produced are shown in figure 5.5.

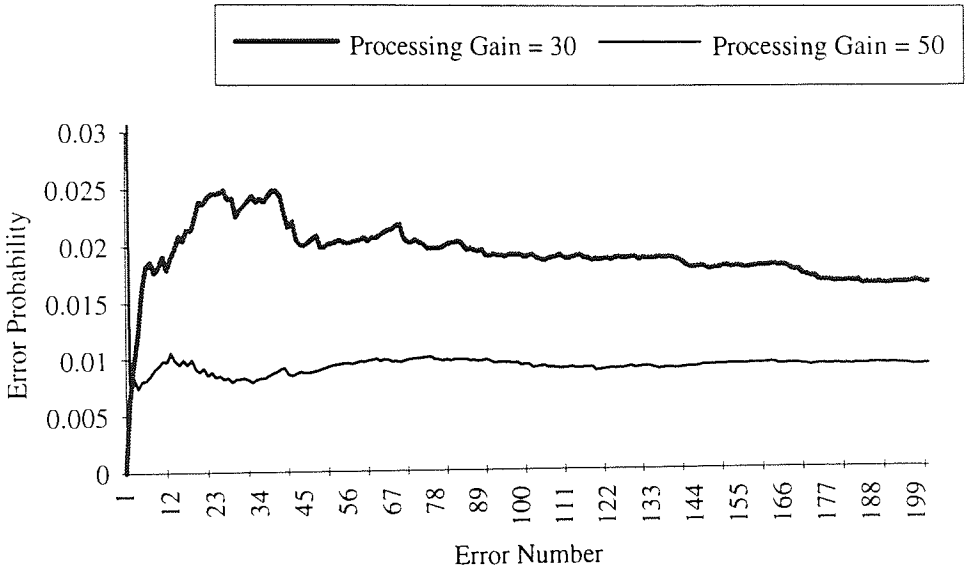


Figure 5.5
Error Rate against the Number of Erroneous Bits

Figure 5.5 shows how the error-rate moves to its statistical average over many error bits, and the two lines can be seen to fit into the tolerance lines shown [Rubinstein81]. The two plots here represent a processing gain of 30 and 50, producing two separate error-rates that both tend to stabilise after around 90 errors. This indicates that, to achieve good results which are statistically reliable, the model would be required to produce at least 90 errors.

Taking these plots as typical of the output of the DS-CDMA simulation, and using the equation provided in section 2.4, a model length may be decided upon using these statistics. Thus it was decided to measure the error-rate at the point of 101 errors. This leads on to a set of parameters which will be used throughout the simulations. This allows a direct comparison of error-rates to be made easily over all the models and modulation schemes considered.

Parameters	Value
data_rate_bps	1bps
ave_chip_rate	10 to 100cps
model_length	101 errors
num_of_users	10
frequency_hz	1000Hz
samples_per_second	16000sps
gain	ave_chip_rate
unwanted_user_amp	1
variation	0.2 of a chip
user_v_spacing	1/num_of_users

Table 5.1
Model Parameters Used

Table 5.1 summarises the parameters which were used for the majority of the simulations; where other parameters were used, this is stated in the results.

5.3 Model Results

The following sections present a series of results based on the simulation runs for the different variation functions, various types of modulation, spreading sequences, orthogonal coding and gaussian noise. Analysis of the results is given in a subsequent section.

5.3.1 The Results for Different Variation Functions

The aim of varying the chip duration is to decrease the interference generated from users' transmissions. The amount of variation induced into the signal is important in two respects.

- When the chip rate is increased, bandwidth expansion occurs which detracts from the gain achieved in altering the chip rate.

- The return given in terms of a reduced bit error rate is not constant and does peak and trough for a cosine variation waveform as shown in figure 5.6.

Triangular waveforms were tried and gave only a slight reduction in the error probability from 0.07 to 0.06 with a processing gain of 20. The reduction for the cosine case was much larger and thus was used in all further models.

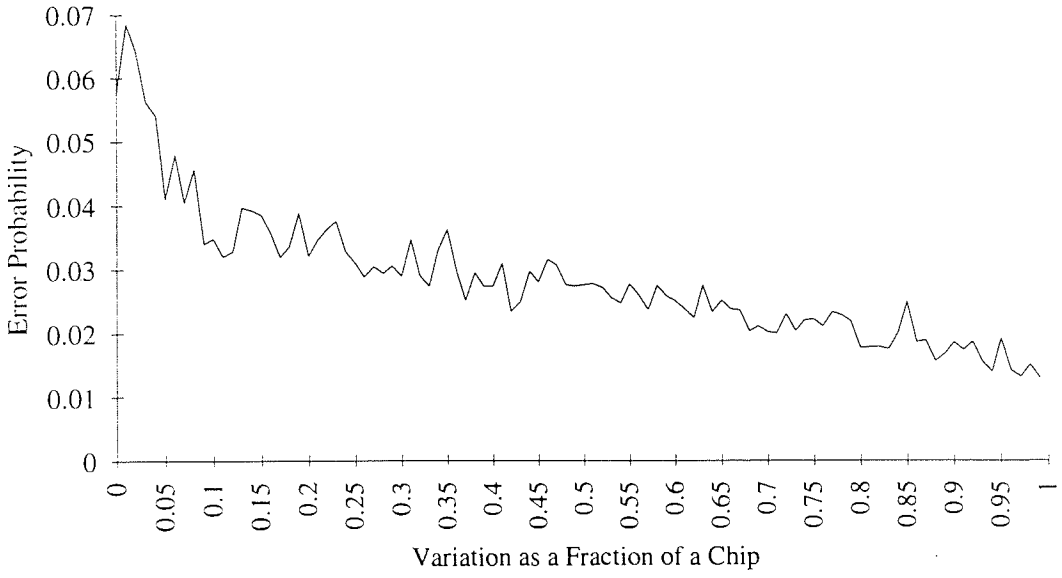


Figure 5.6
Probability of Error as a Function of the Variation

Figure 5.6 above shows the effects on the error probability for variations between 0 and 0.99 of a chip period in increments of 0.01. The numerical values are provided in table C.1. These results are taken over 101 errors and the resulting line can be split into two areas. The first is a high probability of error near the lower valued (V<0.1) variations. In this portion of the line the variations are very small and do not provide a significant difference in the transmitted signals. The graph shows there is a point where the variation becomes large enough to provide a gain and in this case is around a variation of 0.1. In the second part of this graph the output varies around a trend line that shows a decreasing error probability. This second portion is related to the bandwidth expansion introduced by larger (though still extremely small when compared to the chip duration) chip variants. Taking the gradient and y axis intercept of this second portion, the resulting equation is;

$$P_e = -0.025\,V + 0.04 \qquad 0.01 < V \leq 0.99 \qquad 5.2$$

From these results, a decision was made to use a variation of 0.2 in all the simulations that follow.

5.3.2 The Results for Different Spreading Codes

In this section, the spreading code used in constructing the spread spectrum modulated signal is changed and the effect upon the error probability is recorded. The four codes that were outlined in chapter 4, namely Maximal, Gold, Bent and the long spreading code which was developed by Qualcomm, are used. The probability of error results in this section were gathered using 10 users continuously transmitting into a central base station whilst the system processing gain increased. The error probability for each spreading sequence is compared to the theoretical case, which is given by equation 4.3, with the actual values that are provided in tables C.2 and C.3. The factor ψ is set to 1 for the theoretical plots shown.

The first graph, figure 5.7, shows the results of using the maximal length sequences. Only one code generator is used and the users in the system are sequentially provided with a chip from this sequence.

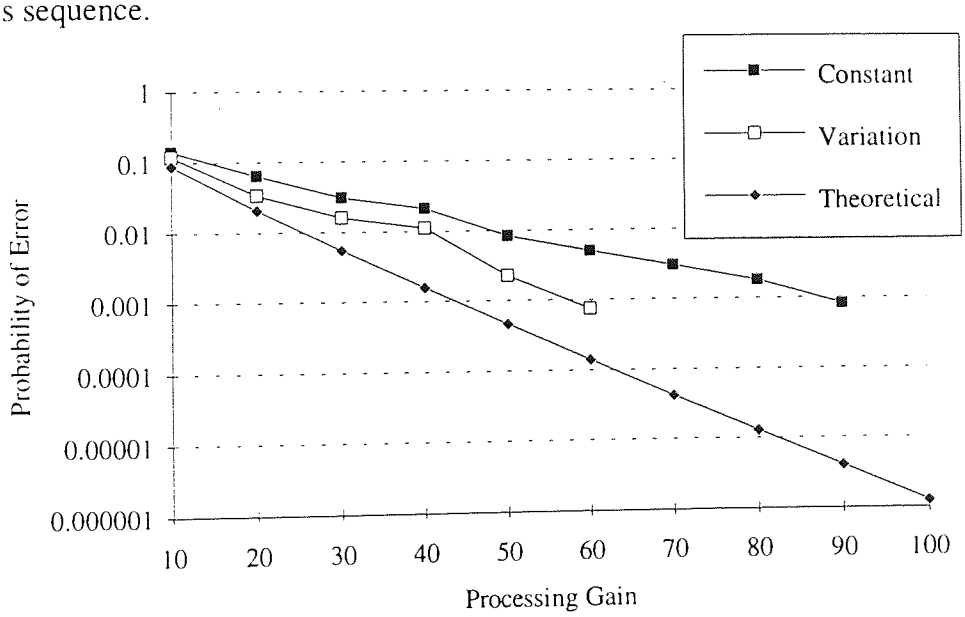


Figure 5.7
Error Probability against Processing Gain
for Maximal Sequences

There is a difference in the error probabilities of the theoretical case and those modelled. The theoretical assumes that there is no correlation between codes and is therefore an idealised case. Factors introduced by the model itself may also account for an increase in the error probability.

The second code used in the simulation for generating the spreading sequences was the Gold code.

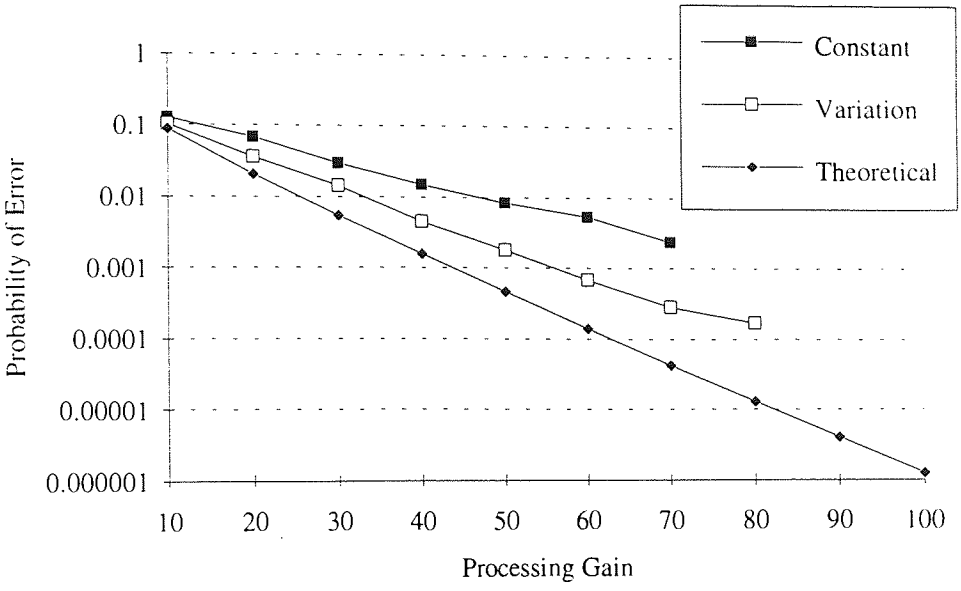


Figure 5.8
Error Probability against Processing Gain
for Gold Codes

The third code used for generating the spreading sequences was the Bent sequences as outlined in section 4.4.2.

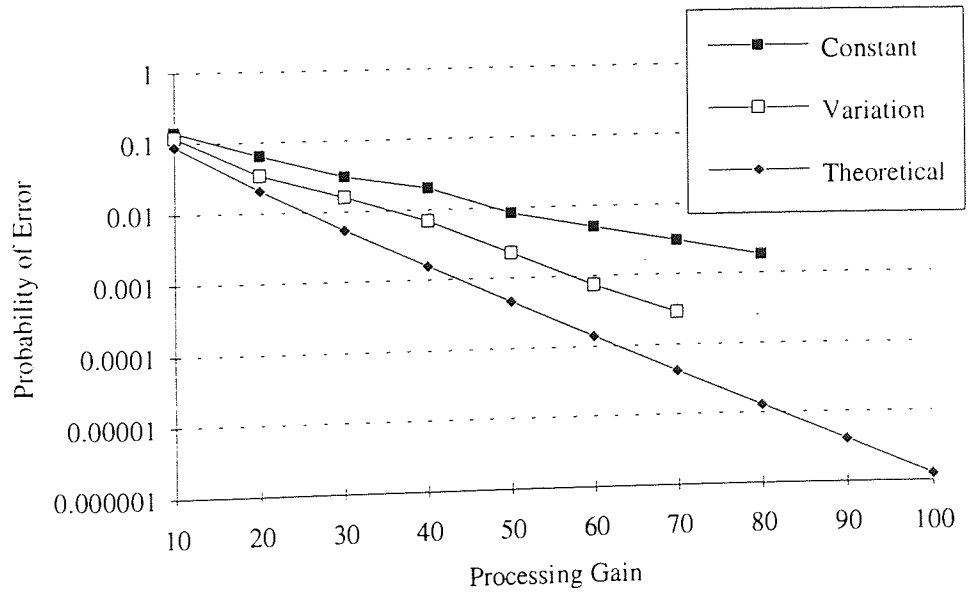


Figure 5.9
Error Probability against Processing Gain
for Bent Sequences

The fourth code spreading sequence is the Qualcomm long code sequence, whose error probability is shown in figure 5.10.

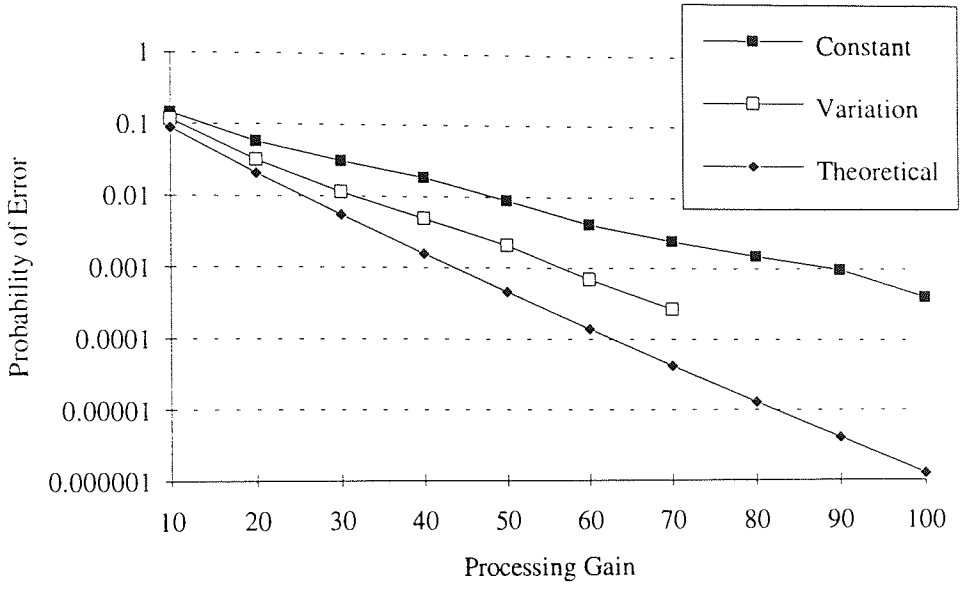


Figure 5.10
Error Probability against Processing Gain for the Qualcomm Long Spreading Codes

The results presented here are further supplemented by taking the partial cross-correlation of the codes in 100 bit blocks. This technique requires the code to be split into 100 chip blocks and the cross-correlation with another 100 chip block from a sequence in the same family to be measured. This is done so that a direct comparison may be made of differing length codes from various families of sequences for there suitability in a DS-CDMA system. The results obtained are expressed as percentages of the maximum correlation measured. The highest was the maximal length with 30%, then the bent with 27% followed by the Qualcomm and Gold with 20%. This reflects the error probability results obtained throughout the simulations. The bent sequence is derived for a reduced cross-correlation over the entire period and is thus not suited to partial correlation, which raises the question as to its suitability in asynchronous DS-CDMA systems.

It can be seen from the four graphs shown in figures 5.7 to 5.10, that introducing variations into these modulation schemes provides an improvement in performance in the form of a reduced error-probability.

5.3.3 The Results for Different Modulation Schemes

Three modulation schemes were included in the model, Binary Phase Shift Keying (BPSK), Quadrature Phase Shift Keying (QPSK) and Minimum Shift Keying (MSK). The probability of error results in this section were gathered using 10 users continuously transmitting into a central base station whilst the system processing gain increased. The Qualcomm spreading code is used in this simulation. The theoretical probability of error has been derived for all these modulation schemes in Ziemer92 and all the results obtained from the model will be compared to the theoretically calculated values.

The first modulation scheme implemented was Binary Phase Shift Keying (BPSK) and the error probability results are shown below in figure 5.11.

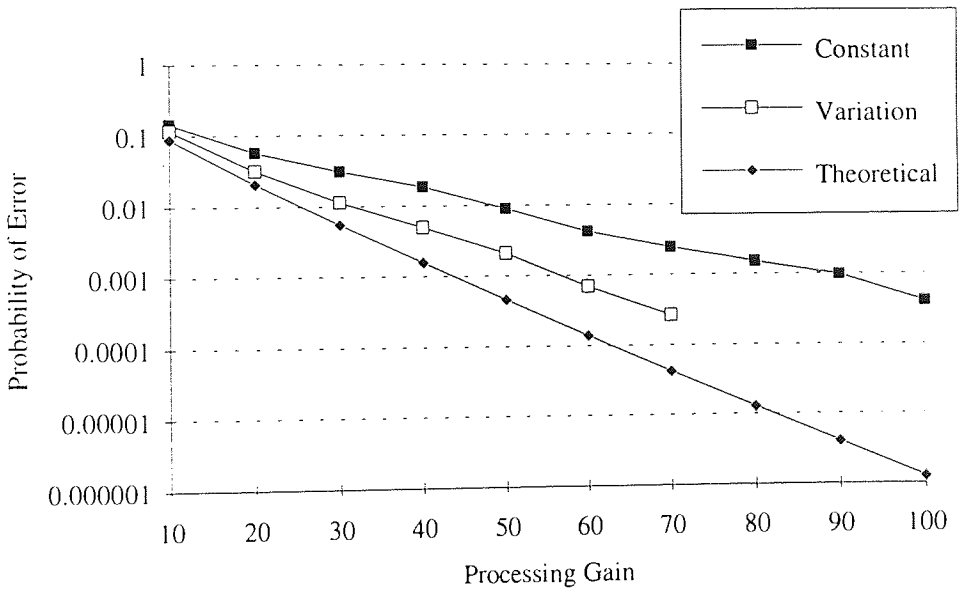


Figure 5.11
Error Probability against Processing Gain
for Binary Phase Shift Keying

The second modulation scheme implemented was Quadrature Phase Shift Keying and the error probability which results from its use is shown in figure 5.12.

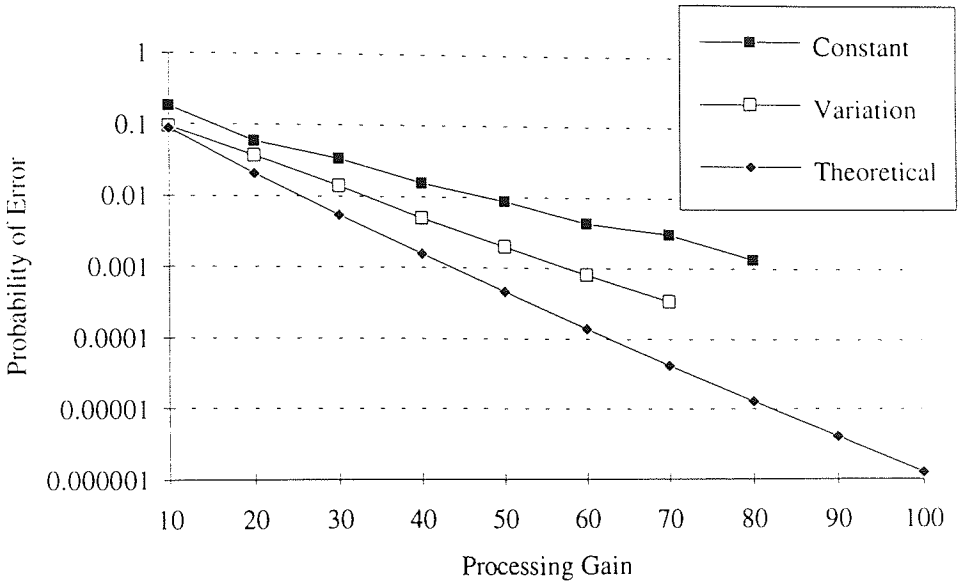


Figure 5.12
Error Probability against Processing Gain
for Quadrature Phase Shift Keying

The third and final modulation scheme implemented was Minimum Phase Shift Keying and the error probability which results from its use is shown below in figure 5.13.

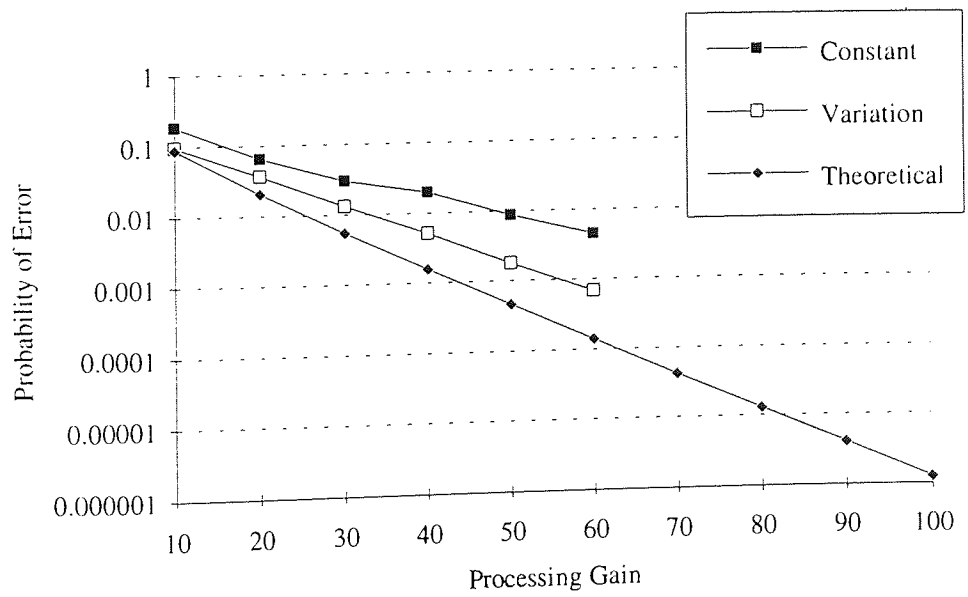


Figure 5.13
Error Probability against Processing Gain
for Minimum Shift Keying

The graphs which have been presented in this chapter plot the error probability against the processing gain. These results require a cross reference with the number of users that the introduction of variations to the chip may permit.

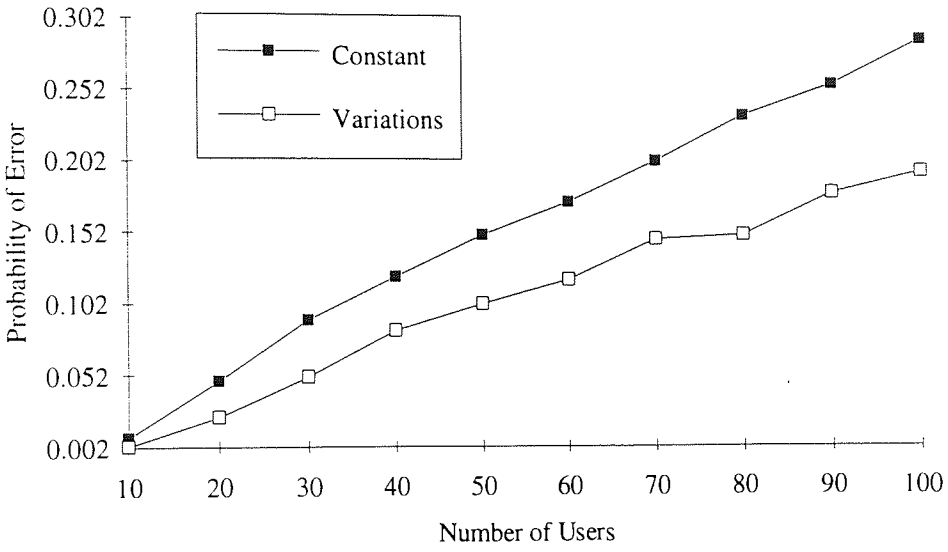


Figure 5.14
Error Probability against Number of Users

The graph shown in figure 5.14 is the results from a simulation with a constant processing gain of 50. For an error probability of 0.15, it can be seen that with constant duration 50 users may communicate, whereas with variations 80 users are accommodated, providing 60% more users in the system.

5.3.4 Results of the Orthogonal Coding Model

When Orthogonal coding is used, the bit error rate within the system should be equal to, or very near to zero depending upon whether the system is communicating in a synchronous or asynchronous mode. One feature of a cellular CDMA system is that the bits and chips are not synchronous and the chips or bits do overlap and cause interference to each other during the required bit period.

