SIMULATION OF MULTISTAGE DETECTOR FOR SPREAD-SPECTRUM APPLICATIONS

by

Viraj Kumar Bais

Project submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfilment of the requirements for the degree of

Master of Science
in
Electrical Engineering

APPROVED:

Brian D. Woerner
Dr. Brian D. Woerner, Chairman

[illegible]
Dr. J. H. Reed

[illegible]
Dr. A. Safaai-Jazi

June, 1993
Blacksburg, Virginia

c.2

LD
5655
V851
1993
B347
c.2

Simulation of Multistage Detector for Spread-Spectrum Applications

by

Viraj Kumar Bais

Dr. Brian D. Woerner, Chairman

Electrical Engineering

(ABSTRACT)

The conventional single-user correlation receiver is not optimized for use with a multiple-access channel shared by users transmitting asynchronously independent data streams by modulating a set of assigned signal waveforms with low mutual cross-correlations. Furthermore, the performance is severely degraded as the number of users in a radio-network with fixed bandwidth grows or as the relative powers of the interfering signals become large, as in the "near-far" problem. The optimum multiuser detector based on a dynamic programming algorithm has substantially better performance, but has a variable decoding delay and is computationally intensive due to the NP-hardness of the problem of computing the most likely sequence of symbols from the sufficient statistics and signal cross-correlations. Interest has grown in suboptimal demodulation schemes which exhibit a low order of complexity to ensure their practical implementation, but not exhibiting the impairment of the conventional single-user detector.

The multistage detection strategy for coherent demodulation in asynchronous code-division multiple-access systems is such a suboptimal detection scheme based on successive estimation and removal of the multiple-access interference, and requires a computational complexity per symbol which is linear in the number of users, in contrast to the exponential complexity of the optimum demodulator. This project presents simulation results for the performance of a multistage detector for varying number of interfering users, and different combinations of desired user versus interfering users' signal energies. It is shown that the performance approaches that of single-user communications as the interfering signals become stronger. The near-far problem is therefore alleviated. Results also indicate that one stage in the receiver is required to reject interference from users at each of the various power levels. This receiver could be modified for use in practical CDMA systems by designing a multistage receiver which forms successive estimates of the channel state in addition to the interference.

ACKNOWLEDGEMENTS

Many thanks are due to my advisor Professor Brian D. Woerner, who provided friendly and optimistic guidance throughout the course of this project. His suggestions, patience, and approachability encouraged me to put in my best endeavour. I also appreciate the helpful comments and advice from members of my committee: Professors J. H. Reed and A.S. Jazi.

I am deeply indebted to my parents. Their unfailing support and pride in my small achievements has always been a source of inspiration to me. I also appreciate the support of my friends with whom I have shared the anxieties and satisfactions of educational experience at Virginia Tech. This experience will be one of my cherished memories.

Table of Contents

List of Figures

List of Tables

1 INTRODUCTION

Over the recent few years the interest in wireless communications has been spectacular, and it represents the fastest growing segment of the telecommunications industry. Cellular radio systems around the world have been enjoying 33 to 50 percent growth rates. It has been projected that 50 to 75 million subscribers could be using wireless systems for various types of personal communications by mid-1990s, and that cellular and PCN systems will achieve a market penetration rate of at least 12 percent in many developed countries by the end of this century. There has been intense corporate research and development aimed at commercializing new wireless communications services called PCS (Personal Communications Services). Use of low power digital radio is proposed to be integrated into an existing local exchange network to provide universal access to a Personal Communications Network (PCN) over a large geographical area. The implementation of a PCN will be accomplished by a sizable infrastructure of compact, low powered base stations comprising a microcellular structure, with base stations mounted on lamp posts, building rooftops, or other locations where high user density is expected, and connected to the public telephone network via wired (optical fiber) or wireless (microwave) links [Cox91]. Of course, the widescale deployment of such extensive high-grade wireless telephone system will require engineering tools and techniques that allow accurate system design and propagation prediction.

The use of spread-spectrum techniques for military communications in hostile scenarios, like jamming, has been commonplace over the past decade. In the recent years, however, entirely new opportunities such as wireless LAN's, personal communication networks, and digital cellular radios have created the need for research on how spread-spectrum systems can be reoptimized for most efficient use in these nonhostile environments. Widespread use of spread-spectrum for personal wireless communications is likely in the future. In 1990, more than a dozen experimental license applications were tendered by the FCC for U.S. wireless communication systems using direct-sequence code-division multiple-access (DS-CDMA). U.S. CDMA field trials using the existing cellular spectrum in the 800-900 MHz and 1.8 GHz bands are currently underway by major cellular and personal communications companies.

The characteristics of spread-spectrum that make it advantageous for mobile communications are numerous. With the increasing need to communicate over a "fixed" available spectrum, it becomes necessary to use this spectrum more efficiently. Spread-spectrum makes effective use of the spectrum by allowing additional users to use the same band as the existing users because the pseudo-random properties of the spread-spectrum signals make them "uncorrelated" with every other spread signal using the same band, and thus difficult to demodulate by receivers other than the intended ones. The use of spread-spectrum also makes it possible to overlay low level direct-sequence waveforms on top of existing narrow-band users, and hence increase the overall spectrum capacity to a still higher level. This is especially important in the context of current unavailability of spectrum for PCN in the UHF and low microwave regions and the consequent proposal of overlaying spread-spectrum mobile radio systems on existing point-to-point microwave links in the 1850MHz-1990MHz band used by utility companies, banks, and public safety offices. CDMA systems degrade gracefully and require no demand assignment control protocol unlike narrowband solutions such as TDMA and FDMA. CDMA systems also allow for reuse of spectrum which increases the spectral efficiency dramatically. Multipath is often a fundamental limitation to system performance in mobile communications environment. Spread-spectrum systems are capable of combating multipath by the virtue of low cross-correlation among the signature waveforms for arbitrary delays. A narrowband communication link can be designed to be tolerant of multipath interference only at the cost of increased receiver complexity in the form of adaptive equalization, and may affect the ability to perform a smooth handover in cellular radio situations. Spread-spectrum is based on the use of noise-like signals, where each user has a uniquely addressable code, and therefore privacy is inherent.

The limitations of present day spread-spectrum systems include the problem of "self-jamming" and the related problem of near-far effect. The self-jamming arises from the fact that in an asynchronous CDMA network, the spreading sequences of the different users are not truly orthogonal, and hence in despreading of a given user's waveform, nonzero contributions to that user's test statistic arise from the transmissions of other users in the network. Spread-spectrum systems tend to be self-interference limited. An associated phenomenon is the near-far problem. In mobile environments, a close-in undesired transmitter can swamp a remote, desired transmission. This intolerable condition may be mitigated by using power control to reduce the transmission power of a close-in user. However, perfect power control has not been achieved in practice [Cam92]. Moreover this still does not guarantee that interfering signals from neighboring cells will not arrive with intolerable power levels, because the waveforms in different cells may experience independent fading.

Therefore it can be seen that while the use of spread-spectrum techniques offer some unique opportunities to system designers of digital cellular radios, there are issues of concern as well. Design of spreading sequences with better correlation properties and detectors robust to near-far effect are ongoing areas of research. The near-far problem is not an inherent shortcoming of DS-CDMA systems, but of the conventional single user receiver. Detection schemes resistant to the near-far problem have in fact been designed. Moreover, an improved receiver design could be substituted into existing systems to offer improved performance without any new signalling standard.

This report is organized as follows. In Chapter 2 we review the conventional, optimal, and suboptimal receiver structures for spread-spectrum systems and their near-far performance. Chapter 3 specifies the parameters used in the simulation of the suboptimal multistage receiver structure, while Chapter 4 presents the simulation results and their interpretation in the context of near-far robustness of the multistage detector and its performance at various stages. Finally, Chapter 5 concludes the report and points out the extensions possible to the simulation of this receiver.

2 RECEIVER STRUCTURES FOR SPREAD SPECTRUM

This chapter describes the binary direct-sequence spread-spectrum communications system model and the notations used subsequently. Various receiver structures used in spread-spectrum communications systems are explained and their performance under various conditions is summarized from literature. The limitations of each receiver are outlined.

2.1 Direct Sequence Spread-Spectrum Model

The block diagram shown in Figure 2.1-1 illustrates the basic elements of a spread spectrum digital communications system [Pro89].

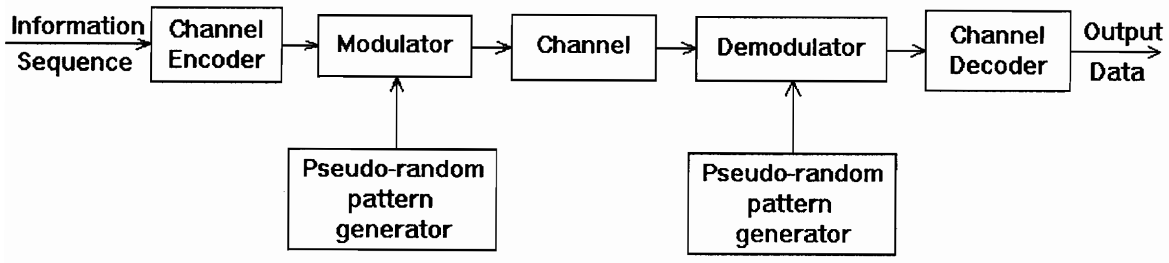


FIGURE 2.1-1 Block diagram of a spread-spectrum digital communication system

It differs from a conventional communications system in the following respect:

(i) Two identical, synchronized pseudo-random pattern generators are used, one interfacing with the modulator at the transmitting end to impress the pseudo-noise (PN) binary valued sequence on the transmitted signal, and the second interfacing with the demodulator at the receiving end to remove the PN sequence from the received signal. The generation of PN sequences for spread-spectrum applications and evaluation of their auto-correlation and cross-correlation properties is a topic that has received considerable attention in the technical literature. Maximum-length shift-register sequences (m-sequences), Gold sequences, and Kasami sequences are by far the most widely used binary PN sequences [Pur79].

(ii) Several asynchronous user signals simultaneously occupy the same channel bandwidth. These systems achieve multiple-access capability by assigning a different signature waveform to each user from a set of waveforms with low mutual cross-correlations. For code division multiple-access (CDMA) communications the primary goal is to be able to recover the information transmitted by a user from the received sum of the signals modulated by several asynchronous users by correlating the received signal with replicas of the assigned signature waveforms known to the receiver.

(iii) In applications where phase coherence between the transmitted signal and the received signal can be maintained over a time interval that is relatively long compared to the reciprocal of the transmitted signal bandwidth, PSK is the appropriate modulation. When the PN sequence generated at the modulator is used in conjunction with the PSK modulation to shift the phase of the PSK signal pseudo-randomly, the resulting modulated signal is called a *direct sequence* (DS) spread-spectrum signal. On the other hand FSK modulation is appropriate in applications where phase coherence cannot be maintained due to time-variant effects on the communications link. When used in conjunction with FSK, the PN sequence selects the frequency of the transmitted signal pseudo-randomly and the resulting signal is called a *frequency-hopped* (FH) spread spectrum signal.

In the following treatment we concentrate on binary phase-shift-keyed direct-sequence spread-spectrum multiple-access communication system over an additive white Gaussian noise channel. For the binary direct-sequence form of spread-spectrum modulation, the binary data signal $b(t)$ is given by

$$b(t) = \sum_{l=-\infty}^{+\infty} b_l \, P_T(t - lT), \qquad b_l \in \{+1, -1\} \tag{2.1 - 1}$$

and a baseband signal of the form

$$a_k(t) = \sum_{j=0}^{N-1} a_k^{(j)} \, \psi(t - jT_c), \qquad a_k^{(j)} \in \{+1, -1\} \tag{2.1 - 2}$$

as shown in Figure 2.1-2, is employed as the spectral-spreading waveform, where $P_T(t)$ is the rectangular pulse of duration T which starts at $t = 0$, $b = \{b_l\}$ is the binary data sequence to be transmitted, and for each k, $\{a_k^{(j)}\}_{j=0}^{N-1}$ is the code sequence assigned to the k^{th} transmitter.

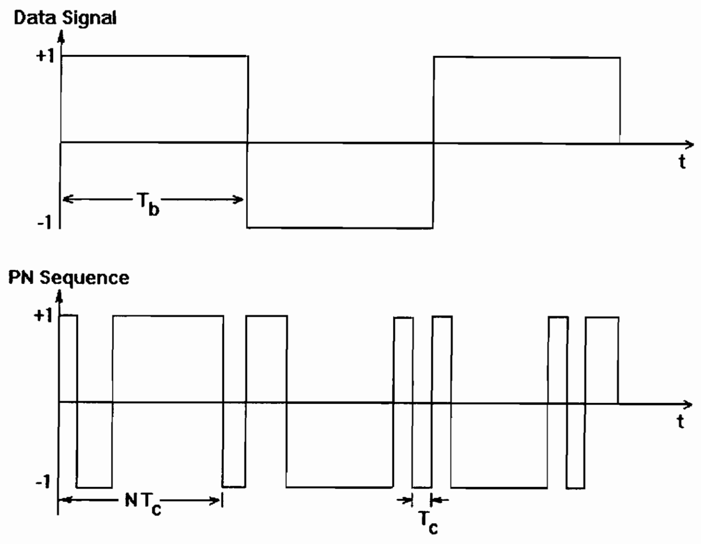


FIGURE 2.1-2 The PN and data signals used in direct-sequence spread-spectrum systems.

The time duration of the pulse, T_c, is called the chip interval and the chip waveform $\psi(\cdot)$ is usually the unit rectangular signal

$$\psi(t) \; = \; P_{T_c}(t) \; = \; \begin{cases} 1, & 0 \le t \le T_c; \\ 0, & \text{otherwise.} \end{cases} \qquad (2.1-3)$$

and is normalized to have energy T_c i.e.

$$T_c^{-1} \int_0^{T_c} \psi^2(t)\, dt \; = \; 1 \qquad (2.1-4)$$

The actual transmitted time-limited bandpass signal for the k^{th} user in a binary direct-sequence spread-spectrum system is

$$s_k(t) \; = \; A\, a_k(t)\, b_k(t)\, cos(\omega_c t + \theta_k) \; = \; \sqrt{2E_k T^{-1}}\, a_k(t)\, cos(\omega_c t + \theta_k) \qquad 1 \le k \le K \qquad (2.1-5)$$

where K is the number of simultaneous users in the system, ω_c is the carrier frequency, E_k is the energy of the k^{th} user's transmitted information bit and $\theta_k \in \{0, 2\pi\}$ is the phase angle of the k^{th} user. The number of chips per data bit, $N = T/T_c$, is the length of the signature sequence or the number of phase shifts that occurred in the transmitted signal $s_k(t)$ during a bit duration T; therefore the bandwidth of the transmitted signal is of the order of N times that of the information signal $b(t)$. In other words the bandwidth expansion factor or processing gain for the spread-spectrum system is N.

In general, the phase angles $\theta_k, 1 \le k \le K$, are not the same in a system with several independent users because individual transmitters are not phase synchronous, and furthermore, the propagation delays for the various signals from transmitter to receiver need not be the same. Thus, as illustrated in Figure 2.1-3, the received signal is given by [Pur81]

$$r(t) \; = \; n(t) + \sum_{k=1}^{K} s_k(t - \tau_k) \; = \; n(t) + \sum_{k=1}^{K} \sqrt{2E_k T^{-1}}\, a_k(t - \tau_k)\, b_k(t - \tau_k)\, cos(\omega_c t + \phi_k) \quad (2.1-6)$$

where $n(t)$ is additive white Gaussian noise (thermal noise) with two-sided spectral density $N_0/2$, τ_k is the time delay associated with the k^{th} signal, and $\phi_k \; = \; \theta_k \; - \omega_c \tau_k$ is the phase angle of the k^{th} carrier. For our purposes only the relative time delays and phase angles need be considered, and so we assume $\phi_i \; = \; 0$ and $\tau_i \; = \; 0$ in the analysis of the receiver which is matched to the i^{th} signal. Without loss of generality, we can also assume an ordering on the time delays τ_k such that $0 \le \tau_1 \le \tau_2 \cdots \le \tau_k \le T$, and, $0 \le \phi_k < 2\pi$, since we are only concerned with time delays modulo T and phase shifts modulo 2π.

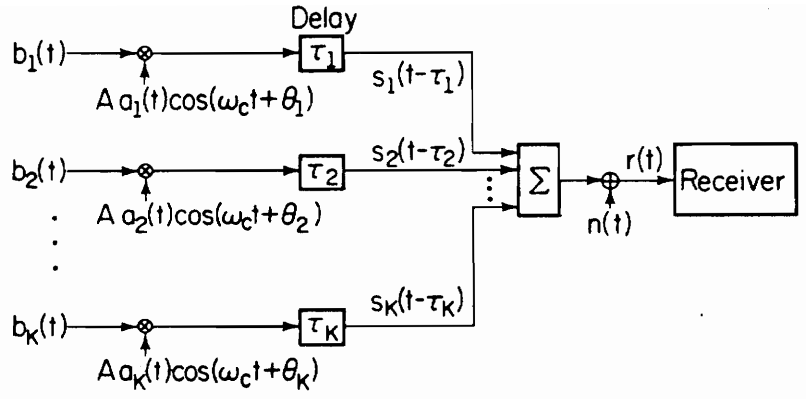


FIGURE 2.1-3 Model of a binary direct-sequence spread-spectrum communications system [Pur81]

In the following sections, the multiple-access communication system model described above is considered for demodulation and various recent attempts to derive detectors for multi-user channels are summarized.

2.2 Conventional Correlation Detector in Multiple-Access Channels

The conventional approach to multi-user demodulation, based on single-user detection, is to demodulate each user's signal as if it were the only one present. This receiver consists of a bank of filters matched to each user's signal waveform in corresponding time and phase synchronism as in Figure 2.2-1. The performance of this receiver in a spread-spectrum system is now examined.

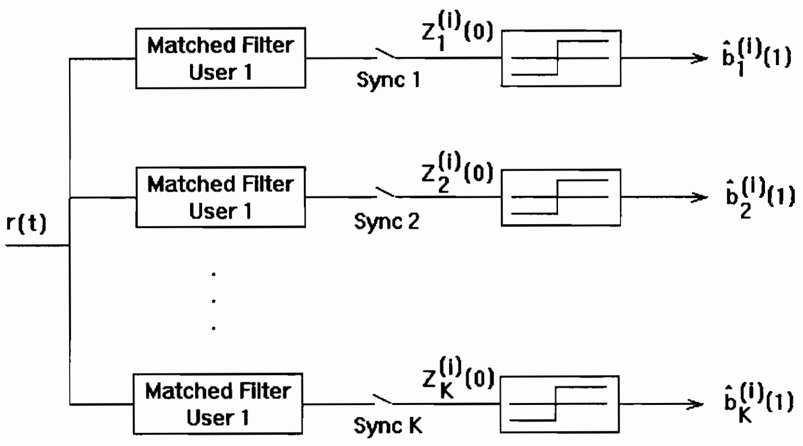


FIGURE 2.2-1 The conventional multiuser detector for the BPSK-CDMA system.

2.2.1 Sufficient Statistics for Coherent Demodulation

The decision statistic of the k^{th} user at the output of matched filter consists of a sum of the desired signal component, additive noise component, and multiple-access interference due to the cross-correlation's of the k^{th} user's signal with the signals from other users. The receiver then makes decisions by comparing the statistic to a threshold. The output of the i^{th} matched filter sampled at the end of the j^{th} time interval is given as

$$Z_i^{(j)}(0) \;=\; \int_{-\infty}^{+\infty} r(t)\, s_i(t + jT - \tau_i)\, dt \qquad (2.2-1)$$

The received signal is $r(t) \;=\; n(t) + \sum_{k=1}^{K} s_k(t - \tau_k)$ and since the users are indexed in order of increasing delays, the bit transmitted by user i during time interval j overlaps with bits transmitted during time intervals j and $(j-1)$ from users $k = i+1, \ldots, K$ and bits transmitted during the time intervals j and $(j+1)$ from users $k = 0, \ldots, i$. Therefore as in [Var90],

$$Z_i^{(j)}(0) \;=\; \eta_i^{(j)} + \sum_{k=1}^{i-1} h_{ik}(-1)\, b_k^{(j+1)} + \sum_{k=1}^{K} h_{ik}(0)\, b_k^{(j)} + \sum_{k=i+1}^{K} h_{ik}(+1)\, b_k^{(j-1)} \qquad (2.2-2)$$

where $\eta_i^{(j)}$ is the component of the statistic due to the additive channel noise, and

$$h_{ik}(j) \;=\; \int_{-\infty}^{+\infty} s_i(t - \tau_i)\, s_k(t + jT - \tau_k)\, dt \qquad (2.2-3)$$

is the signal cross-correlation of the i^{th} and k^{th} users. If $\mathbf{H}(j)$ are $K \times K$ cross-correlation matrices with $(i,k)^{th}$ element as $h_{ik}(j)$ then, since the modulating signals are time limited to $[0, T]$, $\mathbf{H}(j) = 0\; \forall |j| > 1$, and $\mathbf{H}(-j) \;=\; \mathbf{H}^T(j)$. Also $H(1)$ is an upper triangular matrix with a zero diagonal. Since $\omega_c T \;=\; 2\pi N$ is an integer multiple of 2π and $\gg 1$, the matrices $\mathbf{H}(j), j \;=\; -1, 0, +1$ can be related to the normalized baseband correlation functions as in [Ver84]

$$h_{ik}(0) \;=\; \begin{cases} T^{-1}\sqrt{E_i E_k}\, \hat{R}_{ik}(\tau_i - \tau_k)\, \cos\big((\tau_i - \tau_k)\omega_c + \theta_k - \theta_i\big), & \text{if } k \le i; \\ T^{-1}\sqrt{E_i E_k}\, \hat{R}_{ki}(\tau_k - \tau_i)\, \cos\big((\tau_k - \tau_i)\omega_c + \theta_k - \theta_i\big) & \text{if } k > i. \end{cases} \qquad (2.2-4)$$

$$h_{ik}(1) \;=\; \begin{cases} 0, & \text{if } k \le i; \\ T^{-1}\sqrt{E_i E_k}\, \hat{R}_{ik}(T + \tau_i - \tau_k)\, \cos\big((\tau_k - \tau_i)\omega_c + \theta_k - \theta_i\big) & \text{if } k > i. \end{cases} \qquad (2.2-5)$$

where

$$\hat{R}_{ik}(\tau) \;=\; \int_{\tau}^{T} a_i(t - \tau)\, a_k(t)\, dt \quad 0 \le \tau \le T \qquad (2.2-6)$$

and

$$R_{ik}(\tau) \;=\; \int_{0}^{\tau} a_i(t - \tau)\, a_k(t)\, dt \quad 0 \le \tau \le T \qquad (2.2-7)$$

are the continuous-time partial cross-correlation functions of the i^{th} and k^{th} signature sequences. For the purposes of providing more insight into the various components of the decision statistics of a conventional correlation receiver extended for multi-user environment, a similar, though scaled, version of the decision statistic has been introduced in [Pur82]. The output of the i^{th} matched filter is

$$Z_i(0) = \int_0^T r(t)\, a_i(t)\, \cos\omega_c t\, dt \tag{2.2 - 8}$$

since $\phi_i = \tau_i = 0$ as the i^{th} matched filter is synchronized with the i^{th} signal.