The factor under consideration here is, therefore, whether introducing variations to the chip rate within an asynchronous system will increase the probability of error. It is thought that introducing variations will not do so as the orthogonal coding does still have a small error rate which is due to the various chips overlapping. The decision on the bit value is made at the end of the period and, at this point, an average is made for the cross-correlations of the

codes being used. Varying the chip duration, therefore, in an asynchronous orthogonal system should not increase the bit error probability.

The results from the system implemented show that no errors where recorded after a significant run. These were measured in a synchronous system with a processing gain of 64. For a system with a processing gain of 10 the error-rates are 4.38×10^{-4} and 5.86×10^{-4} respectively. These results were calculated over a measurement of less than 10 errors. These results are given in table C.9.

5.3.5 Results of Introducing Gaussian Noise to the Model

The previous results have been presented from a simulation which contains user generated noise. This stems from the understanding that the majority of the noise within the system has been generated by other users' contributions. The form of investigation up to now has centred upon the error probability and the number of users contributing interference to the system. The introduction of Gaussian noise is therefore a method of providing a measure of the effectiveness of this technique for other forms of interference generated in communication systems. The user-transmitted spread spectrum and Gaussian noise signals are shown in figure 5.15 for one user's transmission and Gaussian noise of 10 times the user amplitude. The BPSK transmitted signal is a sinewave and the period shown here is less than one chip period, thus no phase transitions are illustrated.

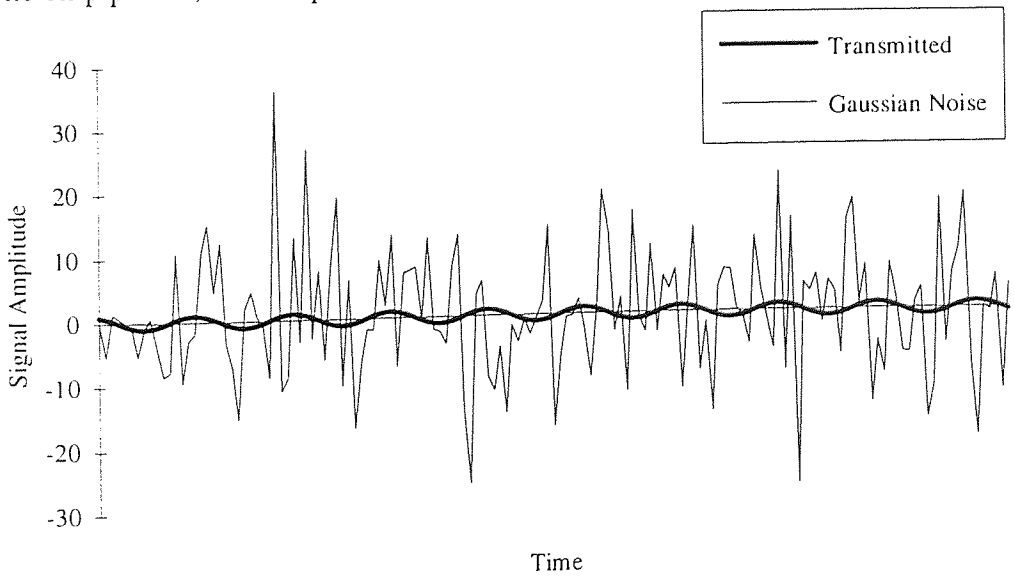


Figure 5.15
Gaussian and Transmitted Signals

The error probability for the system shown in figure 5.15 is around 0.004 and 0.002 for the constant and varying chip duration respectively. Figure 5.16 illustrates an investigation into the effects of Gaussian noise on a DS-CDMA signal. These results were gathered with one user continuously transmitting into a central base station, whilst varying the amplitude of the Gaussian noise introduced into the channel. The simulation used BPSK, a processing gain of 20 and the Qualcomm long spreading code.

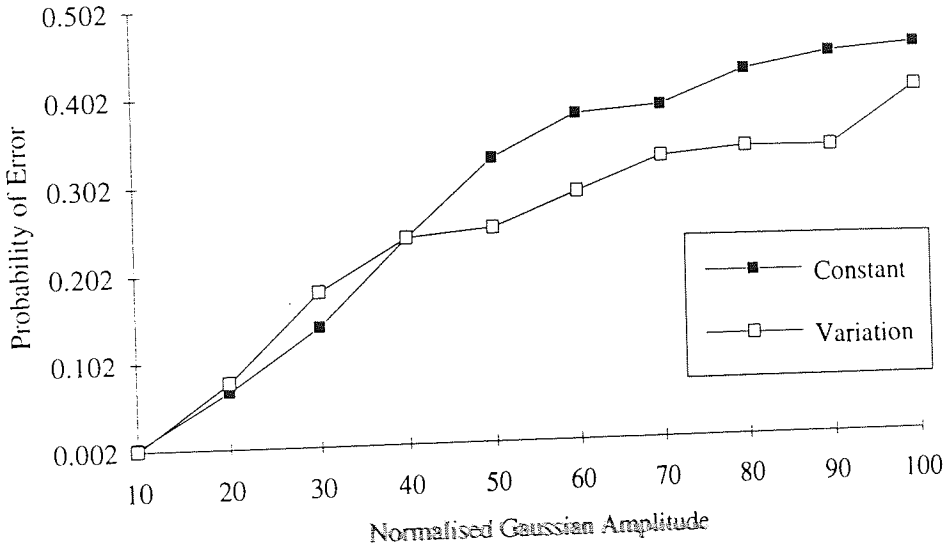


Figure 5.16
Error Probability against Normalised Gaussian Amplitude when Introduced into the System

Figure 5.16 indicates that the variation system is slightly better at communicating in the presence of larger amounts of Gaussian noise than the constant case. Therefore the above results indicate that the variation scheme would be marginally better in a Gaussian noise environment. This would occur mainly in some PMR systems or when used as a secondary system.

5.3.6 The Results of Concurrent Constant and Varying Chip Duration Transmissions

One important consideration that must be examined is the effect on non-varying chip duration users when varying users transmissions are introduced into the system. This is important if this scheme was to be introduced in an existing system with users that do not have chip duration variations.

The results presented in figure 5.17 are taken from a simulation system with 10 users and a processing gain of 20. The number of varying and non-varying users is altered, keeping the total number of system users the same, the probability of error being taken for both the varying chip duration user and the constant user.

Figure 5.17 indicates that when the system contains only users who are transmitting using variations in the chip duration the overall bit error rate is lower than at any other user combination. The general trend in the figure is that the varying users' error probability increases as the proportion of constant duration transmitting users within the system increases. The opposite is true for the constant duration users as this tends to decrease as the proportion of varying duration users decreases.

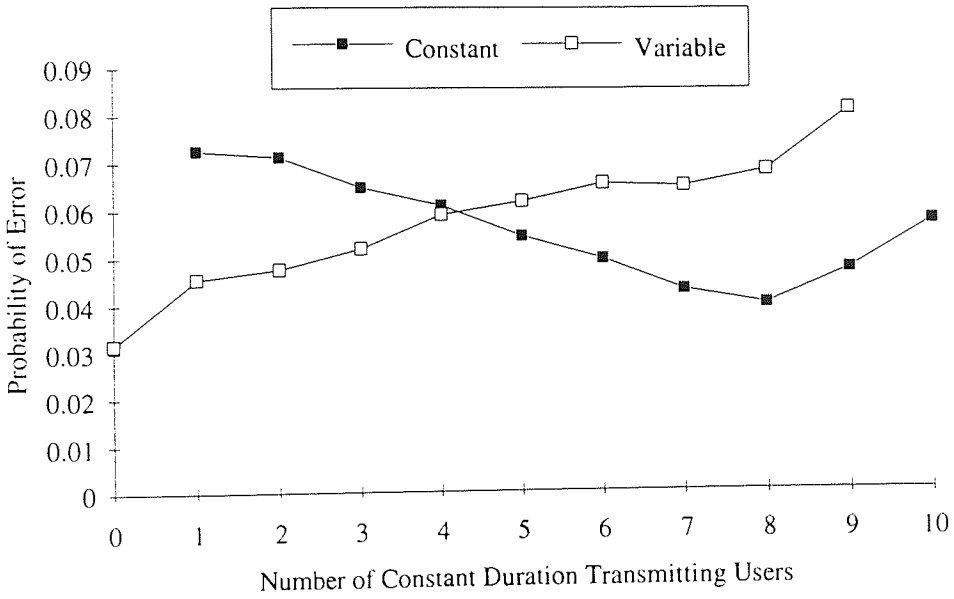


Figure 5.17
Probability of Error for Varying number of Users Transmitting Using Chip Duration Variations

The conclusion drawn from this simulation is that introducing users with varying chip durations into a system with a fixed number of users will increase the error probability for the constant duration users. If constant duration users are introduced to a varying duration system then the varying users will experience higher error-rates related to the number of fixed duration users. These results were obtained for processing gains of between 10 and 50.

5.4 Analysis of Results

In the previous sections various results were presented along with the parameters used. These findings were obtained from a simulation that contained no error control, either in the form of bit interleaving or forward error correction/detection. This was done intentionally, so as to provide a platform which may be used to compare directly the employment of variations. The use of interleaving and FEC would have masked the errors, if not entirely correcting them. Error masking may also occur when the incoming bit stream contains too many errors or in a statistical distribution to which the FEC can not respond.

The analysis of this scheme may be carried out at several points within the DS-CDMA system. The model system diagram given in figure 4.2 has been modified below in figure 5.18 to illustrate the points at which signal waveforms have been taken in analysing the results.

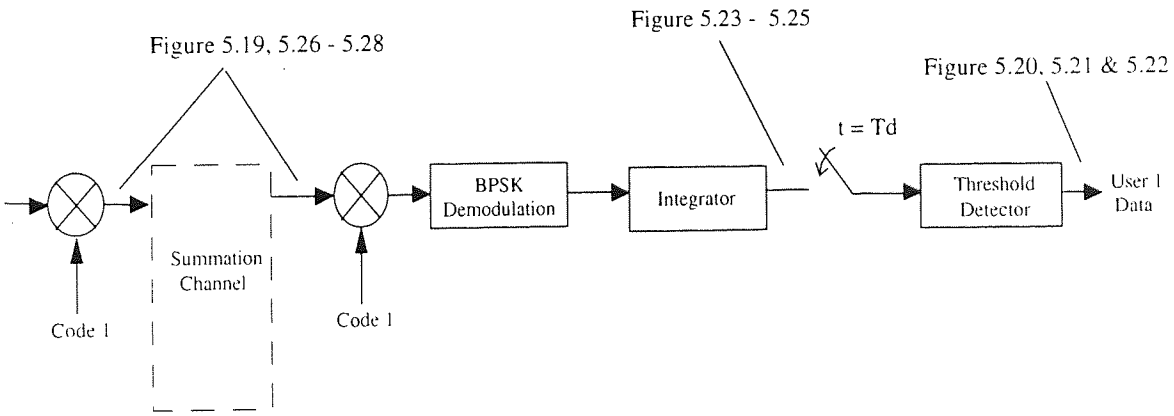


Figure 5.18
Signals Taken for Analysis

The waveforms observed at the first point shows the interference which ten asynchronous users generate at the base station with perfect power control. Figure 5.19 shows the normalised transmitted and the received signals for a period of time less than one chip duration.

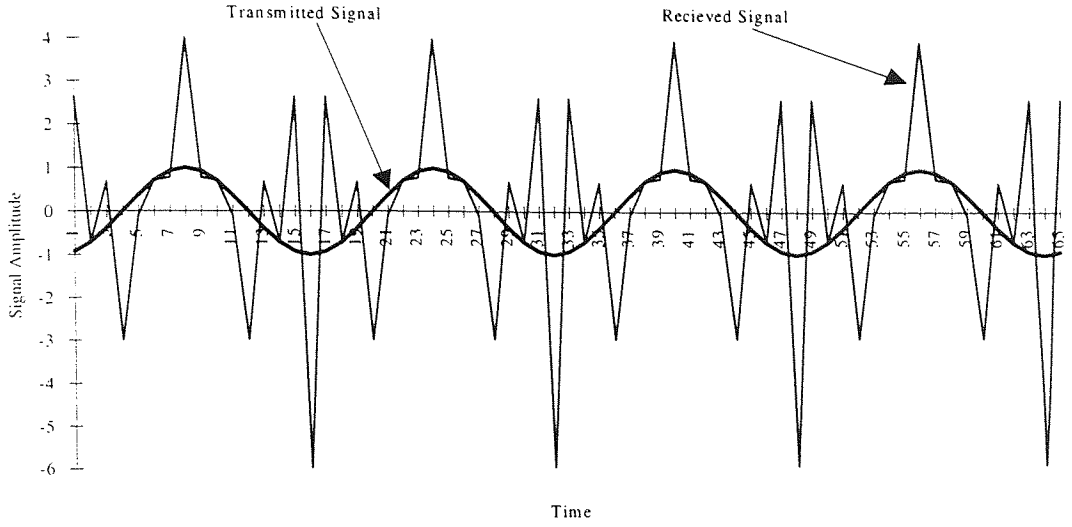


Figure 5.19
DS-CDMA Transmitting and Receiving Signals

The period of figure 5.19 is far less than one bit period and, although the system is asynchronous, the user generated interference within this small time period can be seen to be periodic. The total interference is the sum of all the users transmissions and, as some of the components may cancel each other, the interference is not as high in amplitude as would be expected. Errors are induced by phase and amplitude distortions. Two forms of analysis may now be performed, the first concentrates on the error statistics and the second on the bandwidth utilisation. These are covered in the following sections.

5.4.1 Error Analysis

The results presented for both constant and varying chip durations are compared and a gain is indicated in terms of a reduction in the system bit error-rate. The first analysis is achieved by finding the correlation value ψ (given in equation 4.3) which provides an error probability plot matching those given in figures 5.7 to 5.14. In the BPSK systems shown in figure 5.10, 5.11 and 5.14 this value is 0.5 for the constant duration scheme and 0.8 for the chip duration variation scheme. For the various spreading sequences and modulation schemes used this value alters between 0.5 and 0.55 for the constant duration and for the varying duration scheme 0.77 and 0.8.

To provide a better understanding of the processes involved in varying the chip duration the error statistics can be analysed for any changes in the distribution of errors generated by this form of multiplexing. These errors occur in the simulation in a random manner, which

must be considered at length to ensure the correct operation of the FEC that would be added to a practical system.

This method of analysing the error statistics is to plot the spacing between errors (*i.e.* the number of error free bits) against the number of times this spacing occurs over the simulated bit period. Figure 5.20 shows the error spacing for 10 users with a processing gain of 50, which provides a system error rate of approximately 1 in 200 bits (see table C.3 & C.4) and is measured over a simulation length of 300,000 bits.

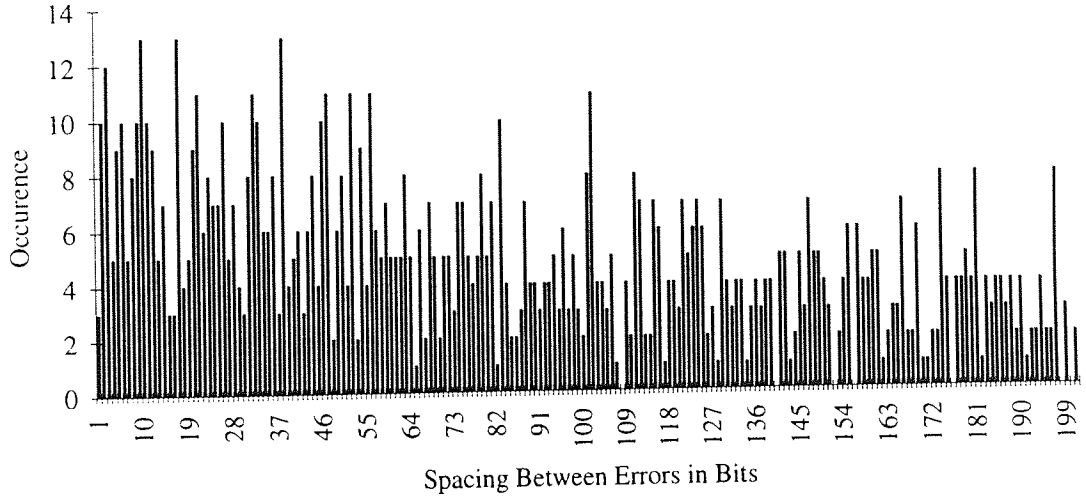


Figure 5.20
Histogram of Error Occurrence

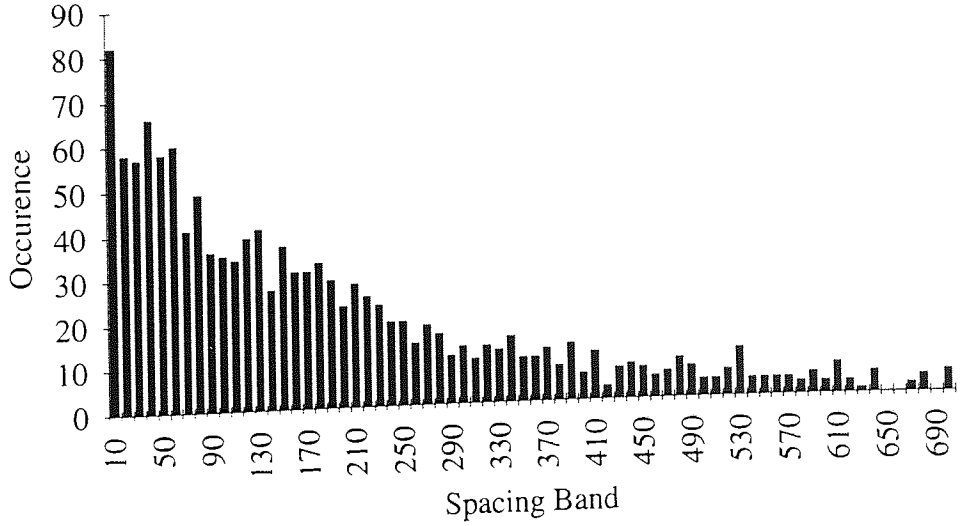


Figure 5.21
Histogram of Error Occurrence in Blocks of 10 Bits

Figure 5.20 indicates that errors occur in a bursty format with no clear statistical distribution evident. Figure 5.21 illustrates the same simulation results shown in figure 5.20, this time with a distribution of errors in blocks of 10 error bits to enable an easier examination. From this graph, it may be seen that the highest occurrence of error spacing is between 0 and 60 bits with a diminishing off to around 10 occurrences at a spacing of 290 bits.

The error statistics are mainly due to the nature of multiplexing. The model does not include any multipath components in which other errors may be included. Thus the errors produced are directly generated by user noise contributions and the correlation function of the spreading codes. The statistics that are produced in the error distribution graph can now be compared with those produced in a constant chip duration environment and an analysis of the change in the error statistics may be determined. The main distributions that are commonly used in communication theory may now be compared with the distributions which have been produced by the simulation. The common distributions used in communication theory are Gaussian (normal), binomial, poisson and Rayliegh, all of which show no resemblance to the distribution found here. One notable point about the error distribution is that it is not guassian as the curve has only one peak at the lower (1 to 10) error bit spacing, thus the simulated distribution is closer to an exponential or chi-squared than to any other statistical distribution.

A family of these exponential curves may be plotted where the error statistics are taken for various processing gains. The plots given in figure 5.22 show the results of processing gains of 10 and 20 with a variation of 0.2.

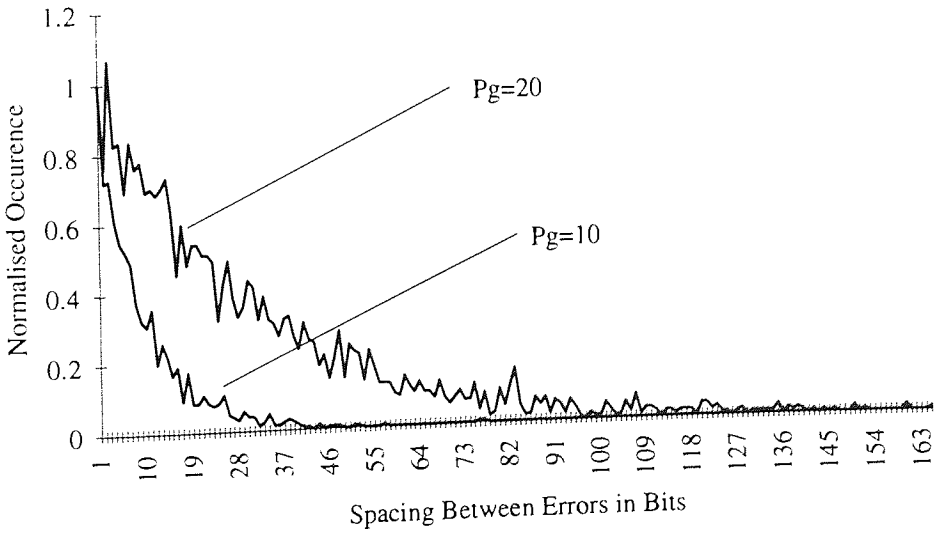


Figure 5.22

Histogram of Error Occurrence for Processing Gains of 10 and 20

When an exponential curve is aligned to the distributions shown in figure 5.22 the close fitting line has the following equation,

$$y = e^{\frac{-x}{B}} \qquad 5.3$$

where B is 10, 30 and 70 for processing gains of 10, 20 and 50 respectively indicating that the error-rate and error statistics are dependent upon the processing gain.

The probability of error equation which is usually given for communication systems assumes that the noise accumulated in the system, and the production of errors is associated with the gaussian distribution of the white noise in the channel. Bit interleaving can be implemented in the system so that a Gaussian distribution may still be maintained. Therefore, considering figure 5.21 above, the centre of the graph can be made to fold out, thus producing a peak in the middle (*i.e.* at 200 bits spacing) of the graph and a curve that is more 'normal'.

One signal that can be taken from the model which indicates the error probability, is the correlator output. This signal indicates the amount of correlation between the locally generated wanted signal and the received set of signals over the bit period. The signal obtained here is the integrate and dump value which provides a good indication of both the bit logical value and the amount of correlation between the two signals. Figures 5.23 and 5.24 show the output values of the integrator during the bit periods for five separate bits. The number of BPSK samples transmitted and received was 3201 during this one bit period and the graph is made up of only every 100th sample, which smoothes out some of perturbation in the line and makes the graphs more manageable. The final value at sample 3201 is the $t=T_b$ sample which is used to determine the data bit value. The upper area or positive values indicate a +1 data bit while the bottom or negative values produce a -1 data bit value. The further they are from the zero point the superior the signal-to-noise ratio and less chance of a bit being in error.

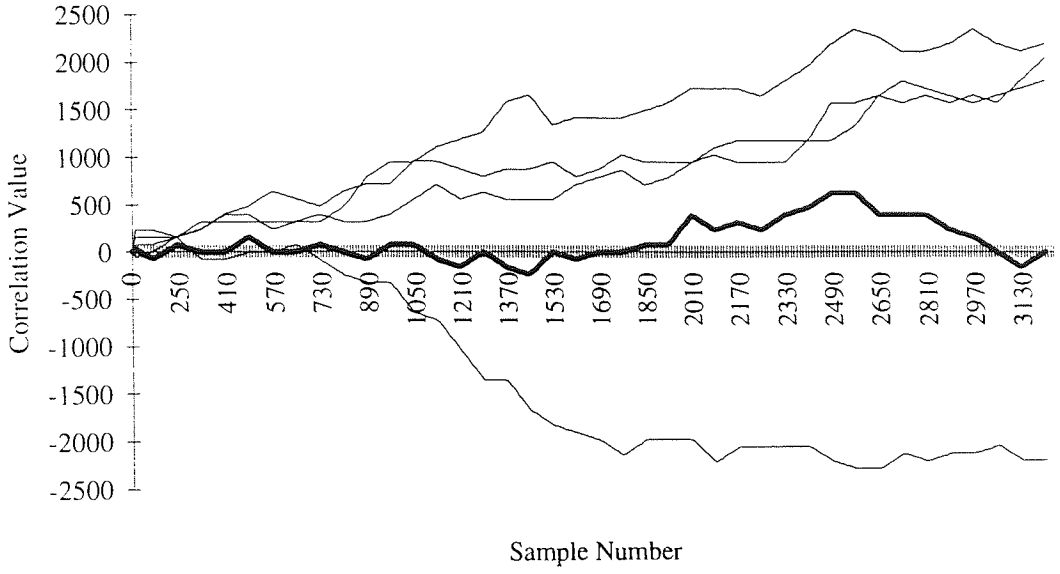


Figure 5.23
Correlation Graph for Constant Duration Chips

In figure 5.24 the second bit (in bold) was in error; this line moves around the zero line throughout the bit period and the final value, which is a negative sample value, produces an erroneous bit.