On substituting $r(t)$ from $(2.1-6)$, the decision statistic for the 1^{st} user at the end of j^{th} time interval, after rejecting the double frequency components of the integrand, can be written as [Pur77]

$$Z_1^{(j)}(0) = \eta_1^{(j)} + T\sqrt{P_1/2}\, b_1^{(j)} + \sum_{k=2}^{K} \sqrt{P_k/2}\, \left[b_k^{(j-1)}\, R_{k1}(\tau_k) + b_k^{(j)}\, \hat{R}_{k1}(\tau_k) \right] \cos\phi_k \tag{2.2 - 9}$$

where $P_k = E_k/T$ are the relative powers of the various users' signals at the receiver.

The first term in the decision statistic for the i^{th} user, $\eta_i = \int_0^T n(t)\, a_i(t)\, \cos\omega_c t\, dt$, is the component of the decision statistic due to additive Gaussian noise and is a random variable with Gaussian distribution with mean $E[\eta_i] = \int_0^T E[n(t)]\, a_i(t)\, \cos\omega_c t\, dt = 0$ and variance $E[\eta_i^2] = \int_0^T \int_0^T E[n(t)n(\tau)]\, a_i(t)\, a_i(\tau)\, \cos\omega_c t \cos\omega_c \tau\, dt\, d\tau = N_0 T/4$. The second term is due to the i^{th} information signal, which is to be separated out from noise and multiple-access interference. The third and the final term $\Xi_i(b,\tau,\phi) = \sum_{k\neq i} \sqrt{P_k/2}\, I_{ki}(b_k, \tau_k, \phi_k) = \sum_{k\neq i} \sqrt{P_k/2}\, [b_k^{(j-1)}\, R_{ki}(\tau_k) + b_k^{(j)}\, \hat{R}_{ki}(\tau_k)] \cos\phi_k$ is the multiple-access interference component of the received signal from the other $K-1$ signals $\{s_k(t) : 1 \leq k \leq K,\, k \neq i\}$. This interference component is unique to DS-SS multiple-access systems and is nonexistent in single-user communications.

The decision statistics in the form $(2.2-2)$ or $(2.2-9)$ are the sufficient statistics for coherent detection of a user's signal by the conventional correlation detector.

2.2.2 Limitations on the Performance of Conventional Detector

If the symbol-epoch synchronism is maintained and the assigned signature waveforms are mutually orthogonal (i.e. have zero cross-correlations), then the multiple-access interference term $\Xi_i(b,\tau,\phi)$ is zero and the channel decouples into single-user channels which means that a bank of conventional single-user matched filter receiver achieves optimum demodulation. Due to lack of symbol synchronism among the users or bandwidth limitations, orthogonality among signature sequences is not practically feasible and therefore other users' signals are not transparent to each matched filter. Under these circumstances, the above conventional correlation receiver approach to multi-user detection of asynchronous signals is nonoptimal. However this demodulation scheme is conventionally used in practice due to advantages of simplicity and expedient modification for decentralized reception. Acceptable performance from the conventional receiver is possible if two conditions are satisfied: first, the assigned signals need to have low cross-correlations $\hat{R}_{ki}(\cdot)$ and $R_{ki}(\cdot)$ for all possible relative delays between the independent data streams transmitted by the asynchronous users. This is done by choosing a signal constellation with a large bandwidth (thereby lowering the bandwidth efficiency) and choosing signature sequences with better cross-correlation properties. However as the number of users in a system with fixed bandwidth grows, severe performance degradation is observed. Second, the powers of the received signals, P_k, cannot be very dissimilar even with good (i.e. quasi orthogonal) signal constellations, as the increase in multi-user interference powers leads to increasing bit-error-rates until the conventional detector is unable to recover the messages transmitted by weak users (the *near-far* problem). The only remedy implemented in practice has been to use power control and reduce the transmission power of a close-in user so that a close-in undesired transmitter cannot swamp a remote, desired transmitter. Perfect power control cannot be achieved in practice. Moreover power control does not guarantee that interference from other users might not arrive with power levels that can be tolerated, because the desired user signal may undergo independent fading in dynamically changing topologies such as in mobile radio situations. Therefore the near-far problem and the consequent degradation in performance of the conventional receiver, even for relatively low bandwidth efficiencies, is not alleviated. This effect has been succinctly characterized in [Ver86b] in terms of asymptotic efficiency, a measure that quantifies the multiple-access limitation of a detector, and is described in Section 2.3.3

2.2.3 Performance Results for the Conventional Detector

Extensive analysis of the conventional detector for spread-spectrum systems has been undertaken over the years. The average probability of error of this detector for the i^{th} user over the bit sequences of the interfering users can be shown to be given by [Lup90]

$$\overline{P}_e = \frac{1}{2(K-1)^2} \sum_{\substack{b_k(0),b_k(-1)\\ k\neq i}} Q\left\{\frac{\sqrt{E_i} - \sum_{j\neq i}[\mathbf{H}_{ij}(0)b_j(0) + \mathbf{H}_{ij}(1)b_j(-1)]\sqrt{E_j}}{\sigma}\right\} \qquad (2.2-10)$$

The various approaches taken to evaluate this bit-error-probability are as follows:

(i) *Guassian Approximation* : When the despread multiple-access interference term $\Xi_i(b,\tau,\phi)$ is approximated as a Guassian distributed random variable, then the first and the third terms can be combined into a new Gaussian (channel noise + multiple-access interference) term, since sum of statistically independent Gaussian random variables is also a Gaussian random variable. The problem of multi-user detection of asynchronous signals therefore reduces to the detection of a single-user in presence of Gaussian noise, for which the conventional correlation detector is optimum. Under this approximation the average bit-error-rate can be shown to be [Pur77]

$$\overline{P}_e = Q\left(\sqrt{\frac{N_0}{2E_b} + \frac{K-1}{3N}}\right) \qquad (2.2-11)$$

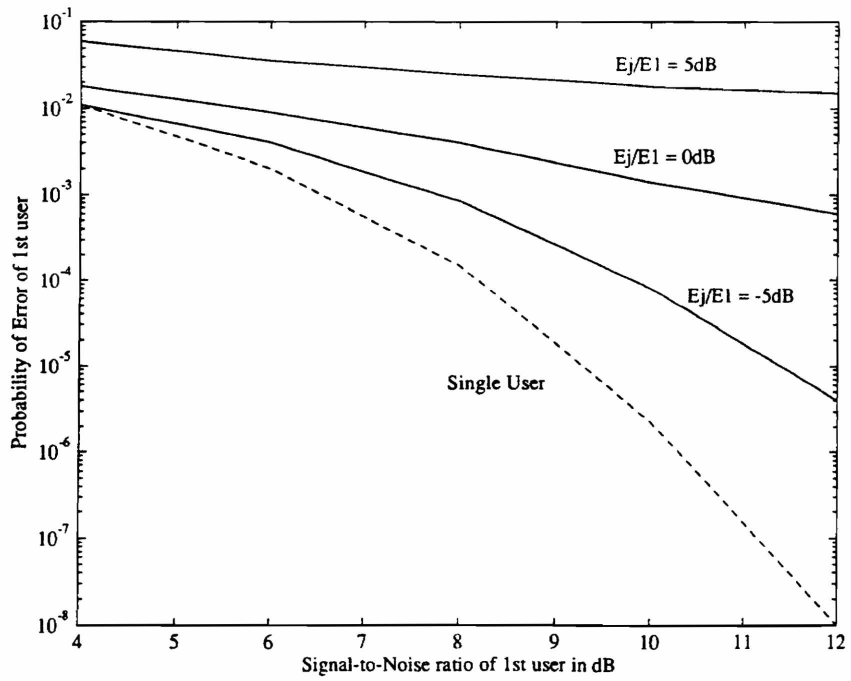


FIGURE 2.2-2 Average bit-error-probability for the conventional receiver in a three-user direct-sequence spread-spectrum system with optimal m-sequences of length 31 [Lup90]

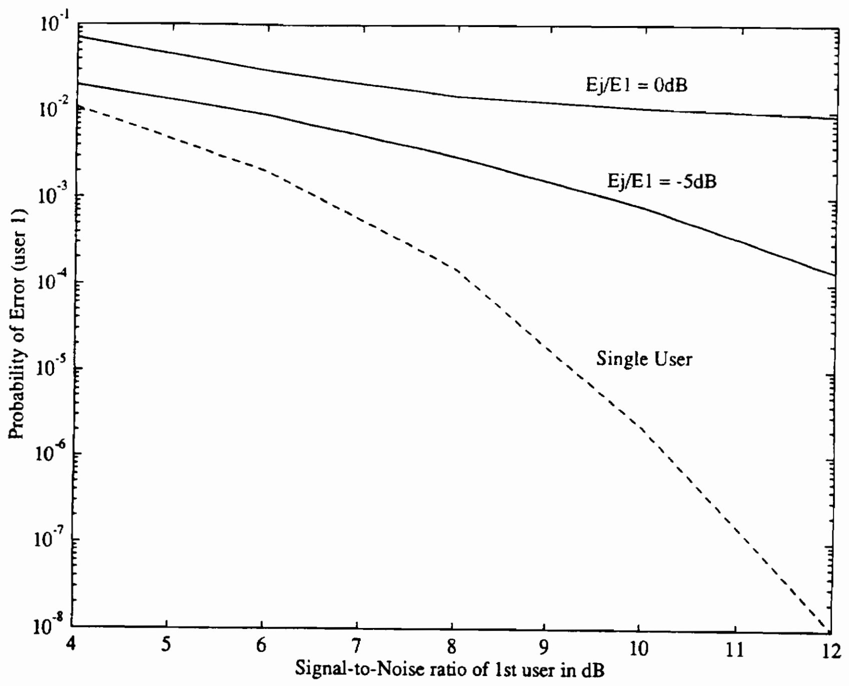


FIGURE 2.2-3 Average bit-error-probability for the conventional receiver in a six-user direct-sequence spread-spectrum system with optimal m-sequences of length 31 [Lup90]

where E_b/N_0 is the signal-to-noise ratio of the desired user, $K-1$ is the number of active interferers, and N is the length of the spreading signature sequence. This approximation is found to hold reasonably well [Leh87] when long spreading sequences are used i.e. $N \gg 1$, and when $K \gg 1$, and P_e is not too small. This latter assumption is reasonable when forward error correction is used, since P_e corresponds to the channel bit-error-rate, not the decoded bit-error-rate.

(ii) *Bounding or Approximating the Bit-Error-Rate* : Different methods for computation of bounds and approximations of the probability of error can be found in [Ger82],[Leh87] and [Pur82]. For example, in [Leh87] the interval containing all possible values of $\Xi_i(b, \tau, \phi)$ is divided into a set of subintervals. For each subinterval, an upper bound is found on the probability that the value of lies in that subinterval. The set of upper bounds corresponding to the set of subintervals forms a vector which is used to obtain an upper bound on the average probability of error. Similarly, a set of lower bounds is used to form a second vector which is used to obtain a lower bound on the average probability of error. The two bounds become tighter as the size of the vectors grow.

(iii) *Computer Simulations*: Figures 2.2-2 and 2.2-3 summarize the simulation results for the average probability of error of correlation receiver obtained in [Ger82], [Pur82]. For comparison purposes the bit-error-rate of a single-user operating in a additive Gaussian noise channel in the absence of any interfering users is included. Figure 2.2-2 shows the average error probability in a three-user direct-sequence spread-spectrum system for various power levels of desired user and interfering users' signals using the set of auto optimal m-sequences of length 31 found in [Pur79] to be optimal with respect to certain peak and mean square correlation parameters. Figure 2.2-3 repeats the same curves for $K = 6$ users. It is obvious by comparing the curves in the same figure that the conventional receiver has a strong near-far limitation as its performance deteriorates rapidly for increasing interfering users' powers, till for an energy ratio above $5dB$ it becomes practically multiple-access limited and the error probability becomes irreducible. Also the performance deteriorates whenever sufficiently many users are active even if their energies are well below the energy of the desired user, as can be seen by comparing the $0dB$ or $-5dB$ curves across the figures.

2.3 Optimum Multi-user Detector

If data demodulation is restricted to single-user detection systems, as described in the previous subsection, then the cross-correlation properties of the signal constellation carry the entire burden of complexity required to achieve a given performance level, and even with signature sequences having good cross-correlation properties the performance degradation is too severe when the number of users increases or the power of some of the interfering users is dominant. More sophisticated receivers, like the minimum error probability detector [Ver86a] are needed to achieve performance improvement.

2.3.1 Maximum Likelihood Formulation of the Multiuser Detection problem

Optimum multi-user detection of asynchronous signals is inherently a problem of sequence detection. The reason is that the observation of the complete intervals of the overlapping symbols of the other users gives additional information about the received signal in the bit interval in question. Hence one-shot approaches where the demodulation of each symbol takes into account the received signal only in the interval corresponding to that symbol, are suboptimal. The optimum decision rules for the multiple-access model with additive linearly modulated signals in additive Gaussian noise channels are now outlined and it is shown that a K-user maximum likelihood sequence detector consists of a bank of K single-user matched filters as the signal processing front end, followed by a Viterbi forward dynamic programming algorithm with $2^{(K-1)}$ states as shown in Figure 2.3-1.

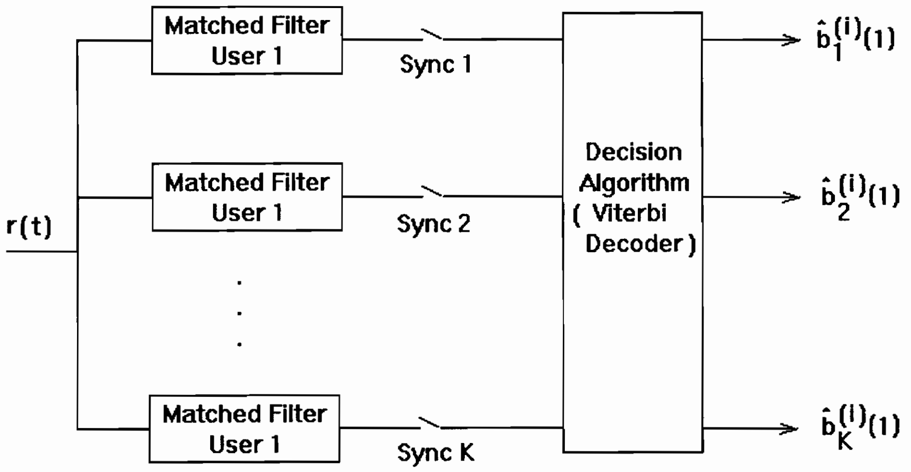


FIGURE 2.3-1 The optimum K-user detector for asynchronous multiple-access Gaussian channel [Ver86a]

The received process in a BPSK-CDMA system is given by

$$r(t) = s(t, \mathbf{B}) + n_t \qquad t \in R \tag{2.3-1}$$

where

$$s(t, \mathbf{B}) = \sum_{i=-M}^{+M} \sum_{k=1}^{K} b_k(i)\, s_k(t - iT - \tau_k) \tag{2.3-2}$$

is the sum of K user signals transmitted, $\mathbf{B}$ is the $K \times (2M+1)$ information matrix so that the $(k,i)^{th}$ element of $\mathbf{B}$, equal to $b_k(i) \in \{-1, +1\}$ is the i^{th} symbol of k^{th} user, and $(2M+1)$ is the length of the packets transmitted by all users. Since all transmitted sequences of symbols are equiprobable, the maximum likelihood sequence detector selects the sequence that maximizes the a posteriori probability

$$p[\mathbf{B}|\{r(t), t \in R\}] = \frac{p[r(t)|\mathbf{B}]\, b[\mathbf{B}]}{p[r(t), t \in R]} = C\, exp(\Omega[\mathbf{B}, r(t)]/2\sigma^2) \qquad (2.3-3)$$

or equivalently, the log-likelihood function

$$\Omega[\mathbf{B}, r(t)] = 2\int_{-\infty}^{+\infty} s(t, \mathbf{B})\, r(t)\, dt - \int_{-\infty}^{+\infty} s^2(t, \mathbf{B})\, dt \qquad (2.3-4)$$

On substituting $s(t, \mathbf{B})$ from (2.3-2), the first term on the right-hand side can be expressed as

$$\int_{-\infty}^{+\infty} s(t, \mathbf{B})\, r(t)\, dt = \sum_{i=-M}^{+M} \mathbf{b}^T(i)\, \mathbf{y}(i) \qquad (2.3-5)$$

where $y_k(i)$ denotes the output of a matched filter for the i^{th} symbol of the k^{th} user, i.e.,

$$y_k(i) = \int_{\tau_k+iT}^{\tau_k+iT+T} s_k(t - iT - \tau_k)\, r(t)\, dt \qquad (2.3-6)$$

Hence, even though $y_k(i)$ is not a sufficient statistic for the detection of $b_k(i)$, the whole sequence of outputs of the bank of K matched filters $\mathbf{y}$ is a sufficient statistic for the selection of the most likely sequence $\mathbf{B}$. This implies that the maximum-likelihood multi-user coherent detector consists of a front end of matched filters, one for each user, followed by a decision algorithm, which selects the sequence $\mathbf{B}$ that maximizes the log-likelihood function $\Omega[\mathbf{B}, r(t)]$ of $(2.3-4)$. The efficient solution of this combinatorial optimization problem is the central issue in the derivation of the optimum multiuser detector.

2.3.2 Maximization Algorithm

An exhaustive maximization of the joint a posteriori probability needs its computation for each of the $2^{(2M+1)K}$ possible values of $\mathbf{B}$. The time complexity per binary division of such an approach has the inconvenient feature of not only being dependent on the packet size $(2M+1)$, but in an exponential way. Such a brute force maximization is practically useless. In our case a significant computational gain over the exhaustive scheme can be obtained by exploiting the additive decomposibility of the log-likelihood function $\Omega[\mathbf{B}, r(t)]$. The key to this efficient maximization of $\Omega[\mathbf{B}, r(t)]$ lies in its sequential dependence on the symbols $b_k(i)$, which allows it to be put as a sum of terms that depend only on a few variables at a time. Suppose that we can find a discrete-time system $x_{i+1} = f_i(x_i, u_i)$ (with initial condition x_{i_0}, a transition-payoff function $\lambda_i(x_i, u_i)$, and a bijection between the set of transmitted sequences $b_k(i)$ and a subset of control sequences $\{u_i, i = i_0, .., i_f\}$ such that $\Omega[\mathbf{B}, r(t)] = \sum_{i=i_0}^{i_f} \lambda_i(x_i, u_i)$ subject to $x_{i+1} = f_i(x_i, u_i), x_{i_0}$, and $\mathbf{B} \leftrightarrow \{u_i, i = i_0, .., i_f\}$. Then the maximization of $\Omega[\mathbf{B}, r(t)]$ is equivalent to a discrete-time deterministic control problem with additive cost and finite input and state spaces, and can therefore be solved by a forward or backward dynamic programming algorithm. It turns out that there is no unique additive decomposition of the log-likelihood function $\Omega[\mathbf{B}, r(t)]$ into λ_i's resulting in decision algorithms with very different computational complexities. The decision algorithm suggested in [Sch79] has 4^K states and $O(8^K/K)$ time complexity per bit, while the decision algorithm in [Ver84] has 2^K states and $O(4^K/K)$ time complexity. The decomposition

$$\lambda_i(x,u) = u\left[2y_{(i \, mod \, k)}(\lfloor i/k \rfloor) - u\int_0^T s^2_{(i \, mod \, k)}(t)\, dt - 2x^T\mathbf{H}^{(i \, mod \, k)}\right] \qquad (2.3-7)$$

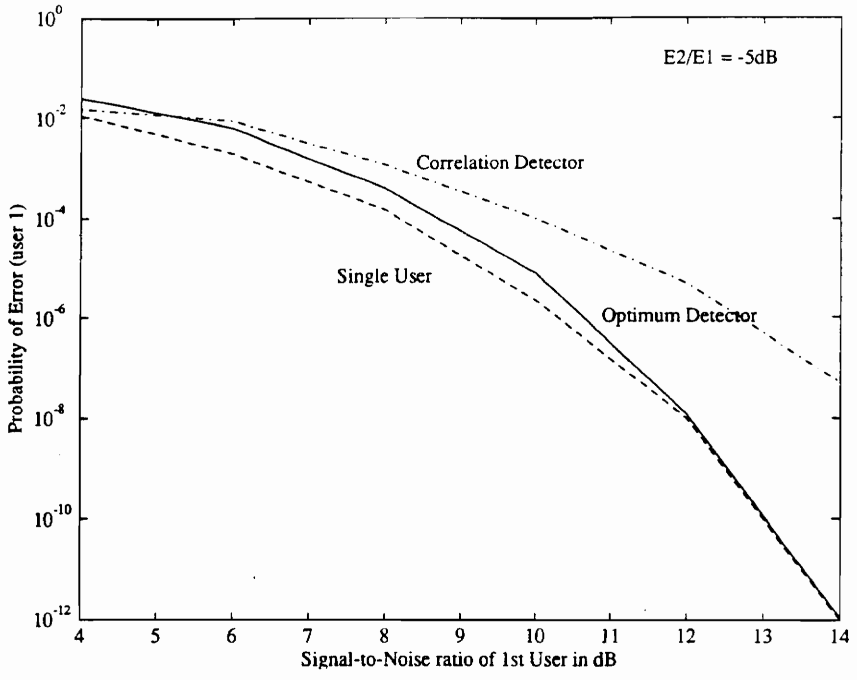
with $x_{i+1} = \left[x_i^2 \, x_i^3 \ldots x_i^{K-1} \, u_i\right]^T$, $x_{i_0} = 0$ where $\mathbf{H}^j$ is the j^{th} column of the cross-correlation matrix with elements

$$H_{ij} = \begin{cases} \int_{-\infty}^{+\infty} s_{i+j}(t-\tau_{i+j})\, s_j(t-\tau_j - T)\, dt, & \text{if } i+j \leq K; \\ \int_{-\infty}^{+\infty} s_{i+j-K}(t-\tau_{i+j-K})\, s_j(t-\tau_j)\, dt & \text{if } i+j > K. \end{cases} \qquad (2.1-4)$$

is shown by Verdu in [Ver86a] to fully exploit the sequential dependence of the log-likelihood function on the transmitted symbols, and the resulting optimum Viterbi decision algorithm performs K times the number of stages in [Ver83] (a state space dimensionality of 2^{K-1} with each stage connected to two states in the previous stage, for binary modulation) but with a time complexity per bit of $O(2^K)$. It has been shown [Ver84] that the multiuser detector which maximizes the probability of error has the same structure, but it uses a backward-forward dynamic programming algorithm instead.

2.3.3 Performance Results for Optimum Detector

Figure 2.3-2 compares the worst case performance of the optimum sequence detector with that of the conventional receiver for the case of 2 users employing a subset of the three m-sequences of length 31 generated to maximize a signal-to-multiple-access-interference functional[Gab80]. It is obvious that the optimum multiuser detector affords important performance gains over conventional single-user detector extended for multiuser environment. The error probability of the conventional detector is lower than the upper bound on the optimum sequence detector for small signal-to-noise ratios where the multiple-access interference plays a subordinate role compared to the background noise because the optimum detector pays a penalty for combating the interference instead of simply ignoring it. The opposite effect of an increase in the energy of the interfering users is also apparent: while the conventional error probability grows rapidly until it becomes multiple-access limited, the optimum sequence detector performance approaches the single-user lower bound. This effect has been characterized in [Ver86b] in terms of the asymptotic efficiency as follows. Let $P_k(\sigma)$ denote the bit-error-rate of the k^{th} user when the spectral level of the background white Gaussian noise is σ, and let $e_k(\sigma)$ be such that $P_k(\sigma) = Q(\sqrt{e_k(\sigma)}/\sigma)$. Then, $e_k(\sigma)$ is actually the energy that the k^{th}



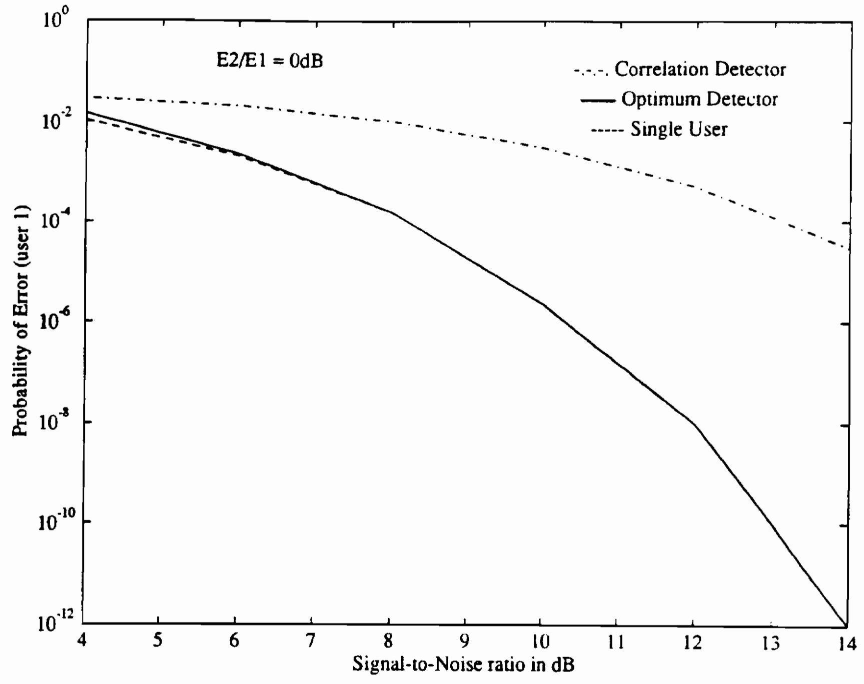


FIGURE 2.3-2 Minimum bit-error-probability for the optimum multiuser detector for a two-user direct-sequence spread-spectrum system with optimal m-sequences of length 31 [Ver86a]

user would require to achieve bit-error-rate $P_k(\sigma)$ in the same white Gaussian channel but without interfering users i.e. $e_k(\sigma)$ is the effective energy of the k^{th} user. Then the asymptotic efficiency (for high SNR) of a transmitter whose bit-error-rate curve and energy are given by $P_k(\sigma)$ and E_k respectively is $\eta_k = \lim_{\sigma\to 0}[e_k(\sigma)/E_k]$. The asymptotic efficiencies for the conventional detector and the optimum multiuser detector in a two-user case is shown in Figure 2.3-3. Note that the optimum multiuser detector is near-far resistant, and in fact has an asymptotic efficiency of unity for sufficiently powerful interference, whereas the conventional detector succumbs to the near-far problem and its asymptotic efficiency drops to zero for large interference.

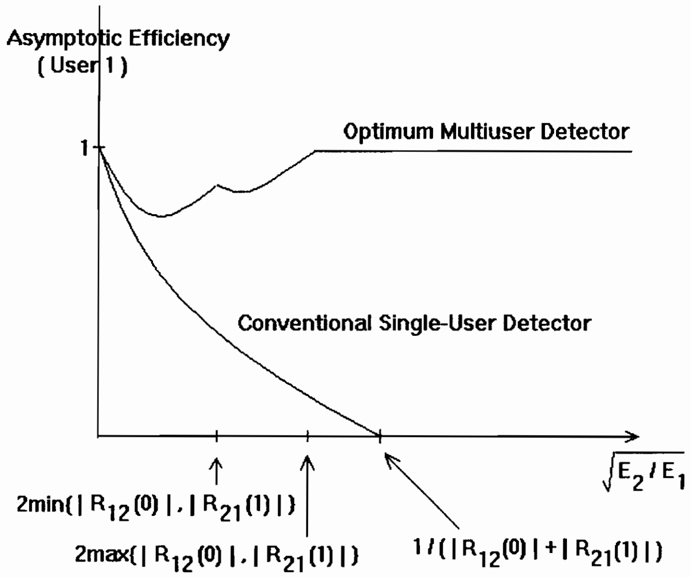


FIGURE 2.3-3 Asymptotic efficiencies for optimum and conventional detectors in a two-user case for infinite transmitted sequence length and user energies constant over time [Ver86b]

Despite its substantially higher performance the computational and storage complexity of this receiver are unpractical, being exponential in the number of users. It has also been shown that the multiuser optimum detection problem is NP-hard. In addition to requiring intensive complexity, the maximum likelihood detector solution also has a variable decoding delay, because optimum decisions cannot be made in a Viterbi forward dynamic programming algorithm until all states share a common shortest subpath, which is unacceptable in many applications. The knowledge of partial cross-correlations between the signals of every pair of active users and the received signal energies is also required in the Viterbi algorithm.

2.4 Suboptimal Detection Schemes for Asynchronous CDMA Systems

From earlier discussion it is clear that despite its simplicity, the conventional receiver has limited use for CDMA systems in the desirable range of bandwidth efficiencies. In addition, its vulnerability to near-far effects is considerable. On the other hand, the optimum receiver is computationally intensive, and has an unacceptable variable decoding delay. Hence, there is a need for suboptimum receivers which perform reliably in higher bandwidth efficiencies situation and are robust to near-far effects, with a reasonable computational complexity to ensure their practical implementation. Recent attempts to derive detectors for multiuser channels include [Var90], [Por88] and [Lup90].

2.4.1 Decorrelating Detection Strategy

The decorrelating detector in [Lup90] is shown in Figure 2.4-1 to be implemented as K separate single-input (discrete-time) single-output filters. This filter is a modification of the conventional single-user matched filter where instead of correlating the channel output with the signature waveform of the user of interest, its projection on the subspace orthogonal to the space spanned by the interfering signals is used. It has the near-far resistance of the optimum multiuser detector and requires the knowledge of the signal waveforms of the interfering users but, in contrast to the optimum receiver, has a computational complexity which is linear in the number of users and does not require knowledge of the received signal energies. But a stable, noncausal realization of the filter used in such a decorrelating detector is possible only if $det[H^T(1)e^{j\omega} + H(0) + H(1)e^{-j\omega}] \neq 0, \quad \forall\omega \in [0, 2\pi]$, (where H is the signature sequence cross-correlation matrix introduced earlier) i.e., if the cross-correlation properties of the signature sequences is not "poor".

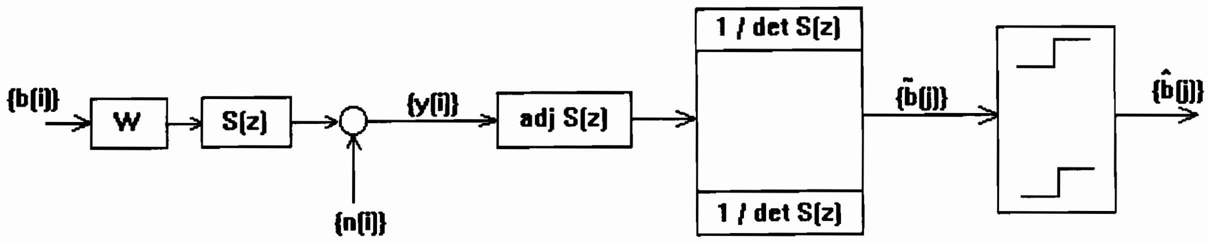


FIGURE 2.4-1 Interpretation of the suboptimum decorrelating detector [Lup90]

2.4.1 Multistage Detection Strategy

In the remainder of this section we are concerned with the multiuser detection strategy for coherent demodulation in an asynchronous CDMA system described in [Var90]. The resulting detector which processes the sufficient statistics via a multistage algorithm has been used for simulation purposes in Section 3.

It has been shown that the outputs of a bank of K matched filters matched to the modulating signal of each user, sampled in corresponding time synchronism, i.e.,

$$Z_i^{(j)}(0) = \int_{-\infty}^{+\infty} r(t)\, s_i(t + jT - \tau_i)\, dt \qquad (2.4-1)$$

are the sufficient statistics for purposes of demodulation. In fact, this set of sufficient statistics is minimally sufficient for the demodulation of any one bit[Ver86a].

The multistage suboptimum solution to the maximization of log-likelihood function $\Omega[\mathbf{B}, r(t)]$ used here is as follows: Consider the demodulation of the j^{th} bit of the i^{th} user and assume we are in the $(m+1)^{st}$ stage. Let the m^{th} stage estimates of bits be denoted as $\hat{b}_k^{(l)}$ for all k and for all bit intervals l. Then the $(m+1)^{st}$ stage estimate of $b_i^{(j)}$ is

$$b_i^{(j)}(m+1) = arg\Big\{max_{\substack{b_i^{(j)} \in \{+1,-1\} \\ b_k^{(l)} = \hat{b}_k^{(l)}(m), k \neq i}} \Omega[\mathbf{B}, r(t)]\Big\} \qquad (2.4-2)$$

Note that the maximization is performed by setting $b_k^{(j)}$ to be equal to its m^{th} stage estimate for all k and for all j but not $k = i$ and $l = j$ simultaneously. An additive decomposition of the log-likelihood function can be found in the form [Var90]

$$\Omega[\mathbf{B}, r(t)] = \sum_p \langle \mathbf{b}^{(p)}, 2\mathbf{z}^{(p)}(0) - \mathbf{H}(0)\mathbf{b}^{(p)} - 2\mathbf{H}(1)\mathbf{b}^{(p-1)} \rangle \qquad (2.4-3)$$

where $\langle \mathbf{x}, \mathbf{y} \rangle$ denotes the inner product of the vectors $\mathbf{x}$ and $\mathbf{y}$. On substituting this decomposed form of log-likelihood function in $(2.4-2)$, we have

$$b_i^{(j)}(m+1) = arg\Big\{max_{\substack{b_i^{(j)} \in \{+1,-1\} \\ b_k^{(l)} = \hat{b}_k^{(l)}(m), k \neq i}} \sum_{p=j}^{j+1} \langle \mathbf{b}^{(p)}, 2\mathbf{z}^{(p)}(0) - \mathbf{H}(0)\mathbf{b}^{(p)} - 2\mathbf{H}(1)\mathbf{b}^{(p-1)} \rangle \Big\} \qquad (2.4-4)$$

After each inner product is expanded, the terms dependent on $b_i^{(j)}$ are retained. Then substituting the m^{th} stage estimates of all bits except $b_i^{(j)}$ and noting the properties of $\mathbf{H}(\cdot)$ we have the following expression for the estimate

$$b_i^{(j)}(m+1) = sgn[z_i^{(j)}(m)] \qquad (2.4-5)$$

where

$$z_i^{(j)}(m) = z_i^{(j)}(0) - \sum_{k=1}^{i-1} h_{ik}(-1)\, \hat{b}_k^{(j+1)}(m) - \sum_{k \neq i} h_{ik}(0)\, \hat{b}_k^{(j)}(m) - \sum_{k=i+1}^{K} h_{ik}(1)\, \hat{b}_k^{(j-1)}(m) \quad (2.4-6)$$

When compared to $(2.2-2)$ the above result has a simple interpretation. The channel adds noise and multiple-access interference to the received signal, while the $(m+1)^{st}$ stage of the receiver obtains its decision statistic by *subtracting* the *estimate* of the multiple-access interference which is reconstructed using the m^{th} stage estimates of the information bits. (Such interference cancellation techniques for demodulation in a spread-spectrum multiple-access system are not new). Coupled with this multistage solution is the need to choose the initial estimate of the information bits $\hat{\mathbf{b}}^{(j)}(1)\,\forall j$. For reasons of conceptual simplicity the conventional detector can be chosen as the first stage, i.e.

$$\hat{\mathbf{b}}^{(j)}(1) = sgn[\mathbf{z}^{(i)}(0)], \quad \forall j \qquad (2.4-7)$$

As for any iterative solution, it is important that this initial estimate is reasonably good.

A realizable and efficient implementation of the multistage algorithm described by $(2.4-5)$ and $(2.4-6)$ is shown in Figure $(2.4-2)$. The first part of the figure shows a bank of K matched filters followed by a set of K M-stage processors. The detailed implementation of k^{th} such processor as a snap-shot at time $t = (i+2)T$ is shown in the next second part. For convenience, construction and subtraction of the multiple-access interference is assumed to require a time duration of T and sign determination is assumed to be instantaneous in this implementation. The following aspects are worth noticing:

a) This is a fixed decoding delay implementation with a decoding delay of $2(M-1)T$ where M is the number of stages in the receiver, and T is the bit duration.

b) It is clear from the multistage algorithm that at any time we need to store the current bit estimate and the previous two bit estimates for each user at each stage. The storage requirement of this detection scheme is thus $3(M-1)K$ for binary modulation. In other words the storage complexity of this implementation is $O(MK)$.

c) Reconstruction of the multiple-access interference in each stage requires no more than $2(K-1)$ additions per bit, as is obvious from $(2.4-6)$. Therefore, for an M-stage detector, $2(M-1)(K-1)$ additions per bit are needed. Thus, the computational complexity per symbol for the multistage detection scheme is $O(MK)$, which is linear in the number of users K. In contrast, the computational complexity of the optimal detection scheme had an exponential dependence on the number of users.

The suboptimum detection schemes described in this section eliminate the near-far problem of the conventional correlation detector, while maintaining a fixed decoding delay and a computational complexity that grows linearly as the number of users. These schemes therefore emerge as the solution of choice in practical DS-CDMA systems with a large number of users.

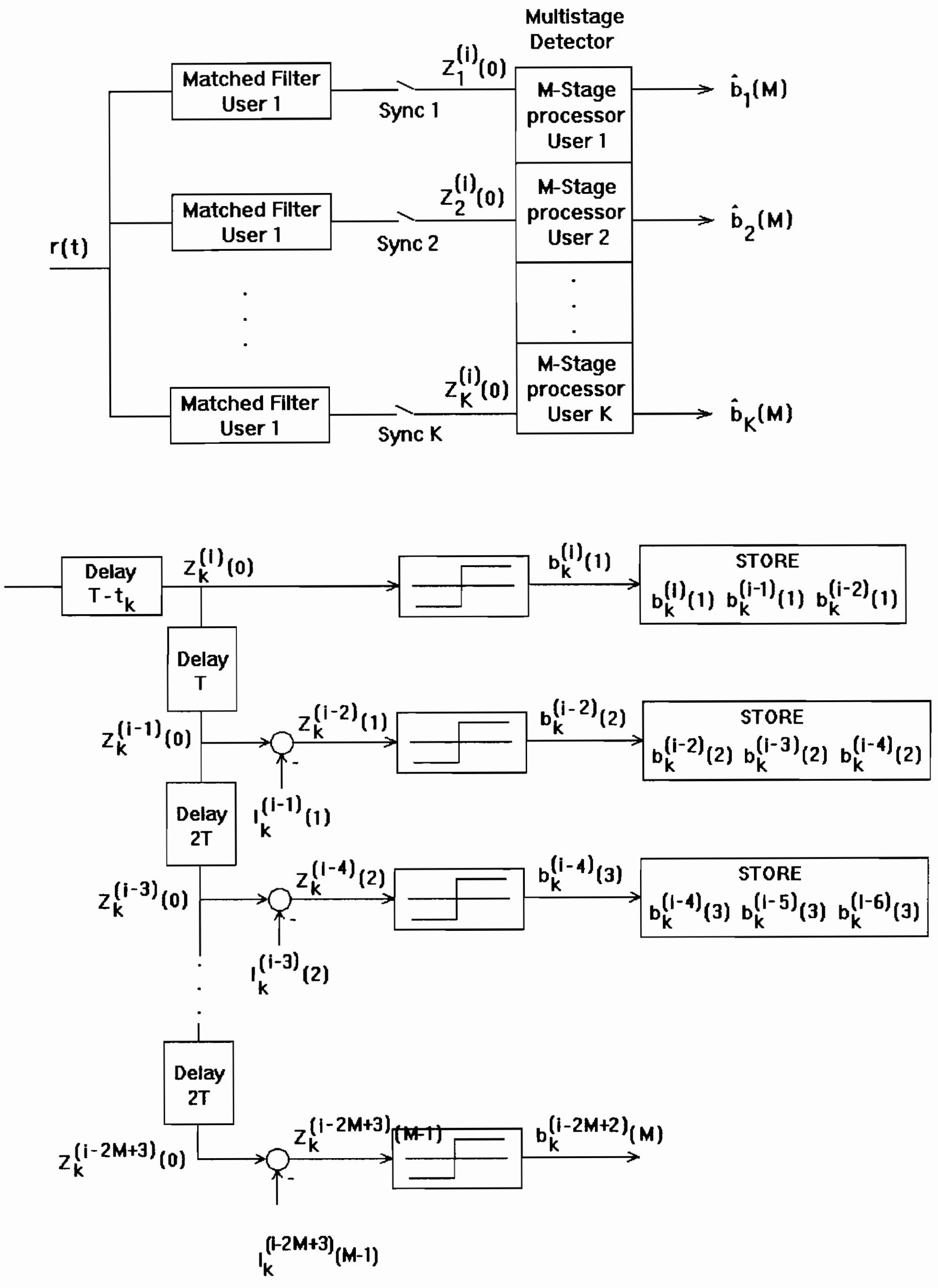


FIGURE 2.4-2 The suboptimum multistage detector for BPSK-CDMA system. The detailed implementation of the $k^{(th)}$ M-stage processor of the first part is shown in the second part, where $I_k^{(i-2M+1)}(m)$ denotes the estimate of the multiple-access interference reconstructed in the $m^{(th)}$ stage based on the estimates $\hat{b}_j^{(i-2m)}(m-1)$, $\hat{b}_j^{(i-2m+1)}(m-1)$, and $\hat{b}_j^{(i-2m+2)}(m-1)$ $\forall j \neq k$, obtained form other $K-1$ processors [Var90]

3 COMPUTER SIMULATION OF MULTISTAGE DETECTOR

The multistage receiver shown in the block diagram of Figure (2.4 − 2) was simulated using a C code on a SUN SPARCstation. The sailent features of the simulation are summarized below.