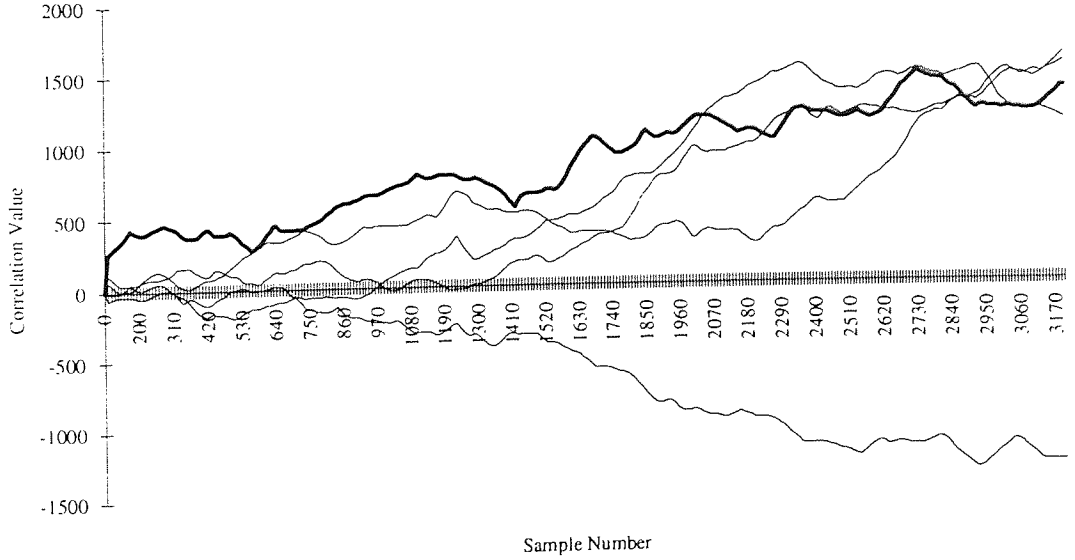


Figure 5.24
Correlation Graph for Variations in the Chip Duration

The constant duration correlator values in both the upper and lower sections of the graph are higher in value than for the variable chip duration scheme, indicating that the signal-to-

noise ratio when variations are implemented is lower. These values are typical of the correlator output and the statistics for the modulus of this output are shown in table 5.2.

	Constant	Variable
Mean	848.036	***850.1***
Standard Error	21.84552	18.96347
Median	872	846
Mode	958	798
Standard Deviation	488.4806	424.0361
Variance	238613.3	179806.6
Range	2636	2060
Minimum	8.93E-14	4.44E-15
Maximum	2636	2060
Sum	424018	***425050***

Table 5.2
Statistics of Correlator Output

The main variables that account for the variation error probability being lower are the mean and the sum. The mean over the sampled 500 bits is higher as is also the sum. Figure 5.23 shows that a bit in error is produced by a lower than average correlator output value. The signal with variations has statistics which show that there is a smaller standard deviation or variance in the correlator output which further indicates that the varying chip duration signal is less prone to fluctuations.

Another way of representing the received signal and noise components is by a phasor diagram as is shown in figure 5.25. This diagram shows the received QPSK sample phases after correlation with the locally generated signals over a one bit period, with a chip variation of 0.2 and a processing gain of 10.

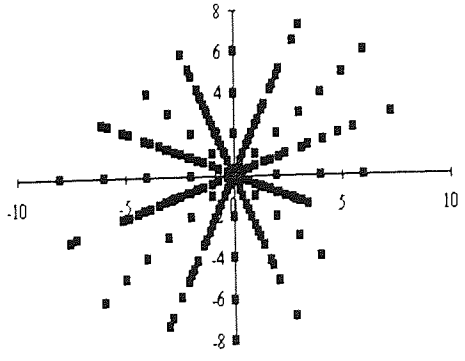


Figure 5.25
QPSK Constellation for Variations in the Chip Duration

The QPSK signal transmitted may be one of four phases and would be represented on the phasor diagram as four points at 45° to the vertical and horizontal axis. The received signal shown contains 16 phases of various amplitudes (distance from the centre) which are generated by contributions from the other transmitting users. A small number of samples is received in the correct phase producing in this instance an error-rate of 1 in 33. This illustrates that the user generated noise produces both phase and amplitude distortion, the phase being attributed to summation of user codes (cross-correlation) for the number of users transmitting in the system. When comparing the modulation schemes in Ziemer92 it was stated that phase modulation is susceptible to phase distortion. The phase distortion shown in figure 5.25 is not random and to some extent is deterministic (see figure 5.19) due to the interference being generated by users who are communicating using similar spreading codes, enabling techniques to be used which may combat this noise such as interference cancellation.

5.4.2 Spectral Analysis

When increasing the capacity of the DS-CDMA system, the spectral profile must be considered. The main investigation is as to whether the addition of variations has increased the bandwidth of the system and, if so, by what amount. The bandwidth occupied by the system is determined [Smirnov90, Yven79] by a combination of the chip rate, or the duration between chip transitions, and the modulation scheme employed. The spectral density is determined by taking the Fourier transform of the transmitted time domain signal $S(t)$, (first given in equation 2.24 for BPSK) given below in equation 5.4.

$$S(t) = A\, c(t)\, d(t) \cos(\omega_o t) \qquad 5.4$$

The first term, A defines the amplitude of the transmitted signal and is maintained at a constant power controlled level throughout the simulations. The second term $c(t)$ relates to the spreading code and, in the simulations, various spreading codes were used. For analysis we shall assume that a maximal length code is used. The auto-correlation function of a maximal sequence is very easily defined as it is a two valued function (Table 2.2) and therefore, taking the fourier transform over the period of this sequence, provides the power spectrum shown in figure 5.26 [Dixon76, Ziemer92].

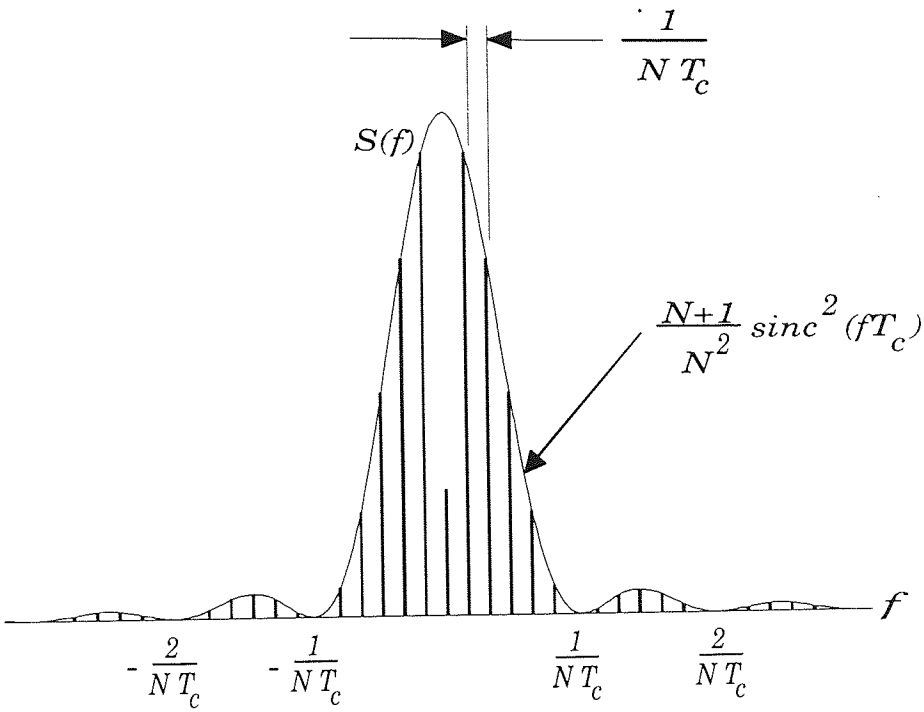


Figure 5.26
Power Spectrum of a Maximal Length Sequence

In modulating the code, it is multiplied by the carrier frequency and the data stream *d(t)*. This, in the frequency domain, means that the above spectrum is convolved with the spectrum of the other two components producing a similar spectrum in the required band. Using the Fourier transforms given in appendix D, equation 5.4 may be transformed from the time domain to give the following frequency domain equation for a single users transmission:

$$G(f) = A\left[\frac{1}{T_c}.\mathrm{comb}_{\frac{1}{T_c}}\left\{|T_c|\mathrm{Sa}(fT_c)\right\} \otimes \frac{1}{T_d}.\mathrm{comb}_{\frac{1}{T_d}}\left\{|T_d|\mathrm{Sa}(fT_d)\right\} \otimes \frac{1}{2}\left[\delta(f+f_o)+\delta(f-f_o)\right]\right] \qquad 5.5$$

where A is the amplitude of the transmitted signal, T_c is the duration of the chip, T_d is the duration of the data and f_o is the carrier frequency. The maximal length sequence used in the simulation was 2^{39} ($\approx 5.497\times10^{11}$) in length, producing a distance between frequency bins of 1.818×10^{-12} Hertz. Therefore the transmitted spectrum is as shown in figure 5.27 for a centre frequency of 1000Hz and a chip rate of 10π. Only the power outline is illustrated.

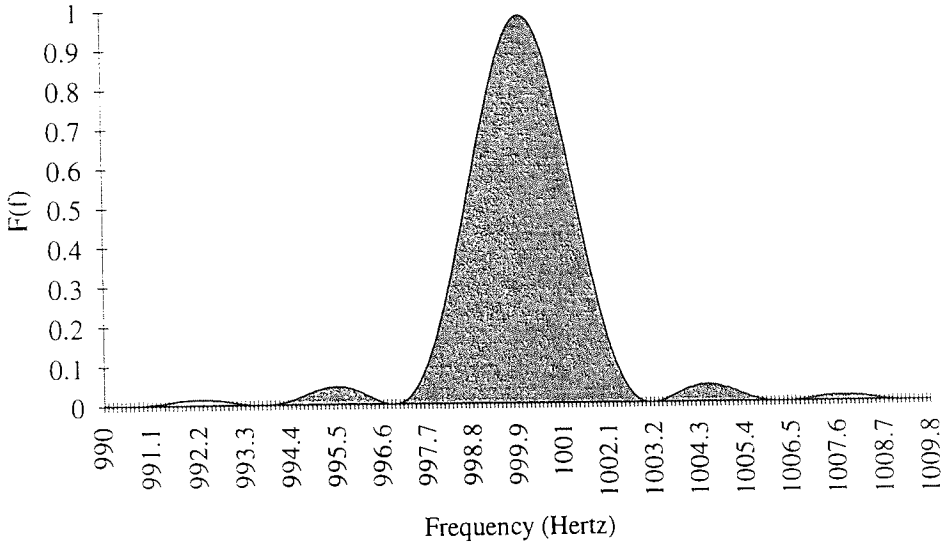


Figure 5.27
Spectrum of Spread Spectrum Signal

Equation 5.5 may now be altered to take account of the varying chip durations. The value of T_c is to be replaced by the variation function (given in equation 5.1) and in equation 5.6 is designated as T_c'.

$$G(f) = A\left[\frac{1}{T_c'}.\text{comb}_{\frac{1}{T_c'}}\left\{\left|T_c'\right|\text{Sa}(fT_c')\right\}\otimes\frac{1}{T_d}.\text{comb}_{\frac{1}{T_d}}\left\{\left|T_d\right|\text{Sa}(fT_d)\right\}\otimes\frac{1}{2}\left[\delta(f+f_o)+\delta(f-f_o)\right]\right] \qquad 5.6$$

By varying the chip duration, the signal spectrum is altered. When averaged, the spectrum is determined by the maximum and minimum chip duration values. Figure 5.28 illustrates only one half of this spectrum. Three spectra are shown, the first, which is the middle line (bold) is the spectrum for the varying chip duration and is an average of the other two, which are the maximum chip duration (top) and minimum chip duration (bottom) signal's spectrum.

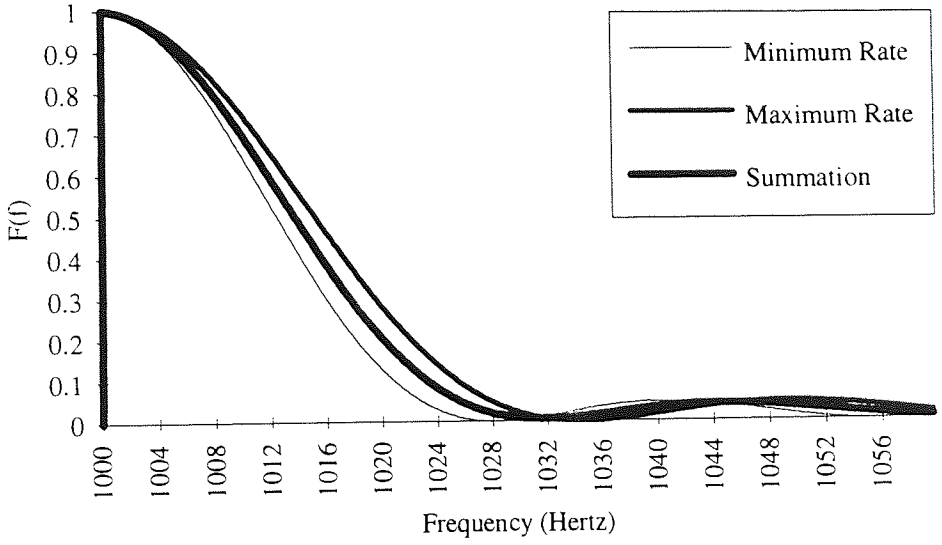


Figure 5.28
Spectrum of BPSK Spread Spectrum Signal with Variations

Figure 5.28 indicates that the variation spectral density will occupy more bandwidth when the chip duration is decreased and less when the chip duration is increased. The spectrum over one bit period will thus be an average of the minimum and maximum chip durations and, therefore, the same as the case where no variations are employed. When allocating bandwidth for this scheme the maximum spectrum should be used, so that interference to adjacent channels is kept to a minimum. The variation used in all the simulations were either 0.2 or 0.1 of a chip period which kept this bandwidth expansion to a small level. These results are similar for all the modulation schemes which are under investigation here.

5.4.3 Analysis of Bandwidth Efficiency

Chapter 1 stated that the basic measure in validating any emerging techniques is the improvement in spectral efficiency. It is therefore important that the techniques effectiveness is firstly compared with the standard constant duration system.

In the simulated system a single cell structure is considered and the data rate is the same for all users throughout the system/cell. Therefore the basic bandwidth efficiency equation given in equation 3.3 may be used and, taking these factors into consideration, the equation is reduced to be the ratio of users to utilised bandwidth. We shall use the results presented in figure 5.14, which represents the error probability against number of users for a constant chip duration and with a variation of 0.2 for a system with a processing gain of 50. The error probability of 0.15 may be used as an arbitrary value for this comparative analysis.

The number of users in a constant duration system is 50 and in the variable duration system 70. This provides a bandwidth efficiency for the constant scheme of 1.0. Taking the maximum bandwidth occupied in the variable chip duration scheme the bandwidth efficiency is 1.166. Therefore an increase in the system bandwidth efficiency of 16% has been obtained by varying the chip duration. If the average is now taken as illustrated by the spectrum shown in figure 5.28 which is equal to the nominal bandwidth, the resulting bandwidth efficiency is 1.4 or an increase of 40%. The reason for both calculations is that if the system employed had only one user stack in the bandwidth (see figure 3.11A), then the maximum chip rate must be used in the bandwidth calculations so that transmissions do not occur outside the allocated bandwidth. If there were say, three user stacks (see figure 3.11B) then the middle stack may accommodate overlap from the outer two stacks.

5.5 Summary

This chapter has shown that including variations to the chip duration of a spread spectrum signal decreases the error-rate of the DS-CDMA system. This decrease in the error-rate is dependent on the amount of variation induced and is related to the maximum bandwidth used. This maximum bandwidth is proportional to the chip duration variation and with the expansions seen during these simulations it has no noticeable effect within the system. The increase in capacity is between 16% and 40% depending on the system implementation.

The varying chip duration scheme was compared with the nominal scheme in Gaussian noise environments and found to have a better error-rate at higher noise levels. The error probability when orthogonal coding was introduced produced an almost equal result. These results were obtained over many hundreds of thousands of bits, producing a small number of errors and therefore a low confidence in the results. The concurrent system results have produced curves illustrating that increasing either the number constant or varying duration users will increase the error-rate of the other type of user.

Chapter 6

Amplitude Variation Spread Spectrum Modulation

6.0 Introduction

This chapter takes the methodology of adding variations to the chip duration one step further by adding variations to the transmitted amplitude of the users' signal. These amplitude variations are produced by using a sparse sequence generator, therefore keeping the average transmitted power to a nominally low average. The aim of introducing amplitude variations is to increase the capacity of the system. The capacity of a code division multiplexing system is inherently limited by amplitude variations, especially when generated in the close proximity of reception. This research therefore, investigates both the use of this form of additional information added to the transmitted signal and the volume of channel noise in the system. The chapter starts with only amplitude variations and then moves on to consider both forms of variation, amplitude and the duration of the chips.

6.1 Adding Amplitude Variations to the Model

In this section the sparse amplitude variation generators are described and the method of their implementation within the model is outlined. The approach taken in varying the amplitude of the signal is important as the gain induced by this process should not detract from the overall system power control. It is intended to change the amplitude of the chip using the chip pattern produced by a sparse code generator as outlined in section 2.3.2.4. This type of code generator will produce a larger sparse amplitude chip at intervals which are determined by the generator taps.

The simulation has one sparse generator per system and therefore the chip amplitude of each user is determined by sequentially taking the amplitude provided by the generator. The first sparse generator which was used in these simulations is shown in figure 6.1. This generator outputs a larger amplitude pulse on average every 7 bits. This generator thus allows one in seven users to have larger amplitudes and therefore an increased power. This sequence generator was named *sparse1_gen()* and its structure is illustrated in figure 6.1. It has a register length of 4 and a sequence period of 15. The output is multiplied by the maximum sparse amplitude A_s.

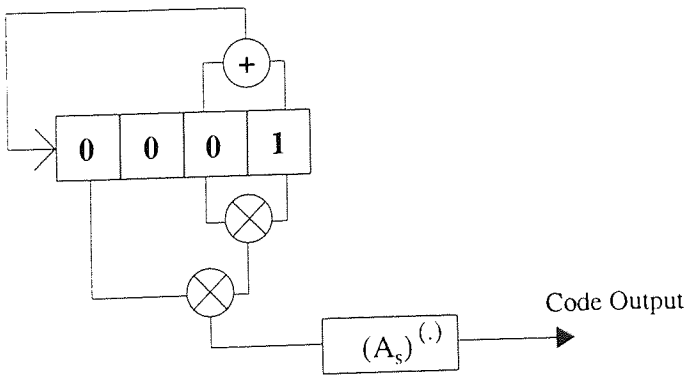


Figure 6.1
First Sparse Amplitude Code Generator

Multiplier Taps	Sparse Output Sequence	Sparse Ratio
1,3,4	000000100001000	0.1333

Table 6.1
First Sparse Generator Sequence

Table 6.1 shows the generator's output sparse sequence and the sparse ratio, which is the ratio of 1s to 0s in the sequence. This generator was chosen so that in every chip period there would be at least one larger amplitude, which provides at least two users with a larger sparse amplitude per chip. If the number of users was, say, set to 10, the sparse amplitudes generated using this first sequence can be seen from table 6.2 to produce the larger sparse amplitudes for users 1 and 5 half the time and none for the other users. The line in table 6.2 shows where the sparse sequence starts to repeat the sequence.

User	1st Chip	2nd Chip	3rd Chip
1	1	0	1
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0
6	1	1	0
7	0	0	0
8	0	0	0
9	0	0	0
10	0	0	0

Table 6.2
First Sparse Generator Sequence Distribution into Chips

This sparse sequence provides a reduced error probability for user one and five as they are continually communicating using the larger amplitudes, whereas the others are not. The probability of error in this system for user one and five is 0.017, whereas the other users have a rate of 0.22, thus the averaging over the system is an error rate of 0.1794. This may be compared with a nominal system rate of 0.145. A second sparse generator was constructed which provides a longer code sequence and various distributions of sparse bits. This second sparse generator, named *sparse2_gen()* is shown in figure 6.2 and will be used in all further results presented in this chapter and those in appendix C.

The second sparse code generator has a register length of 15 and the feedback is the addition of taps 5 and 15. A search was made for the multiplication taps which would provide the best distribution of sparse bits within the sequence. Several different multiplying tap combinations were found to be of use and are shown in table 6.3, with their corresponding generated sequence.

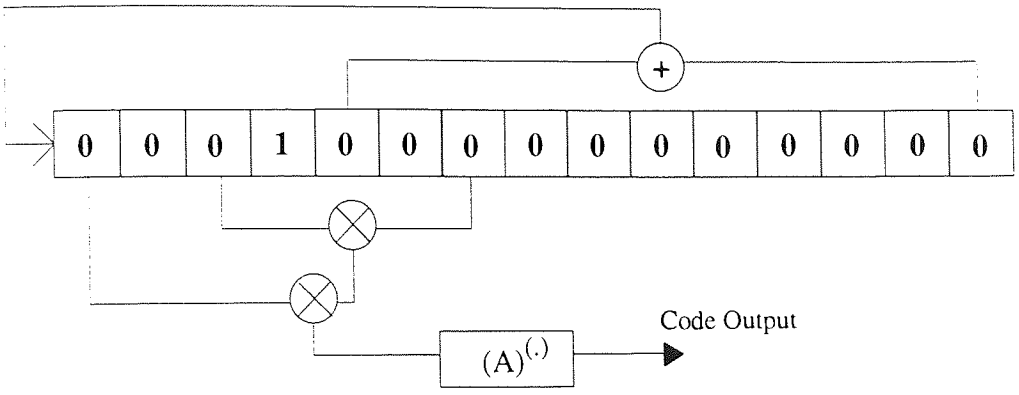


Figure 6.2
Second Sparse Amplitude Code Generator

The sparse ratio is also given in table 6.3 and is determined by the ratio of 1's to 0's in the sequence. This ratio in conjunction with the sparse chip amplitude will determine the addition power induced into the system and is used in comparing the sequences later in this chapter.

Multiplier Taps	Sparse Output Sequence	Sparse Ratio
1,3	10010100101100000000110000000000000	0.22857
1,3,4	10010000000001000000000000000010000	0.14285
1,3,5	10000100000000000000000000000000000	0.05714
1,3,7	10000000000100000000100000000000000	0.08571
1,3,4,7	10000000000000000000000000000000000	0.02857

Table 6.3
Second Sparse Generator Sequences

The greater the sparse ratio the more power there will be in the DS-CDMA system and therefore the probability of error will be greater. It was hoped that a gain would be achieved by introducing the sparse amplitude variations and thus a trade off would occur between increased power and the additional coding provided.

6.2 Model Parameters Used

In section 5.2 the model parameters for the variable chip duration model were stated. The system parameter values that are given in table 5.1 are also viable for this set of simulations. The spreading code used in all simulations in this chapter is the Qualcomm

long spreading code, so that a direct comparison may be made with those results presented in chapter 5.

The one additional parameter that requires definition is the sparse chip amplitude A_s, which is to be varied for different simulations to investigate the relationship between sparse amplitudes and the error performance. The range of sparse amplitudes used in this set of simulations was from twice to one hundred times the nominal (non-sparse) value. This means that the voltage amplitude of the signal will vary from it's nominal or power controlled value to a value which may be as high as one hundred times that value set. This sparse bit amplitude will alter the total system power received in combination with the sparse ratio defined in the previous section.

6.3 Model Results

The following sections present a series of results based on the simulation for both differing types of spreading sequences and gaussian noise. The results presented in these sections are given in table form in appendix C section 5.

6.3.1 The Results of Different Sparse Ratio and Amplitude Variations

The first set of results shows the change in the system probability of error when sparse amplitude changes are added to the system. The results of these simulations are shown in figure 6.3 below, where the amplitude of the sparse chip was varied between 2 and 5 times the nominal constant value.

This graph indicates what would be expected [Gilhousen91, Hulbert93, Newson93] when the amplitudes of users' transmissions are increased, as the received error probability is determined by the users interference contributions. It clearly shows that increasing the transmitted amplitudes decreases the capacity of the DS-CDMA system and that a gain is not induced by using this form of coding. This characteristic of spread spectrum communications is commonly referred to as the near-far effect, which has been discussed in section 2.4. Techniques are available which are capable of resolving this problem and have been examined in 2.5 and Chapter Three.

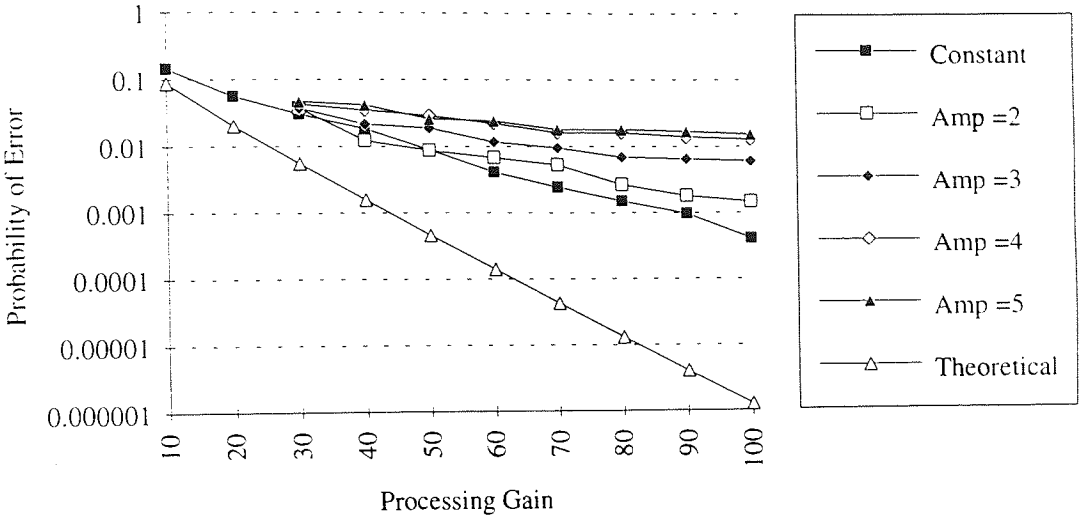


Figure 6.3

A Comparison of Different Sparse Generator Amplitudes

The effect of having sparse amplitudes of four times the nominal amplitude in the system with a sparse ratio of 0.142 provides a total increase in system power of 5.71 times. This gives rise to a comparative increase in the error probability of 3.263 times that of the nominal case. It is noteworthy that increasing the sparse amplitude again has the effect of increasing the probability of error by a diminishing amount. If the amplitude is now changed to 100 times the nominal value, which relates to an increase in power over the four times amplitude case of 24.86 times, (these results given in table C.14) then the comparative error probability increases by a factor of 1.85 times, for the same sparse code.