3.1 Decision Statistics Formulation and Algorithm Used in Simulation

The decision statistics at the output of the bank of matched filtes in the receiver are given by (2.2-9). However for the purpose of identifying the bits of various users generated in different time intervals correctly in the simulation the statistics need to be extended to the form

$$Z_i^{(j)}(0) = \eta_i^{(j)} + T\sqrt{P_i/2}\, b_i^{(j)} + \sum_{k=1}^{i-1} \sqrt{P_k/2}\left[b_k^{(j)}\hat{R}_{ik}(\tau_i - \tau_k) + b_k^{(j+1)} R_{ik}(\tau_i - \tau_k)\right]\cos(\phi_i - \phi_k)$$

$$+ \sum_{k=i+1}^{K} \sqrt{P_k/2}\left[b_k^{(j)}\hat{R}_{ki}(\tau_k - \tau_i) + b_k^{(j-1)} R_{ki}(\tau_k - \tau_i)\right]\cos(\phi_k - \phi_i) \qquad (3.1-1)$$

where $Z_i^{(j)}(0)$ is the decision statistic of the i^{th} user at the end of j^{th} time interval, $\eta_i^{(j)}$ is Gaussian distributed random variable with mean equal to zero and variance $N_0T/4$ if $N_0/2$ is the two-sided spectral density of the channel white Guassian noise, P_i is the power of the i^{th} user i.e. energy per bit of i^{th} user per bit duration, $b_k^{(j)}$ is the bit transmitted by the i^{th} user in the j^{th} bit interval, ϕ's and τ's are the relative phases and delays of the signals from different users, and $\hat{R}_{ik}(\tau)$,$R_{ik}(\tau)$ are the continuous-time partial cross-correlation sequences of the i^{th} and k^{th} users described by (2.1 − 6) and (2.1 − 7). These can be further expressed in terms of the discrete aperiodic cross-correlation function $C_{ik}(\cdot)$ and the continuous-time partial autocorrelation functions $\hat{R}_\psi(\cdot)$ and $R_\psi(\cdot)$ of the chip waveforms as follows[Pur77]

$$R_{ik}(\tau) = C_{ik}(\gamma - N)\,\hat{R}_\psi(\tau - \gamma T_c) + C_{ik}(\gamma + 1 - N)\,R_\psi(\tau - \gamma T_c), \quad 0 \le \tau \le T \qquad (3.1-2)$$

$$\hat{R}_{ik}(\tau) = C_{ik}(\gamma)\,\hat{R}_\psi(\tau - \gamma T_c) + C_{ik}(\gamma + 1)\,R_\psi(\tau - \gamma T_c), \quad 0 \le \tau \le T \qquad (3.1-3)$$

where

$$\gamma = \lfloor \tau/T_c \rfloor \quad 0 \le \gamma \le (N-1) \qquad (3.1-4)$$

$$R_\psi(s) = \int_0^s \psi(t)\,\psi(t + T_c - s)\,dt \quad 0 \le s \le T_c \qquad (3.1-5)$$

$$\hat{R}_\psi(s) = \int_s^{T_c} \psi(t)\,\psi(t - s)\,dt \quad 0 \le s \le T_c \qquad (3.1-6)$$

$$C_{ik}(\gamma) = \begin{cases} \sum_{j=0}^{N-1-\gamma} a_j^{(i)}\, a_{j+\gamma}^{(k)} & 0 \le \gamma \le (N-1); \\ \sum_{j=0}^{N-1+\gamma} a_{j-\gamma}^{(i)}\, a_j^{(k)} & (1-N) \le \gamma \le 0. \end{cases} \qquad (3.1-7)$$

For rectangular chip waveforms, $\psi(t) = P_{T_c}(t)$, the autocorrelation functions can be expressed as $R_\psi(s) = s$, $0 \le s \le T_c$ and $\hat{R}_\psi(s) = T_c - s$, $0 \le s \le T_c$. In this formulation $0 \le R_{ik}(\cdot) \le T$, and

$0 \le \hat{R}_{ik}(\cdot) \le T_c$, and $0 \le C_{ik}(\cdot) \le N$. Therefore, the functions depend on signature sequences only through the function $C_{ik}(\cdot)$ and on the chip waveforms only through the functions $R_{\psi}(\cdot)$ and $\hat{R}_{\psi}(\cdot)$.

This formulation of the decision statistics is equivalent to the formulation of $(2.2-2)$ except that a common factor of $\sqrt{2P_i}$ has been omitted without affecting the result of the threshold operation performed on it subsequently.

Using this formulation of decision statistics, the reconstructed multiple-access interference for the i^{th} user in the $(m+1)^{st}$ stage becomes

$$I_i^{(j)}(m+1) = \sum_{k=1}^{i-1} \sqrt{P_k/2}\left[\hat{b}_k^{(j)}\,\hat{R}_{ik}(\tau_i-\tau_k) + \hat{b}_k^{(j+1)}\,R_{ik}(\tau_i-\tau_k)\right]\cos(\phi_i-\phi_k)$$

$$+ \sum_{k=i+1}^{K} \sqrt{P_k/2}\left[\hat{b}_k^{(j)}\,\hat{R}_{ki}(\tau_k-\tau_i) + \hat{b}_k^{(j-1)}\,R_{ki}(\tau_k-\tau_i)\right]\cos(\phi_k-\phi_i) \qquad (3.1-8)$$

3.2 Specifications of Parameters Used in Simulation

The choice of parameters like signature sequences, random number generator and the user phase and delay distribution used in the simulation is important factors affecting the final simulation results. The specification of these parameters for the case of test simulations used in this study are now summarized, along with justifications for the same.

3.2.1 Signature Sequences

The most popular sequences used in literature for simulation of spread-spectrum systems are m-seuqences generated to maximize a signal-to-multiple-access-interference functional[Gab80],[Pur79]. However, in our simulation we use randomly generated Gold Sequences of length 31, for the following reasons:

a) Chip sequences with better cross-correlation can expand the range of signal energies for acceptable performance by conventional receiver, which rapidly becomes multiple-access limited, and therefore expand the range where improvement can be seen between the first (conventional) stage of the multistage receiver and the subsequent stages, for the same range of signal energies. Gold sequences possess considerably better cross-correlation properties than the m-sequences. It has been proven [Gold67] that certain pairs of m-sequences of length $n = 2^m - 1$ exhibit a three-valued cross-correlation function with values $\{-1, -t(m), t(m)-2\}$ where

$$t(m) = \begin{cases} 2^{(m+1)/2}+1, & m \text{ odd;} \\ 2^{(m+2)/2}+1, & m \text{ even.} \end{cases} \qquad (3.1-9)$$

If Φ_{max} is the peak cross-correlation for the family of possible sequences generated by a m stage shift register with different feedback connections, then $t(m)/\Phi_{max}$ is 3.44 for $n = 31$ i.e., a threefold difference in the peak values. The two m sequences of length n with periodic cross-correlation function that takes on the possible values $\{-1, -t(m), t(m)-2\}$ are called *preferred sequences.* From a pair of preferred sequences, say $\mathbf{a} = [\mathbf{a_1}\,\mathbf{a_2}\cdots\mathbf{a_n}]$ and $\mathbf{b} = [\mathbf{b_1}\,\mathbf{b_2}\cdots\mathbf{b_n}]$, we construct a set of sequences of length n by taking the modulo-2 sum of $\mathbf{a}$ with the n cyclicly shifted versions of $\mathbf{b}$ or vice-versa. Thus we obtain n new periodic sequences with period $n = 2^m - 1$. We may also include the original sequences $\mathbf{a}$ and $\mathbf{b}$ for a total of $n+2$ sequences. The shift register connections to obtain 33 Gold sequences of length 31 are shown in Figure 3.2-1. The contents of the shift register flip-flops, which affect which sequence is generated, were randomized between successive generations in order to get a random sequence order.

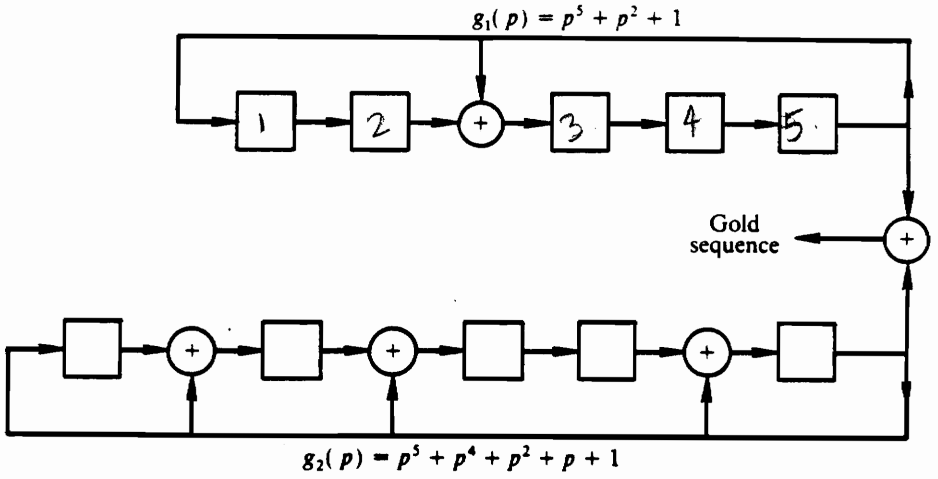


FIGURE 3.2-1 Generation of Gold sequences of length 31 [Pro89]

The tap values in each shift register stages are set according to the polynomials

$$g_1(p) = p^5 + p^2 + 1$$

$$g_2(p) = p^5 + p^4 + p^2 + p + 1$$

in order to generate the preferred sequences.

b) The bit-error-rate obtained is averaged with respect to all possible combinations of gold sequences of length 31 that might be selected by randomly assigning the signature sequences to the users, and changing the assignments at regular intervals. In comparision to the approach of choosing a fixed, best possible combination of K sequences (K = number of simultaneous users) from the set of all possible gold sequences, the result can be expected to be pessimistic, but more realistic.

3.2.2 Random Number Generator

The generation of random numbers for transmitted bits and the channel noise is an important aspect of this simulation. The spectral properties of the C library functions *rand() and srand()* leave a great deal to be desired. Not only is there a sequential dependence between the random numbers generated, but the lower bits of the numbers generated are not very random, and hence if m deviates are generated, they will tend to be concentrated along planes in the m dimensional space, with a maximum of $2^m - 1$ such planes, rather than being uniformly distributed in the space. Thus, if the process using these generators is occuring in a small volume of the whole space, the discreteness of the planes is more pronounced. This is the case with our simulation expected to give results in a reasonable time frame like two weeks of simulation time.

The random number generator used for simulation purposes produces uniform deviates between zero and one using the linear congruential formula

$$X_{n+1} = (a X_n + c) \, mod \, m \qquad n \geq 0$$

where a, c, and m are constants chosen to give the best random deviates on the architecture used to run the simulations. This portable random number generator is based on three linear congruential generators to generate different significant bits of the final deviate, and a shuffler to randomize the order of the generated deviates. Hence the generator achieves the required accuracy while eliminating sequential correlation between the deviates generated. Deviates drawn from Gaussian distribution for the purpose of simulating the performance of additive-white-gaussian-noise channel are generated by using the Box-Muller transformation on the uniform deviates derived from the above linear congruential generator in the following manner: if X_1 and X_2 are uniformly distributed over the range 0.0 to 1.0, then the deviates obtained by the transformations $Y_1 = \sqrt{-2\sigma^2 ln X_1} \sin(2\pi X_2)$ and $Y_2 = \sqrt{-2\sigma^2 ln X_1} \cos(2\pi X_2)$ have Gaussian distribution with mean 0 and variance σ^2, as can be verified by evaluating the Jacobian $\mid \frac{\partial(X_1,X_2)}{\partial(Y_1,Y_2)} \mid$. Since in $(3.1-1)$ the variance of $\eta_i^{(j)}$ is $\sigma^2 = N_0T/4$, and power of k^{th} user's signal can be calculated as $P_k = N_0 \times 10^{(SNR_1(dB)/10)} \times (Ratio\, of\, Relative\, Powers\, of\, k^{th}\, and\, 1^{st}\, users)$, therefore $\sqrt{N_0T}$ appears as a multiplicative factor in all terms on the right-hand-side of the equation. Therefore both the noise variance N_0 and the bit duration T can be excluded from the decision statistic without affecting the result of the threshold operation performed on $Z_i^{(j)}(0)$. Consequently, both N_0 and T have been set to unity in the simulation.

3.2.3 User Delays and Phases

In an asynchronous CDMA system, the K-user coherent receiver locks to the signaling interval and phase of each active user which transmits an independent data stream and hence the received phase and delay of each user at the receiver is different. For simulation purposes the relative user delays are assumed to be uniformly distributed in the interval $[0, T]$ and relative user phases are assumed to be uniformly distributed in the interval $[0, 2\pi]$ which are well known to be reasonable assumptions because of the dynamic environment of most CDMA systems and the fact that $\omega_c T_c \gg 2\pi$. Using the convexity of the Q-function and assuming the eye-open condition [Hay83] it has been shown that the bit-error-rate is upper bounded by its average for the base-band case (same phases for all users) on a discrete grid of time delays corresponding to the chip boundaries. However, the bit-error-rate curve obtained from our simulation is averaged with respect to all possible user delays and phases.

The algorithm used in simulation can therefore be given briefly as follows

1. *Read the number of receiver stages M, the number of users K and their relative powers.*
2. *Generate K Gold sequences starting from a random setting of bits in the shift register stages, and assign one chip sequence to each user. Generate user phases and delays as random numbers uniformly distributed over $[0, 2\pi]$ and $[0, T]$ respectively. Sort the user delays in acsending order.*
3. *Evaluate the partial cross-correlation functions $R_{ik}(\cdot)$ and $\hat{R}_{ik}(\cdot)$ for the chip sequences and relative delays generated above.*
4. *Generate a few data bits $\in \{-1, +1\}$ for each user, and the corresponding decision statistics from $(3.1-1)$, and demodulate them using the detector function which allows the delay lines and the three bit stores in the detector module are filled up.*
5. *Now, for each set of data bits generated for K users, calculate the corresponding decision statistics from $(3.1-1)$ and pass them to the detector module for demodulation. The detector module outputs M decisions on the bits transmitted during bit intervals $j, j-1, \ldots j-2(M-1)$ from its M stages for each user, where j is the current bit interval. Compare these demodulated bits against the corresponding transmitted bits for user 1, and increase the number of errors of user 1 in case of a disparity.*
6. *After about 100 bits or so the chip sequences, and user delays and phases have to be changed. Start again from step 2.*
7. *Repeat the process for approximately 10^8 bits*

The operation of the simulation program is described in Appendix A.

4 SIMULATION RESULTS AND THEIR INTERPRETATION

The multistage detector was simulated for different conditions of number of interfering users and their relative signal energies using the technique described in the previous section. In this section we present results of the simulation runs on a Sun SPARCstation and draw conclusions regarding the near-far robustness of the detector and the dependence of its bit-error-rate on the number of stages used.

4.1 Listing of Results for Different Interference Conditions

The uncoded bit-error-rates for user 1 at each stage of a seven stage detector were computed using the software for a direct-sequence spread-spectrum CDMA system with $k = 2, 3, 4$ users, the signal-to-noise ratio of user 1 varying in the range $5dB$ to $10dB$, and the signal energies of other users relative to user 1 varying in the range $0dB$ to $\pm 6dB$. The results are summarized in Tables 4.1-1 to 4.1-8 in the form of bit-error-rate of the 1^{st} user at each stage versus the signal-to-noise ratio of 1^{st} user and the corresponding results for a subsequent stage, hypothesized in Section 4.2.2 to have the best bit-error-rate versus computational complexity tradeoff, are plotted in Figures 4.1-1 to 4.1-8. The number of interfering users and their relative powers are noted along with each table and figure.

4.2 Interpretation of the Results

The results summarized in Tables 4.1-1 to 4.1-8 are now used in comparing the performance of the multistage receiver with that of a conventional receiver operating under the same interference conditions and a matched filter operating in the absence of interfering signals. Performance gains achievable through multistage detection are pointed out and their justification in terms of the receiver block diagram is used to draw an inference regarding the optimum number of stages needed in a receiver to avoid wasteful computation and decoding delay associated with redundant stages.

4.2.1 Near-Far Robustness of Multistage Detector

A bank of K matched filters is used as the first stage in the detector, and hence the bit-error-rate performance of Stage 1 equals that of a conventional coherent correlation receiver operating under the same interference conditions and selection of signature sequences. The performance of stage 1 has a strong near-far limitation and degrades rapidly for increasing interfering users' powers, as is obvious by comparing the bit-error-rates in the first columns of the Tables. For example the bit-error-rate of user 1 rise from a value of 8.528×10^{-3} at a signal-to-noise ratio of $5dB$ in Table 4.1-1 (single interfering user with a power level 3dB below that of the 1^{st} user) to 9.016×10^{-3} at $5dB$ in Table 4.1-2 (three equal power interferers with power levels 3dB below that of the 1^{st} user). The same phenomena can be observed between Tables 4.1-3 and 4.1-4 (increase in number of interferers at the same power level as that of the desired user from one to three), and between Tables 4.1-5 and 4.1-6 (increase in number of interferers at $3dB$ higher power level from one to three). Similarly, the bit-error-rate of user 1 at a signal-to-noise ratio of, say, $5dB$ rises from a value of 8.528×10^{-3} to 2.415×10^{-2} when the power level of a single interfering user rises from $3dB$ below that of user 1 (the case of Table 4.1-1) to 3dB above that of user 1 (the case of Table 4.1-5). In other words, the bit-error-rate curve for the first (conventional detector) stage shifts towards higher error probabilities as the number of interferers at the same power level increases, or the energy of the interferer(s) increases (the latter being the near-far limitation).

In contrast the performance of all the subsequent stages in the multistage detector improves with increasing powers of interfering users, approaching that of single-user operating in the absence of interference. This is obvious if one compares the bit-error-rates for any particular stage in Tables 4.1-1, 4.1-3, and 4.1-5 which correspond, respectively, to the power of signals from the (single) interfering user varying from 3dB below that of the signal power from the desired user (Table 4.1-1) to the same power as user 1 (Table 4.1-3) to 3dB above the power of user 1 (Table 4.1-5). In other words, the bit-error-rate values in column 1 (stage 1, or the conventional detector) increase from Table 4.1-1 to Table 4.1-3 to Table 4.1-5 , corresponding to increasing interference, whereas the values in subsequent columns (higher order stages) decrease and tend to the single-user channel values. The improvement in performance over the conventional detector is therefore two-fold.

The simple explanation for this is that, if an interfering user has a high energy, it is estimated better by the first stage and hence is rejected more successively in the subsequent stages. This behavior is also noted in [Var90] where it is shown that the performance of a two-stage detector with two users improves as the signal strength of the interfering user increases, and in [Vdu86] where it is shown that the asymptotic efficiency of the optimum detector is equal to one for sufficiently high interference. The near-far rejection by these multiuser detectors eliminates a principal shortcoming of current radio networks using direct-sequence spread-spectrum communications.

4.2.2 Bit-Error-Rate Dependence on the Number of Stages

Since the simulation of the multistage detector was carried out for seven stages, it affords an opportunity to study the bit-error-rate performance of various stages of the detector. This study has not been undertaken until now, but it is an important consideration in deciding on the number of receiver stages to use in a practical implementation of the multistage detection strategy for multiple-access communications.

It is obvious that no single stage gives the best performance for all the cases of the number of interfering users and their relative energies. The suboptimal multistage detection approach attempts to reach at a decision on the information bit transmitted by a particular user in the a past bit interval by setting the interfering bits from other users during that time interval to their best estimates currently known (obtained from the preceding stage), and uses this information in arriving at a conclusion regarding bits transmitted by the other users during the same past bit interval. Thus, instead of the joint maximization of log-likelihood function for the information matrix transmitted by all users in a particular time interval, the approach is to maximize it in an iterative manner. This approach searches for the maxima of the log-likelihood function along hyperplanes in the information space determined by the bit estimates of the previous stage rather than in the whole space, which might lead it not to converge to the global maxima. This can be observed in fluctuations in the bit-error-rate with increasing stage index.

However it can be concluded that, depending on the number of separate power "levels", there is a minimum number of stages until which the performance can be *guaranteed* to improve from stage-to-stage. The reasoning can be applied as follows. The first stage of the receiver (the conventional receiver stage) forms good estimates for the information bits of all users transmitting at the highest power "level" because these users will exprience moderate interference from each other but relatively little interference from users at lower power levels. The number of such users that can transmit simultaneously at this power level without the ensuing impairment in the performance of the bank of matched filters of this stage is determined by the interference tolerability of the conventional receiver. The bit estimates at this stage of the users transmitting at the lower power levels will be severely impaired because of the strong interfering power their matched filters will face from users at the highest power level. At the second stage the bit estimates for these lower powers will improve when the receiver reconstructes the interference experienced from high power users based on their good bit estimates obtained from the first stage and subtracts it off from the decision statistics of the low power users. Thus the second stage will obtain good bit estimates for all users operating at the second "level" of powers. The users operating at power levels lower than the second will still experience sustantial interference from the users at second power level even though largest interference from users at the highest power level has been neutralized. Therefore at the second stage the receiver decodes only the users at the second level with sufficient accuracy. Extending this reasoning it can be hypothesized that one stage is needed in the receiver for each level of transmitting powers of the users. Further, the amount of improvement observed at each stage over the previous stage should decrease because the power level of neutralized interferers keeps decreasing with the stage index. The performance of stages with indexes beyond the number of user power levels is not guaranteed to improve because of the unpredictable dynamics of suboptimal multistage maximization algorithm. In other words as with any iterative solution, a performance improvement may not materialize unless the initial estimates are significantly better. Therefore limiting the number of stages to the number of users' power levels would be a good tradeoff between bit-error-rate versus computational complexity and receiver decoding delay.

This conclusion is supported by the simulation results.

- In case of any number of interfering users operating at 3dB lower power than that of the desired user (the cases of Table 4.1-1 and 4.1-2 for one and three interferers respectively) the first stage should form good estimates for the desired user's information bits and use these good estimates to reliably reconstruct and reject the interference experienced by other users in the second stage. The second stage will then form better estimates of the interfering users' information bits than was possible in the first stage. Using these good estimates for the interfering users, the desired user should be able to neutralize the interferers more reliably in the third stage. Thus an improvement in bit-error-rate of the desired user should be guaranteed till the third stage. This can be readily observed in Tables 4.1-1 and 4.1-2 where the second stage achieves most of the bit-error-rate improvement and the third stage affords further, though somewhat less, improvement.
- For the case of equal power interferers (Tables 4.1-3 and 4.1-4), the first stage bit estimates for the interfering users are reasonably accurate because there are no users operating at higher power levels. These estimates can be used by the desired user to reliably reject the interference experienced by it in the second stage. Therefore the bit-error-rate is guaranteed to improve from the first stage to the second, but not beyond it, a fact which can be verified from Tables 4.1-3 and 4.1-4.
- Similarly, in the case of all interfering users' signals having 3dB more power than the desired user signal, the receiver will form good estimates of the interfering users' bits in the first stage and use them in the second stage to subtract off the interference component from decision statistic of the desired user in the second stage. Therefore the second stage is again guaranteed to achieve improved performance, as in the case of equal energy interferers above.
- To verify these conclusions with greater precision, simulations were performed for the case of users with powers in 3dB cascades. Table 4.1-7 summarizes the results for the case of two interfering users transmitting at one-half and one-fourth the power of the desired user signal. Therefore in this case the weaker interferer is at the lowest power level, the second interferer is at the second power level, and the desired user is at the highest power level, where power levels are set to $3dB$ in this case. Similarly, Table 4.1-7 summarizes the results for the case of two interfering users transmitting at double and quadraple the power of the desired user signal, and in this case the desired user is at the lowest power level, the weaker interferer is at the second power level, and the stronger interferer is at the highest power level. In both cases the

lowest bit-error-rate is obtained by the third stage, as can be expected from the above argument because there are three power levels at which the users are transmitting.

- It is also clear that the bit-error-rate does decrease beyond the stage with index equal to the number of user power levels in the system. For example in the simulation result for cascaded power case of Table 4.1-8, the bit-error-rate is not minimum at the 3^{rd} stage but decreases to a still lower value in the 6^{th} stage after a second round of interference rejection in stages 4, 5, and 6. Similarly in Tables 4.1-1 and 4.1-2 stage 5 often accomplishes lower bit-error-rates than stage 3; and in Table 4.1-3 through 4.1-6 stage 4 often accomplishes better results than stage 2. However, this is not a consistent result and is more pronounced only in cases of strong interference. Moreover, very small bit-error-rate gains are achievable by these higher-indexed stages in the cases of strong interference (of the order of 0.9%). This does not seem to compensate for the additional computational complexity and decoding delay involved in implementing these stages. Therefore, limiting the number of stages to the number of user power levels to achieve only the first round of bit-error-rate reductions appears to be a good compromise in most cases of moderate interferences. Further cost-benefit analysis may be required in this regard with more extensive simulation.

TABLE 4.1-1 Bir-Error-Rate of a User in a 7-Stage Detector

One Interfering User at $3dB$ Lower Power

SNR	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7
$5dB$	8.528×10^{-3}	8.170×10^{-3}	8.108×10^{-3}	8.161×10^{-3}	8.127×10^{-3}	8.154×10^{-3}	8.133×10^{-3}
$6dB$	4.517×10^{-3}	4.121×10^{-3}	4.067×10^{-3}	4.118×10^{-3}	4.059×10^{-3}	4.108×10^{-3}	4.093×10^{-3}
$7dB$	2.352×10^{-3}	1.979×10^{-3}	1.938×10^{-3}	1.982×10^{-3}	1.934×10^{-3}	1.970×10^{-3}	1.958×10^{-3}
$8dB$	1.109×10^{-3}	8.229×10^{-4}	8.014×10^{-4}	8.207×10^{-4}	8.016×10^{-4}	8.189×10^{-4}	8.121×10^{-4}
$9dB$	5.011×10^{-4}	3.129×10^{-4}	3.030×10^{-4}	3.132×10^{-4}	3.027×10^{-4}	3.129×10^{-4}	3.114×10^{-4}
$10dB$	2.312×10^{-4}	1.279×10^{-4}	1.213×10^{-4}	1.229×10^{-4}	1.217×10^{-4}	1.256×10^{-4}	1.249×10^{-4}

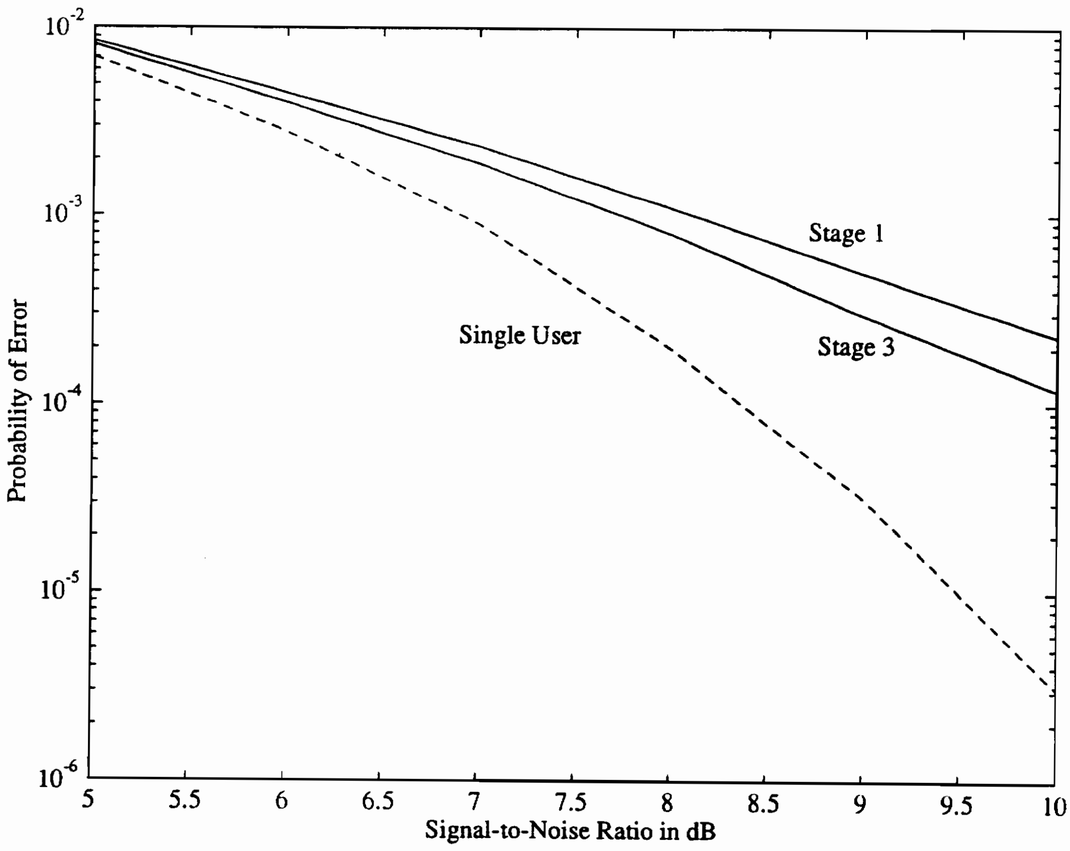


FIGURE 4.1-1 Bir-Error-Rates vs. Signal-to-Noise ratio for Stage 1 and Stage 3

One Interfering User at $3dB$ Lower Power

TABLE 4.1-2 Bir-Error-Rate of a User in a 7-Stage Detector
Three Interfering Users each at $3dB$ Lower Power

SNR	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7
$5dB$	9.016×10^{-3}	8.167×10^{-3}	8.062×10^{-3}	8.144×10^{-3}	8.057×10^{-3}	8.121×10^{-3}	8.101×10^{-3}
$6dB$	5.814×10^{-3}	4.117×10^{-3}	3.951×10^{-3}	4.075×10^{-3}	3.948×10^{-3}	4.062×10^{-3}	4.048×10^{-3}
$7dB$	3.462×10^{-3}	1.878×10^{-3}	1.741×10^{-3}	1.819×10^{-3}	1.736×10^{-3}	1.802×10^{-3}	1.791×10^{-3}
$8dB$	1.907×10^{-3}	7.630×10^{-4}	7.028×10^{-4}	7.641×10^{-4}	7.017×10^{-4}	7.508×10^{-4}	7.147×10^{-4}
$9dB$	1.004×10^{-3}	3.011×10^{-4}	2.462×10^{-4}	2.936×10^{-4}	2.547×10^{-4}	2.886×10^{-4}	2.793×10^{-4}
$10dB$	5.023×10^{-4}	1.102×10^{-4}	8.527×10^{-5}	1.007×10^{-4}	9.024×10^{-4}	9.641×10^{-5}	9.463×10^{-5}

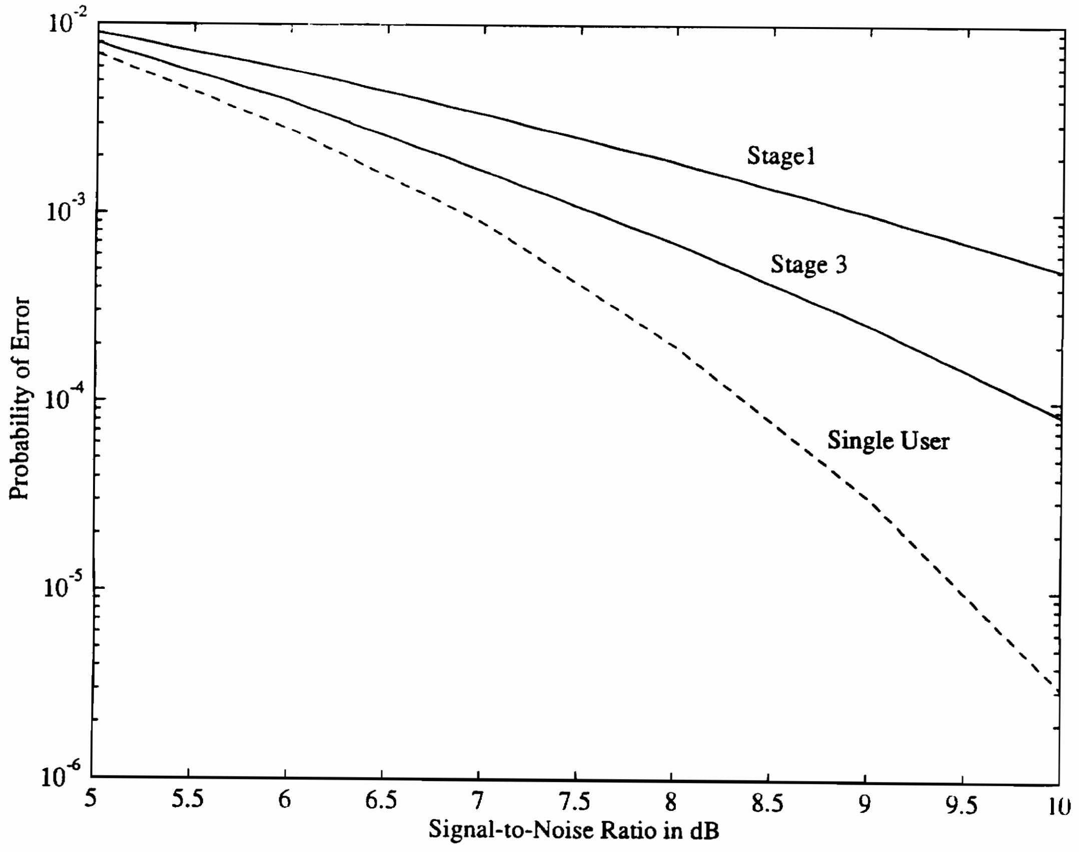


FIGURE 4.1-2 Bir-Error-Rates vs. Signal-to-Noise ratio for Stage 1 and Stage 3
Three Interfering Users each at $3dB$ Lower Power

TABLE 4.1-3 Bir-Error-Rate of a User in a 7-Stage Detector

One Interfering User having same Power as the Desired User

SNR	**Stage 1**	**Stage 2**	**Stage 3**	**Stage 4**	**Stage 5**	**Stage 6**	**Stage 7**
$5dB$	1.518×10^{-2}	7.521×10^{-3}	7.733×10^{-3}	7.512×10^{-3}	7.682×10^{-3}	7.606×10^{-3}	7.670×10^{-3}
$6dB$	8.502×10^{-3}	3.508×10^{-3}	3.657×10^{-3}	3.492×10^{-3}	3.613×10^{-3}	3.508×10^{-3}	3.582×10^{-3}
$7dB$	5.017×10^{-3}	1.314×10^{-3}	1.362×10^{-3}	1.316×10^{-3}	1.379×10^{-3}	1.331×10^{-3}	1.347×10^{-3}
$8dB$	2.815×10^{-3}	5.012×10^{-4}	5.306×10^{-4}	5.066×10^{-4}	5.237×10^{-4}	5.090×10^{-4}	5.152×10^{-4}
$9dB$	1.653×10^{-3}	1.622×10^{-4}	1.768×10^{-4}	1.622×10^{-4}	1.781×10^{-4}	1.698×10^{-4}	1.713×10^{-4}
$10dB$	9.002×10^{-4}	5.129×10^{-5}	5.653×10^{-5}	5.127×10^{-5}	5.591×10^{-5}	5.202×10^{-5}	5.276×10^{-5}

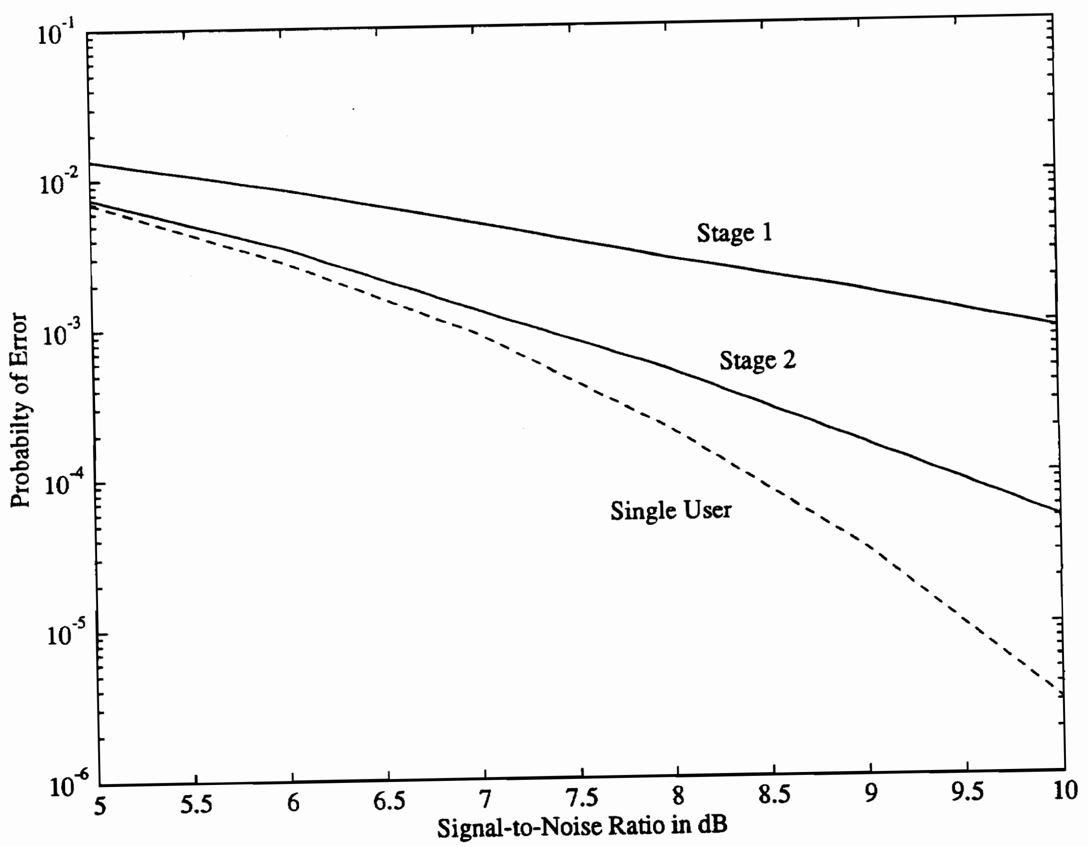


FIGURE 4.1-3 Bir-Error-Rates vs. Signal-to-Noise ratio for Stage 1 and Stage 2

One Interfering User having same Power as the Desired User

TABLE 4.1-4 Bir-Error-Rate of a User in a 7-Stage Detector
Three Interfering Users at the same Power Level as the Desired User

SNR	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7
$5dB$	2.311×10^{-2}	7.014×10^{-3}	7.218×10^{-3}	7.032×10^{-3}	7.157×10^{-3}	7.059×10^{-3}	7.084×10^{-3}
$6dB$	1.809×10^{-2}	3.021×10^{-3}	3.123×10^{-3}	3.016×10^{-3}	3.097×10^{-3}	3.014×10^{-3}	3.395×10^{-3}
$7dB$	1.237×10^{-2}	1.109×10^{-3}	1.134×10^{-3}	1.096×10^{-3}	1.109×10^{-3}	1.172×10^{-3}	1.189×10^{-3}
$8dB$	7.821×10^{-3}	4.083×10^{-4}	4.124×10^{-4}	4.074×10^{-4}	4.126×10^{-4}	4.134×10^{-4}	4.174×10^{-4}
$9dB$	5.070×10^{-3}	1.227×10^{-4}	1.273×10^{-4}	1.224×10^{-4}	1.263×10^{-4}	1.218×10^{-4}	1.196×10^{-4}
$10dB$	3.082×10^{-3}	3.356×10^{-5}	3.665×10^{-5}	3.351×10^{-5}	3.572×10^{-5}	3.426×10^{-5}	3.575×10^{-5}

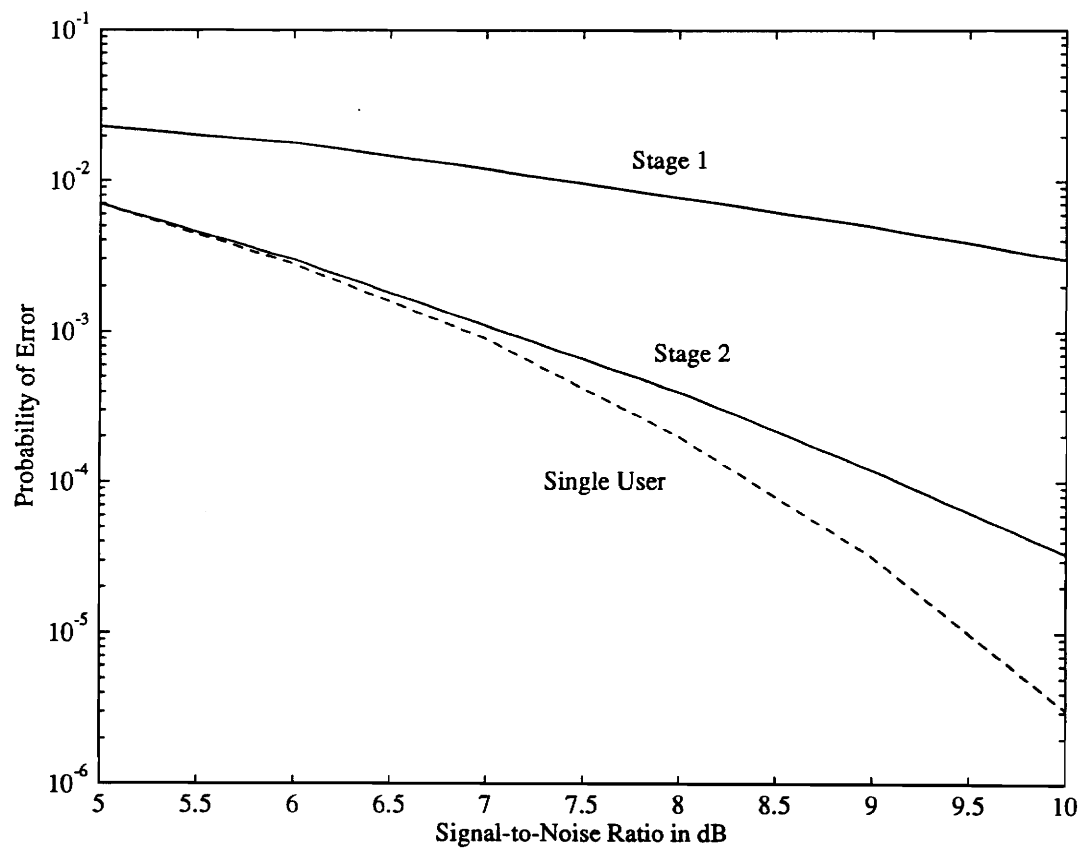


FIGURE 4.1-4 Bir-Error-Rates vs. Signal-to-Noise ratio for Stage 1 and Stage 2
One Interfering User at the same Power Level as the Desired User

TABLE 4.1-5 Bir-Error-Rate of a User in a 7-Stage Detector

One Interfering User at $3dB$ Higher Power

SNR	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7
$5dB$	2.415×10^{-2}	7.041×10^{-3}	7.183×10^{-3}	7.087×10^{-3}	7.129×10^{-3}	7.114×10^{-3}	7.183×10^{-3}
$6dB$	1.908×10^{-2}	2.952×10^{-3}	3.093×10^{-3}	2.961×10^{-3}	3.046×10^{-3}	3.012×10^{-3}	3.032×10^{-3}
$7dB$	1.351×10^{-2}	1.061×10^{-3}	1.085×10^{-3}	1.056×10^{-3}	1.092×10^{-3}	1.079×10^{-3}	1.112×10^{-3}
$8dB$	9.014×10^{-3}	2.303×10^{-4}	2.505×10^{-4}	2.329×10^{-4}	2.456×10^{-4}	2.404×10^{-4}	2.373×10^{-4}
$9dB$	6.041×10^{-3}	5.031×10^{-5}	5.823×10^{-5}	5.017×10^{-5}	5.915×10^{-5}	5.442×10^{-5}	5.728×10^{-5}
$10dB$	4.120×10^{-3}	9.016×10^{-6}	8.952×10^{-6}	1.047×10^{-5}	1.085×10^{-5}	1.081×10^{-5}	1.041×10^{-5}