The increase in user generated interference is dependent upon two factors, one being the sparse amplitude which has been investigated above and the other being the sparse ratio. The sparse ratio determines the average increase in power which the user contributes into the system and therefore relates to an increase in the error probability. The sparse codes and the ratios which have been given in table 6.3 can now be used in the simulation and the relative error probabilities obtained. The results of these simulations are similar to those shown in figure 6.3, where the amplitude of the sparse chip is varied and therefore these results have not been made into a graph.

6.3.2 The Results of Introducing Both Amplitude and Chip Variations

Section 6.3.1 indicated that increasing the amplitude of the sparse generator output decreases the capacity. This section introduces both sparse amplitude and chip duration variations to the transmitted signal. Figure 6.4 compares the results gained in the previous sections, namely sparse amplitude variation with varying both the chip duration and the sparse amplitude. The results obtained using only amplitude variations (top curves) has an increased error rate and a greater line spread (variance of error probability) when compared with the results obtained when both amplitude and chip duration variations are added (bottom set of curves). The graph shown in figure 6.4 also contains the constant or nominal system results which were shown in the previous chapter, adding to the comparison between results.

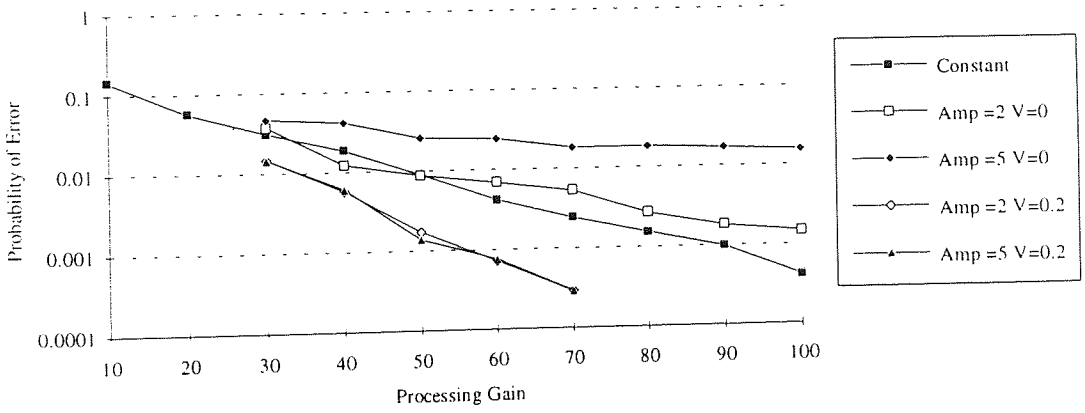


Figure 6.4
Comparison of Chip and Amplitude Variations against only Amplitude Variations

From figure 6.4, it can be seen that introducing chip duration variation decreases the error probability of the signal from the case when only sparse amplitude variations are used. These constant duration signal results can now be compared with those obtained in the previous chapter. Figure 6.5 below shows these sets of curves and compares them to the constant or normal scheme simulation results, along with the ideal theoretical results which have been given in equation 4.3.

These results indicate that introducing the amplitude variation to the chip duration scheme does not increase the system capacity by any noticeable amount as the probability of error results are similar to those without amplitude variations. The results do, however, show that when chip variations are included the amplitude of the signal is not as important in determining the probability of error.

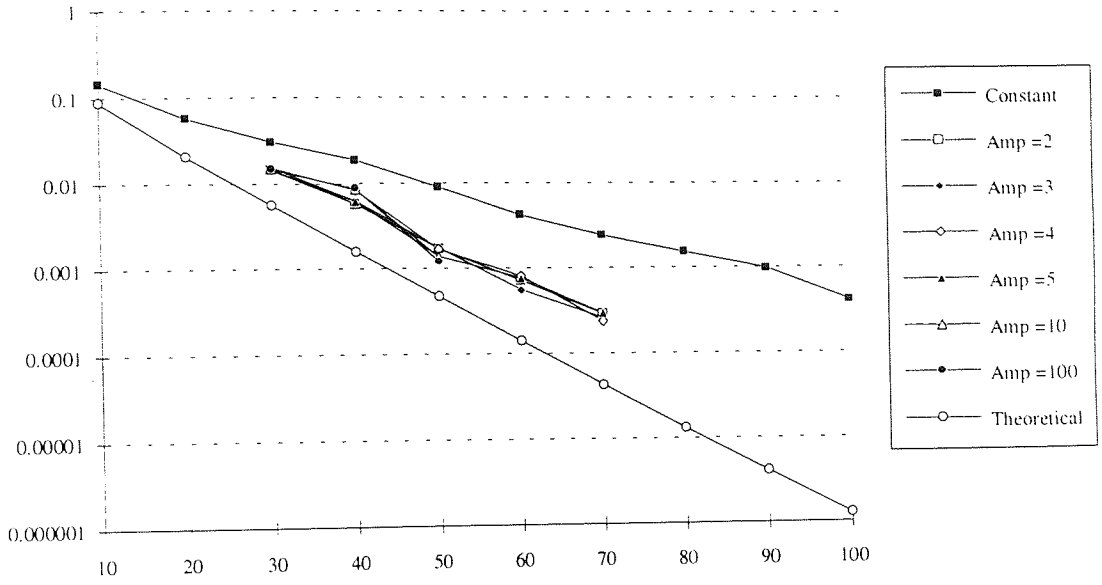


Figure 6.5
Comparison of Chip and Amplitude Variations with the Constant Duration Results and Theoretical Results

6.3.3 Results of the Gaussian Noise Model

The reasons for investigating the effects of Gaussian noise are the same as those presented in section 5.3.5, namely providing a direct comparison with other multiple access systems. The following results were again gathered using 10 users continuously transmitting into a central base station whilst varying the system processing gain and the amplitude of the gaussian noise introduced into system. This allows direct comparison with the results obtained in section 5.3.5.

The graph given in figure 6.6 shows the results obtained from a simulation using BPSK, the Qualcomm long spreading code and a gaussian noise channel. The graph shows two plots, the first is for when only amplitude variations are used, the second, with both amplitude and chip duration variations. These are shown for constant and variation respectively. The varying chip duration plot indicates a better error probability at lower gaussian noise levels. At higher levels of noise, the use of only amplitude variations is preferred with the curve having a smaller gradient than both forms of variation.

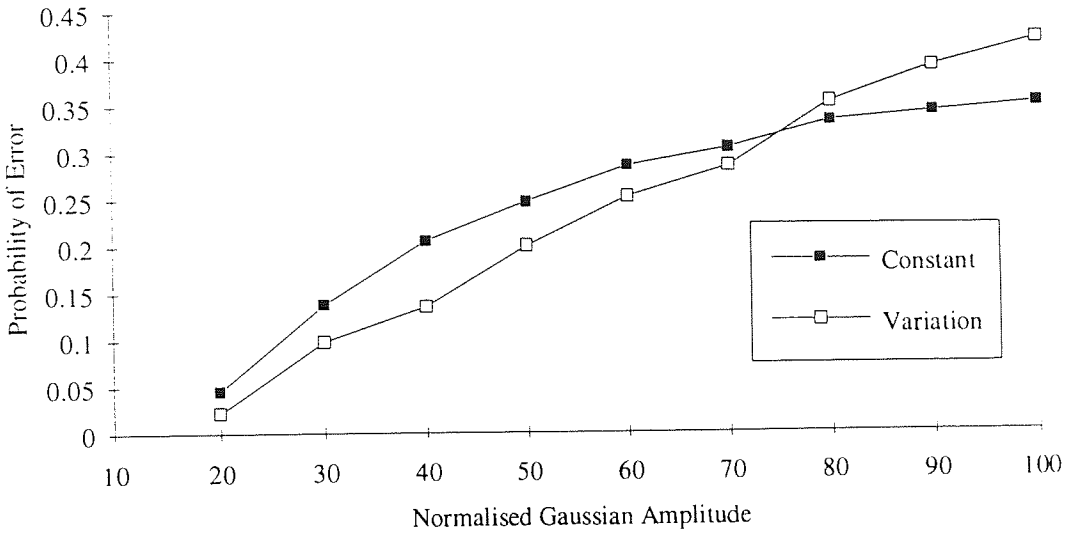


Figure 6.6
Error Probability against Processing Gain
when Gaussian Noise is Introduced into the System

6.4 Analysis of Results

The analysis of the results in this section proceeds by firstly measuring the gain in terms of the error-rate and following on with the bandwidth efficiency and spectral analysis. The results used here are all using the second sparse amplitude generator and varying the chip duration.

6.4.1 Error Analysis

In a previous section the results of amplitude variations were presented, along with the parameters that were used to obtain the results. These results were obtained from the model which contains no error control either in the form of bit interleaving or forward error correction or detection. This was done intentionally to provide a platform to compare directly the use of variations.

Figure 6.7 illustrates similar simulation results to those shown in figure 5.20, except these results are for both amplitude and amplitude-chip variations with a distribution of errors in blocks of 10 error bits to enable an easier examination. From this graph, it may be seen that the highest occurrence of error spacing is near 0 to 10 bits, diminishing off to around 2 occurrences at a spacing of 90 to 100 bits.

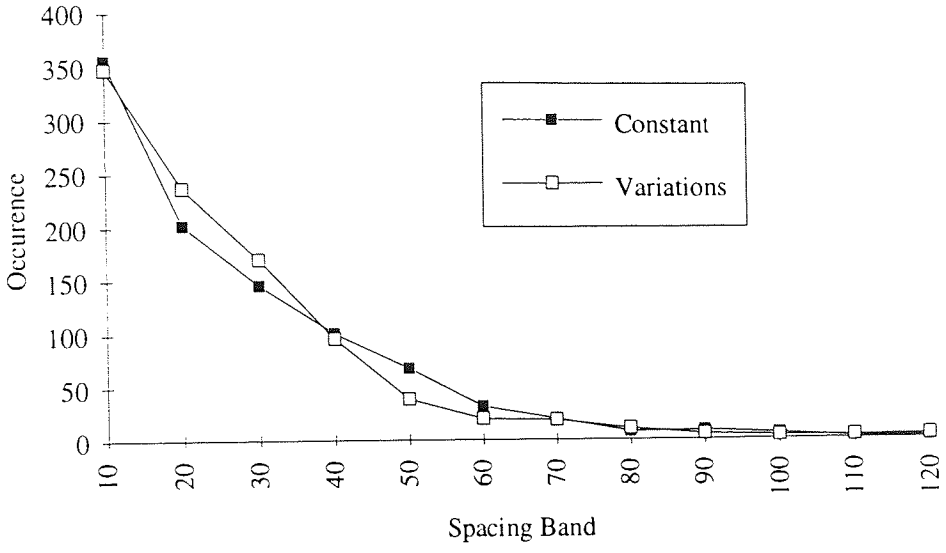


Figure 6.7
Histogram of Error Occurrence in Blocks of 10 Bits
for Both Amplitude and Amplitude-Chip Duration Variations

The results presented in figure 6.7 indicate that having both forms of variation has a worse error probability (integral of the area under the curve) than having only sparse amplitude variations. These results were obtained with a processing gain of 20 and sparse amplitude of three times the nominal value.

	Constant	Variations
Mean	2927.404	**3009.588**
Standard Error	69.08262	64.74193
Median	2876	2999
Mode	3342	2836
Standard Deviation	1544.734	1447.674
Variance	2386204	2095759
Range	9138	7352
Minimum	26	20
Maximum	9164	7372
Sum	1463702	**1504794**

Table 6.4
Statistics of Correlator Output for both Amplitude
and Amplitude-Chip Duration Variations

The error statistics in this section results from the nature of multiplexing when combined with the total user generated interference. The correlation graphs that were generated in the previous chapter (figures 5.23 & 5.24) may again be generated for the error performance of

both cases where the amplitude and both amplitude and chip duration variation are used. These graphs are not generated here because their information content is limited. Only the typical statistics, also given in the previous chapter, are calculated and given in table 6.4.

The mean and sum statisitcal parameters shown in table 6.4 account for the results with both forms of variation having a lower error probability. The mean over the sampled 500 bits is higher, as is also the sum indicating that the spacing between errors is larger. The signal with chip duration and amplitude variations statistics indicate that there is less of a standard deviation or variance in the correlator output. Thus the varying chip duration signal is less prone to fluctuations as the correlator values have a smaller deviation after correlation.

The phasor diagram of the amplitude variation system shown in figure 6.8 indicates the variations received in the signal amplitude.

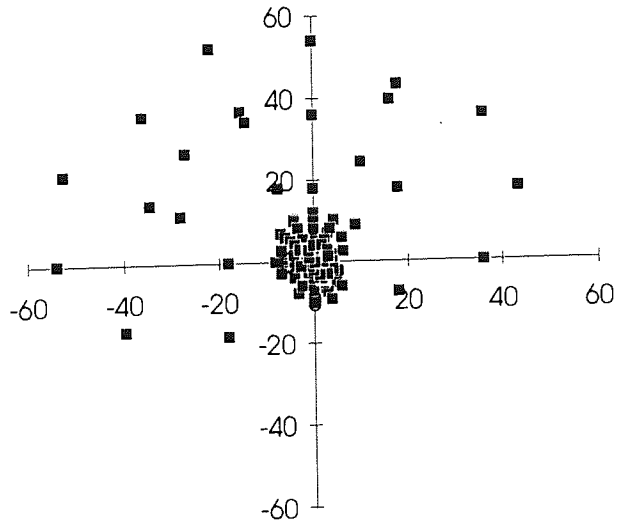


Figure 6.8

QPSK Constellation for Both Amplitude and Chip Duration Variations

The results shown in figure 6.8 are of a sparse amplitude of three times the nominal, a chip duration variation of 0.2 and a processing gain of 20. This, therefore, may be directly compared to the constellation diagram given in figure 5.25. The first notable element is the wider distribution of points on the graph which are the result of amplitude variations in the system. These are sparsely distributed as these higher amplitudes (distance from the centre) are controlled by the sparse code generator and its output amplitude. The centre of the constellation is very similar to that in figure 5.25 and it must thus be assumed that error statistics when the amplitude is at it's nominal value are the same as for the nominal system.

6.4.2 Spectral Analysis

In increasing the capacity of the DS-CDMA system, the spectral usage must now be considered, with the main question being, has the addition of variations increased the bandwidth of the system and by what amount?

Equation 5.5 gave the spectrum of a typical spread spectrum signal and the variations to the chip duration were then added in equation 5.6. Equation 6.1 is the time domain signal when both forms of variation are included in the system, $v^k(t)$ has been included and is the sparse amplitude variation code.

$$S^k(t) = A^k\, v^k(t)\, c^k(t)\, d^k(t)\, \cos(\omega_o t + \alpha^k) \qquad 6.1$$

The Fourier transform of the sparse code is not as straight-forward as the random data and spreading code. In this transformation the ratio of the bit period to sparse bit spacing and amplitude is important. Therefore, using the Fourier transforms given in appendix D, $v(t)$ may be transformed into the frequency domain and is given in equation 6.2.

$$v(f) = A_s \xi . Sa(f\xi) . e^{-j2\pi f\xi/p} \qquad 6.2$$

The sparse ratio ξ is defined as the number of sparse chips with amplitude A_s to non-enlarged amplitude chips. Equation 6.1 may now be transformed using the equation given in 6.2.

$$S(f) = A\left[A_s \xi . Sa(f\xi) . e^{-j2\pi f\xi/p} \otimes \frac{1}{T_c} . \mathrm{comb}_{\frac{1}{T_c}} \left\{ |T_c| \mathrm{Sa}(fT_c) \right\} \otimes \frac{1}{T_d} . \mathrm{comb}_{\frac{1}{T_d}} \left\{ |T_d| \mathrm{Sa}(fT_d) \right\} \otimes \frac{1}{2} [\delta(f + f_o) + \delta(f - f_o)] \right] \qquad 6.3$$

where A is the amplitude of the nominal transmitted signal, T_c is the duration of the chip, T_d is the duration of the data and f_o is the carrier frequency. The sparse amplitude term in the frequency domain induces a further spreading of the signal spectrum proportional to the sparse ratio which is negligible. Equation 6.3 may now be altered to take account of the varying chip durations. This requires the value of T_c to be replaced by the variation function that is defined in equation 5.1 and, when applied to equation 5.6, was designated as T_c'.

$$S(f) = A\left[A_s \xi' . Sa(f\xi') . e^{-j2\pi f\xi/p} \otimes \frac{1}{T_c'} . \mathrm{comb}_{\frac{1}{T_c'}} \left\{ |T_c'| \mathrm{Sa}(fT_c') \right\} \otimes \frac{1}{T_d} . \mathrm{comb}_{\frac{1}{T_d}} \left\{ |T_d| \mathrm{Sa}(fT_d) \right\} \otimes \frac{1}{2} [\delta(f + f_o) + \delta(f - f_o)] \right] \qquad 6.4$$

The sparse ratio ξ' (the ratio of sparse amplitude bits to non sparse amplitude bits) in equation 6.4 now changes as the duration of the chips are varied. By varying the chip duration and the inclusion of sparse amplitude variations, the signal spectrum is again altered from that of the nominal case and also from only chip duration variations. This produces an average spectrum that is determined by both the maximum and minimum chip duration values and sparse ratios.

The spectrum must now be considered for four schemes, firstly the nominal case, secondly the case where there is only chip duration variations, thirdly, where only amplitude variations are used and fourthly, where there are both sparse amplitude and chip duration variations.

The spectrum of the first two cases has been analysed in chapter 5. If the spectrum is considered for the system that has been designated as the nominal case, then it will resemble the spectrum illustrated in figure 5.27. The varying chip duration spectrum is shown in figure 5.28 in bold. The nominal system occupies the least spectrum of the four stated cases.

In the following analysis it is assumed that the chip duration and the sparse ratio are the same *e.g.* 0.1 and the bandwidth of the varying duration scheme is that of the minimum chip duration, so that a direct comparison may be made . The analysis can be made using equation 6.3 which defines the spectrum for the case where only sparse amplitude variations are used and equation 6.4 that defines the case where both schemes are used. The sparse amplitude and varying chip duration schemes each occupy a bandwidth which is dependant upon the sparse ratio or amount of variation. Therefore, if these schemes have the same ratios, they will increase the bandwidth by a similar amount. Employing both forms of variation, the fourth case will exhibit the largest bandwidth when compared with the other cases under consideration. The total bandwidth increase, in this case will be twice that of cases two and three. The values which we are considering here still provide a negligible effect on the overall system bandwidth.

6.4.3 Analysis of Bandwidth Efficiency

In chapter 1, it was stated that the basic measure of any emerging techniques is the increase in spectral efficiency. The first consideration here is that this measure of efficiency is calculated for a given error probability. Introducing sparse amplitude variations has

increased the error-rate and therefore a direct comparison is more difficult. In the simulated system a single cell structure is used and the data rate is the same for all users throughout the system. Therefore the basic bandwidth efficiency equation given in [Burr93] can again be considered to be the ratio of users to bandwidth used. Using the results presented in figure 6.4, we obtain the error probabilities for the above cases three and four.

Taking an error probability of 0.01, the processing gain of the sparse amplitude scheme and nominal schemes is 50. The additional bandwidth occupied by the sparse amplitude scheme is around 22%, having a sparse ratio of 0.11. This can be directly related to the decrease in the bandwidth efficiency when this scheme is employed. For the case where both forms of variation are employed the bandwidth efficiency is lower than having only chip variations and better than the nominal case. Taking the same error probability as above, the processing gain required when both schemes are employed is 34, providing a bandwidth efficiency of 1.104. Therefore the bandwidth efficiency increase for this system is 10%, which has been provided by employing both forms of variation. This is a lower bandwidth efficiency than only varying the chip duration.

6.5 Summary

The research presented in this chapter may be split into two areas. Firstly, it has been shown that a DS-CDMA system is sensitive to amplitude variations. Increasing the system noise causes an increase in the error probability. The chosen method of amplitude variations increased the system interfering power. If a method was used which did not change the average power then this scheme may have provided a system gain. One such method would be to vary the amplitude using a sinewave around the nominal value.

Secondly, and probably the most important result from this chapter, is that the varying chip duration scheme almost eliminates the effects of varying the amplitude. When a range of amplitudes (Figure 6.5) from twice to 100 times the nominal value are applied to the system, only a slight change in error probability can be seen. This change is so small that no clear results can be gauged from the variance. Therefore, the varying duration scheme has been shown to increase system capacity and provide an immunity to amplitude fluctuations. These amplitude fluctuations in a DS-CDMA mobile radio communication system are of the utmost concern and a great deal of power control has been required to combat these effects.

Chapter 7

Conclusions

7.0 Introduction

This research set out to increase the capacity of a direct-sequence code division multiple access radio network and in the previous chapters it is shown that this increase was achieved using two new methods; toroidal sectorisation and varying the chip duration. These techniques produced a total increase in user capacity of between 92% and 140%.

7.1 Discussion of Results

In Chapter Two the capacities of both PMR and cellular DS-CDMA mobile systems were derived, providing a basis from which increases in the capacity can be made. The PMR capacity equation was derived from first principles and compared to the modelled results. These were found to be similar, indicating that the assumptions made in this derivation were correct.

Chapter Three developed toroidal sectorisation and results were presented of the gains provided. These were compared to cell clustering and the conventional form of 'arc' sectorisation. The typical increase in capacity was shown to be 80% for clustering and 50% for toroidal sectorisation. This form of sectorisation in conjunction with cell clustering and 'arc' sectorisation can provide a very powerful cell planning technique for capacity hotspots and for irregularities within the cell structures.

In Chapters Five and Six, two strategies were introduced to vary a DS-CDMA signal in order to increase the capacity. In the nominal case where the duration of the chip and the amplitude of the transmitted signal are constant, the probability of a bit in error is determined by the cross-correlation of the codes in conjunction with the power and number of users within the system. The maximum, or upper bound is approached by using orthogonal coding techniques which are, in practice, used to ensure the maximum utilisation of the multiplexing technique and of the system resources available to it.

The results presented in Chapter 5, where the chip duration is varied, produced a reduction in error probability when compared with the constant duration case. This gain may be calculated in terms of the simulated probability of error for a given number of users and the processing gain or, conversely, in terms of the number of users for a given processing gain and the probability of error. The increase in capacity in terms of the number of users was shown to be around 60%.

The simulation was carried out using a variety of modulation schemes such as BPSK, QPSK and MSK together with spreading codes derived from maximal, Gold, Bent and from the Qualcomm long spreading code. The results for all the chip variations were shown to be similar in reducing the probability of error for all the above system parameter permutations. This was seen in the correlation factor ψ, which is around 0.5 for constant chip duration, and, is 0.8 for the chip duration variations.

The orthogonal sequences used in the simulation were those derived by Walsh and also used in the American Second Generation system developed by Qualcomm. It was demonstrated that when a 64 bit orthogonal code was used in a synchronised network with a processing gain of 64, the ideal case, no errors were measured for either the constant chip or the varying chip duration. A greatly increased error-rate occurred when these parameters were changed, which is expected. The chip duration variation scheme had a slightly higher error probability than the nominal scheme. This is accounted for by the number of errors measured being small and the difference in error-rates being statistically insignificant when compared to the confidence band.

The effect of inducing Gaussian noise into the additive channel showed the variation scheme to have a better error probability. This would be important in a system that experienced large amounts of Gaussian noise or noise that may be modelled as such.

In considering the results from the concurrent, Gaussian noise and amplitude variation simulations, it can be seen that having chip duration variations produces a DS-CDMA system which is less prone to these forms of noise and, therefore, provides a good basis for further investigation.

The addition of amplitude variations to the nominal system provided a decrease in system capacity. This decrease is related to the total additional power induced into the system and is determined by the sparse chip ratio and by the amplitude of these chips. When amplitude variations are implemented in conjunction with chip duration variations, the probability of error is very similar to that when only the chip duration is varied. These results are very important to the implementation and design of future spread spectrum systems as they prove that the varying chip duration technique reduces, if not eliminates, the effect known as the near-far effect. This effect has very serious implications in the design of a DS-CDMA network and the use of variable chip durations provides the additional benefit of reducing the design requirements on the power control determined by the near-far effect. In a fading system, the continually changing amplitudes is a normal occurrence and, therefore, it can be seen that the chip varying system is immune to changes in the amplitude.

7.2 Further Work

One of the advantages of implementing DS-CDMA within a cellular environment is that variants of the basic system may be easily accommodated using the same spectrum without having to re-plan the allocated bandwidth or reprogram the handsets. To some extent this can be extended to the fixed system architecture by using the base stations and other common trunking structures that are already implemented.

Given more time, a fading channel can be implemented in the model. This would give the techniques under analysis a rigorous testing. Phase shift modulation schemes are prone to phase distortion while amplitude schemes are less so. The fading channel becomes important in the analysis of the chip varying scheme when the fading paths are of small or similar duration. Chapter Three proposes methods for increasing the capacity of a DS-CDMA cellular system, and from there one path was taken in each of Chapters Five and

Six. Clearly there are other paths; probably the most beneficial being the use of directional antennas within the handset. This would increase both signal paths and, at the same time, decrease user interference within the system. Further investigations may include the following;

- Varying the amplitude whilst keeping the average the same requires investigation. This may also be done by using a sinewave to provide three forms of coding upon a DS-CDMA transmitted signal.

- An increase in capacity may be achieved by combining source and channel coding. Certain functions provided such as data compression, VODEC, security/encryption, multiplexing/spreading and forward error correction/detection may be combined. These techniques have always been implemented to provide the smallest amount of transmitted bit redundancy. If bits could be added which are easily determined at the receiver, a system gain would be achieved.