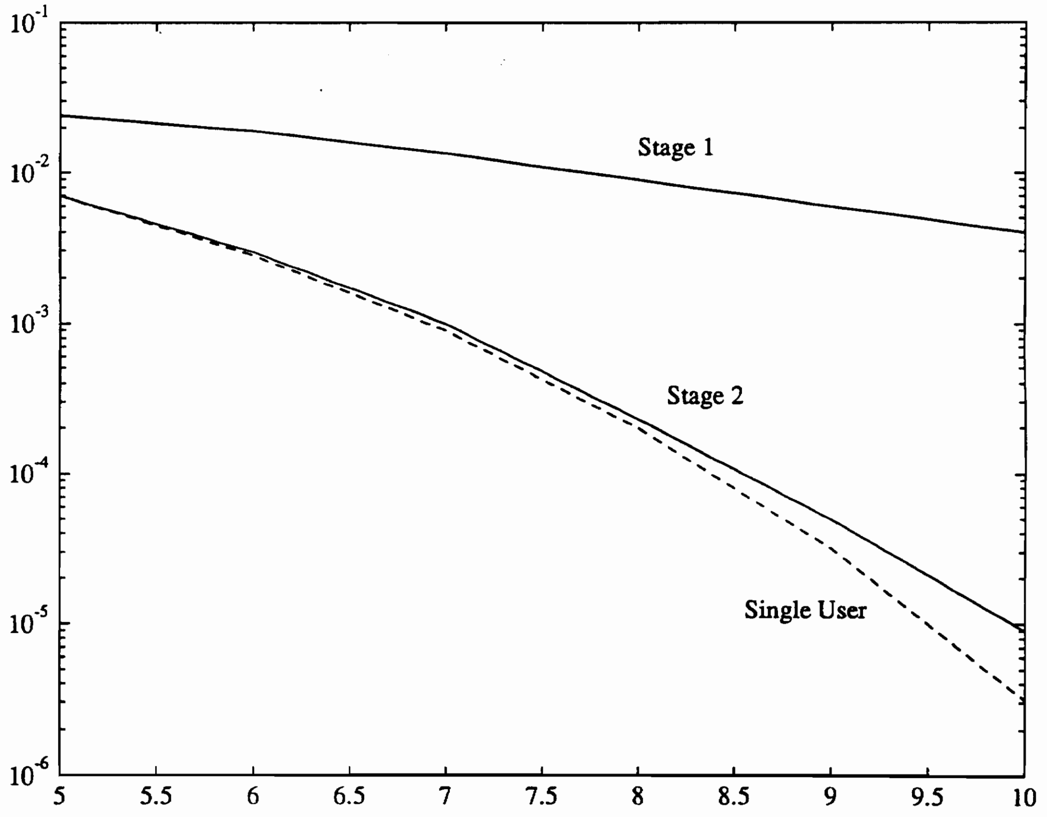


FIGURE 4.1-5 Bir-Error-Rates vs. Signal-to-Noise ratio for Stage 1 and Stage 3

One Interfering User at $3dB$ Higher Power

TABLE 4.1-6 Bir-Error-Rate of a User in a 7-Stage Detector
Three Interfering User each at $3dB$ Higher Power

SNR	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7
$5dB$	5.062×10^{-1}	7.013×10^{-3}	7.142×10^{-3}	7.046×10^{-3}	7.119×10^{-3}	7.073×10^{-3}	7.207×10^{-3}
$6dB$	4.088×10^{-1}	2.857×10^{-3}	3.016×10^{-3}	2.843×10^{-3}	3.044×10^{-3}	3.083×10^{-3}	3.079×10^{-3}
$7dB$	3.040×10^{-1}	9.099×10^{-4}	1.064×10^{-3}	9.027×10^{-4}	1.052×10^{-3}	9.367×10^{-4}	1.039×10^{-3}
$8dB$	2.593×10^{-1}	2.124×10^{-4}	2.426×10^{-4}	2.117×10^{-4}	2.439×10^{-4}	2.131×10^{-4}	2.378×10^{-4}
$9dB$	2.241×10^{-1}	3.453×10^{-5}	4.877×10^{-5}	3.387×10^{-5}	3.953×10^{-5}	3.411×10^{-5}	3.975×10^{-5}
$10dB$	2.253×10^{-1}	4.039×10^{-6}	6.257×10^{-6}	4.020×10^{-6}	5.856×10^{-6}	4.003×10^{-6}	5.432×10^{-6}

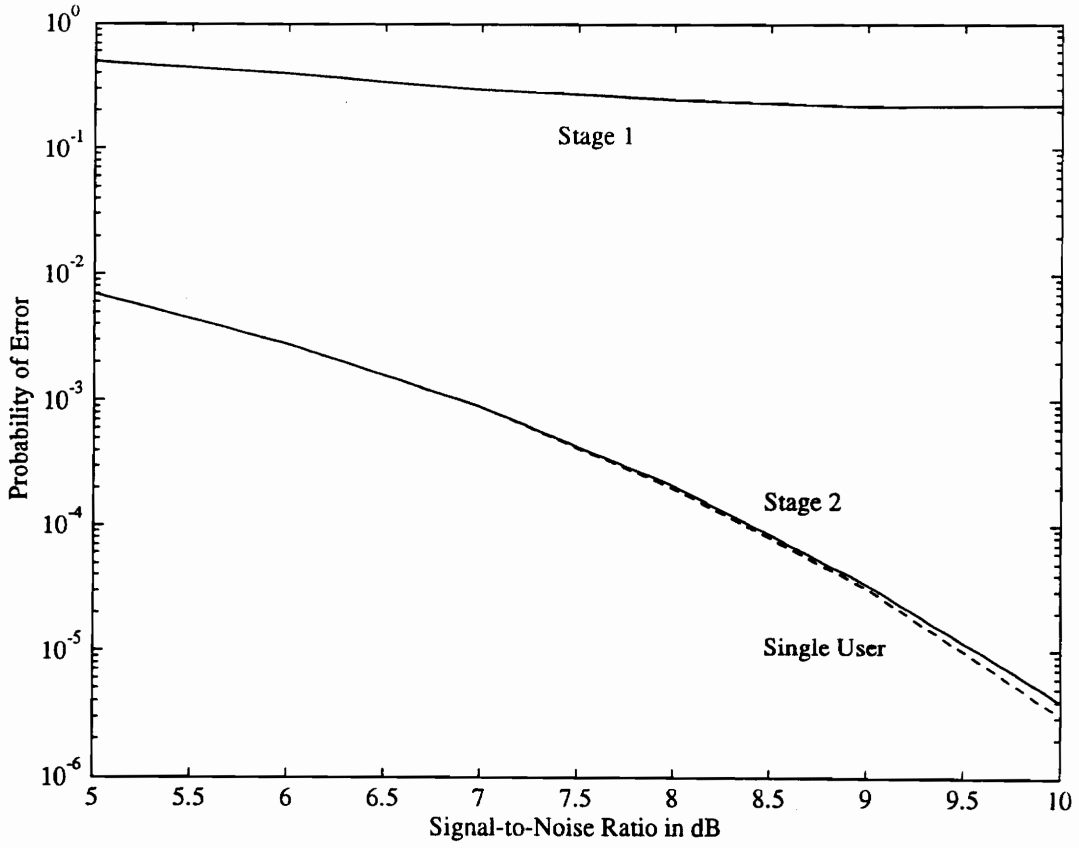


FIGURE 4.1-6 Bir-Error-Rates vs. Signal-to-Noise ratio for Stage 1 and Stage 3
Three Interfering User each at $3dB$ Higher Power

TABLE 4.1-7 **Bir-Error-Rate of a User in a 7-Stage Detector**

Two Interfering Users at $3dB$ and $6dB$ Lower Powers

SNR	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7
$5dB$	8.963×10^{-3}	8.306×10^{-3}	7.953×10^{-3}	8.317×10^{-3}	8.298×10^{-3}	8.314×10^{-3}	8.044×10^{-3}
$6dB$	5.413×10^{-3}	4.336×10^{-3}	3.871×10^{-3}	4.042×10^{-3}	3.976×10^{-3}	4.016×10^{-3}	3.882×10^{-3}
$7dB$	3.105×10^{-3}	2.040×10^{-3}	1.686×10^{-3}	1.814×10^{-3}	1.827×10^{-3}	1.792×10^{-3}	1.673×10^{-3}
$8dB$	1.676×10^{-3}	8.204×10^{-4}	6.066×10^{-4}	6.341×10^{-4}	6.293×10^{-4}	6.304×10^{-4}	6.017×10^{-4}
$9dB$	8.217×10^{-4}	2.964×10^{-4}	2.041×10^{-4}	2.427×10^{-4}	2.316×10^{-4}	2.224×10^{-4}	2.039×10^{-4}
$10dB$	4.027×10^{-4}	9.047×10^{-5}	5.578×10^{-5}	6.011×10^{-5}	5.593×10^{-5}	5.812×10^{-5}	5.546×10^{-5}

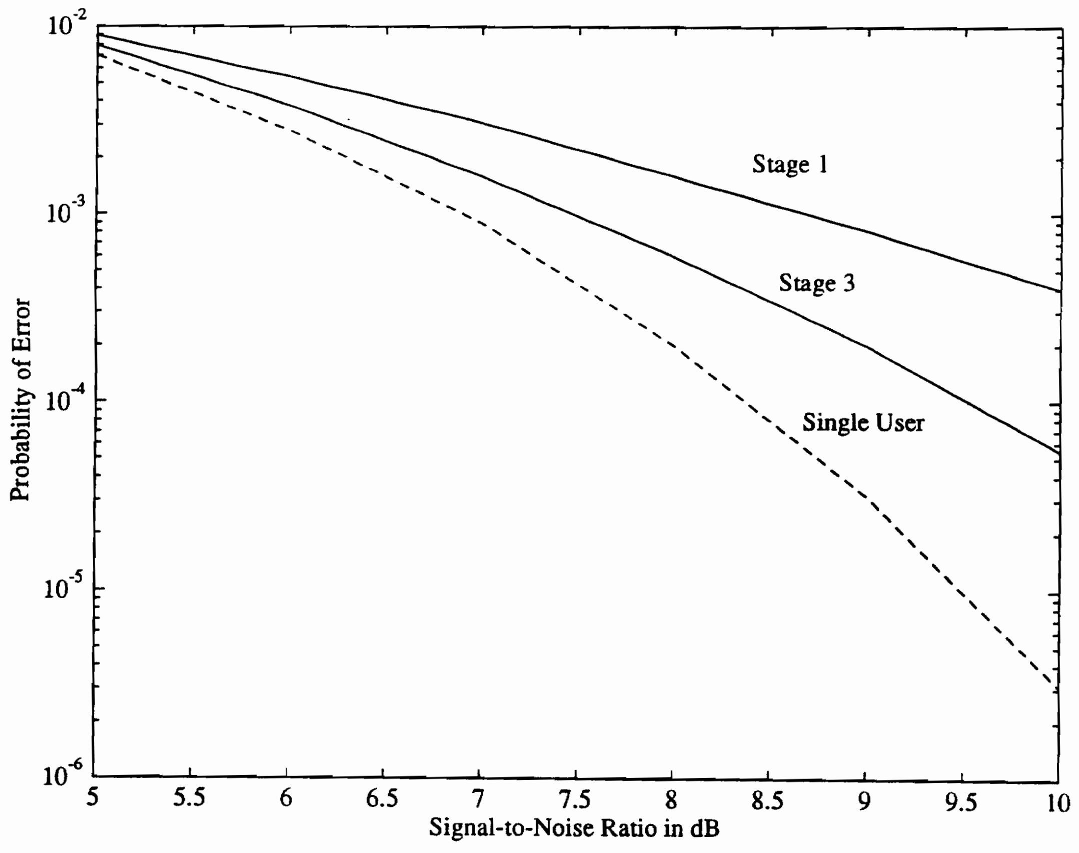


FIGURE 4.1-7 **Bir-Error-Rates vs. Signal-to-Noise ratio for Stage 1 and Stage 3**

Two Interfering Users at $3dB$ and $6dB$ Lower Powers

TABLE 4.1-8 Bir-Error-Rate of a User in a 7-Stage Detector
Two Interfering Users at $3dB$ and $6dB$ Higher Powers

SNR	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7
$5dB$	4.037×10^{-1}	7.516×10^{-3}	7.017×10^{-3}	7.543×10^{-3}	7.582×10^{-3}	6.974×10^{-3}	7.003×10^{-3}
$6dB$	3.478×10^{-1}	3.116×10^{-3}	2.858×10^{-3}	3.132×10^{-3}	3.127×10^{-3}	2.849×10^{-3}	2.836×10^{-3}
$7dB$	2.524×10^{-1}	9.140×10^{-4}	9.027×10^{-4}	9.174×10^{-4}	9.182×10^{-4}	8.960×10^{-4}	8.942×10^{-4}
$8dB$	1.810×10^{-1}	2.097×10^{-4}	2.052×10^{-4}	2.151×10^{-4}	2.185×10^{-4}	2.049×10^{-4}	2.036×10^{-4}
$9dB$	1.205×10^{-1}	3.338×10^{-5}	3.277×10^{-5}	3.308×10^{-5}	3.282×10^{-5}	3.227×10^{-5}	3.257×10^{-5}
$10dB$	8.097×10^{-2}	3.746×10^{-6}	3.705×10^{-6}	3.752×10^{-6}	3.814×10^{-6}	3.691×10^{-6}	5.640×10^{-6}

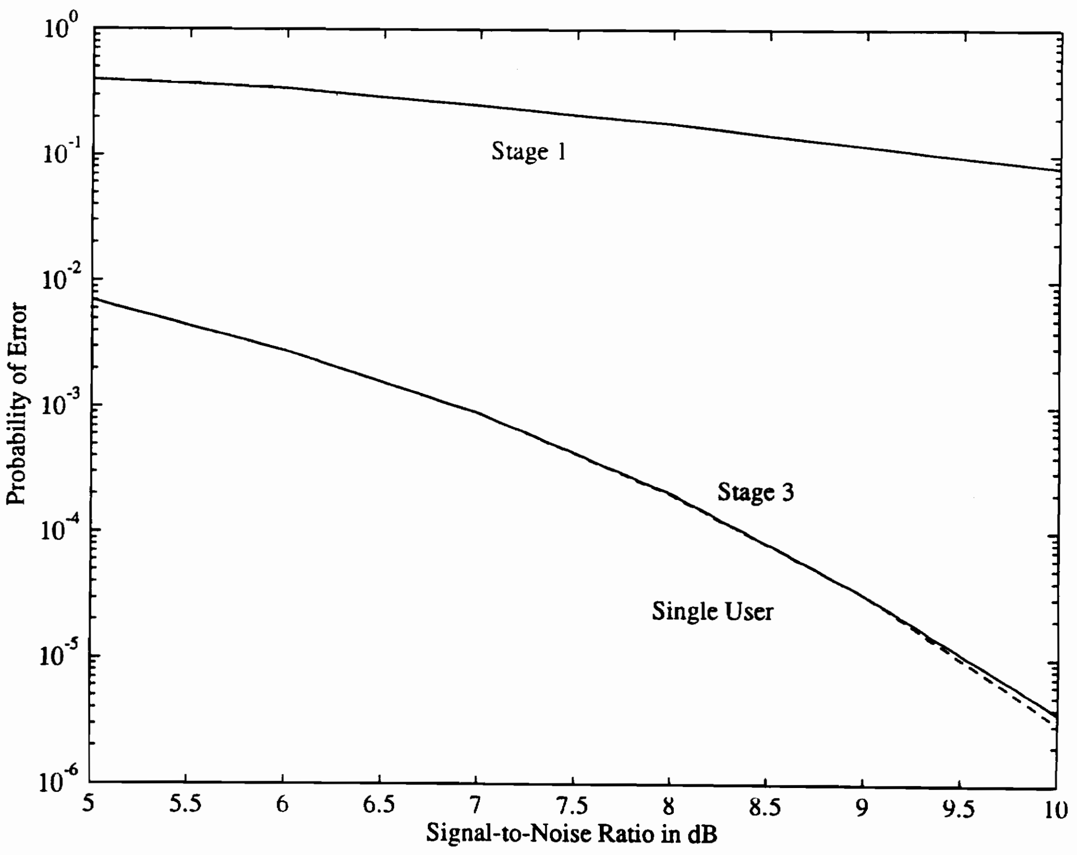


FIGURE 4.1-8 Bir-Error-Rates vs. Signal-to-Noise ratio for Stage 1 and Stage 3
Two Interfering Users at $3dB$ and $6dB$ Higher Powers

5 CONCLUSIONS

An important aspect of spread-spectrum code-division multiple-access radio networks has been addressed in this project: coherent demodulation of asynchronous users in near-far situations with high bandwidth efficiency and a reasonable computational complexity. Currently, spread-spectrum systems use a nonoptimal conventional multiuser detector which shifts the entire complexity required to achieve a given performance level on the cross-correlation properties of the signal constellation. These systems suffer from near-far problem and the performance degradation is too severe as the number of users in the fixed system bandwidth grows. However, as interest in new wireless technologies (eg. PCS/PCN), where spread-spectrum has emerged as a *defacto* standard, continues to grow, a greater number of users need to be supported, while providing greater reliabilty and range of services in a fixed spectrum allocated. The suboptimum multistage detection strategy achieves significant performance gains over the conventional receiver in the range of desirable banwidth efficiencies, and is robust to near-far situations, while maintaining a reasonable computational complexity and fixed decoding delay. Clearly, accurate simulation tools are needed to accurately predict the performance gains achievable by any detection scheme proposed to be used in the next generation spread-spectrum systems. This project fulfils part of this need in the context of multistage detection strategy.

5.1 Extensions of this Research

The key features of the simulation and their justification have been presented in Section 3. The following extensions to the simulation software are desirable:

- The performance of the multistage detector has been examined for various hypothetical cases of user powers, in order to gain an insight of the funtioning of each stage in the detector, and to demonstrate the benefits achievable. These distributions of user powers do not occur in a practical spread-spectrum network with the use of power control by users. Simultion for a realistic case like log-normal distribution of user powers about their (equal) mean would be helpful in visualizing the performance gain achievable over conventional receiver in practical spread-spectrum systems.
- It is observed that the performance of higher order stages in the detector improves with increasing interference, i.e. better initial bit estimates for the interfering users.Therefore, alternate implementation of the first stage of the detector can be explored. Similarly, in cases where exact values of the energies is not available, use of suitable estimates can be examined.
- Extension of the receiver design to realistic multipath channels could be made. One could envision a multistage RAKE receiver which forms successive estimates of the significant multipath components from each user.

- The signature sequences most widely used in literature are m-sequences generated to maximize a signal-to-multiple-access-interference functional[Gab80],[Pur79]. These signature sequences could be used for simulation by modifying the ChipSequencesGenerator function in the software, so that the accuracy of predictions made by the software can be analyzed against a wealth of simulation data already accumulated over the years.
- The RandomNumberGenerator function in the code has been written primarily with the portability question in mind and may not be suitably matched to the architecture of a particular machine. The suitability of this generator for the machine on which simulations are performed should be verified by comparing the bit-error-rate results of detection in a Gaussian noise channel in the absence of multiple-access interference against the plot for $P_e = \frac{1}{2} erfc(\sqrt{\frac{E_b}{N_0}})$, which is the probability of error for binary PSK signalling in a Gaussian noise channel.

REFERENCES

[Cam92] Rick Cameron and Brian Woerner, "Performance of CDMA with imperfect power control,"*IEEE Veh. Tech. Conf.*, Denver, CO, May 1992.

[Cox91] D. C. Cox, "A radio system proposal for widespread low-power tetherless communications,"*IEEE Trans. Commun.*, vol. 39, No. 2, Feb. 1991, pp. 324-335.

[Gab80] F. D. Gaber, and M. B. Pursley, "Optimal phases of maximal sequences for asynchronous spread-spectrum multiplexing," *IEE Electron. Lett.*, vol. 16, Sept. 1980, pp. 756-757.

[Ger82] E. A. Geraniotis, and M. B. Pursley, "Error probability for direct-sequence spread-spectrum multiple-access communications - Part II: Approximations," *IEEE Trans. Commun.*, vol. COM-30, May 1982, pp. 985-995.

[Hay83] S. Haykin, "*Communication Systems*," New York: Wiley, 1983.

[Leh87] J. S. Lehnert, and M. B. Pursley, "Error probability for binary direct-sequence spread-spectrum communications with random signature sequences," *IEEE Trans. Commun.*, vol. COM-35, Jan. 1987, pp. 87-98.

[Lup90] R. Lupas, and S. Verdu, "Near-far resistance of multiuser detectors in asynchronous channels," *IEEE Trans. Commun.*, vol. 38, No. 4, April 1990, pp. 496-508.

[Pro89] J. G. Proakis, "*Digital Communications*," New York: McGraw-Hill, 1989.

[Pur77] M. B. Pursley, "Performance evaluation for phase-coded spread-spectrum multiple-access communication, Part I: System analysis," *IEEE Trans. Commun.*, vol. COM-25, Aug. 1977, pp. 795-799.

[Pur79] M. B. Pursley, and H. F. A. Roefs, "Numerical evaluation of correlation parameters for optimal phases of binary shift-register sequences," *IEEE Trans. Commun.*, vol. COM-27, Oct. 1979, pp. 1597-1604.

[Pur81] M. B. Pursley, "Spread-spectrum multiple-access communications," in *Multi User Communications*, G. Longo, Ed. New York: Springer-Verlag, 1981, 139-199.

[Pur82] M. B. Pursley, D. V. Sarwate, and W. E. Stark, "Error probability for direct-sequence spread-spectrum multiple-access communications - Part I: Upper and lower bounds," *IEEE Trans. Commun.*, vol. COM-30, May 1982, pp. 975-984.

[Var90] M. K. Varanasi, and B. Aazhang, "Multistage Detection in asynchronous code-division multiple-access communications," *IEEE Trans. Commun.*, vol. 38, No. 4, April 1990, pp. 509-519.