- Correct management of frequency and spatial separations will reduce [Bozward95] the total interference. The main provision of a mobile radio system is to allow users to move around the network, therefore, providing capacity requirements which vary in time and place. This is further complicated in the proposed Third Generation by having numerous cell topologies that all require careful consideration in implementation to achieve the optimum capacity. This management function should be performed in real time to take both system resources and capacity requirements into consideration when allocating the system's frequencies and channels.

- Adaptive Processing may be introduced into both mobile and cellular base stations in the form of adaptive antennae, interference cancellation and joint detection. This will be a major benefit to system capacity in a DS-CDMA system, however, battery life and signal processing power limitations will hinder its full potential.

- The design of a DS-CDMA system has to take into account so many factors which all contribute noise to users in the system. Increasing the modulation index is one method which may be developed which would provide a more resilient method of communication.

- Further investigation into concurrent systems is required. The use of DS-CDMA systems as the secondary radio network within a given bandwidth provides benefits to both the licensing authority and to the paying customers. Research is required to

determine the interference which may, in present day and in future systems, be contributed to and by the non-DS-CDMA users.

Appendices

Index

Appendix A

C++ Code Listings for Model Library

A.0 Introduction

PARAM.H	GENS.H	INTER.H	MOD.H
calculate_ber	clear_seq	system_parameters	psk_mod
file_chip	m_gen	refresh_screen	receive_sin_psk_sample
file_sample	gold_gen	display_user	receive_cos_psk_sample
error_handler	qum_gen		decide_bit
check_trans	set_user_mask		qpsk_mod
	sparse1_gen		msk_mod
	sparse2_gen		receive_msk_sample
	bent_gen		channel
	gaussian_gen		
	ran1_gen		

Table A.1

Contents of C++ Library Files Produced for the Model

```
//*********************************************param.h*****************/
// Definition of Simulation Parameters        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//*************************************************************/
#include <fstream.h>
#include <iostream.h>
#include "process.h"
#include <math.h>
#include <stdlib.h>
double const pi=3.141592653589793;
int const max_users=21;                                  //maximum amount of user class
int const tab=0;                                         //tab first chip so is longer
unsigned long int data_bit;                              //bit number
float sample_num;
unsigned int user_num;
unsigned int interval;
int samples_per_cycle,error;
float gaussian_noise;
double channel_noise,de_spread_sample[2];
unsigned int count_plus_one,count_nega_one;
unsigned int count_data_plus_one,count_data_nega_one;
ofstream outfile;
char* output_filename;
class user_parameters
        {
public:  int chip[2];                                    //chip value
         int data_in[2];                                 //data in from maximal sequence gen
         double trans;                                   //user transmission
         double rec;                                     //received signal
         double rx_spread_sample[2];                     //recieved acumative sample
         int data_out[2];                                //data received
         int mask[43];                                   //class of SS user parameters
         unsigned int next_chip;                         //next chip will start at sample number
         unsigned int chip_rate_cps;                     //user chip rate
         int amplitude;                                  //user maximum amplitude value
         unsigned int samples_per_v_chip;                //amount of samples in variable chip
         int walsh_data_in[6];
         int walsh_code_out[64];
         int walsh_code_in[64];
         int walsh_data_out[6];
         int qum_gen();                                  //chip generator
         int m_gen();                                    //maximal length data generator
         int max_gen();                                  //maximal length data generator
         int gold_gen();                                 //gold data generator
         int sparse1_gen();                              //amplitude variation generator 1
         int sparse2_gen();                              //amplitude variation generator 2
         int bent_gen();
         void gen_chip();                                //determines how many samples per chip
         void gen_bit();                                 //determines how many samples per bit
         double psk_mod();                               //user PSK modulator
         double msk_mod();                               //mimimum shift keying modulator
         double qpsk_mod();                              //quadrature phase shift keying modulator
         void display_user();                            //display on screen user parameters
         void triangle_chip_var();                       //triangular variation function generator
         void cosine_chip_var();                         //cosine variation function generator
         void channel();                                 //additive user noise channel
         void delay_channel();
```

```
        void receive_cos_psk_sample();      //recieve sample from cosine quadrature
        void receive_sin_psk_sample();      //recieve sample from sine quadrature
        void receive_msk_sample();
        void decide_bit(int);                       //make decision on bit value
        void walsh_coding();
        void walsh_decoding();
        };
struct system_parameters                    //system parameters
        {
        unsigned int data_rate_bps;         //system data rate in bits per second
        unsigned int ave_chip_rate;         //average chip rate in chips per second
        float variation;
        unsigned long int samples_per_bit;  //samples per data bit
        unsigned long int samples_per_second;       //samples per data bit
        unsigned long int samples_per_chip; //samples per chip
        unsigned long int model_length;             //amount of data bits model will run
        unsigned int frequency_hz;          //transmission frequency in hertz
        unsigned int num_of_users;          //number of transmitting users in system cell
        unsigned int user_v_spacing;        //spacing on varaiation function so to make even spacing over
function
        unsigned int unwanted_user_amp;     //other transmission's amplitude
        unsigned int gain;                  //system processing gain
        char* quick_run;                    //save all samples to disk if long run
        };
system_parameters systems;                  //set up one system
user_parameters user[max_users];            //set up array of class user
extern void file_sample();                  //file samples to disk
extern void file_chip();                    //file chip value to disk
extern void test();                         //test proceedure for debugging
extern void error_handler(char* s);         //display message and exit on error
extern void calulate_ber();                 //calculate and display bit error rate
extern void system_parameters();            //get system parameters from keyboard
extern void set_default_values();           //default setting to start program with
extern void refresh_screen();               //menu to be displayed
#include "d:\work\model\lib\inter.h"        //file contains all keyboard and screen programs
extern void clear_seq();                    //clear all sequences and leave in random part of sequence
extern float ran1_gen(int*);
extern float gaussian_gen(int*);
extern void define_ran1();
extern void undefine_ran1();
#include "d:\work\model\lib\gens.h"         //file contains all pn and sequence generators
#include "d:\work\model\lib\mod.h"          //file contains all modulators and demodulators
int w64[64][64];                            //walsh lookup table
void make_walsh_table();                            //walsh table generator

//*************************************************************/
// Calculate Bit Error Rate        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
// Calls Functions : error_handler;
// Global Variables : data_bit,error,output_filename,systems,user[];
// Local Variables : error_rate;
//*************************************************************/
void calulate_ber()
{
float error_rate=(float)error/(float)systems.model_length;
cout << "\n" << error_rate << "          Error Prob.";
```

```
cout << "/a=" << systems.unwanted_user_amp << "/b=" << systems.data_rate_bps << "/c=" <<
systems.ave_chip_rate << "/f=" << systems.frequency_hz << "/m=" << systems.model_length << "/s=" <<
systems.samples_per_second << "/u=" << systems.num_of_users << "/v=" << systems.variation;
cout << "\n *** Code Statistics *** \n Spreading Code Ratio is " << (float)count_plus_one/count_nega_one << "\n Data
Code Ratio is " << (float)count_data_plus_one/count_data_nega_one << " \n";
ofstream outfile(output_filename,ios::app);   //Open file to append data
if (!outfile)error_handler(output_filename);
outfile << "\n " << output_filename << " Pb          " << error_rate ;
outfile << "/a=" << systems.unwanted_user_amp << "       /b=" << systems.data_rate_bps << "/c=" <<
systems.ave_chip_rate << "/f=" << systems.frequency_hz << "/m=" << systems.model_length << "/s=" <<
systems.samples_per_second << "/u=" << systems.num_of_users << "/v=" << systems.variation;
}
//*****************************************************************/
// File All User Chip Values        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : error_handler;
// Global Variables : output_filename,user,user_num;
// Local Variables : None;
//*****************************************************************/
void file_chip()
{
ofstream outfile(output_filename,ios::app);
if (!outfile)error_handler(output_filename);
outfile << sample_num << "   ";
for (user_num=1;user_num<=systems.num_of_users;user_num++)
            {
            outfile << user[user_num].chip[0] << "   ";
            }
outfile << "\n";
}
//*****************************************************************/
// File User Sample Parameters        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : error_handler;
// Global Variables : systems,user[];
// Local Variables : None;
//*****************************************************************/
void file_sample()
{
if (*systems.quick_run=='n')
            {
            ofstream outfile(output_filename,ios::app);   //Open file to append data
            if (!outfile)error_handler(output_filename);
//          outfile << user[user_num].data_in[0] << "          " << user[user_num].chip[0] << "          " <<
user[user_num].trans << "     " << user[user_num].rec << " " << user[user_num].rx_spread_sample[0] << "   " <<
de_spread_sample << "       " << user[user_num].data_out[0] <<"\n";
            outfile << user[user_num].data_in[0] << "          " << user[user_num].data_in[1] << "      " <<
user[user_num].chip[0] << "   " << user[user_num].chip[1] <<"          " << user[user_num].trans << "          " <<
user[user_num].rec << "       " << de_spread_sample[0] << "          " << de_spread_sample[1] << "          " <<
user[user_num].data_out[0] <<"\n";
            }
}
//*****************************************************************/
// File Error Handler        Last Modified 13/03/95 11am
```

```
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : None;
// Global Variables : None;
// Local Variables : None;
//*****************************************************************/
void error_handler(char* s)
{
        cerr << "\n Cannot Open " << s;
        cerr << "\n Program Terminating";
        exit(-1);
}
//*****************************************************************/
// Test Program        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : None;
// Global Variables : None;
// Local Variables : test_int;
//*****************************************************************/
void test()
{
static int test_int;
test_int=test_int+1;
cout << test_int << "test ";
}
```

```
//*************************************************gens.h*****************/
// Spreading Code Generators          Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simulation in C++
//*************************************************************/
#define M1 259200
#define IA1 7141
#define IC1 54773
#define RM1 (1.0/M1)
#define M2 134456
#define IA2 8121
#define IC2 28411
#define RM2 (1.0/M2)
#define M3 243000
#define IA3 4561
#define IC3 51349
int seq_s1[16],seq_m1[45],seq_g1[14],seq_g2[14],seq_q1[43];
//*************************************************************/
// Clear Generator sequences        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simulation in C++
//
// Calls Functions : None;
// Global Variables :seq_m1,seq_q1,seq_g1,seq_g2,seq_s1,user[],user_num;
// Local Variables : i,j,n;
//*************************************************************/
void clear_seq()          //clear sequence for pn codes
{
int n=45;                                          //n is the longest generator length
for (int i=0;i<=n;i++)                             // Set gens to ones
          {
          seq_m1[i]=1;                                    //maximal length array
          seq_q1[i]=1;                             //qual maximal length array
          seq_g1[i]=seq_g2[i]=1;                   //gold arrays
          }
 seq_g1[1]=0;                                      //set gold reg set one to correct starting state
 seq_g1[2]=1;
 seq_g1[3]=1;
 seq_g1[4]=0;
 seq_g1[5]=1;
 seq_g1[6]=1;
 seq_g1[7]=0;
 seq_g1[8]=1;
 seq_g1[9]=1;
 seq_g1[10]=0;
 seq_g1[11]=1;
 seq_g1[12]=1;
 seq_g1[13]=1;                                           //set reg set two to correct starting state
 seq_g2[13]=0;
 seq_s1[1]=0;
 seq_s1[2]=0;
 seq_s1[3]=0;                                            //load sparse gen with correct bit value
 seq_s1[4]=1;
 seq_s1[5]=0;
 seq_s1[6]=0;
 seq_s1[7]=0;
 seq_s1[8]=0;
 seq_s1[9]=0;
```

```
seq_s1[10]=0;
seq_s1[11]=0;
seq_s1[12]=0;
seq_s1[13]=0;
seq_s1[14]=0;
seq_s1[15]=0;
for (user_num=1;user_num<=systems.num_of_users;user_num++)    //give each user a data bit
        {
        for (i=0;i<=50;i++)                    //start qumgen in random part
        {
                user[user_num].qum_gen();
                user[user_num].m_gen();
                user[user_num].gold_gen();
        }
}
//*****************************************************************/
// User Mask Generator        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : None;
// Global Variables : user[],user_num;
// Local Variables : add,base,input_number;
//*****************************************************************/
void set_user_mask()
{
int base=1;                                            //start with bit number base
int add=1;                                             //and use every add bits
for (user_num=1;user_num<=systems.num_of_users;user_num++)    //give each user a data bit
        {
        float input_number=user_num;
        if ((input_number/256 >=1))
                {
                user[user_num].mask[base+(8*add)]=1;
                input_number=input_number-256;
                }
        if ((input_number/128 >=1))
                {
                user[user_num].mask[base+(7*add)]=1;
                input_number=input_number-128;
                }
        if ((input_number/64 >=1))
                {
                user[user_num].mask[base+(6*add)]=1;
                input_number=input_number-64;
                }
        if ((input_number/32 >=1))
                {
                user[user_num].mask[base+(5*add)]=1;
                input_number=input_number-32;
                }
        if ((input_number/16 >=1))
                {
                user[user_num].mask[base+(4*add)]=1;
                input_number=input_number-16;
                }
        if ((input_number/8 >=1))
                {
```

```
                    user[user_num].mask[base+(3*add)]=1;
                    input_number=input_number-8;
                    }
          if ((input_number/4 >=1))
                    {
                    user[user_num].mask[base+(2*add)]=1;
                    input_number=input_number-4;
                    }
          if ((input_number/2 >=1))
                    {
                    user[user_num].mask[base+(1*add)]=1;
                    input_number=input_number-2;
                    }
          if ((input_number/1 >=1))
                    {
                    user[user_num].mask[base+(0*add)]=1;
                    }
//cout << "user" << user_num <<  user[user_num].mask[5]<< "!"<<  user[user_num].mask[4]<< "!"<<
user[user_num].mask[3]<< "!"<<  user[user_num].mask[2]<< "!" <<user[user_num].mask[1]<< "!"
<<user[user_num].mask[0] <<"\n";
          }
}
//*****************************************************************/
// Maximal Length Code Generator        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : None;
// Global Variables :seq_m1;
// Local Variables : chip_out,i,n,sum_m1;
//*****************************************************************/
int user_parameters::m_gen()
{
static int sum_m1;
int chip_out,i;
//cout << "Starting to generate maximal sequence ";
int const n=5; //15; //39;                    //change n number of registers with different code used
//          sum_m1=seq_m1[1]+seq_m1[2]+seq_m1[15];
                              //generator for pn code  #100003 degree 15
//          sum_m1=seq_m1[1]+seq_m1[4]+seq_m1[39];
                              //generator for pn code  #10000000000021 degree 39
          sum_m1=seq_m1[5]+seq_m1[2];
                              //generator for pn code  #45 degree 5
          sum_m1 = sum_m1%2;                  //modulo 2 addition
//          cout << "seq[n]=" << seq_m1[n];        //test mgen output
          if (seq_m1[n]==0)
                    {
                    chip_out=-1;
                    }
          else chip_out=1;                                    //if seq[n] is greater than 0 then =1
          for (i=n;i>1;i--)                                   //step bit around registers by one
                    {
                    seq_m1[i]=seq_m1[i-1];
                    }
     seq_m1[1] = sum_m1;
 return  chip_out;
 }
 //*****************************************************************/
```

```
// Maximal Length Code Generator        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : None;
// Global Variables :seq_m1;
// Local Variables : chip_out,i,n,sum_m1;
//*****************************************************************/
int user_parameters::max_gen()
{
static int sum_m1;
int chip_out,i;
//cout << "Starting to generate maximal sequence ";
int const n=39;                                    //change n number of registers with different code used
//          sum_m1=seq_m1[1]+seq_m1[2]+seq_m1[15];
                        //generator for pn code  #100003 degree 15
          sum_m1=seq_m1[1]+seq_m1[4]+seq_m1[39];
                        //generator for pn code  #10000000000021 degree 39
          sum_m1 = sum_m1%2;                       //modulo 2 addition
          if (seq_m1[n]==0)
                    {
                    chip_out=-1;
                    }
          else chip_out=1;                                  //if seq[n] is greater than 0 then =1
          for (i=n;i>1;i--)                                 //step bit around registers by one
                    {
                    seq_m1[i]=seq_m1[i-1];
                    }
     seq_m1[1] = sum_m1;
 return  chip_out;
 }
 //*****************************************************************/
 // Gold Code Generator        Last Modified 13/03/95 11am
 // Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
 // Direct Sequence Code Division Multiple Access Simmulation in C++
 //
 // Calls Functions : None;
 // Global Variables :data_bit,sample_num,user[],user_num;
 // Local Variables : chip_out,i,n,sum_g1,sum_g2;
 //*****************************************************************/
 int user_parameters::gold_gen()
 {
 static int sum_g1,sum_g2,chip_out;
 int const n=13;
 if ((data_bit==1) && (sample_num==1))//if first sample of first bit
          {
          for (int i=1;i<=13;i++)                   //load user mask into gold generator array
                    {
                    seq_g2[i]=user[user_num].mask[i];
                    }
          }
 sum_g1=seq_g1[13]+seq_g1[4]+seq_g1[3]+seq_g1[1];          //generator for pn code
 sum_g2=seq_g2[13]+seq_g2[12]+seq_g2[10]+seq_g2[9]+seq_g2[7]+seq_g2[6]+seq_g2[5]+seq_g2[1];     //generator
 for pn code
 //sum_g1=seq_g1[11]+seq_g1[2];
 //sum_g2=seq_g2[13]+seq_g2[5]+seq_g2[3]+seq_g2[1];
 sum_g1 = sum_g1%2;                                        //modulo 2 addition
 sum_g2 = sum_g2%2;                                        //modulo 2 addition
```

```
if ((seq_g1[n]==0 && seq_g2[n]==0) || (seq_g1[n]==1 && seq_g2[n]==1)) chip_out=-1;
else chip_out=1;                                        //if seq_g[n] is greater than 0 then =1
for (int i=n;i>1;i--)                                   // Step bit around registers by one
        {
        seq_g1[i]=seq_g1[i-1];
        seq_g2[i]=seq_g2[i-1];
        }
if (sum_g1==0) seq_g1[1]=0; else seq_g1[1]=1;
if (sum_g2==0) seq_g2[1]=0; else seq_g2[1]=1;
return chip_out;
}
//*****************************************************************/
// Qualcomm Long Spreading Code Generator        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : None;
// Global Variables :mask,user[],user_num;
// Local Variables : chip_out,i,n,sum_q1;
//*****************************************************************/
int user_parameters::qum_gen()
{
static int sum_q1,chip_out;
// cout << "\nStarting to generate qualcomm sequence ";
        sum_q1=(seq_q1[42]+user[user_num].mask[42])%2+
            (seq_q1[35]+user[user_num].mask[35])%2+
            (seq_q1[33]+user[user_num].mask[33])%2+
            (seq_q1[31]+user[user_num].mask[31])%2+
            (seq_q1[27]+user[user_num].mask[27])%2+
            (seq_q1[26]+user[user_num].mask[26])%2+
            (seq_q1[25]+user[user_num].mask[25])%2+
            (seq_q1[22]+user[user_num].mask[22])%2+
            (seq_q1[21]+user[user_num].mask[21])%2+
            (seq_q1[19]+user[user_num].mask[19])%2+
            (seq_q1[18]+user[user_num].mask[18])%2+
            (seq_q1[17]+user[user_num].mask[17])%2+
            (seq_q1[16]+user[user_num].mask[16])%2+
            (seq_q1[10]+user[user_num].mask[10])%2+
            (seq_q1[7]+user[user_num].mask[7])%2+
            (seq_q1[6]+user[user_num].mask[6])%2+
            (seq_q1[5]+user[user_num].mask[5])%2+
            (seq_q1[3]+user[user_num].mask[3])%2+
            (seq_q1[2]+user[user_num].mask[2])%2+
            (seq_q1[1]+user[user_num].mask[1])%2;        //generator for pn code
                                                        //modulo 2 addition
        int temp = sum_q1%2;
//      cout << "seq[n]=" << seq[n]; test mgen output
        if (temp==0)
                {
                chip_out=-1;
                }