[Ver84] S. Verdu, "Optimum multi-user signal detection," Ph.D. dissertation, Dep. Elec. Comp. Eng., Univ. Illinois, Urbana-Champaign, 1984.

[Ver86a] S. Verdu, "Minimum probability of error for asynchronous Gaussian multiple-access channels," *IEEE Trans. Inform. Theory*, vol. IT-32, Jan. 1986, pp. 85-96.

[Ver86b] S. Verdu, "Optimum multiuser asymptotic efficiency," *IEEE Trans. Comm.*, vol. COM-34, Sept. 1986, pp. 890-897.

APPENDIX

C CODE AND OPERATING INSTRUCTIONS FOR THE MULTISTAGE DETECTOR SOFTWARE

OPERATING INSTRUCTIONS FOR THE MULTISTAGE DETECTOR SOFTWARE

If the receiver code is stored in a file called receiver.c, then the following series of commands should run the desired simulation on a UNIX system

```
cc receiver.c -o receiver -lm
nohup receiver 7 3 1 2 4 &
```

The first command compiles the C code and forms an executable file by the name *receiver*

The second command uses the nohangup command to run the simulation as a background process without terminating the simulation if the user ends the session. The second command line argument after the executable filename viz., 7 is taken as the number of stages in the receiver, the third command line argument of 3 is taken as the number of users in the system, and the remaining arguments specify the relative powers of all the users in integer/decimal notation. The desired user is the first user. Thus the above case specifies simulation for a 7 stage receiver with three users in the system and the two interferers at a power level of 3dB and 6dB above that of the desired user. The results for the desired user are written in a file with the same name as the executable file but with a '.r' extension. Thus in the above case, a file with the name *receiver.r* will be created and the SNR vs. BER of the first user written into it. The simulation takes about a week to run, and the activeness of the process can be checked by the *ps* command on the UNIX systems.

```
/***************************************************************************/
/*    PROGRAM TO SIMULATE A MULTISTAGE CDMA RECEIVER
  */
/***************************************************************************/

#include "header.c"
/* include the header file containing relevant type definitions and certain constants */

/* The following global variables are needed */
int NumberOfReceiverStages = 0, NumberOfUsers = 0, CurrentBitSubscript = 0, seed = 1;
double *RelativePowers, *RelativeDelays, *RelativePhases;




/* Declaration of functions used later. For ANSI C compilers, define the function prototypes here */

int ChipSequencesGenerator();
int PartialCrossCorrelationsGenerator();
int RandomDecisionStatisticisticsGenerator();
int MultistageDetector();
double RandomNumberGenerator();
double R_value();
double R_Hat_value();




int
main(argc, argv)
int argc;
char *argv[];
{

        /* The data structures of the following local variables are shown separately */

        int DecimalPlaces=0, SNR1dB, smallest, SufficientErrors=0;
        register long int i, j, k, l;
        double UserPower=0.0, temp;
        char c, *prog=argv[0];       /* the first command line argument is the name by which the
                                        function is invoked */
        int **GoldSequences;
        double *StagewiseErrors;
        struct Bit **DataBitsSent, **DemodulatedDataBits;
        struct Statistic *DecisionStatistics, **DelayLines;
        struct CrossCorrelation **PartialRs;
        struct ThreeBitStore **BitEstimates;
        FILE *fp;
```

```
if (argc-- == 1){
        printf("\n\n\t Number of Users and Number of Stages in the Receiver not specified \n\t
                The default setting is 2 users with equal Powers and 2 Stages in the Receiver
                \n\n");
        NumberOfReceiverStages = 2;
        NumberOfUsers = 2;
        RelativePowers = (double *) calloc(2, sizeof(double));
        RelativePowers[0] = RelativePowers[1] = 1.0;
}
else{

        /* The 2nd command line argument is the number of stages in the receiver */

        for (argv++, argc--, i = 0; (c = (*argv)[i]) != '\0'; i++){
                if (c < '0' || c > '9'){
                        fprintf(stderr, "\n\n Error reading # of stages: integer expected \n\n");
                        exit(1);
                }
                else
                        NumberOfReceiverStages = NumberOfReceiverStages*10 + (c-'0');
        }

        /* The 3rd command line argument is the number of users in the system */

        argv+=1; argc-=1;
        while (c= *(argv[0]++)){
                if (c < '0' || c > '9'){
                        fprintf(stderr, "\n\n Error reading # of Users: integer expected \n\n");
                        exit(-1);
                }
                else
                        NumberOfUsers = NumberOfUsers*10 + (c-'0');
        }

        /* check if the relative power of all users is specified */

        if (argc != NumberOfUsers){
                printf("\n Usage: %s < # of Receiver Stages  # of Users  Relative Power of
                       User 1 ... >\n", prog);
                exit(-1);
        }

        /* Read the relative power of each user in integer or decimal format */

        RelativePowers = (double *) calloc(NumberOfUsers, sizeof(double));
        for (; *++argv != NULL; argc--){
                while (c = *(argv[0]++)){
                        if ((c < '0' || c > '9') && (c != '.')){
                                fprintf(stderr, "\n\n Error reading Relative Power of User %d:
                                        a positive number expected \n\n", i);
                                        exit(-1);
                                }
                                else if (c == '.')
                                        DecimalPlaces = 1;
                                else if (DecimalPlaces == 0)
                                        UserPower = UserPower*10.0 + (c-'0');
                                else{
                                        UserPower = UserPower + (c-'0') / (pow(10.0, ((double)
                                                        DecimalPlaces)));
                                        DecimalPlaces +=1;
                                }
                        }
                        RelativePowers[NumberOfUsers-argc] = UserPower;
                        UserPower = 0.0;
                        DecimalPlaces = 0;
                }

}

/* List all the command line arguments read */

printf("\n\n Simulating the Suboptimum %2d Stage Receiver for the case of %2d Users with
        Relative Powers ", NumberOfReceiverStages, NumberOfUsers);
for (i = 0; i < NumberOfUsers; i++)
        printf("%.3f ", RelativePowers[i]);
printf("\n\n");



/* Allocate memory for all the relevant variables */



/* RelativeDelays is an array of size NumberOfUsers storing user delays as floating point
numbers between 0 and BitDuration */

if ((RelativeDelays = (double *) calloc(NumberOfUsers, sizeof(double))) == NULL){
        fprintf(stderr, "\n\n %s: Cannot allocate space for relative delays of users \n\n", prog);
        exit(-2);
}


/* RelativePhases is an array of size NumberOfUsers storing user phases as floating point
numbers between 0 and 2π */

if ((RelativePhases = (double *) calloc(NumberOfUsers, sizeof(double))) == NULL){
        fprintf(stderr, "\n\n %s: Cannot allocate space for relative phases of users \n\n", prog);
        exit(-2);
}
```

```
/* GoldSequences is an array of size NumberOfUsers, i_th element of which is a pointer to an
array of integers {-1,1} of size ChipSequencesLength storing the signature sequence of the
i_th user */

if ((GoldSequences = (int **) calloc(NumberOfUsers, sizeof(int *))) == NULL){
        fprintf(stderr, "\n\n %s: Cannot allocate space for chip sequences \n\n", prog);
        exit(-2);
}
for (i = 0; i < NumberOfUsers; i++)
        if ((GoldSequences[i] = (int *) calloc(ChipSequencesLength, sizeof(int))) == NULL){
                fprintf(stderr, "\n\n %s: Cannot allocate space for chip sequences \n\n", prog);
                exit(-2);
        }


/* PartialRs is an array of size NumberOfUsers, i_th element of which is a pointer to an
array of size ChipSequencesLength the j_th element of which stores the partial crosscorrelation
(between the i_th and the j_th user) values R and R_hat  */

if ((PartialRs = (struct CrossCorrelation **) calloc(NumberOfUsers, sizeof(struct
    CrossCorrelation *))) == NULL){
        fprintf(stderr, "\n\n %s: Cannot allocate space for signal cross-correlation values \n\n",
               prog);
        exit(-2);
}
for (i = 0; i < NumberOfUsers; i++)
        if ((PartialRs[i] = (struct CrossCorrelation *) calloc(NumberOfUsers, sizeof(struct
            CrossCorrelation))) == NULL){
                                fprintf(stderr, "\n\n %s: Cannot allocate space for signal
                                        cross-correlation values \n\n", prog);
                                exit(-2);
        }


/* DataBitsSent is an array of size 2*NumberOfReceiverStages, each element of which is a
pointer to an array of bits {-1,1} of size NumberOfUsers j_th element of which is the
information bit transmitted by the j_th user in a particular bit interval */

if ((DataBitsSent = (struct Bit **) calloc(2*NumberOfReceiverStages, sizeof(struct Bit *))) ==
    NULL){
        fprintf(stderr, "\n\n %s: Cannot allocate space for transmitted data bits \n\n", prog);
        exit(-2);
}
for (i = 0; i < 2*NumberOfReceiverStages; i++)
        if ((DataBitsSent[i] = (struct Bit *) calloc(NumberOfUsers, sizeof(struct Bit))) ==
            NULL){
                fprintf(stderr, "\n\n %s: Cannot allocate space for transmitted data bits \n\n",
                       prog);
                exit(-2);
```

```
          }

/* DecisionStatistics is an array of size NumberOfUsers i_th element of which is the decision
statistic for the i_th user during a particular bit interval */

if ((DecisionStatistics = (struct Statistic *) calloc(NumberOfUsers, sizeof(struct Statistic))) ==
     NULL){
          fprintf(stderr, "\n\n %s: Cannot allocate space for decision statistics of the receiver \n\n",
                    prog);
          exit(-2);
}

/* DelayLines implements the delay lines in the receiver structure. It is an array of size
NumberOfUsers i_th element of which is a pointer to an array of size
(2*NumberOfReceiverStages-2) which stores the received decision statistics of the i_th user in
the past (2*NumberOfReceiverStages-2) bit intervals */

if ((DelayLines = (struct Statistic **) calloc(NumberOfUsers, sizeof(struct Statistic *))) ==
     NULL){
          fprintf(stderr, "\n\n %s: Cannot allocate space for delay lines in the multistage detector
                    \n\n", prog);
          exit(1);
}
for (i = 0; i < NumberOfUsers; i++)
          if ((DelayLines[i] = (struct Statistic *) calloc(2*NumberOfReceiverStages-2,
               sizeof(struct Statistic))) == NULL){
                    fprintf(stderr, "\n\n %s: Cannot allocate space for delay lines in the multistage
                              detector \n\n", prog);
                    exit(-2);
          }

/* BitEstimates implements the three bit stores in the receiver structure. It is an array of size
NumberOfUsers i_th element of which is a pointer to an array of size (NumberOfReceiverStages-
1) which implements the three bit stores for the i_th user */

if ((BitEstimates = (struct ThreeBitStore **) calloc(NumberOfUsers, sizeof(struct ThreeBitStore
    *))) == NULL){
          fprintf(stderr, "\n\n %s: Cannot allocate space for the bit estimates of the multistage
                    detector \n\n", prog);
          exit(1);
}
for (i = 0; i < NumberOfUsers; i++)
          if ((BitEstimates[i] = (struct ThreeBitStore *) calloc(NumberOfReceiverStages-1,
               sizeof(struct ThreeBitStore))) == NULL){
                    fprintf(stderr, "\n\n %s: Cannot allocate space for bit estimates of the
                              multistage detector \n\n", prog);
                    exit(-2);
```

```
        }
/* DataBitsSent is an array of size NumberOfReceiverStages, each element of which is a pointer
to an array of bits {-1,1} of size NumberOfUsers j_th element of which is the demodulated bit
obtained from the receiver for the j_th user during a particular bit interval */

if ((DemodulatedDataBits = (struct Bit **) calloc(NumberOfReceiverStages, sizeof(struct Bit *)))
    == NULL){
        fprintf(stderr, "\n\n %s: Cannot allocate space for demodulated data bits of different
                stages\n\n", prog);
        exit(-2);
}
for (i = 0; i < NumberOfReceiverStages; i++)
        if ((DemodulatedDataBits[i] = (struct Bit *) calloc(NumberOfUsers, sizeof(struct Bit)))
            == NULL){
                fprintf(stderr, "\n\n %s: Cannot allocate space for demodulated data bits of
                        different stages \n\n", prog);
                exit(-2);
        }


/* StagewiseErrors is an array of size NumberOfReceiverStages, i_th element of which stores the
number of errors of the i_th user for the particular SNR ratio in the current simulation  */

if ((StagewiseErrors = (double *) calloc(NumberOfReceiverStages, sizeof(double))) == NULL){
        fprintf(stderr, "\n\n %s: Cannot allocate space for user errors \n\n", prog);
        exit(-2);
}


/* Open a file for writing results of SNR vs. bit-error-rates of different stages. The file name is
the name by which the program is invoked, with a ".r" extension */

if ((fp = fopen(strcat(prog,".r"), "w+")) == NULL){
        fprintf(stderr, "\n\n %s: error opening file for writing results\n\n", prog);
        exit(-2);
}
fprintf (fp, "\n\n \t SNR1 (in dB)");
for (i = 0; i < NumberOfReceiverStages; i++)
        fprintf (fp, "\t BER of Stage %d", (i+1));
fprintf (fp, "\n");



/* START THE SIMULATION NOW */

for (SNR1dB = SNR1dBmin; SNR1dB <= SNR1dBmax; SNR1dB+=1){

        for (i = 0; i < NumberOfReceiverStages; i++)
                StagewiseErrors[i] = 0.0;

        /* initialize the random number generator */
        temp = RandomNumberGenerator(-1);

        SufficientErrors = 0;


/* For a particular SNR repeat until NumberOfErrorsGenerated (a constant defined in
the header file) errors have been generated for all users */

        for (j=0; SufficientErrors == 0; j++){

                /* Generate signature sequences for each user */
                if (ChipSequencesGenerator(GoldSequences) != NumberOfUsers){
                        perror("\n\n Warning: Error in generation of chip sequences \n\n");
                        exit(-3);
                }




        /* Generate user delays to be uniformly distributed random numbers between 0
        and T (bit duration), and user phases to be uniformly distributed random
        numbers between 0 and 2π. Number the users according to increasing delays,
with user 1 (the desired user) as the reference */

                RelativeDelays[0] = RelativePhases[0] = 0.0;
                for (k = 1; k < NumberOfUsers; k++){
                        RelativePhases[k] = 2.0 * (22.0/7.0) *
                                                RandomNumberGenerator(seed);
                        RelativeDelays[k] = BitDuration * RandomNumberGenerator(seed);
                }
                for (k = 1; k < NumberOfUsers-1; k++){
                        smallest = k;
                        for (l = k+1; l < NumberOfUsers; l++)
                                if (RelativeDelays[l] < RelativeDelays[smallest])
                                        smallest = l;
                        if (smallest != k){
                                temp = RelativeDelays[k];
                                RelativeDelays[k] = RelativeDelays[smallest];
                                RelativeDelays[smallest] = temp;
                        }
                }


        /* Generate R and R_hat values for all pairs of user signature sequences */

                if (PartialCrossCorrelationsGenerator(GoldSequences, PartialRs) !=
                    NumberOfUsers){
                        perror("\n\n Warning: Error in calculation of chip sequences cross-
                                correlation values \n\n");
                        exit(-4);
```

```
}
/* Generate the first two transmitted bits. No decision statistics are calculated
corresponding to these bits and they are not demodulated either */
for (i = 0; i < 2; i++){
        for (k = 0; k < NumberOfUsers; k++)
                DataBitsSent[CurrentBitSubscript][k].Value =
                        (RandomNumberGenerator(seed) > 0.5) ? 1: -1;
        if (debug == 1)
                for (k = 0; k < NumberOfUsers; k++)
                        DataBitsSent[CurrentBitSubscript][k].ID =
                                                        i * NumberOfUsers + k;

        CurrentBitSubscript = ((CurrentBitSubscript + 1) %
                                        (2 * NumberOfReceiverStages));
}




/* Next, generate 3*(NumberOfReceiverStages-1) bits, the corresponding
decision statistics, and pass them through the receiver. The purpose is to fill up
the three bit stores at the receiver so that decisions on demodulated data bits
are available from all stages of the receiver when receiver functioning is
started */

for (i = 0; i < 3*(NumberOfReceiverStages-1); i++){

        for (k = 0; k < NumberOfUsers; k++)
                DataBitsSent[CurrentBitSubscript][k].Value =
                        (RandomNumberGenerator(seed) > 0.5) ? 1: -1;
        if (debug == 1)
                for (k = 0; k < NumberOfUsers; k++)
                        DataBitsSent[CurrentBitSubscript][k].ID =
                                        (i+2) * NumberOfUsers + k;


        if (RandomDecisionStatisticsGenerator(DataBitsSent, PartialRs,
           DecisionStatistics, SNR1dB) != NumberOfUsers){
                perror("\n\n WARNING: Error in generation of decision
                        statistics \n\n");
                exit(-5);
        }

        if (MultistageMultiuserDetector(DecisionStatistics, DelayLines,
            PartialRs, BitEstimates, DemodulatedDataBits, SNR1dB) !=
            NumberOfReceiverStages){
                perror("\n\n WARNING: error in demodulation of user bits
                        \n\n");
                exit(-6);
        }
        CurrentBitSubscript = ((CurrentBitSubscript+1) %
                                (2*NumberOfReceiverStages));
}


 /* Next, generate transmitted data bits for each user, generate corresponding
decision statistics from equation (3.1-1) which adds on noise and multiple-
access interference from other users as the data bits pass through the
Gaussian-Noise multiuser channel, invoke the receiver function to get
decisions on demodulated data bits of various users from each stage. Repeat
for 100 or so bits before generating new delays, phases and signature
sequences */

for (i = 0; i < DelayChangeInterval; i++){

        for (k = 0; k < NumberOfUsers; k++)
                DataBitsSent[CurrentBitSubscript][k].Value =
                        (RandomNumberGenerator(seed) > 0.5) ? 1: -1;
        if (debug == 1)
                for (k = 0; k < NumberOfUsers; k++)
                        DataBitsSent[CurrentBitSubscript][k].ID =
                                (i+2+3*(NumberOfReceiverStages-1))*
                                NumberOfUsers + k;

        if (RandomDecisionStatisticsGenerator(DataBitsSent, PartialRs,
           DecisionStatistics, SNR1dB) != NumberOfUsers){
                perror("\n\n WARNING: Error in generation of decision
                        statistics \n\n");
                exit(-5);
        }

        if (MultistageMultiuserDetector(DecisionStatistics, DelayLines,
            PartialRs, BitEstimates, DemodulatedDataBits, SNR1dB) !=
            NumberOfReceiverStages){
                perror("\n\n WARNING: error in demodulation of user bits
                        \n\n");
                exit(-6);
        }


        /* Compare the decisions on the demodulated data bits from various
        stages against the corresponding transmitted data bits. Increment the
        number of errors at a particular stage if the bit estimate at that stage
        does not match the corresponding transmitted bit */

        for (l = 0; l < NumberOfReceiverStages; l++){

                if (DemodulatedDataBits[l][0].Value !=
                    DataBitsSent[(CurrentBitSubscript - (2*l+1) +
                    2*NumberOfReceiverStages) %
                   (2*NumberOfReceiverStages)][0].Value)
```

```
                                        StagewiseErrors[l] += 1.0;

                                if (debug == 1)
                                        for (k = 0; k < NumberOfUsers; k++)
                                                if (DataBitsSent[(CurrentBitSubscript -
                                                  (2*l+1) + 2*NumberOfReceiverStages) %
                                                  (2*NumberOfReceiverStages)][k].ID !=
                                                   DemodulatedDataBits[l][k].ID){
                                                perror("\n\n\n Error in the functioning of
                                                        the Receiver: demodulated bits not
                                                        being compared against the
                                                        appropriate transmitted bits
                                                        \n\n\n");
                                                exit(-6);
                                        }

                        }

                        CurrentBitSubscript = ((CurrentBitSubscript+1) %
                                                        (2*NumberOfReceiverStages));

                } /* end DelayChangeInterval */

                /* Check if sufficient errors have been generated. If yes, terminate iterating for
                the current value of SNR and proceed with the next */

                SufficientErrors = 1;
                for (l = 0; l < NumberOfReceiverStages; l++)
                        if (StagewiseErrors[l] < NumberOfErrorsGenerated){
                        SufficientErrors = 0;
                        break;
                }

        } /* end iterating for current SNR value */

        /* Calculate the error probabilities for the current SNR and write the results in the
        result file */
        fprintf (fp, "\n \t %d", SNR1dB);
        for (i = 0; i < NumberOfReceiverStages; i++)
                fprintf (fp, " \t %e", StagewiseErrors[i] / (((double) j)  * DelayChangeInterval));

    } /* finished iterating for all SNR values */

    fclose(fp);
    return 0;

} /* end main */
```

```
/*************************************************************************************/
/*        The following function generates uniform random numbers in the range 0 to 1 using a linear
congruational algorithm. See Numerical Recipes for more information. The constants ia1, ia2, ia3, ic1,
ic2, ic3, m1, m2, m3 should be matched to the system architecture for obtaining good deviates
                                                                                                    */
/*************************************************************************************/


double
RandomNumberGenerator(idum)
int idum;
{
        long int m1 = 259200, m2 = 134456, m3 = 243000;
        long int ia1 = 7141, ia2 = 8121, ia3 = 4561;
        long int ic1 = 54773, ic2 = 28411, ic3 = 51349;
        static long int Ix1, Ix2, Ix3;
        static double RanArr[97];
        int j;
        double ret_value;

        if (idum < 0){
                /* Initialize the generator for arguments < 0 */
                Ix1 = (ic1 - idum) % m1;
                Ix1 = (ia1 * Ix1 + ic1) % m1;