                                                //if seq[n] is greater than 0 then =1
        else chip_out=1;
        for (int i=42;i>1;i--)                  // Step bit around registers by one
                {
                seq_q1[i]=seq_q1[i-1];
                }
        seq_q1[1]=(seq_q1[42]+seq_q1[35]+seq_q1[33]+seq_q1[31]+seq_q1[27]+seq_q1[26]+seq_q1[25]+seq_q1[2
2]+seq_q1[21]+seq_q1[19]+seq_q1[18]+seq_q1[17]+seq_q1[16]+seq_q1[10]+seq_q1[7]+seq_q1[6]+seq_q1[5]+seq_q1
[3]+seq_q1[2]+seq_q1[1])%2;
```

```
        //if (user_num==2) cout << chip_out << " $ ";
        return chip_out;
}
//*****************************************************************/
// Bent Code Generator        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : None;
// Global Variables : None;
// Local Variables : None;
//*****************************************************************/
int user_parameters::bent_gen()
int user_parameters::bent_gen()
{
int lqt[6][12];
int y[6][1];
int y1[3];
int y2[3];
int y3[1][1];
int bent_bit_out;
for (int j=0;j<=6;j++) z[j]=0;        //clear sequence
z[user_num-1]=1;                       //code select
for (int r1=0;r1<6;r1++)
        {
        for (int c1=0;c1<12;c1++)
                {
                lqt[r1][c1]=0;
                }
        }
for (int r2=0;r2<6;r2++)
        {
        for (int c2=0;c2<1;c2++)
                {
        y[r2][c2] = 0;
                }
        }
for (int r3=0;r3<1;r3++)
        {
        for (int c3=0;c3<1;c3++)
                {
                y3[r3][c3] = 0;
                }
        }
for (int r4=0;r4<3;r4++)
        {
        y1[r4] = 0;
        y2[r4] = 0;
        }
for (r1=0;r1<6;r1++)
        {
        for (int c1=0;c1<12;c1++)
                {
                for (int colrow1=0;colrow1<12;colrow1++)
                        {
                        lqt[r1][c1] = lqt[r1][c1] + (l[r1][colrow1] * q[colrow1][c1]);
                        }
                }
```

```
        }
for (r2=0;r2<6;r2++)
        {
        for (int c2=0;c2<1;c2++)
                {
                for (int colrow2=0; colrow2<12; colrow2++)
                        {
                        y[r2][c2] = y[r2][c2] + (lqt[r2][colrow2] * max_seq1[colrow2]);
                        }
                }
        }
for (r3=0;r3<1;r3++)
        {
        for (int c3=0;c3<1;c3++)
                {
                for (int colrow3=0; colrow3<12; colrow3++)
                        {
                        y3[r3][c3] = y3[r3][c3] +(qc[r3][colrow3] * max_seq1[colrow3]);
                        }
                }
        }
 int output1,output2,output3;
 y2[0]=y[0][0];
 y2[1]=y[1][0];
 y2[2]=y[2][0];
 y1[0]=y[3][0];
 y1[1]=y[4][0];
 y1[2]=y[5][0];
 output1=((z[0]*y[0][0])+(z[1]*y[1][0])+(z[2]*y[2][0])+(z[3]*y[3][0])+(z[4]*y[4][0])+(z[5]*y[5][0]))%2;
 output2=((y1[0]*y2[0])+(y1[1]*y2[1])+(y1[2]*y2[2]))%2;
 output3=(y1[0]+y1[1]+y1[2]+y1[3])%2;
 bent_bit_out = (output1+output2+output3+y3[0][0])%2;
 static int sum_temp;
 sum_temp = max_seq1[11];
 max_seq1[11] = max_seq1[10];
 max_seq1[10] = max_seq1[9];
 max_seq1[9] = max_seq1[8];
 max_seq1[8] = max_seq1[7];
 max_seq1[7] = max_seq1[6];
 max_seq1[6] = (max_seq1[5]+sum_temp)%2;
 max_seq1[5] = max_seq1[4];
 max_seq1[4] = (max_seq1[3]+sum_temp)%2;
 max_seq1[3] = max_seq1[2];
 max_seq1[2] = max_seq1[1];
 max_seq1[1] = (max_seq1[0]+sum_temp)%2;
 max_seq1[0] = sum_temp;
 return  bent_bit_out;
 }
 //*************************************************************/
 // First Sparse Code Generator        Last Modified 13/03/95 11am
 // Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
 // Direct Sequence Code Division Multiple Access Simmulation in C++
 //
 // Calls Functions : None;
 // Global Variables : seq_s1;
 // Local Variables : chip_out,i,n,sum_s1;
 //*************************************************************/
 int user_parameters::sparse1_gen()
```

```
{
static int sum_s1,chip_out;
int i;
int const n=4;
//cout << "Starting to generate sparse 1 sequence ";
        sum_s1=(seq_s1[3]+seq_s1[4])%2;     //generator for sparse code
        if ((seq_s1[1]*seq_s1[3]*seq_s1[4])%2==1)
                {
                chip_out=3;
                }
        else chip_out=1;                                    //if seq[n] is greater than 0 then =1
        for (i=n;i>1;i--)                                   // Step bit around registers by one
                {
                seq_s1[i]=seq_s1[i-1];
                }
        if (sum_s1==0)                          //if sum equals zero then return zero to array
                {
                seq_s1[1]=0;
                }
        else seq_s1[1]=1;                       //else return a logic one
return  chip_out;
}
//*****************************************************************/
// Second Sparse Code Generator        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : None;
// Global Variables : None;
// Local Variables : None;
//*****************************************************************/
int user_parameters::sparse2_gen()
{
int sum_s1,chip_out;
int i;
int n=15;
//cout << "Starting to generate sparse 2 sequence ";
        sum_s1=(seq_s1[5]+seq_s1[15])%2;   //generator for sparse code
        if (((seq_s1[1]*seq_s1[3]*seq_s1[7])%2)==1)
//      if (((seq_s1[1]*seq_s1[3])%2)==1)
                {
                chip_out=3;
                }
        else chip_out=1;                                    //if seq[n] is greater than 0 then =1
        for (i=15;i>1;i--)                                  // Step bit around registers by one
                {
                seq_s1[i]=seq_s1[i-1];
                }
        seq_s1[1]=sum_s1;                                   //if sum equals zero then return zero to array
 cout << chip_out;
 return chip_out;
 }
 //*****************************************************************/
 // Random Number Generator        Last Modified 13/03/95 11am
 // David Bozward (Aston University and Roke Manor Research Ltd.)
 // Direct Sequence Code Division Multiple Access Simmulation in C++
 // (Taken From Numerical C Recipes)
 // Calls Functions : None;
```

```
// Global Variables : M1,IA1,IC1,RM1,M2,IA2,IC2,RM2,M3,IA3,IC3;
// Local Variables : idum,iff,ix1,ix2,ix3,j,r[],temp;
//*****************************************************************/
float ran1_gen(int *idum)
{
static long ix1,ix2,ix3;
static float r[98];
static int iff=0;
int j;
if (*idum < 0 || iff == 0)
        {
        iff=1;
        ix1=(IC1-(*idum)) % M1;
        ix1=(IA1*ix1+IC1) % M1;
        ix2=ix1 % M2;
        ix1=(IA1*ix1+IC1) % M1;
        ix3=ix1 % M3;
        for (j=1;j<=97;j++)
                {
                ix1=(IA1*ix1+IC1) % M1;
                ix2=(IA2*ix2+IC2) % M2;
                r[j]=(ix1+ix2*RM2)*RM1;
                }
        *idum=1;
        }
ix1=(IA1*ix1+IC1) % M1;
ix2=(IA2*ix2+IC2) % M2;
ix3=(IA3*ix3+IC3) % M3;
j=(int)(1 + ((97*ix3)/M3));
if (j > 97 || j < 1) cout << "RAN1 ERROR: This cannot happen.";
float temp=r[j];
r[j]=(ix1+ix2*RM2)*RM1;
return temp;
}
//*****************************************************************/
// Gaussian Number Generator        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
// (Taken From Numerical C Recipes)
// Calls Functions : None;
// Global Variables : None;
// Local Variables : fac,gset,iset,r,v1,v2;
//*****************************************************************/
float gaussian_gen(int *idum)
{
static int iset=0;
static float gset;
float fac,r,v1,v2;

if  (iset == 0)
        {
        do      {
                v1=2.0*ran1_gen(idum)-1.0;
                v2=2.0*ran1_gen(idum)-1.0;
                r=v1*v1+v2*v2;
                }
        while (r >= 1.0 || r == 0.0);
        fac=sqrt(-2.0*log(r)/r);
```

```
            gset=v1*fac;
            iset=1;
            return v2*fac;
            }
else        {
            iset=0;
            return gset;
            }
}
```

```
//*************************************************inter.h*******************/
// User Interface Routines                Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//*************************************************************************/
//*************************************************************************/
// Acquire System Parameters        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : clear_seq(),error_handler(),refresh_screen(),set_default_values();
// Global Variables : interval,output_filename,systems,user[];
// Local Variables : command;
//*************************************************************************/
void system_parameters()
{
#define STOP 0
#define BLANK ' '
//cout << "\n Initalising System Parameters";
set_default_values();                                    //initilise default value parameters
char command = BLANK;
while (command != STOP)
        {
        refresh_screen();                        //display DS-CDMA system parameters
        cin >> command;
        switch (command)
                {
                case 'a':
                        {
                        cout << "Enter Value : ";
                        cin >> systems.unwanted_user_amp;
                        break;
                        }
                case 'b':
                        {
                        float temp;
                        cout << "Enter Value : ";
                        cin >> temp;
                        if (temp>0)
                                {
                                systems.data_rate_bps=(unsigned int) temp;;
                                }
                        break;
                        }
                case 'c':
                        {
                        float temp;
                        cout << "Enter Value : ";
                        cin >> temp;
                        if (temp>0)
                                {
                                systems.ave_chip_rate=(unsigned int) temp;;
                                systems.frequency_hz=2*systems.ave_chip_rate;
                }
                        break;
                        }
                case 'd':
                        {
```

```
                cout << "Enter File Name : ";
                cin >> output_filename;
                break;
                }
        case 'f':
                {
                float temp;
                cout << "Enter Value : ";
                cin >> temp;
                if (temp>0)
                        {
                        systems.frequency_hz=(unsigned int) temp;;
                        }
                break;
                }
        case 'm':
                {
                float temp;
                cout << "Enter Value : ";
                cin >> temp;
                if (temp>0)
                        {
                        systems.model_length=(unsigned int) temp;;
                        }
                break;
                }
        case 'q':
                {
                cout << "Enter Value (y/n) : ";
                cin >> systems.quick_run;
                if ((systems.quick_run!="y")||(systems.quick_run!="n"))
                        {
                        systems.quick_run="n";
                        }
                break;
                }
        case 'r':
                {
                command=STOP;
                break;
                }
        case 's':
                {
                float temp;
                cout << "Enter Value : ";
                cin >> temp;
                if ((temp>0)&&(temp>systems.frequency_hz*16))
                        {
                        systems.samples_per_second=(unsigned int) temp;
                        }
                else cout << "\n Value is required to be greater than " << systems.frequency_hz*16 << "\n
\n";
                break;
                }
        case 'u':
                {
                float temp;
                cout << "Enter Value : ";
```

```
                        cin >> temp;
                        if ((temp>0)&&(temp<max_users))
                                {
                                systems.num_of_users=(unsigned int) temp;;
                                }
                        else cout << "Value must be Between 0 and " << max_users << "\n \n";
                        break;
                        }
                case 'v':
                        {
                        float temp;
                        cout << "Enter Value : ";
                        cin >> temp;
                        if ((temp>=0)&&(temp<=1))
                                {
                                systems.variation=(float) temp;;
                                }
                        else cout << "Value must be Between 0 and 1 \n \n";
                        break;
                        }
                case 'x':
                        {
                        exit(1);
                        break;
                        }
                default : cout << "Not Valid";
                break;
                }
        }
if (*systems.quick_run=='n')
        {
        ofstream outfile(output_filename,ios::out);  //open file and get ready to append data to it
        if (!outfile)error_handler(output_filename);
        outfile << "/n data_in" << "    " << "chip" << "    " << "trans" << "   " << "rec" << "       " <<
"rx_spread_sample" << "     "<< "de_spread_sample" << "           "<< "data_out" <<"\n";
        }
for (unsigned int i=1;i<=systems.num_of_users;i++)          //give each user an average chip rate
        {
        user[i].chip_rate_cps=systems.ave_chip_rate;    //give each user the average chip rate
        user[i].amplitude=systems.unwanted_user_amp; //give each user the required amplitude settings
        }
user[1].amplitude=1;                                            //ensure user 1 amplitude is always 1
systems.gain=(systems.ave_chip_rate/systems.data_rate_bps);       //Calculate processing gain
systems.samples_per_chip=systems.samples_per_second/systems.ave_chip_rate;
systems.samples_per_bit=systems.samples_per_second/systems.data_rate_bps; //
systems.gain*systems.samples_per_chip;          //calculate how many samples per bit
samples_per_cycle=(int)(systems.frequency_hz/(systems.ave_chip_rate*systems.samples_per_chip));
                                        //how many samples per frequency cycle
systems.user_v_spacing=1/systems.num_of_users; //space users evenly over cosine variation
user[1].data_out[0]=5;          //set data_out to a value to ensure that real data is wrote to it later
user[1].data_out[1]=5;
                                        //set the error count to zero
error=0;
}
//*************************************************************/
// Displays Parameter Menu        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
```

```
// Calls Functions : None;
// Global Variables : output_filename,systems;
// Local Variables : None;
//*****************************************************************/
void refresh_screen()
{
cout << " a          " << "Amplitude of Other Users  = " << systems.unwanted_user_amp << "  \n";
cout << " b          " << "Data Rate              = " << systems.data_rate_bps << " bps  \n";
cout << " c          " << "Chip Rate              = " << systems.ave_chip_rate << " cps \n";
cout << " d          " << "Data Saved to File      = " << output_filename<< " \n";
cout << " f          " << "Frequency              = " << systems.frequency_hz << " Hz \n";
cout << " m          " << "Model Length           = " << systems.model_length << " Data Bits \n";
cout << " q          " << "Quick Run (No Saved Output)= " << systems.quick_run << "  \n";
cout << " r          " << "Run Program             " << " \n";
cout << " s          " << "Samples per Second      = " << systems.samples_per_second << " samples  \n";
cout << " u          " << "Number of Users         = " << systems.num_of_users << "  \n";
cout << " v          " << "Chip Variation         = " << systems.variation << "  \n";
cout << " x          " << "Exit Program            " << "  \n";
cout <<  "\n Required Change Letter = " ;
}
//*****************************************************************/
// Displays User Parameters        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : None;
// Global Variables : user[];
// Local Variables : None;
//*****************************************************************/
void user_parameters::display_user()
{
        cout << "\n #"<<    user[user_num].next_chip;
        cout <<  "/" << user[user_num].chip_rate_cps;
        cout <<  "/" << user[user_num].samples_per_v_chip;
        cout <<  "/" << user[user_num].chip[0];
        cout <<  "/" << user[user_num].data_in[0];
        cout <<  "/" << user[user_num].trans;
        cout <<  "/" << channel_noise;
        cout <<  "/" << user[user_num].data_out[0];
        cout <<  "/" << user[user_num].amplitude << "# \n";
}
```

```
//*****************************************************mod.h*****************/
// Modulator and De-modulator Routines  Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//*****************************************************************/
//*****************************************************************/
// PSK Modulator        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : None;
// Global Variables : sample_num,systems,user[];
// Local Variables : None;
//*****************************************************************/
double user_parameters::psk_mod()
{
return
user[user_num].amplitude*user[user_num].chip[0]*user[user_num].data_in[0]*(cos(2*pi*(systems.frequency_hz*sample
_num)/(user[user_num].chip_rate_cps*systems.samples_per_chip)));
}
//*****************************************************************/
// QPSK Modulator        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : None;
// Global Variables : sample_num,systems,user[];
// Local Variables : path_one,path_two;
//*****************************************************************/
double user_parameters::qpsk_mod()
{
double
path_one=user[user_num].chip[0]*user[user_num].data_in[0]*(cos(2*pi*(systems.frequency_hz*sample_num)/(user[use
r_num].chip_rate_cps*systems.samples_per_chip)));
double
path_two=user[user_num].chip[1]*user[user_num].data_in[1]*(sin(2*pi*(systems.frequency_hz*sample_num)/(user[user
_num].chip_rate_cps*systems.samples_per_chip)));
return path_one+path_two;
}
//*****************************************************************/
// De-Spread Sample        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : None;
// Global Variables : de_spread_sample,channel_noise,systems,user[];
// Local Variables : None;
//*****************************************************************/
void user_parameters::receive_sin_psk_sample()
{
user[user_num].rec=channel_noise;                                    //receive sample
de_spread_sample[1]=user[user_num].amplitude*user[user_num].chip[1]*user[user_num].amplitude*user[user_num].re
c*sin(2*pi*(systems.frequency_hz*sample_num)/(user[user_num].chip_rate_cps*systems.samples_per_chip));
          //correlate rec with locally generate version
user[user_num].rx_spread_sample[1]=user[user_num].rx_spread_sample[1]+de_spread_sample[1];          //add to
previous sample receieved
}
```

```
void user_parameters::receive_cos_psk_sample()
{
user[user_num].rec=channel_noise;                                    //receive sample
de_spread_sample[0]=user[user_num].amplitude*user[user_num].chip[0]*user[user_num].amplitude*user[user_num].re
c*cos(2*pi*(systems.frequency_hz*sample_num)/(user[user_num].chip_rate_cps*systems.samples_per_chip));
//correlate rec with locally generate version
user[user_num].rx_spread_sample[0]=user[user_num].rx_spread_sample[0]+de_spread_sample[0];      //add to
previous sample receieved
}
//*****************************************************************/
// Integrate and Dump Data Recovery        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : None;
// Global Variables : bit,error,percentage,rx_spread_sample,user[];
// Local Variables : None;
//*****************************************************************/
void user_parameters::decide_bit(int bit)
{
if (user[user_num].rx_spread_sample[bit] < 0 )
          {
          user[user_num].data_out[bit]=-1;
          }
else
          {
          user[user_num].data_out[bit]=1;
          }
user[user_num].rx_spread_sample[bit] = 0;
cout << "\n" << user[user_num].data_in[bit] << " = " << user[user_num].data_out[bit] << "                ";
if (user[user_num].data_in[bit]!=user[user_num].data_out[bit])
          {
          ++error;
          }
cout << " Error Rate = " << error << " in " << data_bit << " (" << bit << ")    ";
float percentage=(float)error*100/(float)data_bit;
cout << percentage << "% ";
}
//*****************************************************************/
// MSK Modulator        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : None;
// Global Variables : pi,sample_num,user[];
// Local Variables : None;
//*****************************************************************/
double user_parameters::msk_mod()
{
return
user[user_num].amplitude*user[user_num].chip[0]*user[user_num].data_in[0]*cos((pi*sample_num*(180/pi))/(systems.s
amples_per_chip*2))*cos(2*pi*systems.frequency_hz*sample_num*(180/pi))-
(user[user_num].chip[1]*user[user_num].data_in[1]*sin((pi*sample_num*(180/pi))/(systems.samples_per_chip*2))*sin(2*
pi*systems.frequency_hz*sample_num*(180/pi)));
}
//*****************************************************************/
// De-Spread MSK Sample        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
```

```
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : None;
// Global Variables : channel_noise,de_spread_sample,pi,sample_num,user[];
// Local Variables : None;
//*****************************************************************/
void user_parameters::receive_msk_sample()
{
user[user_num].rec=channel_noise;                    //receive sample

de_spread_sample[0]=user[user_num].rec*user[user_num].amplitude*user[user_num].chip[0]*cos((pi*sample_num*(18
0/pi))/(systems.samples_per_chip*2))*cos(2*pi*systems.frequency_hz*sample_num*(180/pi));
user[user_num].rx_spread_sample[0]=user[user_num].rx_spread_sample[0]+de_spread_sample[0];
                                    //add to previous sample receieved

de_spread_sample[1]=user[user_num].rec*user[user_num].amplitude*user[user_num].chip[1]*-
sin((pi*sample_num*(180/pi))/(systems.samples_per_chip*2))*sin(2*pi*systems.frequency_hz*sample_num*(180/pi));
user[user_num].rx_spread_sample[1]=user[user_num].rx_spread_sample[1]+de_spread_sample[1];
                                    //add to previous sample receieved
}
//*****************************************************************/
// Additive Channel        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : None;
// Global Variables : channel_noise,systems,user[],user_num;
// Local Variables : None;
//*****************************************************************/
void user_parameters::channel()
{
/*if (user_num!=1)    // reduce all other users to fraction of user 1
            {
            user[user_num].trans=user[user_num].trans*systems.unwanted_user_amp;
            } */
channel_noise = channel_noise + user[user_num].trans;
//                                  user[user_num].display_user();
//                                  cout << "\n cn=" << channel_noise << " u=" << user_num <<" tx=" <<
user[user_num].trans;
}
```

Appendix B

C++ Code Listings of Models

B.0 Introduction

Model File Name	Description
bpsk.cpp	Simulates Chip Variation in a BPSK System
walsh.cpp	Simulates Chip Variation in a Orthogonal BPSK System
ampvar.cpp	Simulates Amplitude Variations in a BPSK System

Table B.1
Description of C++ Model Files

```
//*****************************************************bpsk.cpp**************/
// Varying Chip Duration Simulation        Last Modified 6/04/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//*********************************************************************/
#include "d:\work\model\lib\param.h"
//*********************************************************************/
// Simulation Main Proceedure        Last Modified 6/04/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
// Calls Functions : calculate_ber(),clear_seq(),error_handler(),file_chip(),file_sample(),
//          set_user_mask(),system_paremeters(),user[];
// Global Variables : channel_noise,data_bit,error,output_filename,sample_num,systems,user[],user_num;
// Local Variables : None;
//
//*********************************************************************/
void main()
{
output_filename="afpsk.xld";
cout << "\n Starting " << output_filename << " Model \n";
                        //output to screen the model type and standard output filename
system_parameters();                    //user interface and control
set_user_mask();                        //set generator mask for each user
clear_seq();
float hbp[31],hlp[31],hhp[31],f[31];    //filter initialisation
int n;
float sampling_rate=systems.samples_per_second;
float low_freq=systems.frequency_hz-(2*systems.ave_chip_rate);
float high_freq=systems.frequency_hz+(2*systems.ave_chip_rate);
float ol=(2*pi*low_freq)/sampling_rate;
float ou=(high_freq*2*pi)/sampling_rate;
float oc=(ou-ol)/2;
float oo=0.5*pi;
int amplitude=100;
double channel_sum;
cout << ol << "   " << ou << "   " << oc << "   " << oo << "\n";      //output filter coefficients
cout << "hlp[n]" << "              " << "hhp[n]" << "    " << "hbp[n]" << "\n";
for (int t=0;t<=30;t++)   //clear arrays
          {
          hlp[t]=0;
          hhp[t]=0;
          hbp[t]=0;
          f[t]=0;
      }
int i=19;
for(n=1;n<=20;n++)        //generate twenty bin values
          {
          hlp[n]=(1/(pi*n))*(sin(oc*n));
          hhp[n]=2*cos(oo*n*pi);
          hbp[n]=amplitude*hlp[n]*hhp[n];
          hbp[i]=hbp[n];
      --i;
//        if ((hbp[n] < 0.00001) && (hbp[n] >-0.00001)) hbp[n]=0;
                                    //make zero if coefficient is too small
          cout << hlp[n] << "              " << hhp[n] << "                " << hbp[n] << "\n";
          }                         //end of filter bits
data_bit=1;
sample_num=0;
```

```
while (data_bit<=systems.model_length)
        {
        ++sample_num;
        double channel_sum=0;
        for (user_num=1;user_num<=systems.num_of_users;user_num++)
        {                                                   //gen bit
                user[user_num].gen_chip();
                user[user_num].gen_bit();
                user[user_num].trans=user[user_num].qpsk_mod();
                channel_sum = channel_sum + user[user_num].trans;
        }
//*****************
                f[0]=channel_sum;
                for(n=19;n>=0;n--)              // move signal through filter
                        {
                        f[n+1]=f[n];
                }
                channel_noise=0;
                for(n=0;n<=19;n++)              //add all filter bins * signal
                        {
                        channel_noise=channel_noise+(f[n]*hbp[n]);
                        }
         user_num=1;
         user[user_num].receive_sin_psk_sample();
         user[user_num].receive_cos_psk_sample();
         file_sample();
         }
 calulate_ber();                                            //at end of model run calculate ber
 outfile.close();                                           //close open file
 cout << "\n The End of the Simulation";            //inform user end of program has been reached
 }
 //*****************************************************************/
 // Calculate Triangle Varying Chip Durations       Last Modified 13/03/95 11am
 // Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
 // Direct Sequence Code Division Multiple Access Simmulation in C++
 //
 // Calls Functions : None;
 // Global Variables : systems,user[];
 // Local Variables : None;
 //
 //*****************************************************************/
 void user_parameters::triangle_chip_var()
 {
 if (user[user_num].samples_per_v_chip >= systems.samples_per_chip+systems.variation)  //ie max value
          {
          user[user_num].samples_per_v_chip=(systems.samples_per_chip-systems.variation);
          }
 else if (user[user_num].samples_per_v_chip <= systems.samples_per_chip-systems.variation)
          {
          user[user_num].samples_per_v_chip=(systems.samples_per_chip-systems.variation);
          }
 user[user_num].samples_per_v_chip=user[user_num].samples_per_v_chip+interval;
 }
 //*****************************************************************/
 // Calculate Cosine Varying Chip Durations       Last Modified 13/03/95 11am
 // Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
 // Direct Sequence Code Division Multiple Access Simmulation in C++
 //
```

```
// Calls Functions : None;
// Global Variables : pi,sample_num,systems;
// Local Variables : None;
//
//*****************************************************************/
void user_parameters::cosine_chip_var()
{
float
cosine_function=systems.samples_per_chip+(systems.variation*systems.samples_per_chip*cos((systems.user_v_spac
ing+user_num)+(2*pi*sample_num)/(systems.samples_per_bit*systems.samples_per_chip)));
user[user_num].samples_per_v_chip= int (cosine_function - (int
(cosine_function)%(systems.samples_per_second/systems.frequency_hz)));
}
//*****************************************************************/
// Calculate Cosine Varying Chip Durations        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : None;
// Global Variables : pi,sample_num,systems;
// Local Variables : None;
//
//*****************************************************************/
void user_parameters::gen_chip()
{
if (sample_num==1)                                    //first time in
        {
        user[user_num].cosine_chip_var();      //cosine variations
        user[user_num].chip[0]=user[user_num].qum_gen();          //set chip value
        user[user_num].next_chip=user[user_num].samples_per_v_chip;
                                                      //set sample number of next chip
        if (user_num==1)
                {
                if (user[1].chip[0]==1)++count_plus_one; //check stats on spreading code
                if (user[1].chip[0]==-1)++count_nega_one;
                }
        }
if (sample_num==user[user_num].next_chip)      //every chip enter
        {
        user[user_num].cosine_chip_var();      //cosine variations
        user[user_num].chip[0]=user[user_num].qum_gen();                  //set chip value
        user[user_num].next_chip=user[user_num].next_chip+user[user_num].samples_per_v_chip;
                                      //set next enter
        if (user_num==1)
                {
                if (user[1].chip[0]==1)++count_plus_one; //check stats on spreading code
                if (user[1].chip[0]==-1)++count_nega_one;
                }
        }
}
//*****************************************************************/
// Generates New Bit Values       Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : user[];
// Global Variables : sample_num,user[];
// Local Variables : None;
```

```
//
//*************************************************************/
void user_parameters::gen_bit()
{
static unsigned long int next_bit;
if (sample_num==1)
        {
        user[user_num].data_in[1]=user[user_num].m_gen();
        user[user_num].data_in[0]=user[user_num].m_gen();
        next_bit=(user_num*2)+(systems.samples_per_chip*systems.gain)+20;
                                                //set sample number of next chip
        }
if (sample_num==next_bit)
        {
        if (user_num==1)   //reset sample number as it gets too big and makes errors
                {
                user[1].decide_bit(0);
                ++data_bit;
                sample_num=1;
                next_bit=(user_num*2)+(systems.samples_per_chip*systems.gain);
                                        //set sample number of next chip
                }
         else next_bit= data_bit*systems.samples_per_chip*systems.gain;
         user[user_num].data_in[1]=user[user_num].m_gen();
         user[user_num].data_in[0]=user[user_num].m_gen();
         if (user[1].data_in[0]==1)++count_data_plus_one;
         if (user[1].data_in[0]==-1)++count_data_nega_one;
         }
 }
 //*************************************************************/
 // Set System Default Values        Last Modified 13/03/95 11am
 // Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
 // Direct Sequence Code Division Multiple Access Simmulation in C++
 //
 // Calls Functions : None;
 // Global Variables : systems;
 // Local Variables : None;
 //
 //*************************************************************/
 void set_default_values()                      //set values ready for display
 {
 systems.data_rate_bps=1;
 systems.ave_chip_rate=10;
 systems.frequency_hz=100;                      //frequency always at least twice chip rate
 systems.model_length=500;
 systems.samples_per_second=1600;               //16 samples per hertz
 systems.num_of_users=10;                       //10 users standard
 systems.unwanted_user_amp=1;                   //equal amplitude ie perfect power control
 systems.quick_run="y";                         //ie quick run
 systems.variation=0.2;                         //no variation
 }
```

```
//*****************************************************walsh.cpp************/
// Walsh Coding & Varying Chip Duration Simulation        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//*************************************************************************/
#include "d:\work\model\lib\param.h"
//*************************************************************************/
// Simulation Main Proceedure        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : calculate_ber(),clear_seq(),error_handler(),file_chip(),file_sample(),
//          set_user_mask(),system_paremeters(),user[];
// Global Variables : channel_noise,data_bit,error,output_filename,sample_num,systems,user[],user_num;
// Local Variables : i,werror;
//
//*************************************************************************/
void main()
{
int i;
int werror=0;
output_filename="walsh.xld";
cout << "\n Starting " << output_filename << " Model \n";
                        //output to screen the model type and standard output filename
system_parameters();                                    //user interface and control
set_user_mask();                                //set generator mask for each user
clear_seq();                                            //clear all code generators ready for start
make_walsh_table();                                     //creates walsh look up table
for (data_bit=0;data_bit<=systems.model_length;)        //for length of code
        {
        for (user_num=1;user_num<=systems.num_of_users;user_num++)
                                                        //give each user a data bit
                {
                for (int i=0;i<=5;i++)
                {
                        user[user_num].walsh_data_in[i]=user[user_num].m_gen();
                                                        // mgen produces + or - 1 for data in
                        }
                user[user_num].walsh_coding();
                }
        data_bit=data_bit+6;                            //increase data_bit count by 6
        for (i=0;i<63;i++)                      //run model in 64 walsh bit chunks
                {
                for (user_num=1;user_num<=systems.num_of_users;user_num++)
                        {
                        user[user_num].data_in[0]=user[user_num].walsh_code_out[i];
                        }
                for (sample_num=1;sample_num<=systems.samples_per_bit;sample_num++)
                        {
                        channel_noise=0;  //set channel to zero as only user noise required
                        for (user_num=1;user_num<=systems.num_of_users;user_num++)
                                {
                                user[user_num].gen_chip();              //generate chip
                                user[user_num].trans=user[user_num].psk_mod();
                                user[user_num].channel(); //add all users transmissions together
                                }
                        user_num=1;                     //receive only user 1 transmissions
                        user[user_num].receive_cos_psk_sample();
```

```
                    file_sample();
                    }
                user_num=1;
                if (user[user_num].rx_spread_sample[0] < 0 )          //decide bit
                    {
                    user[user_num].walsh_code_in[i]=-1;
                    }
                else
                    {
                    user[user_num].walsh_code_in[i]=1;
                    }
                user[user_num].rx_spread_sample[0] = 0;
                }                                       //end of 64 walsh bits
        user[user_num].walsh_decoding();
        for (int i=0;i<=5;i++)
                {
                cout << "\n" << user[user_num].walsh_data_in[i] << " = " << user[user_num].walsh_data_out[i] << "
          ";
                if (user[user_num].walsh_data_in[i] != user[user_num].walsh_data_out[i])
                    {
                    ++werror;
                    }
                }
        cout << " Error Rate = " << werror << " in " << data_bit<< "  ";
        float percentage=(float)werror*100/(float)data_bit;
        cout << percentage << "%  ";
        }
calulate_ber();                                         //at end of model run calculate ber
outfile.close();                                        //close open file
cout << "\n The End";                                   //inform user end of program has been reached
}
//*****************************************************************/
// Walsh Lookup Table Generator        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : None;
// Global Variables : systems,user[];
// Local Variables : col,col_a,row_a,col_b,col_b_start,row_b_start,count,quad,quadrant,row,row_b,w8,w16,w32;
//
//*****************************************************************/
void make_walsh_table()
{
int w16[16][16];
int w32[32][32];
int col, row, quadrant, quad_count;
int col_a, row_a, col_b, row_b;
int row_b_start, col_b_start;
int  w8[8][8]={{0,0,0,0,0,0,0,0},
               {0,1,0,1,0,1,0,1},
               {0,0,1,1,0,0,1,1},
               {0,1,1,0,0,1,1,0},
               {0,0,0,0,1,1,1,1},
               {0,1,0,1,1,0,1,0},
               {0,0,1,1,1,1,0,0},
               {0,1,1,0,1,0,0,1}};          //standard portion of table which is to be repeated
 quadrant=1;
 for (quad_count=0; quad_count<4; quad_count++)
```

```
{
if (quadrant==4)
        {
        row_b_start=8; col_b_start=8; quadrant=5; row_a=0; col_a=0;
        }
if (quadrant==3)
        {
        row_b_start=8; col_b_start=0; quadrant=4; row_a=0; col_a=0;
        }
if (quadrant==2)
        {
        row_b_start=0; col_b_start=8; quadrant=3; row_a=0; col_a=0;
        }
if (quadrant==1)
        {
        row_b_start=0, col_b_start=0; quadrant=2; row_a=0; col_a=0;
        }
if (quadrant<=4)
        {
        for (row_b=row_b_start; row_b<row_b_start+8; row_b++)
                {
                col_a=0;
                for (col_b=col_b_start; col_b<col_b_start+8; col_b++)
                        {
                        w16[row_b][col_b]=w8[row_a][col_a]; col_a++;
                        }
                row_a++;
                }
        }
 if (quadrant==5)   //changes values for 4th quadrant
        {
        for (row_b=row_b_start; row_b<row_b_start+8; row_b++)
                {
                col_a=0;
                for (col_b=col_b_start; col_b<col_b_start+8; col_b++)
                        {
                        if (w8[row_a][col_a]==1)
                                {
                                w16[row_b][col_b]=0;
                                }
                        if (w8[row_a][col_a]==0)
                                {
                                w16[row_b][col_b]=1;
                                }
                        col_a++;
                        }
                row_a++;
                }
        }                                   //end of quad_count  16x16 hence 16x16 created
}
quadrant=1;
for (quad_count=0; quad_count<4; quad_count++)
        {
        if (quadrant==4)
                {
                row_b_start=16; col_b_start=16; quadrant=5; row_a=0; col_a=0;
                }
         if (quadrant==3)
```

```
                {
                row_b_start=16; col_b_start=0; quadrant=4; row_a=0; col_a=0;
                }
        if (quadrant==2)
                {
                row_b_start=0; col_b_start=16; quadrant=3; row_a=0; col_a=0;
                }
        if (quadrant==1)
                {
                row_b_start=0, col_b_start=0; quadrant=2; row_a=0; col_a=0;
                }
        if (quadrant<=4)
                {
                for (row_b=row_b_start; row_b<row_b_start+16; row_b++)
                        {
                        col_a=0;
                        for (col_b=col_b_start; col_b<col_b_start+16; col_b++)
                                {
                                w32[row_b][col_b]=w16[row_a][col_a]; col_a++;
                                }
                        row_a++;
                        }
                }
        if (quadrant==5)                        //changes values for 4th quadrant
                {
                for (row_b=row_b_start; row_b<row_b_start+16; row_b++)
                        {
                        col_a=0;
                        for (col_b=col_b_start; col_b<col_b_start+16; col_b++)
                        {
                        if (w16[row_a][col_a]==1)
                                {
                                w32[row_b][col_b]=0;
                                }
                        if (w16[row_a][col_a]==0)
                                {
                                w32[row_b][col_b]=1;
                                }
                        col_a++;
                        }
                row_a++;
                }
        }                                               //end of quad_count 32x32
}
quadrant=1;
        for (quad_count=0; quad_count<4; quad_count++)
                {
                if (quadrant==4)
                        {
                        row_b_start=32; col_b_start=32; quadrant=5; row_a=0; col_a=0;
                        }
                if (quadrant==3)
                        {
                        row_b_start=32; col_b_start=0; quadrant=4; row_a=0; col_a=0;
                        }
                if (quadrant==2)
                        {
                        row_b_start=0; col_b_start=32; quadrant=3; row_a=0; col_a=0;
```

```
                }
        if (quadrant==1)
                {
                row_b_start=0, col_b_start=0; quadrant=2; row_a=0; col_a=0;
                }
        if (quadrant<=4)
                {
                for (row_b=row_b_start; row_b<row_b_start+32; row_b++)
                        {
                        col_a=0;
                                for (col_b=col_b_start; col_b<col_b_start+32; col_b++)
                                        {
                                        w64[row_b][col_b]=w32[row_a][col_a]; col_a++;
                                        }
                                row_a++;
                        }
                }
         if (quadrant==5)               //changes values for 4th quadrant
                {
                for (row_b=row_b_start; row_b<row_b_start+32; row_b++)
                        {
                        col_a=0;
                        for (col_b=col_b_start; col_b<col_b_start+32; col_b++)
                                {
                                if (w32[row_a][col_a]==1)
                                        {
                                        w64[row_b][col_b]=0;
                                        }
                                if (w32[row_a][col_a]==0)
                                        {
                                        w64[row_b][col_b]=1;
                                        }
                                col_a++;
                                }
                         row_a++;
                         }
                 }
                                        //end of quad_count 64x64
         }
}
//*************************************************************/
// Walsh Coder        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : None;
// Global Variables : user[];
// Local Variables : col,i,modulation_symbol_index,walsh_data[];
//
//*************************************************************/
void user_parameters::walsh_coding()
{
int modulation_symbol_index, col;
int walsh_data[6];
                                        //convert data to binary 0/1 from -1/1
for (int i=0;i<=5;i++)
        {
        if (user[user_num].walsh_data_in[i]==1)
                {
                walsh_data[i]=1;
```

```
                }
        else
                {
                walsh_data[i]=0;
                }
        }
modulation_symbol_index=walsh_data[0]+(2*walsh_data[1])+(4*walsh_data[2])+(8*walsh_data[3])+(16*walsh_data[4])+
(32*walsh_data[5]);
for (col=0; col<64; col++)                      //fills walsh_spread_data array
        {
        if (w64[modulation_symbol_index][col]==0)
                {
                user[user_num].walsh_code_out[col]=-1;
                }
        else
                {
                user[user_num].walsh_code_out[col]=1;
                }
        }
return;
}
//*****************************************************************/
// Walsh Decoder        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : None;
// Global Variables : user[];
// Local Variables : col,ham_dist,i,modulation_symbol_index,temp_ham_dist;
//
//*****************************************************************/
void user_parameters::walsh_decoding()
{
int col,row,modulation_symbol_index,i;
int temp_ham_dist, ham_dist;
ham_dist=63;
temp_ham_dist=0;
for (row=0; row<64; row++)
        {
        temp_ham_dist=0;
                for  (col=0; col<64; col++)
                        {
                        if (user[user_num].walsh_code_in[col]!=w64[row][col])
                                {
                                temp_ham_dist++;
                                }
                        }
                if (temp_ham_dist<=ham_dist)
                        {
                        ham_dist=temp_ham_dist;
                        modulation_symbol_index=row;
                        }
                                                //end of row loop
        }
 if (modulation_symbol_index-32 >= 0)
        {
        walsh_data_out[5]=1;
        modulation_symbol_index=modulation_symbol_index-32;
        }
```

```
else walsh_data_out[5]=-1;
if (modulation_symbol_index-16 >= 0)
        {
        walsh_data_out[4]=1;
        modulation_symbol_index=modulation_symbol_index-16;
        }
else walsh_data_out[4]=-1;
if (modulation_symbol_index-8 >= 0)
        {
        walsh_data_out[3]=1;
        modulation_symbol_index=modulation_symbol_index-8;
        }
else walsh_data_out[3]=-1;
if (modulation_symbol_index-4 >= 0)
        {
        walsh_data_out[2]=1;
        modulation_symbol_index=modulation_symbol_index-4;
        }
else walsh_data_out[2]=-1;
if (modulation_symbol_index-2 >= 0)
        {
        walsh_data_out[1]=1;
        modulation_symbol_index=modulation_symbol_index-2;
        }
else walsh_data_out[1]=-1;
if (modulation_symbol_index - 1 == 0)
        {
        walsh_data_out[0]=1;
        }
else walsh_data_out[0]=-1;
return;
}
//*****************************************************************/
// Calculate Cosine Varying Chip Durations      Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : None;
// Global Variables : pi,sample_num,systems;
// Local Variables : None;
//
//*****************************************************************/
void user_parameters::cosine_chip_var()
{
float
temp=systems.samples_per_chip+(systems.variation*systems.samples_per_chip*cos((systems.user_v_spacing+user_
num)+(2*pi*sample_num)/(systems.samples_per_bit*systems.samples_per_chip)));
user[user_num].samples_per_v_chip= int (temp - (int (temp)%(systems.samples_per_second/systems.frequency_hz)));
}
//*****************************************************************/
// Generates New Chip Values      Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : user[];
// Global Variables : sample_num,user[];
// Local Variables : None;
//
```

```
//*****************************************************************/
void user_parameters::gen_chip()
{
if (sample_num==1)                              //first time in
        {
        user[user_num].cosine_chip_var();       //cosine variations
        user[user_num].chip[0]=user[user_num].qum_gen();        //set chip value
        user[user_num].next_chip=user[user_num].samples_per_v_chip;
                                                //set sample number of next chip
        }
if (sample_num==user[user_num].next_chip)       //every chip enter
        {
        user[user_num].cosine_chip_var();       //cosine variations
        user[user_num].chip[0]=user[user_num].qum_gen();                //set chip value
        user[user_num].next_chip=user[user_num].next_chip+user[user_num].samples_per_v_chip;
                                        //set next enter
        }
}
//*****************************************************************/
// "        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : None;
// Global Variables : systems;
// Local Variables : None;
//
//*****************************************************************/
void set_default_values()                       //set values ready for display
{
systems.data_rate_bps=1;
systems.ave_chip_rate=20;
systems.frequency_hz=100;                       //frequency always at least twice chip rate
systems.model_length=500;
systems.samples_per_second=1600;                //16 samples per hertz
systems.num_of_users=10;                        //10 users standard
systems.unwanted_user_amp=1;                    //equal amplitude ie perfect power control
systems.quick_run="y";                          //ie quick run
systems.variation=0.2;                                  //variation
}
```

```
//*************************************************ampvar.cpp*****************/
// Amplitude & Varying Chip Duration Simulation        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//*****************************************************************/
#include "d:\work\model\lib\param.h"
int count_amp,chip_num;
int sparsegen();
//*****************************************************************/
// Simulation Main Proceedure        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : calculate_ber(),clear_seq(),error_handler(),file_chip(),file_sample(),
//          set_user_mask(),system_paremeters(),user[];
// Global Variables : channel_noise,data_bit,error,output_filename,sample_num,systems,user[],user_num;
// Local Variables : None;
//
//*****************************************************************/
void main()
{
output_filename="ampvar.xld";
cout << "\n Starting " << output_filename << " Model \n";
system_parameters();
set_user_mask();                                    //set generator mask for each user
clear_seq();
for (data_bit=1;error<=1000;data_bit++)             //for length of code
          {
          for (user_num=1;user_num<=systems.num_of_users;user_num++)
                                                    //give each user a data bit
                    {
                    user[user_num].data_in[0]=user[user_num].m_gen();
                                                    // mgen produces + or - 1 for data in
                    }
          if (user[1].data_in[0]==1)++count_data_plus_one;
          if (user[1].data_in[0]==-1)++count_data_nega_one;
          for (sample_num=1;sample_num<=systems.samples_per_bit;sample_num++)
                    {
                    channel_noise=0;
                    for (user_num=1;user_num<=systems.num_of_users;user_num++)
                              {
                              user[user_num].gen_chip();  //generate chip
                              user[user_num].trans=user[user_num].psk_mod();
//                            user[user_num].display_user();
                              user[user_num].channel();
                              }
//                  file_chip();
                    user_num=1;
                    user[user_num].receive_cos_psk_sample();
//                  file_results();
                    }
           user[user_num].decide_bit(0);
          }
 calulate_ber();
 float temp=(float)count_amp/(float)((systems.gain*data_bit)-count_amp);
 cout << "\n Sparse Ratio =" << temp ;
 outfile.close();
 cout << "\n The End of the Simulation";          //inform user end of program has been reached
```

```
}
//*****************************************************************/
// Calculate Triangle Varying Chip Durations        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : None;
// Global Variables : systems,user[];
// Local Variables : None;
//
//*****************************************************************/
void user_parameters::triangle_chip_var()
{
if (user[user_num].samples_per_v_chip >= systems.samples_per_chip+systems.variation)
                                                        //ie max value
        {
        user[user_num].samples_per_v_chip=(systems.samples_per_chip-systems.variation);
        }
else if (user[user_num].samples_per_v_chip <= systems.samples_per_chip-systems.variation)
        {
        user[user_num].samples_per_v_chip=(systems.samples_per_chip-systems.variation);
        }
user[user_num].samples_per_v_chip=user[user_num].samples_per_v_chip+interval;
}
//*****************************************************************/
// Calculate Cosine Varying Chip Durations        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : None;
// Global Variables : pi,sample_num,systems;
// Local Variables : None;
//
//*****************************************************************/
void user_parameters::cosine_chip_var()
{
float
temp=systems.samples_per_chip+(systems.variation*systems.samples_per_chip*cos((systems.user_v_spacing+user_
num)+(2*pi*sample_num)/(systems.samples_per_bit*systems.samples_per_chip)));
user[user_num].samples_per_v_chip= int (temp - (int (temp)%(systems.samples_per_second/systems.frequency_hz)));
}
//*****************************************************************/
// Generates New Chip Values        Last Modified 13/03/95 11am
// Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
// Direct Sequence Code Division Multiple Access Simmulation in C++
//
// Calls Functions : user[];
// Global Variables : sample_num,user[];
// Local Variables : None;
//
//*****************************************************************/
void user_parameters::gen_chip()
{                                                       //first time in
if (sample_num==1)
        {
        chip_num=1;
        user[user_num].amplitude=sparsegen();
        user[user_num].cosine_chip_var();       //cosine variations
```

```
//          user[user_num].triangle_chip_var();     //triangle variations
            user[user_num].chip[0]=user[user_num].qum_gen();        //set chip value
            user[user_num].next_chip=user[user_num].samples_per_v_chip;
                                                    //set sample number of next chip
            }
if (sample_num==user[user_num].next_chip)      //every chip enter
            {
            if (user_num==1)++chip_num;
            user[user_num].amplitude=sparsegen();
            user[user_num].cosine_chip_var();      //cosine variations
//          user[user_num].triangle_chip_var();    //triangle variations
            user[user_num].chip[0]=user[user_num].qum_gen();                    //set chip value
            user[user_num].next_chip=user[user_num].next_chip+user[user_num].samples_per_v_chip;
                                    //set next enter
            if (user_num==1)
                        {
                        if (user[1].amplitude!=1)++count_amp;          //check stats on spreading code
                        if (user[1].chip[0]==1)++count_plus_one;       //check stats on spreading code
                        if (user[1].chip[0]==-1)++count_nega_one;
                        }
            }
 }
 //*****************************************************************/
 // Set Default Values        Last Modified 13/03/95 11am
 // Written By : David Bozward (Aston University and Roke Manor Research Ltd.)
 // Direct Sequence Code Division Multiple Access Simmulation in C++
 //
 // Calls Functions : None;
 // Global Variables : systems;
 // Local Variables : None;
 //
 //*****************************************************************/
 void set_default_values()                      //set values ready for display
 {
 systems.data_rate_bps=1;
 systems.ave_chip_rate=20;
 systems.frequency_hz=100;                      //frequency always at least twice chip rate
 systems.model_length=1000;
 systems.samples_per_second=1600;               //16 samples per hertz
 systems.num_of_users=10;                       //10 users standard
 systems.unwanted_user_amp=1;                   //equal amplitude ie perfect power control
 systems.quick_run="y";                                                    //ie quick run

 systems.variation=0;                                   //no variation
 }
```

Appendix C

Table of Results

C.0 Introduction

This appendix sets out in a clear format the results which have been obtained in the models outlined in chapters 4, 5 and 6. These results are provided in graphical format in the chapters but it is felt that actual values must be given. Unless otherwise stated in the relevant section the parameters used are those given in table 5.1.

C.1 Variation Function BER

The first two tables show the effect of varying the amount of chip duration variation upon the probability of error. The first table shows the simulation with a processing gain of 20 and 10 users and the second with a processing gain of 40 and 10 users.

Tables C.1 and C.2 are read by using the top, horizontal digits first and then vertical digit to for the second significant digit, providing the amount of variation. As an example to read

the error probability for a variation of 0.23, move across the top to the column which is headed 0.2 and then move down to the row which has 3 in the left Coleman reading a value of 3.78. To conserve space in tables C.1 and C.2, the error probability figures are presented as a percentage (*i.e.* 100th).

	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0	5.75	3.49	3.23	2.92	2.77	2.81	2.57	2.07	1.82	1.91
1	6.85	3.21	3.48	3.50	3.14	2.83	2.44	2.05	1.83	1.79
2	6.43	3.29	3.65	2.93	2.38	2.76	2.29	2.37	1.84	1.92
3	5.63	3.98	3.78	2.76	2.53	2.61	2.82	2.09	1.80	1.60
4	5.41	3.94	3.30	3.33	3.02	2.52	2.39	2.26	2.07	1.43
5	4.11	3.86	3.12	3.67	2.85	2.83	2.58	2.28	2.56	1.98
6	4.80	3.59	2.90	3.00	3.20	2.65	2.44	2.17	1.91	1.45
7	4.05	3.20	3.07	2.54	3.12	2.42	2.42	2.4	1.95	1.35
8	4.58	3.37	2.96	2.98	2.80	2.81	2.08	2.34	1.61	1.55
9	3.40	3.91	3.08	2.77	2.79	2.64	2.17	2.25	1.73	1.33

Table C.1

Bit Error Rate as a Function of the Variation for a Processing Gain of 20

The one result which requires to be included in the case where the variation is increased to 100% providing the resulting BER of 28.69% or 0.2869.

	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0	1.900	0.520	0.415	0.491	0.366		0.296	0.337	0.225	
1	1.223	0.479	0.484	0.447	0.380		0.324		0.227	
2	0.995	0.543	0.500	0.430	0.298		0.313		0.187	
3	0.666	0.518	0.493	0.419	0.398	0.388	0.314		0.212	
4	0.635	0.561	0.507	0.358	0.351	0.443	0.302		0.190	
5	0.754	0.481	0.412	0.385		0.357	0.335		0.144	
6	0.600	0.574	0.417	0.473			0.335			
7	0.526	0.381	0.403	0.365			0.313			
8	0.499	0.427	0.417	0.455						
9	0.575	0.457	0.474	0.333						

Table C.2

Bit Error Rate as a Function of the Variation for a Processing Gain of 40

C.2 Various Codes BER

Tables C.3 and C.4 in this section compares the probability of error against the processing gain for the various coding schemes which have been modelled with a constant and varying chip duration.

Processing Gain	BER for Maximal	BER for Gold	BER for Qualcomm	BER for Bent
10	0.143466	0.128173	0.145324	0.136293
20	0.064249	0.069415	0.05758	0.061037
30	0.031347	0.029636	0.03098	0.02978
40	0.0210504	0.015032	0.018773	0.019048
50	0.0084005	0.008368	0.008969	0.00798
60	0.0050735	0.00537866	0.004164	0.00482
70	0.0030562	0.00234731	0.0024349	0.002903
80	0.0018372		0.0015208	0.001745
90	0.0008297		0.0009764	
100			0.0004122	

Table C.3

Bit Error Rate as a Function of the Spreading Code Scheme with a Constant Chip Duration

Processing Gain	BER for Maximal	BER for Gold	BER for Qualcomm	BER for Bent
10	0.12	0.103272	0.1175	0.114
20	0.034	0.0359303	0.03152	0.0323
30	0.016	0.0144348	0.01138	0.0152
40	0.011	0.00446863	0.0049	0.00665
50	0.00225961	0.00179619	0.00207	0.002147
60	0.00074417	0.00069186		0.000707
70		0.00028201		
80		0.00016934		
90				
100				

Table C.4

Bit Error Rate as a Function of the Coding Scheme with a Variation of 0.2

C.3 Various Modulation Schemes BER

Tables C.5 and C.6 in this section compares the probability of error against the processing gain for the various modulation schemes which have been modelled with a constant and varying chip duration.

Processing Gain	BER for BPSK	BER for QPSK	BER for MSK
10	0.145324	0.18264	0.188785
20	0.05758	0.05855	0.065372
30	0.03098	0.03404	0.030158
40	0.018773	0.01567	0.019668
50	0.008969	0.008872	0.008872
60	0.004164	0.004358	0.0045346
70	0.0024349	0.003102	
80	0.0015208	0.001362	
90	0.0009764		
100	0.0004122		

Table C.5

Bit Error Rate as a Function of the Modulation Scheme with a Constant Chip Duration

Processing Gain	BER for BPSK	BER for QPSK	BER for MSK
10	0.117579	0.095373	0.097022
20	0.03152	0.03711	0.037118
30	0.01138	0.014033	0.013308
40	0.004916	0.005044	0.0051386
50	0.00207	0.002003	0.0017839
60	0.000723	0.000806	0.0006896
70	0.000268	0.0003432	
80			
90			
100			

Table C.6

Bit Error Rate as a Function of the Modulation Scheme with a Variation of 0.2

The following table show the BER for BPSK when a variation of 0.1 is used.

Processing Gain	Variations
10	0.111849
20	0.0349965
30	0.0166365
40	0.0051391
50	0.0017174
60	0.0010211
70	0.0003766
80	
90	
100	

Table C.7

Bit Error Rate as a Function of the Processing Gain with a Variation of 0.1

Table C.8 shows the probability of error against the number of users within the system for a BPSK signal for constant chip duration and variation at 0.2. The processing gain in this set of simulations was set to 50 and the simulation was set to run until 101 errors had been detected.

Users	Constant Duration	Variations
10	0.008567	0.002107
20	0.048118	0.022835
30	0.090909	0.050881
40	0.120813	0.08354
50	0.150972	0.102227
60	0.174671	0.11963
70	0.20404	0.148529
80	0.236534	0.151424
90	0.259065	0.181982
100	0.291908	0.197266

Table C.8

Bit Error Rate as a Function of the Number of Users

C.4 Walsh Coding BER

The next table indicates the use of Walsh orthogonal coding, using the same set of parameters as used in section C.2.

Processing Gain	Constant Duration	Variations
10	0.00043819	0.00058628
64	0 in 327887 $P_e \leq 0.0000030498$	0 in 142415 $P_e \leq 0.00000702173$

Table C.9
Orthogonal Coding Bit Error Rate
with and without Varying Chip Duration

C.5 Gaussian Noise BER

The next table indicates the error probability when gaussian noise is introduced into the system. In this system there is only one communicating user with the base station and the gaussian noise amplitude stated in table C.10 is the multiplication of this users amplitude.

Gaussian Noise Amplitude	Constant Duration	Variations
10		
20		
30	0.0014755	0.017288
40	0.0126345	0.013190
50	0.030569	0.037560
60	0.0601907	0.059202
70	0.0853762	0.095825
80	0.1446	0.140278
90	0.2036	0.191651
100	0.1877	0.165303

Table C.10
Bit Error Rate as a Function of the Gaussian Noise
Induced into the System

C.6 Concurrent Users

Table C.11 indicates the results from having 10 users in the system and varying the number who are transmitting using a varying chip rate. The error probability in each situation is taken for both the varying and constant chip users. The system processing gain is set to 20 and the variation is 0.2.

Users	Constant	Variable
0	-	0.03152
1	0.0726	0.0454137
2	0.0713	0.0475966
3	0.0649	0.0518481
4	0.0609	0.0589953
5	0.054474	0.0618494
6	0.049582	0.065627
7	0.0684	0.0772171
8	0.03988	0.0684282
9	0.04724	0.0813
10	0.05758	-

Table C.11

Bit Error Rate as a Function the Number of Users Transmitting Using a Constant Chip Duration

C.7 Amplitude Variations BER

In the tables C.12 to C.14 the figures at the top of the column represents the two amplitude values, the larger being the sparse amplitude value. Table C.12 shows the constant duration simulation results for a system with sparse amplitudes of from 2 to 5.