                Ix2 = Ix1 % m2;
                Ix1 = (ia1 * Ix1 + ic1) % m1;
                Ix3 = Ix1 % m3;

                for (j = 0; j < 97; j++){
                        Ix1 = (ia1 * Ix1 + ic1) % m1;
                        Ix2 = (ia2 * Ix2 + ic2) % m2;
                        RanArr[j] = ((double) Ix1 + ((double) Ix2) / ((double) m2)) / ((double) m1);
                }
                idum = 1;
        }
        Ix1 = (ia1 * Ix1 + ic1) % m1;
        Ix2 = (ia2 * Ix2 + ic2) % m2;
        Ix3 = (ia3 * Ix3 + ic3) % m3;
        j = (int) ((97 * Ix3) / m3);

        if ((j > 96) || (j < 0)){
                fprintf(stderr, "\n\n Error in generation of Random Number index \n\n");
                exit(-7);
        }

        ret_value = RanArr[j];

        if ((ret_value < 0.0) || (ret_value > 1.0)){
                fprintf(stderr, "\n\n Error in generation of Random Numbers \n\n");
                exit(-7);
        }
        RanArr[j] = ((double) Ix1 + ((double) Ix2) / ((double) m2)) / ((double) m1);

        return ret_value;
} /* end function RandomNumberGenerator */


/***********************************************************************************/
/*The following function generates Gold Sequences of ChipSequencesLength for all the K users using a
pair of preferred m-sequences as shown in Figure 3.2-1. The 5-stage shift register connections are
specific to these two preferred sequences. In order to use other sets of chip sequences, this function can
be modified to read in the shift-register connections from a file. The constants
NumberOfShiftRegisterStages and ChipSequenceLength defined in the header file should be modified
accordingly.                                                                      */
/***********************************************************************************/


int
ChipSequencesGenerator(GoldSequences)
int **GoldSequences;
{
        int i, j, k, remainder, RandomBase;
        struct FlipFlop{
                unsigned int Old; /* The value of the previous bit stored in the flip-flop */
                unsigned int New; /* The value of the new bit to be stored in the flip-flop */
                unsigned int InputMod2; /* Whether a stage of the shift-register has modulo-2
                                        connector */
        } ShiftRegister1[NumberOfShiftRegisterFFs], ShiftRegister2[NumberOfShiftRegisterFFs];


        if (NumberOfUsers > (ChipSequencesLength+2)){
                        perror("\n\n This program is designed to operate with 31 bit Gold chip
                                sequences. Number of users cannot exceed 31 + 2 = 33 \n\n");
                        exit(3);
                }

/* The following connections in the two shift-registers are specific to Gold Sequences of length 31 */

        for (i = 0; i < 5; i++)
                ShiftRegister1[i].InputMod2 = 0;
        ShiftRegister1[1].InputMod2 = 1;

        for (i = 1; i < 4; i++)
                ShiftRegister2[i].InputMod2 = 1;
        ShiftRegister2[0].InputMod2 = ShiftRegister2[4].InputMod2 = 0;

        RandomBase = (int) floor((ChipSequencesLength+1) * RandomNumberGenerator(seed));

        for (i = 0; i < NumberOfUsers; i++){
```

```
if ((i+RandomBase) != (ChipSequencesLength+1)){

        /* The first shift register is initialized to contain binary 1 */

        for (j = 1; j < NumberOfShiftRegisterFFs; j++)
                ShiftRegister1[j].Old = 0;
        ShiftRegister1[0].Old = 1;
        /* The second shift register is initialized to contain binary equivalent of a
        random number with value between 0 and (ChipSequencesLength-1), so that
        the function unbiased chip sequences */

        remainder = ((i+RandomBase) % (ChipSequencesLength+2));
        if (remainder > 31)
                return (-1);
        for (j = NumberOfShiftRegisterFFs-1; remainder != 0; j--){
                if ((remainder - (k = ((int) pow(2.0, (double) j)))) >= 0){
                        ShiftRegister2[j].Old = 1;
                        remainder -= k;
                }
                else
                        ShiftRegister2[j].Old = 0;
        }
        if (j >= 0)
                for (; j >= 0; j--)
                        ShiftRegister2[j].Old = 0;
        if ((remainder != 0) || (j >= 0))
                return (-1);
}
else{
        /* supply the second preferred m-sequence as one chip-sequence */
        for (j = 0; j < NumberOfShiftRegisterFFs; j++){
                ShiftRegister1[j].Old = 0;
                ShiftRegister2[j].Old = 0;
        }
        ShiftRegister2[0].Old = 1;
}

for (k = 0; k < ChipSequencesLength; k++){
        GoldSequences[i][k] = (((ShiftRegister1[0].Old + ShiftRegister2[0].Old) % 2)
                                ==1) ? 1 : -1;
        for (j = 0; j < NumberOfShiftRegisterFFs-1; j++){
                ShiftRegister1[j].New = ((ShiftRegister1[j+1].Old +
                                        ShiftRegister1[j].InputMod2 *
                                        ShiftRegister1[0].Old) % 2);
                ShiftRegister2[j].New = ((ShiftRegister2[j+1].Old +
                                        ShiftRegister2[j].InputMod2 *
                                        ShiftRegister2[0].Old) % 2);
        }
        ShiftRegister1[NumberOfShiftRegisterFFs-1].New = ShiftRegister1[0].Old;
        ShiftRegister2[NumberOfShiftRegisterFFs-1].New = ShiftRegister2[0].Old;
        for (j = 0; j < NumberOfShiftRegisterFFs; j++){
```

```
                    ShiftRegister1[j].Old = ShiftRegister1[j].New;
                    ShiftRegister2[j].Old = ShiftRegister2[j].New;
                }
            } /* finished with generation of all chips of this user */

        } /* end for all users */
        return i;
} /* end function ChipSequenceGenerator */



/***************************************************************************************/
/*      The following function generates the partial crosscorrelation functions R and R_hat defined by
        equations 3.1-2 and 3.1-3 respectively, for all pairs of users {(i,j)/ i>j} since only the elements in
        the lower half of the crosscorrelation matrix are needed in calculation of multiple-access
        interference component of decision statistic in equation 3.1-1
*/
/***************************************************************************************/


int
PartialCrossCorrelationsGenerator(GoldSequences, PartialRs)
int **GoldSequences;
struct CrossCorrelation **PartialRs;
{
        int i, k;
        double tau,  PowerTerm, CosTerm, CrossCorrelationTerm;
        extern double *RelativeDelays, *RelativePhases;


        for (i = 0; i < NumberOfUsers; i++){

                for (k = 0; k < i; k++){

                        CosTerm = cos(RelativePhases[i] - RelativePhases[k]);

                        PartialRs[i][k].R_hat =  R_Hat_value(GoldSequences[i], GoldSequences[k],
                                                        (RelativeDelays[i]-RelativeDelays[k]));
                        PartialRs[i][k].R =  R_value(GoldSequences[i], GoldSequences[k],
                                                        (RelativeDelays[i]-RelativeDelays[k]));

                }
        }

        return i;
} /* end function PatialCrossCorrelationGenerator */
```

```
/*************************************************************************************/
/*      The following function generates the decision statistics for all users corresponding to the bit
        transmitted in the previous bit interval (the bits transmitted in the current bit interval are used
        in calculation of multiple-access interference parts) using equation 3.1-1.
*/
/*************************************************************************************/

int
RandomDecisionStatisticsGenerator(DataBitsSent, PartialRs, DecisionStatistics, SNR1dB)
struct Bit **DataBitsSent;
struct CrossCorrelation **PartialRs;
struct Statistic *DecisionStatistics;
int SNR1dB;
{
        int i, j, k;
        static int flag = 0, Reserve;
        double  CosTerm, PowerTerm, CrossCorrelationTerm, GaussianNoiseTerm, v1, v2, r, factor,
                InterferenceTerm, UsefulTerm;


        for (i=0; i < NumberOfUsers; i++){

                /* Calculate the Noise Part as a Gaussian deviate with mean 0 and variance T/4. The
                noise variance N0 appears as a multiplicative factor in all terms on the right side of the
                decision statistic in eq. (3.1-1) and is therefore omitted. Also the cos(2π*X2) and
                sin(2π*X2) terms have been replaced by v1*ln(r)/r and v2*ln(r)/r repectively, where
                X2 is uniform deviate between 0 and 1, v1 and v2 are uniform deviates between -1 and
                +1, and r is a uniform deviate lying in the unit circle. See 'Numerical Recipes in C' for
                this method of generation of Gaussian deviates */

                if (flag == 0){
                        do {
                                v1 = 2.0 * RandomNumberGenerator(seed) - 1.0;
                                v2 = 2.0 * RandomNumberGenerator(seed) - 1.0;
                                r = v1 * v1 + v2 * v2;
                        } while ((r >= 1.0) || (r <= 0.0));

                        factor = sqrt(-1.0 * BitDuration * log(r) / r);
                        Reserve = v1 * factor; /* Store one deviate for future use */
                        GaussianNoiseTerm = v2 * factor; /* and return the other */
                        flag = 1;
                }
                else{
                        GaussianNoiseTerm = Reserve; /* Return the second Gaussian deviate */
                        flag = 0;
                }
                DecisionStatistics[i].Value = GaussianNoiseTerm;

                if (debug ==1) /* the ID of a decision statistic is the same as that of the corresponding
                                information bit */
                DecisionStatistics[i].ID = DataBitsSent[(CurrentBitSubscript-
                                                1+2*NumberOfReceiverStages)%
                                                (2*NumberOfReceiverStages)][i].ID;

            /* the information part of the decision statistic */

            DecisionStatistics[i].Value += sqrt(BitDuration * pow(10.0, ((double) SNR1dB)/10.0) *
                                                RelativePowers[i] / RelativePowers[0]) *
                                                DataBitsSent[(CurrentBitSubscript-
                                                1+2*NumberOfReceiverStages)%
                                                (2*NumberOfReceiverStages)][i].Value;


            /* Interference from users with smaller delays, the third term in eq.(3.1-1) */

            for (k = 0; k < i; k++){
                InterferenceTerm = DataBitsSent[(CurrentBitSubscript-
                        1+2*NumberOfReceiverStages)%
                        (2*NumberOfReceiverStages)][k].Value * PartialRs[i][k].R_hat;
                InterferenceTerm += DataBitsSent[CurrentBitSubscript][k].Value *
                                        PartialRs[i][k].R;
                InterferenceTerm *= sqrt(pow(10.0, ((double) SNR1dB)/10.0) *
                                        RelativePowers[k] / RelativePowers[0]) *
                                        cos(RelativePhases[i]-RelativePhases[k]);
                DecisionStatistics[i].Value += InterferenceTerm;

            }


            /* Interference from users with larger delays, the fourth term in eq.(3.1-1) */

            for (k = i+1; k < NumberOfUsers; k++){
                InterferenceTerm = DataBitsSent[(CurrentBitSubscript-
                        1+2*NumberOfReceiverStages)%
                        (2*NumberOfReceiverStages)][k].Value * PartialRs[k][i].R_hat;
                InterferenceTerm += DataBitsSent[(CurrentBitSubscript-
                                2+2*NumberOfReceiverStages)%
                                (2*NumberOfReceiverStages)][k].Value * PartialRs[k][i].R;
                InterferenceTerm *= sqrt(pow(10.0, ((double) SNR1dB)/10.0) *
                                        RelativePowers[k] / RelativePowers[0]) *
                                        cos(RelativePhases[i]-RelativePhases[k]);
                DecisionStatistics[i].Value += InterferenceTerm;
            }

        } /* end for all Users */

        return i;

} /* end function RandomDecisionStatisticsGenerator */
```

```
/*************************************************************************************/
    /*    The following function simulates the multistage detection scheme shown in Figure 2.4-2        */
/*************************************************************************************/


int
MultistageMultiuserDetector(DecisionStatistics, DelayLines, PartialRs, BitEstimates,
                              DemodulatedDataBits, SNR1dB)
struct Statistic * DecisionStatistics;
struct Statistic **DelayLines;
struct CrossCorrelation **PartialRs;
struct ThreeBitStore **BitEstimates;
struct Bit **DemodulatedDataBits;
int SNR1dB;
{
        int i, j, k, l, subscript;
        double temp, MultipleAccessInterferenceEstimate, InterferenceTerm;


        /* The first stage is the conventional receiver stage */
        for (l = 0; l < NumberOfUsers; l++)
                DemodulatedDataBits[0][l].Value = (DecisionStatistics[l].Value > 0) ? 1 : -1;
        if (debug == 1) /* The ID of the demodulated bit is same as that of the corresponding decision
                           statistic, which is the same as the corresponding transmitted information bit */
                for (l = 0; l < NumberOfUsers; l++)
                        DemodulatedDataBits[0][l].ID = DecisionStatistics[l].ID;

        /* Reconstruct the multiple access interference estimate for all users in each stage of the
        receiver from the esimates of other users' bits obtained from the preceding stage */

        for (j = 1; j < NumberOfReceiverStages; j++){

                for (i = 0; i < NumberOfUsers; i++){

                        /* Interference from users with higher delays */
                        MultipleAccessInterferenceEstimate = 0.0;
                        for (k = i+1; k < NumberOfUsers; k++){
                                InterferenceTerm = BitEstimates[k][j-1].Previous1.Value *
                                                        PartialRs[k][i].R_hat;
                        InterferenceTerm += BitEstimates[k][j-1].Previous2.Value * PartialRs[k][i].R;
                        InterferenceTerm *= sqrt(pow(10.0, ((double) SNR1dB)/10.0) *
                                                RelativePowers[k] / RelativePowers[0]) *
                                                cos(RelativePhases[i]-RelativePhases[k]);
                        MultipleAccessInterferenceEstimate += InterferenceTerm;
                        }

                        /* Interference from users with lower delays */
                        for (k = 0; k < i; k++){
                                InterferenceTerm = BitEstimates[k][j-1].Previous1.Value *
                                                        PartialRs[i][k].R_hat;
                        InterferenceTerm += BitEstimates[k][j-1].Current.Value * PartialRs[i][k].R;
                InterferenceTerm *= sqrt(pow(10.0, ((double) SNR1dB)/10.0) *
                                        RelativePowers[k] / RelativePowers[0]) *
                                        cos(RelativePhases[i]-RelativePhases[k]);
                MultipleAccessInterferenceEstimate += InterferenceTerm;
                }


                /* Subtract the reconstructed multiple-access interference estimate from the
                decision statistic for each user in each stage of the receiver */
                DemodulatedDataBits[j][i].Value = ((DelayLines[i][2*j-1].Value -
                                                MultipleAccessInterferenceEstimate) > 0) ?
                                                1 : -1;
                if (debug == 1){
                        DemodulatedDataBits[j][i].ID = DelayLines[i][2*j-1].ID;
                        if (DelayLines[i][2*j-1].ID != BitEstimates[i][j-1].Previous1.ID){
                                printf("\n\n Error in generation of
                                        MultipleAccessInterferenceEstimate \n\n");
                                exit(-6);
                        }
                }
        } /* end for all users */

        /* Update the three-bit store for this receiver stage for all users */
        for (l = 0; l < NumberOfUsers; l++){
                BitEstimates[l][j-1].Previous2.Value = BitEstimates[l][j-1].Previous1.Value;
                BitEstimates[l][j-1].Previous1.Value = BitEstimates[l][j-1].Current.Value;
                BitEstimates[l][j-1].Current.Value = DemodulatedDataBits[j-1][l].Value;
        }
        if (debug ==1)
                for (l = 0; l < NumberOfUsers; l++){
                        BitEstimates[l][j-1].Previous2.ID = BitEstimates[l][j-1].Previous1.ID;
                        BitEstimates[l][j-1].Previous1.ID = BitEstimates[l][j-1].Current.ID;
                        BitEstimates[l][j-1].Current.ID = DemodulatedDataBits[j-1][l].ID;
        }


} /* end for all receiver stages */

/* Update the delay line element values for all users */
for (k = 0; k < 2*NumberOfReceiverStages-2; k++)
        for (l = 0; l < NumberOfUsers; l++){
                temp = DelayLines[l][k].Value;
                DelayLines[l][k].Value = DecisionStatistics[l].Value;
                DecisionStatistics[l].Value = temp;
        }
if (debug == 1)
        for (k = 0; k < 2*NumberOfReceiverStages-2; k++)
                for (l = 0; l < NumberOfUsers; l++){
                        temp = DelayLines[l][k].ID;
                        DelayLines[l][k].ID = DecisionStatistics[l].ID;
                        DecisionStatistics[l].ID = temp;
```

```
                    }

        return j;

} /* end function MultistageMultiuserDetector */


/*************************************************************************************/
     /*     The following function generates the R crosscorrelation value between two chip sequences
            for a specified relative delay as given by eq.(3.1-2)                                         */
/*************************************************************************************/


double
R_value(ChipSequence1, ChipSequence2, Tau)
int *ChipSequence1;
int *ChipSequence2;
double Tau;
{

        int i, j, gamma, DiscreteCrossCorrelation1, DiscreteCrossCorrelation2;
        double ContinuousCrossCorrelation1, AutoCorrelation1, AutoCorrelation2;


        if (Tau < 0.0){
                printf ("\n\n \t Negative time difference between Chip Sequence delays illegal \n\n");
                return (-1);
        }

        gamma = (int) floor(Tau * ChipSequencesLength / BitDuration);

        AutoCorrelation1 = (Tau - gamma * BitDuration / ChipSequencesLength);
        AutoCorrelation2 = ((BitDuration / ChipSequencesLength) * (gamma+1) - Tau);

        DiscreteCrossCorrelation1 = 0;
        if ( fabs(gamma-ChipSequencesLength) < ChipSequencesLength)
                for (j = 0; j <= gamma-1; j++)
                        DiscreteCrossCorrelation1 += ChipSequence1[j-gamma+ChipSequencesLength]
                                                     * ChipSequence2[j];

        ContinuousCrossCorrelation1 = ((double) DiscreteCrossCorrelation1) * AutoCorrelation2;

        DiscreteCrossCorrelation2 = 0;
        if ( fabs(gamma+1-ChipSequencesLength) < ChipSequencesLength)
                for (j = 0; j <= gamma; j++)
                       DiscreteCrossCorrelation2 += ChipSequence1[j-gamma-1+ChipSequencesLength]
                                                    * ChipSequence2[j];

        ContinuousCrossCorrelation1 += ((double) DiscreteCrossCorrelation2) * AutoCorrelation1;
        if ((AutoCorrelation1 < 0) || (AutoCorrelation1 > BitDuration / ChipSequencesLength) ||
            (AutoCorrelation2 < 0) || (AutoCorrelation2 > BitDuration / ChipSequencesLength)){
                perror("\n\n Error in calculation of continuous-time partial Autocorrelation functions for
                        rectangular chip waveform \n\n");
                return(-1);
        }


        if ((fabs((double) DiscreteCrossCorrelation1) > ChipSequencesLength) || (fabs((double)
            DiscreteCrossCorrelation2) > ChipSequencesLength) ){
                perror("\n\n Error in calculation of discrete-time  CrossCorrelation functions for the
                        chip waveforms in Correlation1 \n\n");
                return(-1);
        }


        if (fabs(ContinuousCrossCorrelation1) > BitDuration){
                perror("\n\n Error in calculation of continuous-time partial CrossCorrelation function
                        for the chip waveforms \n\n");
                return(-1);
        }

        return ContinuousCrossCorrelation1;
} /* end function R_value */



/****************************************************************************************/
/*      The following function generates the R_hat crosscorrelation value between two chip sequences
        for a specified realtive delay as given by eq.(3.1-3)
*/
/****************************************************************************************/

double
R_Hat_value(ChipSequence1, ChipSequence2, Tau)
int *ChipSequence1;
int *ChipSequence2;
double Tau;
{

        int i, j, gamma, DiscreteCrossCorrelation1, DiscreteCrossCorrelation2;
        double ContinuousCrossCorrelation2, AutoCorrelation1, AutoCorrelation2;


        if (Tau < 0.0){
                printf ("\n\n \t Negative time difference between Chip Sequence delays illegal \n\n");
                return (-1);
        }

        gamma = (int) floor(Tau * ChipSequencesLength / BitDuration);
        AutoCorrelation1 = (Tau - gamma * BitDuration / ChipSequencesLength);
        AutoCorrelation2 = ((BitDuration / ChipSequencesLength) * (gamma+1) - Tau);

        DiscreteCrossCorrelation1 = 0;
        if (gamma < ChipSequencesLength)
                for (j = 0; j <= ChipSequencesLength-gamma-1; j++)
                        DiscreteCrossCorrelation1 += ChipSequence1[j] * ChipSequence2[j+gamma];

        ContinuousCrossCorrelation2 = ((double) DiscreteCrossCorrelation1) * AutoCorrelation2;

        DiscreteCrossCorrelation2 = 0;
        if ((gamma+1) < ChipSequencesLength)
                for (j = 0; j <= ChipSequencesLength-gamma-2; j++)
                        DiscreteCrossCorrelation2 += ChipSequence1[j] *
                                                        ChipSequence2[j+gamma+1];

        ContinuousCrossCorrelation2 += ((double) DiscreteCrossCorrelation2) * AutoCorrelation1;

        if ((AutoCorrelation1 < 0) || (AutoCorrelation1 > BitDuration / ChipSequencesLength) ||
           (AutoCorrelation2 < 0) || (AutoCorrelation2 > BitDuration / ChipSequencesLength)){
                perror("\n\n Error in calculation of continuous-time partial Autocorrelation functions for
                        rectangular chip waveform \n\n");
                return(-1);
        }


        if ((fabs((double) DiscreteCrossCorrelation1) > ChipSequencesLength) || (fabs((double)
            DiscreteCrossCorrelation2) > ChipSequencesLength) ){
                perror("\n\n Error in calculation of discrete-time  CrossCorrelation functions for the
                        chip waveforms in Correlation2 \n\n");
                return(-1);
        }


        if (fabs(ContinuousCrossCorrelation2) > BitDuration){
                perror("\n\n Error in calculation of continuous-time partial CrossCorrelation function
                        for the chip waveforms \n\n");
                return(-1);
        }

        return ContinuousCrossCorrelation2;
} /* end function R_hat_value */
```