Processing Gain	1,2	1,3	1,4	1,5
30	0.0375	0.0377219	0.0436644	0.0469613
40	0.0123367	0.0216285	0.0350998	0.0422185
50	0.00874936	0.0190583	0.0292767	0.0259806
60	0.00688724	0.0118138	0.0223586	0.0244957
70	0.0052954	0.00960271	0.0163148	0.018273
80	0.00266988	0.00702286	0.0160327	0.0182143
90	0.00745505	0.0278384	0.0420445	0.0944444
100	0.00146961	0.00611218	0.0128561	0.0156298

Table C.12

Various Amplitudes as a Function of the Processing Gain

Table C.13 shows the error probability for a system with both amplitude and chip duration variations.

Processing Gain	1,2	1,3	1,4	1,5
30	0.0145548	0.0154125	0.0153893	0.0145134
40	0.00573743	0.00626305	0.00830348	0.00612245
50	0.00170597	0.00173357	0.00167241	0.00137344
60	0.00071342	0.00055068	0.00080528	0.00076000
70	0.00028917	0.00025848	0.00023665	0.00029451
80				
90				
100				

Table C.13

Various Amplitudes as a Function of the Processing Gain with Varying Chip Duration

Tables C.14 shows the constant and varying chip duration simulation results for a systems with sparse amplitudes of from 10 and 100.

Processing Gain	1,10 (Constant)	1,10 (Variable)	1,100 (Constant)	1,100 (Variable)
30	0.0658915	0.0148992	0.0776256	0.0147399
40	0.0469613	0.008547	0.0630408	0.0087105
50	0.0447368	0.00134949	0.0544872	0.00119106
60	0.057888	0.000215	0.0688259	0.000252
70	0.0400	0.0000402	0.0605701	0.0000396
80	0.0461121	0.0000157	0.0686406	0.0000134
90	0.0363766		0.01875	
100			0.068	

Table C.14

Various Amplitudes as a Function of the Processing Gain with Varying Chip Duration

C.8 Amplitude Variations with Gaussian Noise BER

Table C.15 indicates the error probability when gaussian noise is introduced into the sparse amplitude system. In this system there is only one communicating user with the base station and the gaussian noise amplitude stated is the multiplication of this users amplitude.

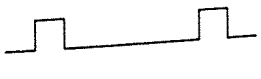
Gaussian Noise Amplitude	Amplitude Variations	Both Variations
10		
20	0.046	0.022
30	0.138	0.098
40	0.207	0.136
50	0.25	0.202
60	0.29	0.256
70	0.31	0.29
80	0.34	0.36
90	0.35	0.40
100	0.36	0.43

Table C.15

Bit Error Rate as a Function of the Gaussian Noise Induced into the System for Amplitude Variations

Appendix D

Fourier Transforms

	Description	Function		Transform
1	Definition	$g(t)$	$\Leftrightarrow$	$G(f)=\int_{-\infty}^{\infty} g(t)e^{-j2\pi ft}dt$
2	Scaling	$g\left(\frac{t}{T}\right)$	$\Leftrightarrow$	$\lvert T\rvert.e^{-j2\pi fT}$
3	Time Shift	$g(t-T)$	$\Leftrightarrow$	$G(f).e^{-j2\pi fT}$
4	Frequency Shift	$g(t).e^{-j2\pi fT}$	$\Leftrightarrow$	$G(f-F)$
5	Multiplication	$g(t).h(t)$	$\Leftrightarrow$	$G(f)\otimes H(f)$
6	Rectangular Function	$rect(t)$	$\Leftrightarrow$	$Sa(f)$
7	Sinc Function	$Sa(t)$	$\Leftrightarrow$	$rect(f)$
8	Repeated function	$rep_T[g(t)]$	$\Leftrightarrow$	$\left\lvert\frac{1}{T}\right\rvert.comb_{\frac{1}{T}}[G(f)]$
9	Sampled Function	$comb_T[g(t)]$	$\Leftrightarrow$	$\left\lvert\frac{1}{T}\right\rvert.rep_{\frac{1}{T}}[G(f)]$
10	Sparse Pulse		$\Leftrightarrow$	$A.\xi.Sa(f\xi).e^{-j2\pi\xi/p}$

Appendix E

References

[Allpress92] Allpress S.A. "Perforamance of Internal Diversity Architectures in Urban Service Areas with Cellular DS-CDMA Systems" IEE Colloquium on Spread Spectrum techniques for radio communication systems London 4th June 1992

[Berlekamp80] Berlekamp E.R. "The Technology of Error-Correcting Codes" *Proceedings of the IEEE Vol. 68, No 5* IEEE pp564-592 May-80

[Bernstein89] Bernstein G.M., Lieberman M.A. "Secure Random Number Generation Using Chaotic Circuits" *MILCOM '89: IEEE Military Communications Conference Part 3* IEEE pp640-644 15/10/89

[Bottomley93] Bottomley G.E. "Signature Sequence Selection in a CDMA System with Orthogonal Coding" *IEEE Trans. on Vehicular Technology Vol. 42, No. 1* IEEE pp62-68 Feb-93

[Bozward93/1] D.V. Bozward, R.L. Brewster *"Efficient Optical Code Division Multiple Access Local Area Networks"* European Optical Conference Berlin 6-9th April 1993

[Bozward93/2] D.V. Bozward, R.L. Brewster *"Techniques to Improve the Capacity of a DS-CDMA System"* IEE Colloquium on Spread Spectrum techniques for radio communication systems London 27th April 1993

[Bozward94] D.V. Bozward, R.L. Brewster *"Variable Chip Duration Spread Spectrum Modulation"* IEE Colloquium on Spread Spectrum techniques for radio communication systems London 15th April 1994

[Bozward95] D.V. Bozward, R.L. Brewster *"Investigation of Intra-cellular Toriodal Sectorisation in a DS-CDMA Cellular System"* Fifth IEE Conference on Telecommunications Brighton 26-29th March 1995

[Brady68] Brady P.T. "A Statistical Analysis of On-Off Patterns in 16 Conversations" *The Bell System Technical Journal* Bell pp73-90 Jan-68

[Burr93] Burr A.G. "Bounds on Spectral Efficiency of CDMA and FDMA/TDMA in a Cellular System" *IEE Colloquium on Spread Spectrum Techniques for Radio Communications systems* IEE pp12/1-12/6 Apr-93

[Chapman91] Chapman R.A., Norman D.M., Zahirhak D.R. "Classification of Correlation Signatures of Spread Spectrum Signals using Neutral Networks" *Proceedings of the IEEE 1991National Aerospace and Electronics Conference* IEEE pp485-491 May-91

[Cheng90] Cheng U., Hurd W.J., Statman J.I. "Spread Spectrum Code Acquisition in the Presence of Doppler Shift and Data Modulation" *IEEE Transactions on Communications Vol. 38 No. 2* IEEE pp241-250 Feb-90

[Chia92] Chia S. "The Universal Mobile Telecommunication System" *IEEE Communications Magazine* IEEE pp54-62 Dec-92

[Chung89] Chung F.R.K., Salehi J.A., Wei V.K. "Optical Orthogonal Codes: Design, Analysis, and Applications" *IEEE Trans. on Information Theory Vol. 35, No.3* IEEE pp595-604 May-89

[Cooper90] Cooper G.R., Martin R.D. "Detection and Identification of Multiple Spread Spectrum Signals" *MILCOM '90: IEEE Military Communications Conference* IEEE pp999-1003 30/09/90

[Davenport58] Davenport W.B., Root W.L. Random Signals and Noise McGraw-Hill 1958

[Dixon76] Dixon R.C. Spread Spectrum Systems John Wiley & Sons 1976

[Dixon94] Dixon R.C. Spread Spectrum Systems 3rd Ed. John Wiley & Sons 1994

[Elhakeem92] Elhakeem A.K., Rahman M.A., Balasubramanian P., Le-Hgoc T. "Modified SUGAR/DS: A New CDMA Scheme" *IEEE Journal on selected Areas in Communications Vol 10 No 4* IEEE pp690-704 May-92

[Fair92] Fair I.J., Wang Q., Bhargava V.K. "A New Zero-Acquisition Time Spreading/Despreading Technique for Spread Spectrum Communication Systems" *1992 IEEE International Conference on Selected Topics in Wireless Communications* IEEE pp219-222 Jun-92

[Falciasecca92] Falciasecca G., Caini C., Missiroli M., Riva G. "A General Approach to Spectral Efficiency Evaluation for High capacity Mobile Radio Systems in Different Scenarios" *42nd IEEE Vehicular Technology Society Conference* IEEE pp1034-1037 May-92

[Fuxjaeger90] Fuxjaeger A.W., Iltis R.A. "Adaptive Parameter Estimation using Parallel Kalman Filtering for Spread Spectrum Code and Doppler Tracking" *ASILOMAR Conference on Signals, Systems and Computers (Part 1)* IEEE pp378-382 Nov-90

[Gilhousen91] Gilhousen K.S., Jacobs I.M., Padovani R., Viterbi A.J., Weaver L. A., Wheatley C. E. "On the Capacity of a Cellular CDMA System" *IEEE Transactions on Vehicular Technology Vol.40 No.2* IEEE pp303-312 May-91

[Goiser90] Goiser A.M.J., Sust M.K "Adaptive Interference Rejection for Non-Coherent Digital Direct Sequence Spread Spectrum Receivers" *GLOBCOM'90 IEEE Global Telecommunications Conference* IEEE pp307.7.1 Dec-90

[Gold67] Gold R. "Optimal Binary Sequences for Spread Spectrum Multiplexing" *IEEE Transactions on Information Theory* IEEE Oct-67

[Goldberg88] Goldberg S.H., Iltis R.A. "Joint Channel Equalisation and Interference Rejection Using a Modified Constant Modulus Algorithm" *IEEE Military Communications Conference* IEEE pp97-101 23-26 Oct-88

[**Graziano86**] Graziano V. "Comparison of Ericsson 3 Sector vs Motorola 6 Sector" Motorola Inc. Sep-86

[**Graziano89**] Graziano V. "Increasing Analogue Capacity" Motorola Inc. 1986

[**Grob90**] Grob U., Welti A.L., Zollinger E., Kung R., Kaufmann H. "Microcellular Direct Sequence Spread Spectrum Radio System Using N-path RAKE Receiver" *IEEE Journal on selected Areas in Communications Vol 8 No 5* IEEE pp772-779 Jun-90

[**Groth71**] Groth E.J. "Generation of Binary Sequences with Controllable Complexity" *IEEE Trans. on Information Theory Vol. 17, No.3* IEEE pp288-296 May-71

[**Hammuda88**] Hammuda H., McGeehan J., Bateman A. "Spectral Efficiency of Cellular Land Mobile Radio Systems" *IEEE Veh. Tech. Conf. 38th* IEEE pp616-622 Sep-88

[**Holmes82**] Holmes J.K. Coherent Spread Spectrum Systems John Wiley 1982

[**Huang92**] Huang C. "Computer Simulation of a Direct Sequence Spread Spectrum Cellular Radio Architecture" *IEEE Trans. on Vehicular Technology Vol. 41, No. 4* IEEE pp545-550 Nov-92

[**Hulbert92**] Hulbert A.P., Goodwin R.J. "Factors Affecting Spectral Efficiency in a CDMA Cellular Network" *IEE Colloquium on Spread Spectrum Techniques for Radio Communication Systems* IEE 1992

[**Hulbert93**] Hulbert A.P. " Myths and Realities of Power Control" *IEE Colloquium on Spread Spectrum Techniques for Radio Communications systems* IEE pp7/1-12/4 Apr-93

[**Jovanovic88**] Jovanovic V.M. " Analysis of Strategies for Serial-Search Spread-Spectrum Code Acquisition-Direct Approach" *IEEE Transactions on Communications Vol. 36 No 11* IEEE pp1208-1220 Nov-88

[**Kasami71**] Kasami T. " The Weight Enumerators for Several Classes of Subcodes of the 2nd Order Binary Reed-Muller Codes" *Information and Control 18* pp369-394 1971

[**Key76**] Key E.L. "An Analysis of the Structure and Complexity of Nonlinear Binary Sequence Generators" *IEEE Trans. on Information Theory Vol. 22, No.6* IEEE pp732-736 Nov-76

[**Kohno90**] Kohno R., Imai H., Hatori M., Pasupathy S. "Combination of an Adaptive Array Antenna and a Chancellor of Interference for Direct-Sequence Spread-Spectrum Multiple Access" *IEEE Journal on selected Areas in Communications Vol 8 No 4* IEEE pp675-682 May-90

[**Kumar91**] Kumar P.V., Moreno O. "Prime-Phase Sequences with Periodic Correlation Properties Better Than Binary Sequences" *IEEE Trans. on Information Theory Vol. 37, No.3* IEEE pp603-616 May-91

[**Lam92**] Lam A.W., Ozluturk F.M. "Performance Bounds for DS/CDMA Communications with Complex Signature Sequences" *IEEE Transactions on Communications Vol. 40 No. 10* IEEE pp1607-1614 Oct-92

[**Lee92**] Lee J.H. "Capacity Increase of a CDMA Cellular System by Using Directional Antennas on Mobile Stations" *42nd IEEE Vehicular Technology Society Conference* IEEE pp993-995 May-92

[**Lehnert87**] Lehnert J.S., Pursley M.B. "Multipath Diversity Reception of Spread Spectrum Multiple Access Communications" *IEEE Transactions on Communications Vol. 31 No. 3* IEEE pp1189-1198 Nov-87

[**Lunayach83**] Lunayach R.S. "Performance of a Direct Sequence Spread-Spectrum System with Long Period and Short Period Code Sequences" *IEEE Transactions on Communications Vol. 31 No. 3* IEEE pp412-419 Mar-83

[**Macario92**] Macario R.C.V Personal and Mobile Radio Systems IEE 1991

[**Mann68**] Mann H.B. Error Correcting Codes Wiley and Son pp165-228 1968

[**Marubayashi92**] Marubayashi G. "Recent Research and Development Activities on Spread Spectrum" *Electronics and Communications in Japan* pp54-61 May-92

[**Miller92**] Miller J.E., Miller S.L. "Coded M-ary Orthogonal Signal Sets in DS-CDMA Systems with Multi-User Interference" *South EASTCON'92* IEEE pp295-298 Apr-92

[**Milstein91**] Milstein L.B., Schilling D.L., Pickholtz R.L. "An Experimental CDMA Personal Communication Network" *10th Annual International Phoenix Conference* IEEE pp425 Mar-91

[**Murota81**] Murota K., Hirade K. "GMSK Modulation for Digital Mobile Radio Telephony" *IEEE Transactions on Communications Vol. 29 No. 7* IEEE pp1044-1050 Jul-81

[**Newson93**] Newson P. "The Effect of Power Control on the Downlink Capacity of a Direct Sequence CDMA System for Mobile Radio" *IEE Colloquium on Spread Spectrum Techniques for Radio Communications systems* IEE pp1/1-1/9 1993

[**Numerical-C**] Press W.H. Numerical Recipes in C Cambridge University Press 1992

[**Nyquist28**] Nyquist N. "Certain Topics in Telegraph Transmission Theory" *Trans. AIEE, vol. 47* pp617-644 Apr-28

[**O'Farrell91**] O'Farrell T. "New Signature Code Sequence Design for CDMA Systems" *Electronics Letters Vol. 27 No. 4* IEE pp371-373 Feb-91

[**Olsen82**] Olsen J.D., Scholtz R.A., Welch L.R. "Bent Function Sequences" *IEEE Trans. on Information Theory Vol. 28, No.6* IEEE pp858-864 Nov-82

[**PAConsulting88**] P.A. Consulting Group The Pan European Cellular Communications market up to the Year 2000 - Management Summary Sep-88

[**Pahlavan90**] Pahlavan K., Chase M. "Spread-Spectrum Multiple-Access Performance of Orthogonal Codes for Indoor Radio Communications" *IEEE Transactions on Communications Vol. 38 No. 5* IEEE pp575-577 May-90

[**Pasupathy79**] Pasupathy S. "Minimum Shift Keying: A Spectrally Efficient Modulation" *IEEE Communications Magazine* IEEE pp14-22 Jul-79

[**Palsule86**] Palsule V.S., Lal P.M.C., Ravi K.V. " Synchronization of Pseudo Noise Spread Spectrum Signals " *Journal of the Institution of Electronics and Telecommunication Engineers Vol 32(3)* pp104-108 Jun-86

[**Peterson81**] Peterson W.W., Weldon E.J Error-Correcting Codes 2nd Ed. MIT Press 1981

[**Petrovic91**] Petrovic R., Holmes S. "CDMA Techniques in Optical Fibre LANs" *Journal of Optical Communications Vol 12 No.3* pp101-106 Sep-91

[**Pickholtz82**] Pickholtz R.L., Schilling D.L., Milstein L.B. "Theory of Spread-Spectrum Communications- A Tutorial" *IEEE Transactions on Communications Vol.30 No.5* IEEE pp855-884 May-82

[**Pickholtz91**] Pickholtz R.L., Milstein L.B., Schilling D.L. "Spread Spectrum for Mobile Communications" *IEEE Transactions on Vehicular Technology Vol.40 No.2* IEEE pp313-322 May-91

[**Polydoros81**] Polydoros A. "Rapid Acquisition Techniques for Direct Sequence Spread spectrum Systems using an Analogue Detector" *Proceedings of the National Telecommunications Conference-New Orleans L.A.* ppA7.1.1-A7.1.5 Dec-81

[**Polydoros84**] Polydoros A., Weber C.L. "A Unified Approach to Serial Search Spread Spectrum Code Acquisition-Part 1: General Theory" *IEEE Transactions on Communications Vol. 32 No. 5* IEEE pp542-549 May-84

[**Poularikas91**] Poularikas A.D., Seely S. Signals and Systems 2nd Ed. PWS-Kent 1991

[Qualcomm92/1] Qualcomm "Proposed EIA/TIA Interim Standard" *Wideband Spread Spectrum Digital Cellular System Dual-Mode Mobile Station - Base Station Compatibility Standard* Apr-92

[Qualcomm92/2] Qualcomm An Overview of the Application of Code Division Multiple Access (CDMA) to Digital Cellular Systems and Personal Cellular Networks May-92

[Reed88] Reed D.E., Wickert M.A. "Spread Spectrum Signals with Low Probability of Chip Rate Detection" *MILCOM '88: 21st Century Military Communications - What's Possible* IEEE pp437-441 23/10/88

[Rothaus93] Rothaus O.S. "Modified Gold Codes" *IEEE Transactions on Information Theory Vol. 39 n2* IEEE pp654-656 Mar-93

[Rubinstein81] Rubinstein R.Y. Simulation and the Monte Carlo Method John Wiley & Sons 1981

[Rustad90] Rustad J.E., Skaug R., Aasen A. "New Radio Networks for Tactical Communication" *IEEE Journal on selected Areas in Communications Vol 8 No 5* IEEE pp713-727 1990

[Salmasi91] Salmasi A., Gilhousen K.S. "On The System Design Aspects of Code Division Multiple Access (CDMA) Applied to Digital Cellular and Personal Communications" *IEEE Veh. Tech. Conf. 41st* IEEE pp57-62 May-91

[Shaft77] Shaft P.D. "Low-Rate Convolutional Code Applications in Spread-Spectrum Communications" *IEEE Transactions on Communications Vol. 25 No. 8* IEEE pp815-821 Aug-77

[Shannon49/1] Shannon C.E. "A Mathematical Theory of Communication" *Bell System Technology Journal Vol 27* pp379-423 Jul-49

[Shannon49/2] Shannon C.E. "Communication in the Presence of Noise" *IRE Vol 37* IRE pp10-21 Jan-49

[Simpson93] Simpson F., Holtzman J.M. "Direct Sequence CDMA Power Control, Interleaving, and Coding" *IEEE Journal on selected Areas in Communications Vol 11 No 7* IEEE pp1085-1095 Sep-93

[Skaug80] Skaug R. "Numerical Evaluation of the Non-periodic Autocorrelation Parameter for Optimal Phases of Maximal Length Sequences" *IEE Proceedings Part F Vol. 127* IEE pp230-237 Jun-80

[Skaug85] Skaug R. Hjelmstad J.F. Spread Spectrum in Communication Peter Peregrinus 1985

[Smirnov90] Smirnov N.I., Gorgadze S.F. "Energy Spectra of Noise-Like Signals of Different Types" *Soviet Journal of Communications Technology and Electronics Vol. 35 No.14* pp105-115 1990

[Snytkin91] Snytkin I.I. "Adaptive Communications Systems Employing Spread-Spectrum Signals Based on Non-Linear Recurrent Sequences of variable Length" *Soviet Journal of Communications Technology and Electronics Vol. 46 No.3* pp161-162 1991

[Steele93] Steele R., Williams J.E.B. "Third Generation PCN and the Intelligent Multimode Mobile Portable" *Electronics and Communication Engineering Journal* IEE pp147-156 Jun-93

[Suehiro88] Suehiro N., Hatori M. "Modulatable Orthogonal Sequences and Their Application to SSMA Systems" *IEEE Trans. on Information Theory Vol. 34, No.1* IEEE pp93-100 Jan-88

[Tachikana91] Tachikana S., Marubayashi G. "Spectral Efficiency of M-ary Spread Spectrum Multiple Access Communication System" *Electronics and Communications in Japan Part 3 : Fundamental Electronic Science Vol. 74 No. 5* pp65-77 May-91

[Torrieri92/1] Torrieri D.J. Principles of Secure Communication Systems 2nd Ed. Artech House 1992

[Torrieri92/2] Torrieri D.J. "Performance of Direct-Sequence Systems with Long Pseudonoise Sequences" *IEEE Journal on selected Areas in Communications Vol 10 No 4* IEEE pp771-781 May-92

[Turin84] Turin G.L. "The Effects of Multipath and Fading on the Performance of Direct Sequence CDMA Systems" *IEEE Journal on selected Areas in Communications Vol 2 No 4* IEEE pp597-603 Jul-84

[Tuttlebee92] Tuttlebee W.H.W. "Cordless Personal Communications" *IEEE Communications Magazine* IEEE pp42-53 Dec-92

[Viterbi79] Viterbi A.J. "Spread Spectrum Communications Myths and Realities" *IEEE Communication Magazine* IEEE pp11-18 May-79

[Viterbi88] Viterbi A.J. "Spread Spectrum Communication: Convergence of Theory and practice" *IEEE 1988 Int. Symp. Inf. Theory Abstracts of Papers Vol 25 No 13* IEEE Jun-88

[Viterbi90] Viterbi A.J. "Very Low Rate Convolutional Codes for Maximum Theoretical Performance of Spread-Spectrum Multiple Access Channels" *IEEE Journal on selected Areas in Communications Vol 8 No 4* IEEE pp641-649 May-90

[Viterbi92] Viterbi A.J, Padovani R. "Implications of Mobile Cellular CDMA" *IEEE Communications Magazine* IEEE pp38-41 Dec-92

[Viterbi93] Viterbi A.M., Viterbi A.J. "Erlang Capacity of a Power Controlled CDMA System" *IEEE Journal on selected Areas in Communications Vol 11 No 6* IEEE pp892-899 Sep-93

[Wan92] Wan Z., Feher K. "Improved Efficiency CDMA by Constant Envelope SQAM" *42nd IEEE Vehicular Technology Society Conference* IEEE pp51-54 May-92

[Wang92] Wang Q., Acres J.G. "Capacity Evaluation of Cellular CDMA" *International Conference on Selected Topics in Wireless Communications* IEEE pp203-210 Jun-92

[Webb92] Webb W.T. "Modulation Methods for PCNs" *IEEE Communications Magazine* IEEE pp90-95 Dec-92

[Welch74] Welch L.R. "Lower Bounds on the Maximum Cross-correlation of Signals" *IEEE Transactions on Information Theory Vol. 20* IEEE pp397-399 May-74

[Widrow85] Widrow B., Stearns S.D. Adaptive Signal Processing Prentice-Hall 1985

[Wiggert88] Wiggert Codes for Error Control and Synchronisation Artech House 1988

[Williams86] Williams C.S. Designing Digital Filters Prentice-Hall 1986

[Yarlagadda89] Yarlagadda R., Hershey J.E. "Analysis and Synthesis of Bent Sequences" *IEE Proceedings Part E Vol. 136* IEE pp112-123 Mar-89

[Yuen79] Yuen C.K., Fraser D. Digital Spectral Analysis Pitman 1979

[Ziemer85] Ziemer R.E., Peterson R.L. Digital Communications and Spread Spectrum Systems Macmillian 1985

[Ziemer92] Ziemer R.E., Peterson R.L. Introduction to Digital Communication Maxwell Macmillan 1992

Appendix F

Publications

[Bozward95] D.V. Bozward, R.L. Brewster *"Investigation of Intra-cellular Toriodal Sectorisation in a DS-CDMA Cellular System"* Fifth IEE Conference on Telecommunications Brighton 26-29th March 1995

[Bozward94] D.V. Bozward, R.L. Brewster *"Variable Chip Duration Spread Spectrum Modulation"* IEE Colloquium on Spread Spectrum techniques for radio communication systems London 15th April 1994

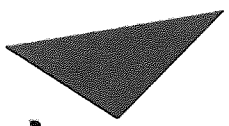
[Bozward93/2] D.V. Bozward, R.L. Brewster *"Techniques to Improve the Capacity of a DS-CDMA System"* IEE Colloquium on Spread Spectrum techniques for radio communication systems London 27th April 1993

[Bozward93/1] D.V. Bozward, R.L. Brewster *"Efficient Optical Code Division Multiple Access Local Area Networks"* European Optical Conference Berlin 6-9th April 1993

Investigation of Intra-cellular Toriodal Sectorisation in a DS-CDMA Cellular System

David Bozward, Tonda Priyanto and Ron Brewster

Department of Electronic Engineering and Applied Physics
Aston University, Birmingham, U.K.



Aston University